# Recursive Multi-Agent Systems

Xiyuan Yang[1,*], Jiaru Zou[1,2,*†], Rui Pan[1], Ruizhong Qiu[1], Pan Lu[2], Shizhe Diao[3], Jindong Jiang[3], Hanghang Tong[1], Tong Zhang[1], Markus J. Buehler[4], Jingrui He[1] ✉, James Zou[2] ✉

[1]UIUC [2]Stanford University [3]NVIDIA [4]MIT
[*]Equal Contribution, Alphabetical Order [†]Project Lead ✉Corresponding Authors

**Project Page:** https://recursivemas.github.io



**Recursive or looped language models have recently emerged as a new scaling axis by iteratively refining the same model computation over latent states to deepen reasoning. We extend such scaling principle from a single model to multi-agent systems, and ask: *Can agent collaboration itself be scaled through recursion?* To this end, we introduce RecursiveMAS, a recursive multi-agent framework that casts the entire system as a unified latent-space recursive computation. RecursiveMAS connects heterogeneous agents as a collaboration loop through the lightweight RecursiveLink module, enabling in-distribution latent thoughts generation and cross-agent latent state transfer. To optimize our framework, we develop an inner-outer loop learning algorithm for iterative whole-system co-optimization through shared gradient-based credit assignment across recursion rounds. Theoretical analyses of runtime complexity and learning dynamics establish that RecursiveMAS is more efficient than standard text-based MAS and maintains stable gradients during recursive training. Empirically, we instantiate RecursiveMAS under 4 representative agent collaboration patterns and evaluate across 9 benchmarks spanning mathematics, science, medicine, search, and code generation. In comparison with advanced single/multi-agent and recursive computation baselines, RecursiveMAS consistently delivers an average accuracy improvement of 8.3%, together with 1.2×–2.4× end-to-end inference speedup, and 34.6%–75.6% token usage reduction.**

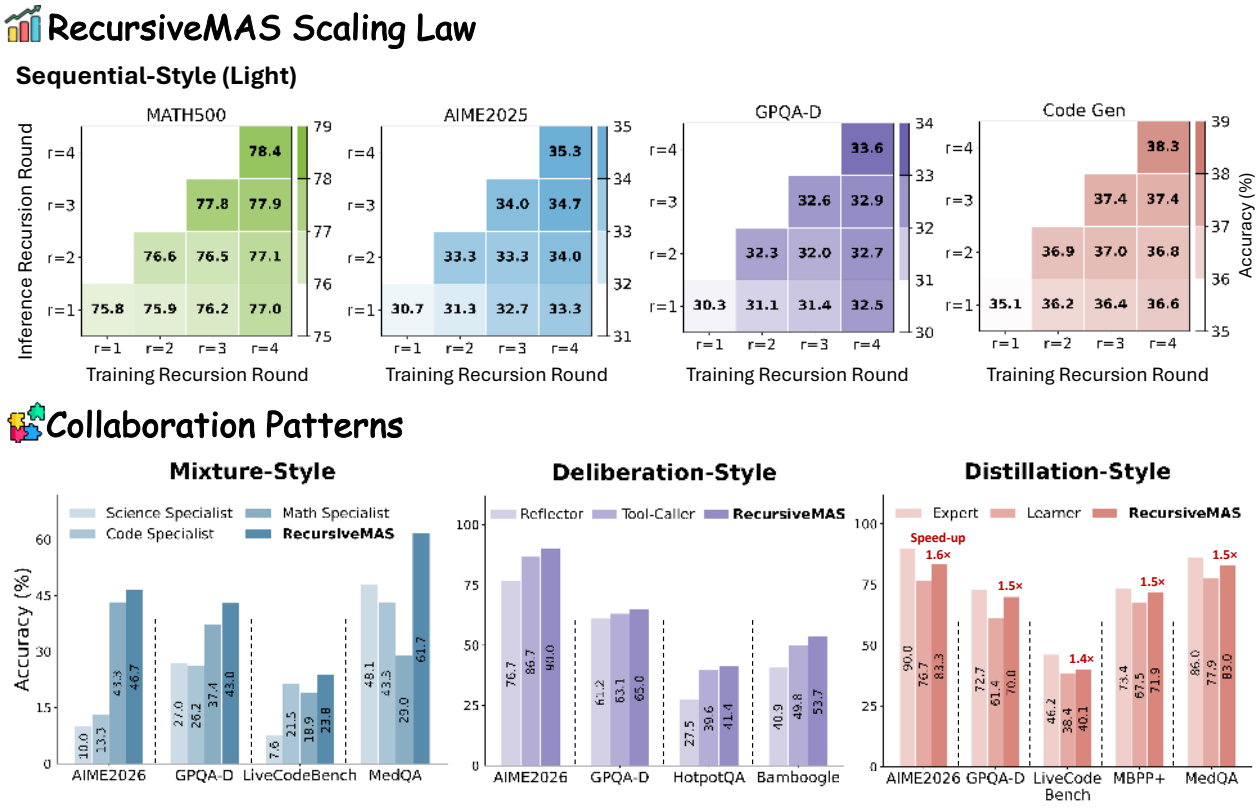


Figure 1 | **Performance Landscape of RecursiveMAS across Training/Inference Recursion Depths (Top):** The *lightweight* RecursiveMAS with sub-1.5B agents shows a clean scaling trend as recursion deepens. **Generalization across Common Collaboration Patterns (Bottom):** The *Scaled* RecursiveMAS with stronger LLM agents (5-10B) seamlessly adapts to diverse multi-agent system structures.

*Contact: jiaru@stanford.edu, xiyuany4@illinois.edu*

## 1. Introduction

To tackle complex tasks, a single language model often falls short due to limited capacity, myopic generation, or inefficient exploration of the solution space (Li et al., 2025b; Shojaee et al., 2025; Song et al., 2026). Once intelligence reaches a threshold, a natural direction is to treat individual models as specialized agents and organize them as a collaborative system (Tran et al., 2025; Xu et al., 2025). A multi-agent system (MAS) (Wang et al., 2025b; Wu et al., 2024) can scale performance by enabling individuals to work together and contribute complementary strengths. Consider a set of heterogeneous agents, each assigned a distinct role and expertise. The system can either arrange agents into a sequential pipeline (Gu et al., 2025; Qian et al., 2024) to progressively decompose and solve a problem, or engage and integrate multiple domain-specialized agents (Babu et al., 2025; Qian et al., 2025; Ye et al., 2025b) for the task.

While MAS establishes a structural foundation, the next question is how to enable the system to evolve over time and adapt to different scenarios. Prior work has explored prompt-based adaptation (Shen et al., 2025; Zhang et al., 2025b; Zhou et al., 2025), where model interactions are improved through the iterative refinement of shared context. Although these updated prompts can help agents generate more aligned responses to the question, each agent itself cannot improve. A more principled line of work is to optimize agents through learning (Motwani et al., 2024; Subramaniam et al., 2025; Zhao et al., 2025). However, training entire agents inside the system is hard, as updating all model parameters is non-trivial (Hu et al., 2025), and the sequential dependency in text-based interactions introduces substantial latency when agents must wait for others to complete generation.

Instead of improving each agent's capabilities as a standalone component, we adopt a higher-level learning perspective and aim to co-evolve and scale the entire system as an integrated whole. We recast agent collaboration through the lens of recursive language models (RLMs) (Jolicoeur-Martineau, 2025; Zhang et al., 2025a; Zhu et al., 2025), where a shared set of layers is iteratively applied and optimized within a continuous latent space. In this view, the entire multi-agent system can be treated as a recursive computation, where each agent acts like an RLM layer, iteratively passing latent representations to the next and forming a looped interaction process.

We call this new system-level agentic recursion framework **RecursiveMAS**. Without updating all model parameters, agents are connected and iteratively optimized solely via the lightweight **RecursiveLink**, a two-layer residual projection module for latent states transmission and refinement. An *inner RecursiveLink* within each agent first consolidates the model's ongoing latent thoughts between input and output spaces during auto-regressive generation. An *outer RecursiveLink* then bridges hidden representations across heterogeneous agents built on different model types and sizes, enabling seamless cross-agent interaction. Together, all agents are chained in a unified loop to perform iterative latent collaboration, with only the last agent producing the textual output in the final recursion round.

Correspondingly, we pair RecursiveMAS with an **Inner-Outer Loop** training paradigm for progressive co-optimization. The *inner loop* provides a preliminary model-level warm start for each agent, by training its inner RecursiveLink to better align with latent thoughts generation. The *outer loop* then trains the outer RecursiveLink across agents at the system-level, with gradients recursively back-propagated through the full computation traces over recursion rounds. By exposing each agent to the feedback of itself and others from previous rounds, RecursiveMAS learns to leverage RecursiveLink for iterative refinement of collaboration, thus enabling the entire system to optimize in a unified manner.

To justify why recursion should occur in latent space rather than text-mediated interaction, we provide two theoretical analyses on runtime complexity and learning dynamics. From an architectural standpoint, RecursiveLink enables direct transformation of latent-space information, avoiding repeated decoding of intermediate agents with more efficient runtime complexity. From the learning perspective,

latent-space connections in RecursiveMAS maintain stable gradient propagation flow across recursion rounds during training, avoiding the gradient vanishing induced by text-based interactions.

Empirically, we evaluate RecursiveMAS on 9 benchmarks spanning mathematics, science, medicine, search, and code generation. We instantiate RecursiveMAS with diverse model families, including Qwen3/3.5, LLama-3, Gemma3, and Mistral, and adapt our framework to 4 representative MAS collaboration scenarios: step-by-step sequential reasoning, mixture-of-experts collaboration, expert-to-learner knowledge distillation, and tool-integrated deliberation. As illustrated in Figure 1, compared with advanced recursive language models and MAS baselines, RecursiveMAS achieves an average accuracy improvement of 8.3%, while delivering 1.2×–2.4× inference speedup and reducing token usage by 34.6%–75.6%. In addition, RecursiveMAS is structure-agnostic and can generalize to various agent collaboration patterns with effective performance. Our additional detailed analyses of scaling laws with deeper recursion, RecursiveLink architectures, semantic distributions across recursions, and training cost further validate the efficiency and performance scalability of the RecursiveMAS.

## 2. Preliminary

**Auto-regressive Generation in Latent Space.** Let $f_\theta(\cdot)$ denote a standard Transformer model (Vaswani et al., 2017) parameterized by $\theta$. Given a question $x$ with corresponding input embeddings $E = [e_1, \ldots, e_t] \in \mathbb{R}^{t \times d_h}$, the model computes the last-layer hidden state $h_t$ through the forward pass. In standard auto-regressive decoding, $h_t$ is projected to the vocabulary space to predict the next token. In contrast, latent generation keeps the recurrence entirely in continuous representation space by directly feeding the previously generated latent embedding $h_t$ back into the next forward pass. Formally, the next latent generation at step $t+1$ is:

$$h_{t+1} = f_\theta([E_{\leq t}; h_t]). \tag{1}$$

We refer to the newly generated latent state $h_{t+1}$ as the model's ongoing *latent thought*.

**Recursive Computation.** A recursive language model (RLM) increases reasoning depth by reusing the same transformation across recurrent steps. Consider a Transformer $f_\theta$ with $L$ layer blocks, denoted as $f_\theta = \mathcal{M}_L \circ \cdots \circ \mathcal{M}_1$. Instead of passing the input through the $L$-layer stack only once to obtain the last representation, a recursive model reuses the same stack for $n$ times of forward iterations, i.e.,

$$H^{(0)} = E, \quad H^{(r)} = f_\theta\big(H^{(r-1)}\big), \quad r = 1, \ldots, n. \tag{2}$$

The last round of latent representation $H^{(n)}$ is obtained through recursive refinement over the same shared Transformer layers, and is subsequently used for the final prediction.

**LLM-based Multi-Agent Evolution.** We define a multi-agent system $\mathcal{S}$ (Tran et al., 2025; Zou et al., 2025) composed of $N$ agents denoted as $\mathcal{A} = \{A_1, \ldots, A_N\}$, where each LLM agent $A_i$ corresponds to $f_{\theta_i}$ with its own last-layer representations $H_i$. We then denote the collective latent state of the system by $\mathcal{H} = \{H_1, \ldots, H_N\}$. Given any input problem $x$ with the ground-truth $y$, the system $\mathcal{S}$ orchestrates interactions among agents to collaboratively produce a final prediction. With this setup in place, we now formalize the evolution of agents under recursive computation.

**Definition 2.1: Recursive Multi-Agent Evolution**

*A **recursive evolution** is the progressive refinement of $\mathcal{H}$, where each agent adjusts its latent representation through iterative interaction with others and its own reasoning state, so that the updated system is better aligned for the given problem, i.e.* $\mathcal{S}^{(0)} \xrightarrow[\text{Evolve}]{H^{(1)}} \mathcal{S}^{(1)} \xrightarrow[\text{Evolve}]{H^{(2)}} \cdots \xrightarrow[\text{Evolve}]{H^{(n)}} \mathcal{S}^{(n)}$.

**Collaboration Pattern.** As MAS architectures are generally not fixed and can vary across tasks, we do not restrict the collaboration pattern to a single style. In this paper, we consider four commonly

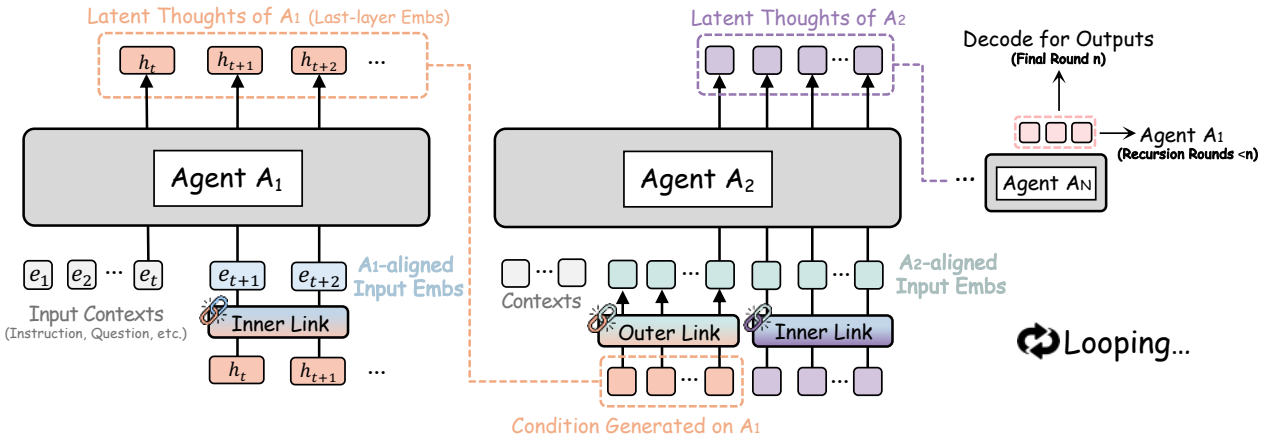


Figure 2 | **Overall Architecture of RecursiveMAS.** Each agent first leverages the inner RecursiveLink to perform latent thoughts generation, and then transfers the generated information to the next agent through the outer RecursiveLink. After the last agent finishes generation, its latent thoughts are fed back to the first agent, thereby forming a recursive loop within the multi-agent system.

adopted collaboration patterns in multi-agent systems: (i) *Sequential Style*, where we follow the chain-of-agents setting to assign three agents with complementary roles of `Planner`, `Critic`, and `Solver` and progressively decompose, judge, refine, and solve the problem; (ii) *Mixture Style*, where a mixture of domain-specialized agents (`Math`, `Code`, `Science`) reasons over the input problem in parallel, and their outputs are aggregated by a `Summarizer` agent to form the final answer; (iii) *Distillation Style*, where a larger, more capable `Expert` agent is paired with a smaller, faster `Learner` agent to distill expert knowledge while retaining higher generation efficiency; and (iv) *Deliberation Style*, where an inner-thinking `Reflector` is paired with a `Tool-Caller` that can invoke external tools (e.g., Python or search APIs). The agents iteratively exchange, critique, and refine candidate solutions until reaching a shared consensus, after which the `Tool-Caller` produces the final answer.

## 3. Building a Recursive Multi-Agent System

We introduce RecursiveMAS, an end-to-end recursive framework that links heterogeneous LLM agents together to scale the entire system through efficient and seamless latent collaboration. In the following, we will first elaborate the detailed architectural design of RecursiveMAS, and then present the corresponding recursive learning algorithm. We also interleave theoretical analyses throughout the method pipeline to support underlying design principles.

### 3.1. A Lightweight RecursiveLink

A language model's last-layer hidden states provide a natural representation of its generated semantics. The RecursiveLink $\mathcal{R}$ is designed to preserve and transmit this information from one embedding space to another. In RecursiveMAS, the transition arises in two cases: (i) *Dense-to-Shallow Transition*, where the previous step's last-layer embeddings are fed back as the next-step input embeddings during latent thoughts generation; and (ii) *Cross-Model Transition*, where one model's newly generated latent representations are passed as conditioning inputs to another model. As illustrated in Figure 3, we bridge these two transitions through the inner and outer links.

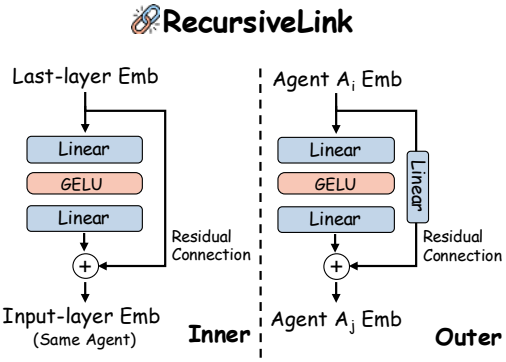


Figure 3 | Illustration on the inner and outer RecursiveLink Design.

**Inner Link.** Each LLM agent $A_i \in \mathcal{A}$ is paired with an inner RecursiveLink $\mathcal{R}_{\text{in}}$ during auto-regressive generation. Given any new last-layer embedding vector $h$, $\mathcal{R}_{\text{in}}$ transforms it as:

$$\mathcal{R}_{\text{in}}(h) = h + W_2\,\sigma(W_1 h), \tag{3}$$

where $W_1$ and $W_2$ are two standard linear layers, $\sigma(\cdot)$ is the GELU activation, and the residual connection preserves the original latent semantics. The transformed embedding is then used as input to the next forward pass of agent $A_i$.

**Outer Link.** An outer RecursiveLink $\mathcal{R}_{\text{out}}$ connects heterogeneous agents with different hidden dimensions. To support this, an additional linear layer $W_3$ is introduced in the residual branch to map the source embedding from agent $A_i$ into the target embedding space of agent $A_j$, i.e.,

$$\mathcal{R}_{\text{out}}(h) = W_3 h + W_2\,\sigma(W_1 h). \tag{4}$$

**Why Residual Connection?**

The residual branch largely preserves the original semantics of the input, allowing the RecursiveLink network to focus on *aligning distributional differences* rather than learning the full projection from scratch. This leads to more stable and efficient training. We also explore other alternatives and empirically validate our proposed design in Section 5.

### 3.2. Chain All Agents Together as a Loop

In recursive language models (RLMs), Transformer layers are connected through hidden states, and the residual stream loops across these layers to increase reasoning depth. Under this view, we cast each agent in RecursiveMAS as an RLM layer, with information flowing and recurring within and across agents as the hidden stream of the system. As shown in Figure 2, each agent contributes by reasoning and interacting with others in the latent space, together forming a recursive loop.

**Latent Thoughts Generation inside Agents.** We start by describing how each agent unfolds reasoning through the auto-regressive generation of latent thoughts. Specifically, given input contexts' embeddings $E_{A_1} = [e_1, e_2, \ldots, e_t]$ for the question and the agent-specific instructions, the first agent $A_1$ passes $E_{A_1}$ through the Transformer and computes the last-layer hidden representation $h_t$ at step $t$. Then, we insert $h_t$ into the inner link $\mathcal{R}_{\text{in}}$ to map the distribution back into the input embedding space for the next step, yielding $e_{t+1} = \mathcal{R}_{\text{in}}(h_t)$. Agent $A_1$ repeats this process auto-regressively for $m$ forward steps, generating a new continuous sequence of latent thoughts $H_{A_1} = [h_t, h_{t+1}, \ldots, h_{t+m}]$.

**Interaction across Heterogeneous Agents.** Once agent $A_1$ completes latent reasoning, its latent thoughts $H_{A_1}$ are sent to the next agent $A_2$ for cross-agent interaction. To achieve seamless information transmission across different types of agents, we first pass $H_{A_1}$ through the outer link $\mathcal{R}_{\text{out}}$ to transform it into input embeddings aligned with agent $A_2$. Next, agent $A_2$ starts latent thoughts generation conditioned on both its own input contexts and transferred information from $A_1$ (i.e., $E_{A_2} \oplus \mathcal{R}_{\text{out}}(H_{A_1})$).

We continue this interaction process across all consecutive agents in RecursiveMAS. In particular, after the last agent $A_N$ completes latent thoughts generation, its latent outputs (representing the system's latent answer to the input question) are passed back to the first agent $A_1$ through the inner-outer RecursiveLink, thereby closing the recursive loop. This recurrent connection allows each new recursion round to condition on information produced in previous rounds, so that each agent can iteratively reflect on earlier system outputs and refine their current generation. Throughout intermediate recursion rounds, all agents collaborate entirely in the latent space. Only after the final recursion round, the agent $A_N$ decodes the textual output as the system's final answer to the question.

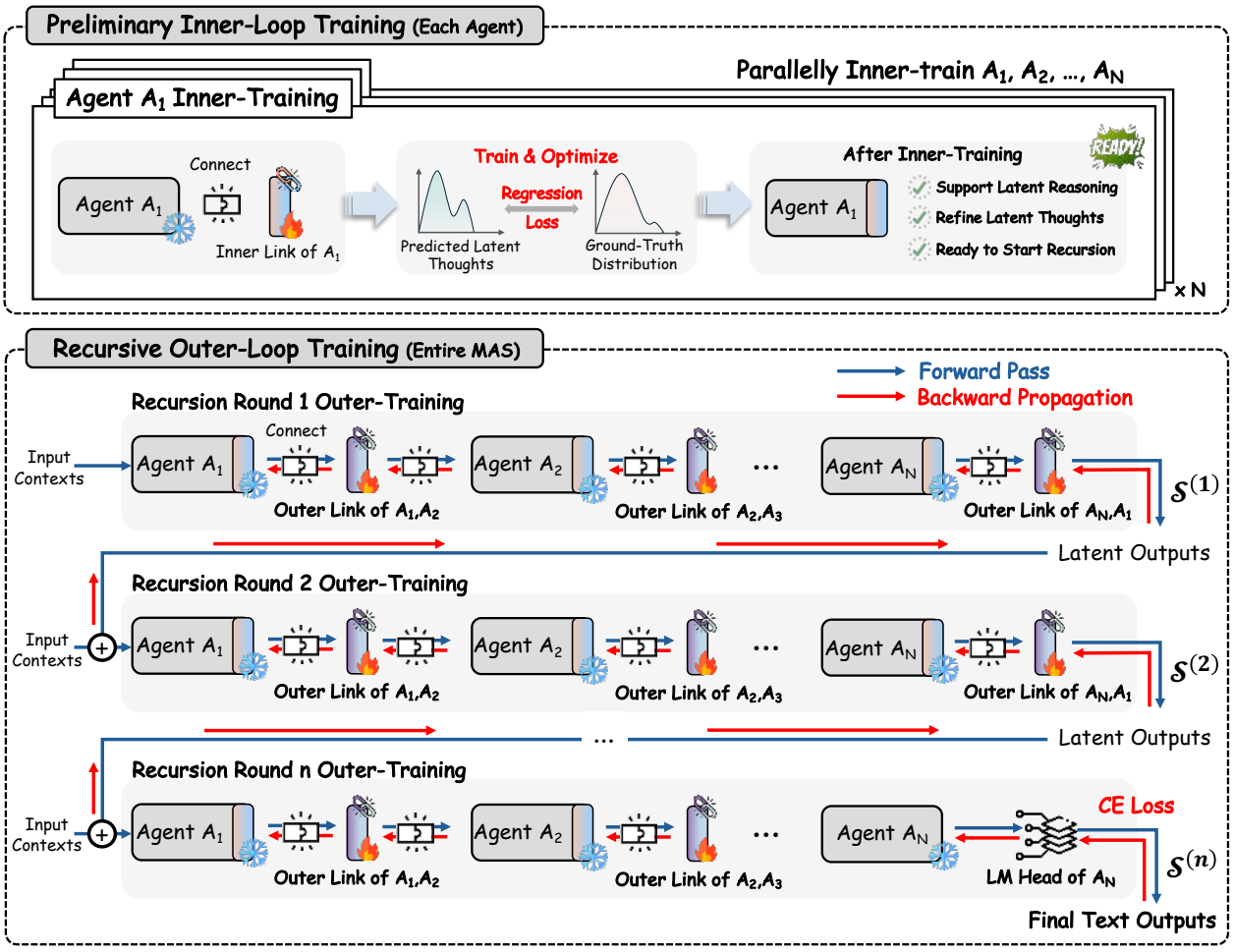


Figure 4 | **Two-Stage Training Pipeline of RecursiveMAS.** We first perform inner-loop training for each agent in parallel to warm up the inner RecursiveLink for latent thoughts generation, and then conduct outer-loop training to recursively optimize the outer RecursiveLink over the entire system.

**End-to-End Complexity Analyses.** To characterize the architectural efficiency of the full RecursiveMAS pipeline, we next analyze its end-to-end runtime complexity with RecursiveLink integrated throughout the system. The following proposition compares RecursiveMAS with a text-based recursive MAS, in which agents follow the same multi-round recursive collaboration structure but communicate through an explicit text medium rather than RecursiveLink-enabled latent interaction.

**Proposition 3.1 (RecursiveMAS Runtime Complexity).** *Without RecursiveLink, a text-based Recursive MAS with the same collaboration structure requires runtime complexity of* $\Theta(N(m|V|d_h + (t+m)d_h^2 + (t+m)^2 d_h))$*; In contrast, with RecursiveLink-enabled collaboration, RecursiveMAS achieves an end-to-end runtime complexity of* $\Theta(N(md_h^2 + (t+m)d_h^2 + (t+m)^2 d_h))$.

*Remark* 3.2. Since $d_h \ll |V|$ in practice, RecursiveMAS replaces the expensive per-step vocabulary-space decoding cost $m|V|d_h$ with a much more efficient latent-space transformation $md_h^2$.

Proposition 3.1 shows the end-to-end runtime advantage of RecursiveMAS. The full proof is provided in Appendix A.1. We also empirically analyze the efficiency advantage of our method in Section 5.

## 4. Learning to Recur as a Whole

With the framework in place, we next present the recursive learning algorithm, which only needs to train on the RecursiveLink to enable co-optimization of the entire system loop. As illustrated in Figure 4, the learning procedure consists of two stages: (i) a preliminary *inner-loop* to equip each agent with stronger latent thoughts generation capabilities; and (ii) an iterative *outer-loop* to progressively optimize the system as one unified entity over recursion rounds.

**Model-Level Inner-Loop Training.** For practical deployment of RecursiveMAS, we directly adopt off-the-shelf text-generation models as agents. To adapt these agents to the latent thoughts generation pattern, we first warm-start them through the inner RecursiveLink $\mathcal{R}_{\text{in}}$. Specifically, given each agent $A_i \in \mathcal{A}$ with parameters $\theta_i$ and the training example $(x, y) \in \mathcal{D}_{\text{train}}$, we construct the target latent thoughts distribution by passing the ground-truth text $y$ through the standard input embedding layer $\text{Emb}_{\theta_i}$ of agent $A_i$. The objective of training the inner link $\mathcal{R}_{\text{in}}$ corresponding to $A_i$ then formulates as:

$$\mathcal{L}_{\text{in}} = 1 - \cos\left(\mathcal{R}_{\text{in}}(H), \text{Emb}_{\theta_i}(y)\right), \tag{5}$$

where $H$ denotes the last-layer latent thoughts generated by agent $A_i$, and $\cos(\cdot,\cdot)$ denotes the standard cosine similarity. The regression objective here encourages each agent to leverage its inner link $\mathcal{R}_{\text{in}}$ to align latent thoughts with the semantic distribution from the input embedding layer, while eliminating the process of explicit decoding and re-encoding.

**System-Level Outer-Loop Training.** Next, we iteratively co-optimize the entire system through the outer RecursiveLink $\mathcal{R}_{\text{out}}$. Let $\mathcal{S}^{(r)}$ denote the system state at recursion round $r = 1, \ldots, n$. During outer-loop training, the system is first unrolled along its looped structure for $n$ forward rounds. After the final textual prediction is produced in the last recursion round, we jointly optimize all outer links that connect the system with the following cross-entropy (CE) objective:

$$\mathcal{L}_{\text{out}} = \text{CE}\Big(\mathcal{S}^{(n)}\big(\mathcal{S}^{(n-1)}(\cdots \mathcal{S}^{(1)}(x))\big),\ y\Big). \tag{6}$$

Throughout training, the computation graph is preserved along the full recursive paths. Gradient backpropagation assigns each outer link a shared credit signal according to its global contribution to the final prediction, thereby enabling information flow to be iteratively optimized as a whole.

**Learning Advantage of RecursiveMAS.** To better understand why latent collaboration of agents in the inner-outer loop training confers a stronger learning advantage, we provide a detailed theoretical analysis below of the gradient propagation process throughout recursive training of RecursiveMAS.

**Theorem 4.1** (**Gradient Stability**). *Under the Realistic Assumptions (stated in Appendix A.2), if tokens are confident with entropy $\leq \epsilon$, where typically $\epsilon \ll 1$: directly applying text-based SFT (denoted by $\mathcal{R}_{text}(h)$) during recursion suffers from gradient vanishing (i.e., gradient norm close to 0); while RecursiveMAS with the RecursiveLink $\mathcal{R}$ maintains stable and near constant gradients (i.e., gradient norm close to 1) during looped backpropagation process. Formally, with probability $\geq 1 - \delta$,*

$$\left\| \frac{\partial \mathcal{R}_{text}(h)}{\partial h} \right\|_2 \leq O(\epsilon) \ll 1, \qquad \left\| \frac{\partial \mathcal{R}(h)}{\partial h} \right\|_2 \geq \Omega\left(1 - \sqrt{\frac{1}{d_h} \log \frac{1}{\delta}}\right). \tag{7}$$

The full proof is provided in Appendix A.3. Theorem 4.1 demonstrates the learning advantage of RecursiveMAS, by allowing gradients to remain informative across recursion rounds. Together, theoretical justifications in Proposition 3.1 and Theorem 4.1 motivate our design of latent-based interaction among agents rather than text mediation, as it makes the whole-system co-optimization of RecursiveMAS easier and more effective. During inference, RecursiveMAS performs recursive generation by following the same $n$ recursion rounds as in the outer-loop training.

Table 1 | **Agent configurations for different collaboration patterns in RecursiveMAS.** We select off-the-shelf models from diverse model families to form heterogeneous agent compositions with complementary strengths. Each assignment is chosen to match the role-specific needs of the corresponding collaboration pattern while preserving both practical efficiency and scalability.

| Collaboration Pattern | Role | Model Size & Version |
|---|---|---|
| Sequential Style (Light) | Planner | Qwen3-1.7B (Yang et al., 2025) |
| | Critic | Llama3.2-1B-Instruct (Grattafiori et al., 2024) |
| | Solver | Qwen2.5-Math-1.5B-Instruct (Qwen et al., 2025) |
| Sequential Style (Scaled) | Planner | Gemma3-4B-it (Team et al., 2025) |
| | Critic | Llama3.2-3B-Instruct (Grattafiori et al., 2024) |
| | Solver | Qwen3.5-4B (Yang et al., 2025) |
| Mixture Style | Code Specialist | Qwen2.5-Coder-3B-Instruct (Hui et al., 2024) |
| | Science Specialist | BioMistral-7B (Labrak et al., 2024) |
| | Math Specialist | DeepSeek-R1-Distill-Qwen-1.5B (Qwen et al., 2025) |
| | Summarizer | Qwen3.5-2B (Yang et al., 2025) |
| Distillation Style | Learner | Qwen3.5-4B (Yang et al., 2025) |
| | Expert | Qwen3.5-9B (Yang et al., 2025) |
| Deliberation Style | Reflector | Qwen3.5-4B (Yang et al., 2025) |
| | Tool-Caller | Qwen3.5-4B (with Tool-Integration) (Yang et al., 2025) |

## 5. Empirical Evaluations

**Tasks and Datasets.** We conduct comprehensive evaluations of RecursiveMAS on nine benchmarks across various domains: (i) *Mathematical Reasoning*, including MATH500 (HuggingFaceH4, 2023), AIME2025 (math ai, 2025), and AIME2026 (MathArena, 2026); (ii) *Scientific and Medical Tasks*, including GPQA-Diamond (Rein et al., 2023) and MedQA (Yang et al., 2024a); (iii) *Code Generation*, including LiveCodeBench-v6 (Jain et al., 2025) and MBPP Plus (Liu et al., 2023); and (iv) *Search QA*, including HotpotQA (Yang et al., 2018) and Bamboogle (Press et al., 2023). We adopt the standard evaluation metric for each dataset. For AIME2025/2026, we report Pass@10 accuracy for testing robustness. Additional benchmark and metrics details are in Appendix B.1.

**Models and Baselines.** We instantiate RecursiveMAS with diverse agent collaboration patterns, including (i) *Sequential Style*, (ii) *Mixture Style*, (iii) *Distillation Style*, and (iv) *Deliberation Style*, following the setups described in Section 2. For each collaboration style, we use off-the-shelf LLMs from diverse model families, covering Qwen (Qwen et al., 2025; Yang et al., 2025), Llama (Grattafiori et al., 2024), Gemma (Team et al., 2025), and Mistral (Jiang et al., 2024), to construct heterogeneous agent compositions. Detailed model configurations and their assigned roles are provided in Table 1.

For baseline comparisons, we evaluate RecursiveMAS against (i) *Single Advanced Agents*, where individual LLM agents from each collaboration pattern are isolated as standalone models to solve problems, such as the final agent in Sequential Style and each domain specialist in Mixture Style. For fair comparison, we provide full supervised and LoRA fine-tuning (Schulman and Lab, 2025) for single models on the same training set. (ii) *Recursion-based Methods*, including single recursive language models, LoopLM (Zhu et al., 2025), and Recursive-TextMAS, where agents collaborate in the same way as RecursiveMAS but interact through text instead of latent thoughts; and (iii) additional *Representative Multi-Agent Frameworks*, including TextGrad (Yuksekgonul et al., 2025) and

Table 2 | **Main results of RecursiveMAS over Different Recursion Rounds.** We report the accuracy (%, "Acc."), end-to-end runtime (s, "Time"), and overall token usage ("Token") across domains. For Code Gen., we evaluate the Light and Scaled settings on MBPP+ and LiveCodeBench, respectively. The average standard deviation of RecursiveMAS across 5 runs is ±0.0041 for accuracy, ±26 for runtime, and ±33 for tokens. We compare with all methods under the same MAS framework structure and recursion budgets. The performance and efficiency advantages of RecursiveMAS become increasingly significant as the recursion round $r$ increases, with improvements highlighted.

| Method | Metric | Math500 | | AIME2025 | | AIME2026 | | GPQA-D | | MedQA | | Code Gen. | | Improve |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Light | Scaled | Light | Scaled | Light | Scaled | Light | Scaled | Light | Scaled | Light | Scaled | |
| ***Recursive Round r=1*** | | | | | | | | | | | | | | |
| Recursive-TextMAS | Acc. | 71.9 | 84.2 | 24.0 | 71.3 | 16.7 | 76.7 | 28.1 | 61.5 | 29.0 | 76.1 | 30.7 | 38.5 | Base |
| | Time | 1368 | 2401 | 2380 | 8462 | 2216 | 9376 | 1056 | 2190 | 1555 | 1522 | 976 | 8867 | Base |
| | Token | 1185 | 1471 | 2993 | 9397 | 2754 | 8854 | 2084 | 3693 | 2382 | 1427 | 1146 | 3154 | Base |
| **RecursiveMAS** | Acc. | 75.8 | 86.3 | 30.7 | 80.0 | 17.3 | 82.7 | 30.3 | 63.1 | 30.3 | 78.2 | 35.1 | 40.1 | ↑ **3.4** |
| | Time | 825 | 1701 | 1829 | 7784 | 1788 | 8134 | 586 | 1965 | 1194 | 1348 | 449 | 7908 | ×**1.2** |
| | Token | 523 | 816 | 1622 | 6338 | 1576 | 7021 | 829 | 2675 | 1369 | 964 | 577 | 2198 | ↓ **34.6%** |
| ***Recursive Round r=2*** | | | | | | | | | | | | | | |
| Recursive-TextMAS | Acc. | 72.5 | 84.4 | 23.3 | 70.7 | 10.0 | 77.3 | 28.7 | 59.1 | 28.3 | 76.1 | 30.0 | 38.0 | Base |
| | Time | 2204 | 3958 | 4247 | 14380 | 3960 | 14110 | 1825 | 4207 | 3097 | 2745 | 1847 | 14792 | Base |
| | Token | 2117 | 2794 | 5318 | 16372 | 4982 | 16213 | 3708 | 6128 | 4436 | 2609 | 1998 | 5369 | Base |
| **RecursiveMAS** | Acc. | 76.6 | 87.1 | 33.3 | 86.0 | 18.7 | 84.0 | 32.3 | 64.6 | 31.2 | 78.3 | 36.9 | 41.3 | ↑ **6.0** |
| | Time | 1096 | 1974 | 2367 | 8178 | 2263 | 8965 | 752 | 2342 | 1427 | 1664 | 627 | 8329 | ×**1.9** |
| | Token | 495 | 953 | 1614 | 5314 | 1552 | 6657 | 813 | 2521 | 1383 | 1008 | 531 | 2020 | ↓ **65.5%** |
| ***Recursive Round r=3*** | | | | | | | | | | | | | | |
| Recursive-TextMAS | Acc. | 69.1 | 85.8 | 18.0 | 73.3 | 16.7 | 74.7 | 28.7 | 58.6 | 28.5 | 77.1 | 29.3 | 36.5 | Base |
| | Time | 2952 | 6010 | 6183 | 19304 | 5907 | 19678 | 3322 | 7537 | 4684 | 3922 | 2310 | 22036 | Base |
| | Token | 3059 | 4100 | 8645 | 23651 | 7813 | 22915 | 5820 | 8091 | 6307 | 3731 | 2676 | 7078 | Base |
| **RecursiveMAS** | Acc. | 77.8 | 88.2 | 34.0 | 86.7 | 20.0 | 86.0 | 32.6 | 66.2 | 31.7 | 79.3 | 37.4 | 42.8 | ↑ **7.2** |
| | Time | 1360 | 2320 | 2727 | 8981 | 2629 | 9623 | 861 | 2638 | 1704 | 1912 | 805 | 10186 | ×**2.4** |
| | Token | 519 | 893 | 1586 | 5342 | 1537 | 6860 | 786 | 2524 | 1378 | 1056 | 595 | 2247 | ↓ **75.6%** |

Mixture-of-Agents (MoA) (Wang et al., 2025b) for more holistic structure-wide evaluations. Detailed baseline implementations are provided in Appendix B.2.

**Training and Implementation Details.** For inner-outer loop training, we freeze all LLM agent parameters and update only the inner/outer RecursiveLink. We curate a diverse training set spanning multiple domains, sourced from s1K (Muennighoff et al., 2025) for mathematical problem solving, m1k (Huang et al., 2025) for medical and scientific tasks, OpenCodeReasoning (Ahmad et al., 2025) for code generation, and ARPO-SFT (Dong et al., 2025) for agentic tool-augmentation (Python Code/Search-API) settings. We use AdamW with a learning rate of 5e-4, a cosine learning rate scheduler, and a batch size of 4. During inference, we set top-p to 0.95 and use a temperature of 0.6 for most reasoning tasks and 0.2 for code generation, as suggested in each model's official report. The maximum output length is adjusted for each task based on its relative difficulty. We perform hyperparameter tuning and report the mean performance over five independent runs. More training/inference details and hyperparameter setups are provided in Appendix B.3.

### 5.1. Scaling Performance via Recursion

We begin by evaluating how RecursiveMAS performs across different recursion depths $r = 1, 2, 3$. As shown in Table 2, we analyze agent collaboration behavior from three complementary perspectives: (i) accuracy, (ii) end-to-end runtime, and (iii) overall system token throughput. We also include a text-based recursive baseline for reference. Across seven math, science, and code generation tasks, both light and scaled versions of RecursiveMAS exhibit a consistent upward trend as recursion depth

Table 3 | **Comparison of RecursiveMAS with Other Methods.** We evaluate RecursiveMAS at recursion round $r = 3$. Under the same training budget and model setups, RecursiveMAS consistently outperforms advanced single-agent methods, alternative MAS frameworks, and recursive computation baselines.

| Method | MATH500 | AIME2025 | AIME2026 | GPQA-D | LiveCodeBench | MedQA |
|---|---|---|---|---|---|---|
| Single Agent (w/ LoRA) | 83.1 | 70.0 | 73.3 | 62.0 | 37.4 | 76.1 |
| Single Agent (w/ Full-SFT) | 83.2 | 73.3 | 76.7 | 62.8 | 38.6 | 77.0 |
| Mixture-of-Agents (MoA) | 79.8 | 60.0 | 63.3 | 47.6 | 27.0 | 57.5 |
| TextGrad | 84.9 | 73.3 | 76.7 | 62.5 | 39.8 | 77.2 |
| LoopLM | 84.6 | 66.7 | 63.3 | 48.1 | 24.9 | 56.4 |
| Recursive-TextMAS | 85.8 | 73.3 | 73.3 | 61.6 | 38.7 | 77.0 |
| **RecursiveMAS** | **88.0** | **86.7** | **86.7** | **66.2** | **42.9** | **79.3** |

increases. When compared with the text-based recursion, RecursiveMAS consistently improves over the baseline by an average of 8.1% at $r = 1$, 19.6% at $r = 2$, and 20.2% at $r = 3$, with performance advantage more pronounced as the recursion deepens. Additionally, under identical MAS architectures, RecursiveMAS delivers steadily increasing efficiency gains across recursion rounds, accelerating end-to-end inference time from 1.2× to 2.4× while reducing output tokens from 34.6% to 75.6%. Additional case studies on the running pipeline of RecursiveMAS across domains are provided in Appendix G.

**Scaling Law on RecursiveMAS (Training v.s. Inference).** We further examine the scaling behavior of recursion in RecursiveMAS by jointly varying the training-time and inference-time recursion rounds. Figure 1 (Up) illustrates the performance landscape of RecursiveMAS under different training and inference settings. Increasing inference depth continues to improve systems trained with fewer rounds, while deeper training shifts the entire performance frontier upward, with the strongest results consistently appearing in the upper-right region where both are large. This trend suggests a complementary training-inference scaling effect in RecursiveMAS: training recursion progressively teaches the system to form refinement-ready latent states, and subsequent inference recursion translates this learned recursive structure into additional test-time gains.

### 5.2. Broader Comparison with Alternative Architectures and Training Frameworks

Table 3 compares RecursiveMAS at the whole-system level against a broader set of baselines, including single fine-tuned agents, representative multi-agent frameworks, and alternative recursive methods. To ensure fair comparison, all methods are instantiated with identical backbone models and comparable training budgets (e.g., matched trainable parameter counts, recursion depth, training set).

Overall, RecursiveMAS delivers a consistent whole-system advantage, achieving an average performance improvement of 8.3% over the strongest baseline on each benchmark. With the same training data, fine-tuning individual agents strengthens performance relative to their off-the-shelf versions, while RecursiveMAS delivers further gains by optimizing cross-agent collaboration at the system level. In addition, RecursiveMAS remains the performance advantage compared to advanced architectures such as TextGrad and LoopLM, especially on reasoning-intensive tasks (e.g., accuracy gains of 18.1% on AIME2025, 13.0% on AIME2026, and 5.4% on GPQA-Diamond).

### 5.3. Can RecursiveMAS Generalize across Diverse Collaboration Patterns?

Beyond the sequential setting, we further instantiate RecursiveMAS under three additional MAS collaboration patterns in Table 1 to assess whether our method is agnostic to any specific system architecture and generalizes across diverse usage scenarios. As shown in Figure 1 (Down), we compare

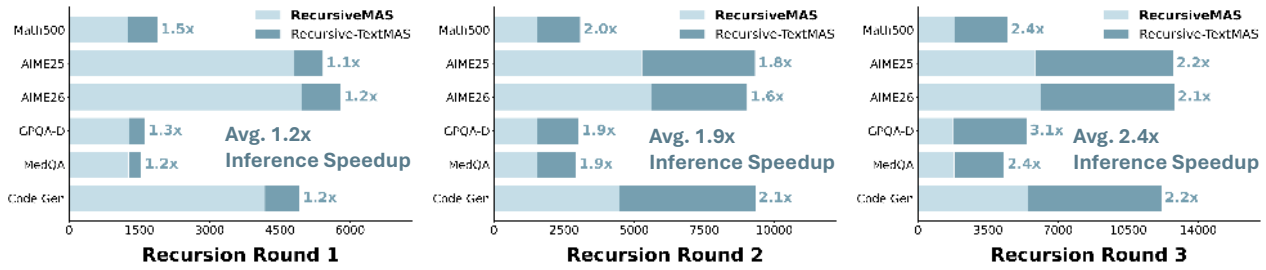


Figure 5 | **Inference Time Speedup of RecursiveMAS across Three Recursion Rounds.** RecursiveMAS exhibits increasing inference speedup as the recursion depth increases.

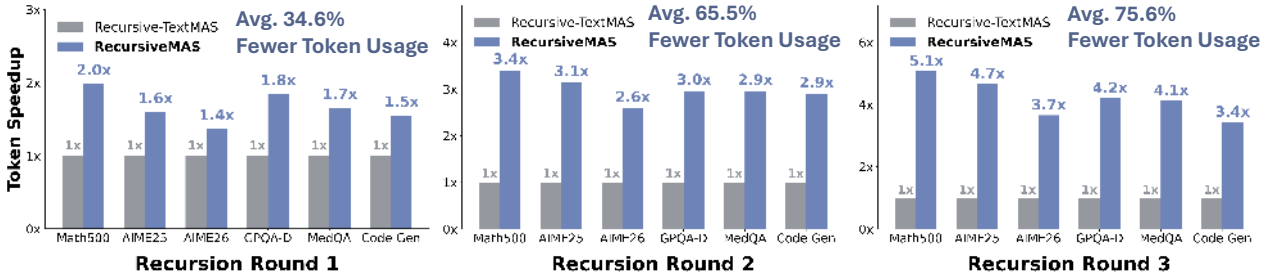


Figure 6 | **Token Reduction of RecursiveMAS across Three Recursion Rounds.** As recursion deepens, RecursiveMAS reduces substantially more tokens than Recursive-TextMAS.

the accuracy of RecursiveMAS against strong standalone agents within each collaboration pattern.

In *Mixture-style*, RecursiveMAS achieves an average improvement of 6.2% over the strongest domain specialist on each benchmark, suggesting that recursive interaction enables non-trivial cross-domain composition beyond what can be attained by selecting one individual specialist alone. In *Deliberation-style*, we evaluate tool use on both mathematical and search-intensive tasks. RecursiveMAS improves the original tool-calling agent by 4.8%, showing that recursive latent coordination remains effective in tool-calling settings through iterative interaction with the Reflector. Finally, in *Distillation-style*, RecursiveMAS improves the learner by 8.0% while retaining 1.5× end-to-end speed advantage over the expert. In this way, RecursiveMAS distills much of the expert's capability into a more efficient system. We leave detailed reports of Figure 1 (Down) in Appendix D.1.

### 5.4. Efficiency Analyses on Latent-space Recursion

**Inference Time Speedup.** We analyze the efficiency of RecursiveMAS to empirically support our complexity advantage in Proposition 3.1. We first compare RecursiveMAS against Recursive-TextMAS to study how our advantage on end-to-end inference time scales with recursion depth. As shown in Figure 5, although deeper recursion rounds introduce cost, we find that RecursiveMAS consistently exhibits efficiency gain, and the advantage further increases as recursion deepens. For example, at recursion round $r = 1$, RecursiveMAS already achieves a 1.2× speedup on average, and this advantage grows to 1.9× and 2.4× at larger recursion rounds of $r = 2/3$. This trend aligns well with our method design, where RecursiveMAS achieves a favorable scaling behavior by conducting recursive collaboration directly in latent space and avoiding repeated intermediate text generation.

**Overall Token Usage Reduction.** We next demonstrate the substantial token usage reduction of RecursiveMAS in Figure 6. Within the comparison, we find that the baseline method suffers from rapidly growing token overhead as recursion round increases, while RecursiveMAS reduces the token usage by 34.6% for the first recursion round, and the reduction scales to 75.6% at $r = 3$. This is because Recursive-TextMAS repeatedly decode the intermediate text at every recursion round, whereas

RecursiveMAS performs most recursive interaction directly in latent space. Overall, RecursiveMAS enables a much more efficient system-level scaling behavior, and the resulting efficiency gain is amplified as the number of recursion rounds increases.

## 6. In-depth Analyses on RecursiveMAS

**RecursiveLink Design.** To validate the effectiveness of RecursiveLink, we compare our 2-layer residual design against three alternatives: (i) a 1-layer network, (ii) a 1-layer network with the residual connection, and (iii) a 2-layer network without the residual connection. We conduct experiments using the scaled sequential-style RecursiveMAS and adapt the same architecture for both $\mathcal{R}_{\text{in}}$ and $\mathcal{R}_{\text{out}}$.

As shown in Table 4, our 2-layer residual design performs best across all three benchmarks, and the residual connection delivers additional improvements across different backbone models. For example, on GPQA-Diamond, equipping a single-layer design with a residual branch improves the performance from 63.2% to 65.3%, which is even higher than the plain 2-layer design (64.5%). These results align with our design intuition in Section 3.1: by preserving latent semantics while learning only the distributional shift, RecursiveLink achieves stable training and stronger inference performance.

Table 4 | **Efficacy on RecursiveLink Design.** We compare accuracy across alternative architectural designs.

| **RecursiveLink Design** | Math500 | GPQA-D | LiveCodeBench |
|---|---|---|---|
| 1-Layer | 84.4 | 63.2 | 40.1 |
| Res+1-Layer | 86.7 | 65.3 | 41.4 |
| 2-Layer | 85.6 | 64.5 | 40.5 |
| **Res+2-Layer (ours)** | **88.0** | **66.2** | **42.9** |

**Semantic Representations in Recursion.** We analyze how the semantic distribution of RecursiveMAS changes across different recursion rounds. Under the scaled sequential setting of RecursiveMAS, we randomly sample 500 question-answer pairs spanning all downstream domains. We then use the solver agent's input embedding layer to map each ground-truth answer string into embedding representations, which serves as the reference semantic distribution. We run RecursiveMAS at recursion rounds $r = 1, 2, 3$ to generate final answers for all these 500 questions, map the generated answers into embeddings using the same input embedding layer, and visualize both the ground-truth reference ("purple") and newly generated distributions ("orange") via PCA projection.

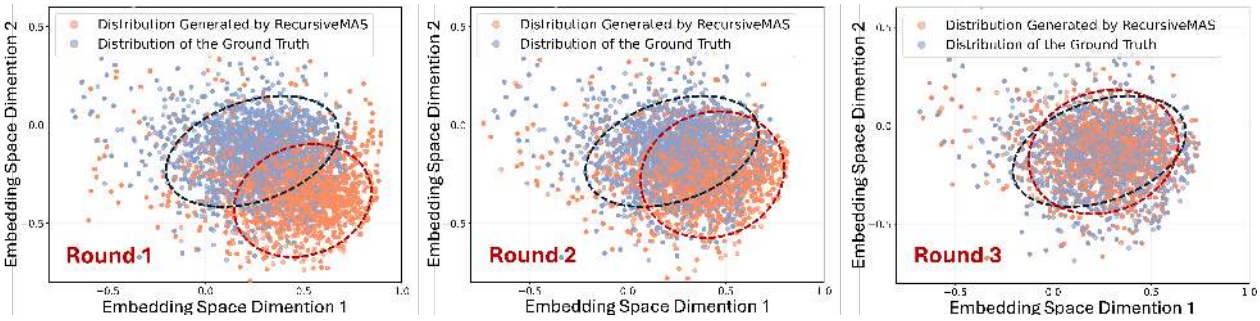


Figure 7 | **Semantic Representations of RecursiveMAS across Differnt Recursion Rounds.** We visualize the semantic distribution of the final answers generated by RecursiveMAS and the corresponding ground-truth across 500 questions. Increasing recursion rounds progressively aligns the generated distribution of RecursiveMAS with the ground truth distribution.

In Figure 7, the generated answers at $r = 1$ remain visibly shifted from the ground-truth distribution, but this discrepancy progressively narrows as depth increases, with the two distributions becoming largely aligned by $r = 3$. This aligning trend suggests that RecursiveMAS iteratively refines the latent embeddings and corresponding answers through recursion. We further take a closer look to examine

Table 5 | **Cost analysis on RecursiveMAS.** We report the peak GPU memory usage (GB), number of trainable parameters, estimated cost, and average accuracy (%) across all downstream tasks.

| Methods | GPU Mem. | Trainable Param. | Cost | Avg. Acc. |
|---|---|---|---|---|
| LoRA Training | 21.67 | 15.92M (0.37%) | $6.64 | 66.9 |
| Full-SFT | 41.40 | 4.21B (100%) | $9.67 | 68.6 |
| **RecursiveMAS** | **15.29** | **13.12M (0.31%)** | **$4.27** | **74.9** |

individual test instances and provide detailed case studies in Appendix F. Our case studies reveal a common pattern in which RecursiveMAS may produce an incorrect answer at an early stage, while deeper recursion successfully corrects it through iterative refinement. Together, these analyses provide further evidence that latent thoughts capture semantically meaningful representations, and that deeper recursion improves alignment toward correct final outputs.

**Optimal Length of Latent Thoughts Generation.** We next study and ablate the latent thoughts length $m$ to examine how much of each agent's internal reasoning is sufficient to support effective collaboration. Under the scaled sequential-style of RecursiveMAS, we evaluate a broad range of $m$. As illustrated in Figure 8, increasing $m$ improves performance in the early regime. Once $m$ reaches a moderate scale (around $m = 80$), performance is stabilized across all benchmarks. The ablation suggests that RecursiveMAS enables effective agent reasoning and interaction with only a modest latent-thought budget, in sharp contrast to text-based collaboration that typically requires longer CoT and costly token generation.

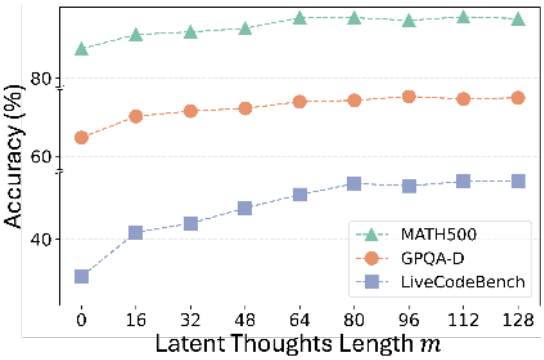


Figure 8 | Effectiveness of RecursiveMAS's latent thoughts with different step lengths.

**Training Cost Analysis** We further analyze the training cost of RecursiveMAS under the scaled sequential-style MAS setting. We compare RecursiveMAS with direct training methods, including LoRA and full supervised fine-tuning with the same training data and backbone setup. For cost estimation, we follow prior methods (Liu et al., 2025; Lu et al., 2023) to measure the cost based on GPU usage. As shown in Table 5, RecursiveMAS utilizes the lowest per-agent GPU memory, trainable parameter count, and estimated cost among all compared training strategies. Meanwhile, RecursiveMAS achieves the highest accuracy across all downstream tasks, suggesting that optimizing the lightweight RecursiveLink provides a better cost-performance trade-off than other training methods.

## 7. Related Works

**LLM-based Multi-Agent Systems.** Current LLMs achieve strong performance on general tasks, but they often exhibit bottlenecks when facing diverse reasoning patterns (Maheswaran et al., 2026; Mirzadeh et al., 2025; Valmeekam et al., 2023) or domain-specific challenges (Chen et al., 2025). To overcome these limitations, Multi-agent systems extend the single LLM paradigm to a collaborative setting (Su et al., 2025; Tran et al., 2025; Wu et al., 2024; Yang et al., 2024b) by organizing a set of agents with distinct roles that jointly address the problem. A standard multi-agent system topology involves a sequential configuration (Li et al., 2023; Qian et al., 2024), where agents are assigned in a linear pipeline to decompose and resolve problems in order. Beyond sequential settings, other

works also explore mixture-style settings (Wang et al., 2025b; Ye et al., 2025b; Yun et al., 2026), where multiple agents with domain expertise reason in parallel, and their outputs are aggregated into a final decision. Another line of work seeks to improve MAS through textual feedback signals. For example, related optimization methods (Shen et al., 2025; Yuksekgonul et al., 2025) leverage an LLM to generate natural language feedback to refine contextual inputs and instructions of each agent. Additionally, another study (Motwani et al., 2024) improves MAS by separately training each agent with role-specific responses. Rather than separate training each individual agent or only leveraging textual feedback, RecursiveMAS treats MAS as a unified whole, and scales the system performance via recursively refining the latent information flow.

**Scaling Reasoning via Recursion.** Recent studies explore recursion as an alternative scaling axis for LLMs (Bae et al., 2025; Geiping et al., 2025; Li et al., 2026; Tang et al., 2026), where the same computation blocks are reused through multiple recurrent rounds (i.e., loops) to increase reasoning depth and iteratively refine hidden representations. One line of work studies recursive language models that apply shared layers to scale latent reasoning. For instance, LoopLM (Zhu et al., 2025) introduces pre-trained looped language models with iterative latent computation. Besides, other work explores other recursive architectures (Jolicoeur-Martineau, 2025; Wang et al., 2025a; Zhang et al., 2025a), including tiny recursive networks and recursive self-calling schemes for long-context inference. While existing methods in agentic AI primarily focus on recursion inside a single language model, RecursiveMAS exhibits the first attempt to extend the recursive scaling paradigm to system-level. Additional related works are provided in Appendix C.

## 8. Conclusion

We introduce RecursiveMAS, a recursive multi-agent framework that scales agent collaboration through system-level recursion. RecursiveMAS first supports latent-thoughts generation within each agent through inner RecursiveLink, then connects heterogeneous agents through outer RecursiveLink, and optimizes the whole system with an inner-outer loop training paradigm. Theoretically, our framework leads to more stable training dynamics and improves efficiency compared to text-based baselines. Our empirical results across mathematical and scientific reasoning, code generation, and search benchmarks show that RecursiveMAS consistently improves accuracy while substantially reducing inference time and token usage. Overall, RecursiveMAS provides a scalable and efficient framework for multi-agent systems to recursively collaborate, refine, and evolve in latent space.

## References

W. U. Ahmad, S. Narenthiran, S. Majumdar, A. Ficek, S. Jain, J. Huang, V. Noroozi, and B. Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding, 2025. URL `https://arxiv.org/abs/2504.01943`.

H. Babu, P. Schillinger, and T. Asfour. Adaptive domain modeling with language models: A multi-agent approach to task planning. In *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*, pages 1701–1708. IEEE, 2025.

S. Bae, Y. Kim, R. Bayat, S. Kim, J. Ha, T. Schuster, A. Fisch, H. Harutyunyan, Z. Ji, A. Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

H. Chen, Z. Fang, Y. Singla, and M. Dredze. Benchmarking large language models on answering and explaining challenging medical questions. In *Proceedings of the 2025 Conference of the Nations of*

*the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3563–3599, 2025.

G. Dong, H. Mao, K. Ma, L. Bao, Y. Chen, Z. Wang, Z. Chen, J. Du, H. Wang, F. Zhang, et al. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025.

Z. Du, R. Wang, H. Bai, Z. Cao, X. Zhu, Y. Cheng, B. Zheng, W. Chen, and H. Ying. Enabling agents to communicate entirely in latent space. *arXiv preprint arXiv:2511.09149*, 2025.

H. Face. Transformers documentation. `https://huggingface.co/docs/transformers/en/index`, 2025.

T. Fu, Z. Min, H. Zhang, J. Yan, G. Dai, W. Ouyang, and Y. Wang. Cache-to-cache: Direct semantic communication between large language models. *arXiv preprint arXiv:2510.03215*, 2025.

J. Geiping, S. McLeish, N. Jain, J. Kirchenbauer, S. Singh, B. R. Bartoldson, B. Kailkhura, A. Bhatele, and T. Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

W. Gu, J. Han, H. Wang, X. Li, and B. Cheng. Explain-analyze-generate: A sequential multi-agent collaboration method for complex reasoning. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7127–7140, 2025.

M. Hu, Y. Zhou, W. Fan, Y. Nie, B. Xia, T. Sun, Z. Ye, Z. Jin, Y. Li, Q. Chen, et al. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*, 2025.

X. Huang, J. Wu, H. Liu, X. Tang, and Y. Zhou. m1: Unleash the potential of test-time scaling for medical reasoning with large language models, 2025. URL `https://arxiv.org/abs/2504.00869`.

HuggingFaceH4. Math-500 dataset. `https://huggingface.co/datasets/HuggingFaceH4/MATH-500`, 2023.

B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

A. Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

Y. Labrak, A. Bazoge, E. Morin, P.-A. Gourraud, M. Rouvier, and R. Dufour. Biomistral: A collection of open-source pretrained large language models for medical domains. In *Findings of the association for computational linguistics: acl 2024*, pages 5848–5864, 2024.

G. Li, H. A. Al Kader Hammoud, H. Itani, D. Khizbullin, and B. Ghanem. Camel: communicative agents for "mind" exploration of large language model society. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Y. Li, J. Chen, F. Wu, J. Yu, H. Qi, W. Xuan, H. Zhao, P. Nie, D. Jin, and X. Tang. Learning multi-step reasoning via persistent latent state propagation. In *Workshop on Latent {\&} Implicit Thinking {\textendash} Going Beyond CoT Reasoning*, 2026.

Z. Li, H. Zhang, S. Han, S. Liu, J. Xie, Y. Zhang, Y. Choi, J. Zou, and P. Lu. In-the-flow agentic system optimization for effective planning and tool use. *arXiv preprint arXiv:2510.05592*, 2025a.

Z.-Z. Li, D. Zhang, M.-L. Zhang, J. Zhang, Z. Liu, Y. Yao, H. Xu, J. Zheng, P.-J. Wang, X. Chen, et al. From system 1 to system 2: A survey of reasoning large language models. *arXiv preprint arXiv:2502.17419*, 2025b.

A. Liu, A. Mei, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, C. Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572, 2023.

Y. Lu, C. Li, H. Liu, J. Yang, J. Gao, and Y. Shen. An empirical study of scaling instruct-tuned large multimodal models. *arXiv preprint arXiv:2309.09958*, 2023.

M. Maheswaran, L. Lakhani, Z. Zhou, S. Yang, J. Wang, C. Hooper, Y. Hu, R. Tiwari, J. Wang, H. Singh, et al. Squeeze evolve: Unified multi-model orchestration for verifier-free evolution. *arXiv preprint arXiv:2604.07725*, 2026.

math ai. AIME 2025 dataset. `https://huggingface.co/datasets/math-ai/aime25`, 2025.

MathArena. Aime 2026 dataset. `https://huggingface.co/datasets/MathArena/aime_2026`, 2026.

S. I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

S. R. Motwani, C. Smith, R. J. Das, R. Rafailov, I. Laptev, P. H. Torr, F. Pizzati, R. Clark, and C. S. de Witt. Malt: Improving reasoning with multi-agent llm training. *arXiv preprint arXiv:2412.01928*, 2024.

N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20275–20321. Association for Computational Linguistics, Nov. 2025.

O. Press, M. Zhang, S. Min, L. Schmidt, N. A. Smith, and M. Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, 2023.

C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 15174–15186, 2024.

C. Qian, Z. Xie, Y. Wang, W. Liu, K. Zhu, H. Xia, Y. Dang, Z. Du, W. Chen, C. Yang, et al. Scaling large language model-based multi-agent collaboration. In *The Thirteenth International Conference on Learning Representations*, 2025.

Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

J. Schulman and T. M. Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250929. https://thinkingmachines.ai/blog/lora/.

M. Shen, R. Shu, A. Pratik, J. Gung, Y. Ge, M. Sunkara, and Y. Zhang. Optimizing llm-based multi-agent system with textual feedback: A case study on software development. *arXiv preprint arXiv:2505.16086*, 2025.

P. Shojaee, I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *SuperIntelligence-Robotics-Safety & Alignment*, 2(6), 2025.

P. Song, P. Han, and N. Goodman. Large language model reasoning failures. *arXiv preprint arXiv:2602.06176*, 2026.

H. Su, S. Diao, X. Lu, M. Liu, J. Xu, X. Dong, Y. Fu, P. Belcak, H. Ye, H. Yin, Y. Dong, E. Bakhturina, T. Yu, Y. Choi, J. Kautz, and P. Molchanov. Toolorchestra: Elevating intelligence via efficient model and tool orchestration, 2025. URL https://arxiv.org/abs/2511.21689.

V. Subramaniam, Y. Du, J. B. Tenenbaum, A. Torralba, S. Li, and I. Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains. *arXiv preprint arXiv:2501.05707*, 2025.

G. Tang, S. Jiang, H. Chang, N. Chen, Y. Li, H. Fan, J. Li, M. Liu, and B. Qin. Looprpt: Reinforcement pre-training for looped language models. *arXiv preprint arXiv:2603.19714*, 2026.

Tavily. Tavily search api. https://www.tavily.com, 2026.

G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, et al. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

K.-T. Tran, D. Dao, M.-D. Nguyen, Q.-V. Pham, B. O'Sullivan, and H. D. Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36:38975–38987, 2023.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

G. Wang, J. Li, Y. Sun, X. Chen, C. Liu, Y. Wu, M. Lu, S. Song, and Y. A. Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025a.

J. Wang, W. Jue, B. Athiwaratkun, C. Zhang, and J. Zou. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

B. Xu, C. Li, W. Wang, W. Fan, T. Zheng, H. Shi, T. Fan, Y. Song, and Q. Yang. Towards multi-agent reasoning systems for collaborative expertise delegation: An exploratory design study. *arXiv preprint arXiv:2505.07313*, 2025.

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

H. Yang, H. Chen, H. Guo, Y. Chen, C.-S. Lin, S. Hu, J. Hu, X. Wu, and X. Wang. Llm-medqa: Enhancing medical question answering through case studies in large language models. *arXiv preprint arXiv:2501.05464*, 2024a.

Y. Yang, Q. Peng, J. Wang, Y. Wen, and W. Zhang. Llm-based multi-agent systems: Techniques and business perspectives. *arXiv preprint arXiv:2411.14033*, 2024b.

Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. 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*, pages 2369–2380, 2018.

H. Ye, Z. Gao, M. Ma, Q. Wang, Y. Fu, M.-Y. Chung, Y. Lin, Z. Liu, J. Zhang, D. Zhuo, et al. Kvcomm: Online cross-context kv-cache communication for efficient llm-based multi-agent systems. *arXiv preprint arXiv:2510.12872*, 2025a.

R. Ye, X. Liu, Q. Wu, X. Pang, Z. Yin, L. Bai, and S. Chen. X-mas: Towards building multi-agent systems with heterogeneous llms. *arXiv preprint arXiv:2505.16997*, 2025b.

X. Yu, Z. Chen, Y. He, T. Fu, C. Yang, C. Xu, Y. Ma, X. Hu, Z. Cao, J. Xu, et al. The latent space: Foundation, evolution, mechanism, ability, and outlook. *arXiv preprint arXiv:2604.02029*, 2026.

M. Yuksekgonul, F. Bianchi, J. Boen, S. Liu, P. Lu, Z. Huang, C. Guestrin, and J. Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

S. Yun, J. Peng, P. Li, W. Fan, J. Chen, J. Zou, G. Li, and T. Chen. Graph-of-agents: A graph-based framework for multi-agent llm collaboration. In *The Fourteenth International Conference on Learning Representations*, 2026.

A. L. Zhang, T. Kraska, and O. Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025a.

Q. Zhang, C. Hu, S. Upasani, B. Ma, F. Hong, V. Kamanuru, J. Rainton, C. Wu, M. Ji, H. Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025b.

W. Zhao, M. Yuksekgonul, S. Wu, and J. Zou. Sirius: Self-improving multi-agent systems via bootstrapped reasoning. *arXiv preprint arXiv:2502.04780*, 2025.

Y. Zheng, Z. Zhao, Z. Li, Y. Xie, M. Gao, L. Zhang, and K. Zhang. Thought communication in multiagent collaboration. *arXiv preprint arXiv:2510.20733*, 2025.

H. Zhou, X. Wan, R. Sun, H. Palangi, S. Iqbal, I. Vulić, A. Korhonen, and S. Ö. Arık. Multi-agent design: Optimizing agents with better prompts and topologies. *arXiv preprint arXiv:2502.02533*, 2025.

R.-J. Zhu, Z. Wang, K. Hua, T. Zhang, Z. Li, H. Que, B. Wei, Z. Wen, F. Yin, H. Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

J. Zou, X. Yang, R. Qiu, G. Li, K. Tieu, P. Lu, K. Shen, H. Tong, Y. Choi, J. He, J. Zou, M. Wang, and L. Yang. Latent collaboration in multi-agent systems, 2025. URL `https://arxiv.org/abs/2511.20639`.

# Table of Contents

# Appendix

## A. Theoretical Analysis

### A.1. Running Complexity Analysis

**Proposition A.1** (**Restate of Proposition 3.1**). *Without RecursiveLink, a text-based Recursive MAS with the same collaboration structure requires runtime complexity of* $\Theta(N(m|V|d_h + (t+m)d_h^2 + (t+m)^2 d_h))$*; In contrast, with RecursiveLink-enabled collaboration, RecursiveMAS achieves an end-to-end runtime complexity of* $\Theta(N(md_h^2 + (t+m)d_h^2 + (t+m)^2 d_h))$.

*Proof of Proposition 3.1.* We analyze the runtime complexity for each agent and then extend the result to the full multi-agent system. For each single agent, given the context length is at most $t$, and the generation length is at most $m$, the Transformer processes a sequence of length at most $t+m$, requiring $\Theta((t+m)d_h^2)$ time for the feed-forward layers and $\Theta((t+m)^2 d_h)$ time for self-attention. This standard Transformer computation is shared by both RecursiveMAS and text-based Recursive MAS.

The computational difference comes from how RecursiveMAS process the generated embeddings. In RecursiveMAS, each of the $m$ latent embeddings is transformed by RecursiveLink, which incurs an additional cost of $\Theta(md_h^2)$. In text-based Recursive MAS, each embedding must be decoded into an explicit token by projecting it to the vocabulary space and computing logits over $|V|$ tokens, resulting in an additional cost of $\Theta(m|V|d_h)$.

Adding all terms together, our proposed RecursiveMAS needs $\Theta(md_h^2 + (t+m)d_h^2 + (t+m)^2 d_h)$ time for each agent, while text-based Recursive MAS needs $\Theta(m|V|d_h + (t+m)d_h^2 + (t+m)^2 d_h)$ time. Since there are $N$ agents in the system, our proposed RecursiveMAS needs $\Theta(N(md_h^2 + (t+m)d_h^2 + (t+m)^2 d_h))$ time, and text-based Recursive MAS needs $\Theta(N(m|V|d_h + (t+m)d_h^2 + (t+m)^2 d_h))$ time in total. □

### A.2. Realistic Assumptions

**Assumption A.1.** *Text-based SFT can be regarded as using* $\mathcal{R}_{text}(h) = W_{in}\,\mathrm{softmax}(W_{out}h)$ *as the recursive link, where* $W_{in}$ *is the token-to-embedding matrix, and* $W_{out}$ *denotes the embedding-to-logits matrix. We also assume* $\|W_{in}\|_2 \le O(1)$ *and* $\|W_{out}\|_2 \le O(1)$*. For RecursiveLink* $\mathcal{R}(h)$*, we assume that* $W_1, W_2$ *follow Kaiming normal initialization, and we only analyze the case where* $W_3 = I$.

### A.3. Learning Advantage Analysis

**Theorem A.2** (**Restate of Theorem 4.1**). *Under the Realistic Assumptions (stated in Appendix A.2), if tokens are confident with entropy* $\le \epsilon$*, where typically* $\epsilon \ll 1$*: directly applying text-based SFT (denoted by* $\mathcal{R}_{text}(h)$*) during recursion suffers from gradient vanishing (i.e., gradient norm close to 0); while RecursiveMAS with the RecursiveLink* $\mathcal{R}$ *maintains stable and near constant gradients (i.e., gradient norm close to 1) during looped backpropagation process. Formally, with probability* $\ge 1-\delta$*,*

$$\left\|\frac{\partial \mathcal{R}_{text}(h)}{\partial h}\right\|_2 \le O(\epsilon) \ll 1, \qquad \left\|\frac{\partial \mathcal{R}(h)}{\partial h}\right\|_2 \ge \Omega\left(1 - \sqrt{\frac{1}{d_h}\log\frac{1}{\delta}}\right). \tag{8}$$

*Proof of Theorem 4.1.* We first analyze the gradient behavior of text-based recursive interaction. By

applying the chain rule to $\mathcal{R}_{\text{text}}(h)$, the gradient matrix is:

$$J_{\text{text}} = \frac{\partial \mathcal{R}_{\text{text}}(h)}{\partial h} = W_{\text{in}} S W_{\text{out}}, \qquad S = \text{diag}(p) - pp^T,$$

where $p = \text{softmax}(W_{\text{out}} h)$ is the next token distribution. Then, by the sub-multiplicativity of the spectral norm,

$$\|J_{\text{text}}\|_2 \le \|W_{\text{in}}\|_2 \|S\|_2 \|W_{\text{out}}\|_2 \le O(1) \cdot \|S\|_2 \cdot O(1) = O(\|S\|_2).$$

Because $S$ represents the covariance matrix of a categorical distribution, it is symmetric and positive semi-definite. Thus, its spectral norm is upper-bounded by its trace:

$$\|S\|_2 \le \text{Tr}(S) = \sum_{i=1}^{|V|} (p_i - p_i^2) = \sum_{i=1}^{|V|} p_i - \sum_{i=1}^{|V|} p_i^2 = 1 - \|p\|_2^2.$$

Using the logarithm inequality $\ln z \le z - 1$ (for all $z > 0$), we can lower-bound the entropy:

$$\epsilon \ge \sum_{i=1}^{|V|} p_i(-\ln p_i) \ge \sum_{i=1}^{|V|} p_i(1 - p_i) = 1 - \|p\|_2^2.$$

Substituting this inequality back into the norm bound yields:

$$\|S\|_2 \le 1 - \|p\|_2^2 \le \epsilon.$$

Therefore, combining the constants, we have:

$$\left\| \frac{\partial \mathcal{R}_{\text{text}}(h)}{\partial h} \right\|_2 = \|J_{\text{text}}\|_2 \le O(\epsilon).$$

We next analyze the gradient behavior of RecursiveMAS. Applying the chain rule to $\mathcal{R}(h)$, the gradient matrix is:

$$J = \frac{\partial \mathcal{R}(h)}{\partial h} = I + W_2 \Sigma' W_1,$$

where $\Sigma' = \text{diag}(\sigma'(W_1 h))$. By the triangle inequality for the matrix norm,

$$\Big| \|J\|_2 - 1 \Big| = \Big| \|J\|_2 - \|I\|_2 \Big| \le \|J - I\|_2 = \|W_2 \Sigma' W_1\|_2.$$

Since $W_1, W_2$ follow Kaiming initialization, and the GELU function $\sigma$ has $|\sigma'| \le O(1)$, then by the subgaussian matrix concentration inequality, $\|W_2 \Sigma' W_1\|_2 \le O\left(\sqrt{\frac{1}{d_h} \log \frac{1}{\delta}} + 1\right)$ with probability $\ge 1 - \delta$. It follows that:

$$\|J\|_2 \ge \Omega\left(1 - \sqrt{\frac{1}{d_h} \log \frac{1}{\delta}}\right).$$

□

# B. Experiment Setups

## B.1. Evaluation Datasets

We introduce all datasets used in our experiments as follows:

**Mathematical Reasoning.**

- **MATH500** (HuggingFaceH4, 2023) is a widely used subset of the MATH benchmark, containing mathematical problems across algebra, geometry, number theory, probability, and combinatorics.
- **AIME2025** (math ai, 2025) contains 30 challenging problems from the 2025 American Invitational Mathematics Examination. These problems require olympiad-style derivations and precise numerical answers, providing a compact but difficult benchmark for mathematical reasoning.
- **AIME2026** (MathArena, 2026) follows the same AIME-style questions with 30 challenging competition-level math problems. We use it to further test performance and generalization on difficult mathematical reasoning tasks.

**Scientific and Medical Tasks.**

- **GPQA-Diamond** (Rein et al., 2023) is the most difficult split of GPQA, consisting of graduate-level multiple-choice questions in biology, physics, and chemistry. It requires specialized scientific knowledge and careful multi-step reasoning beyond shallow factual recall.
- **MedQA** (Yang et al., 2024a) contains medical licensing-style questions that assess biomedical knowledge, clinical reasoning, and diagnostic decision-making. The benchmark requires integrating domain-specific evidence from patient scenarios and selecting the most appropriate answer.

**Code Generation.**

- **LiveCodeBench-v6** (Jain et al., 2025) is a contamination-resistant code generation benchmark built from recent programming problems. It evaluates whether models can synthesize functionally correct programs under realistic problem specifications and hidden test cases.
- **MBPP Plus** (Liu et al., 2023) extends the original MBPP benchmark with more comprehensive test cases for Python program synthesis. The stricter execution-based evaluation provides a more reliable measure of functional correctness.

**Search-based Question Answering.**

- **HotpotQA** (Yang et al., 2018) is a multi-hop question answering benchmark based on Wikipedia. It requires models to gather and combine evidence from multiple supporting facts, making it suitable for evaluating search-based reasoning.
- **Bamboogle** (Press et al., 2023) is a compact but challenging benchmark for search-intensive multi-hop reasoning. Its questions often require decomposition and intermediate retrieval steps before composing the final answer.

## B.2. Compared Baselines

We compare our method against the following baselines:

**Single-Agent Fine-tuning Baselines.**

- **Single Agent (w/ LoRA)** uses the final agent from the corresponding collaboration pattern and trains it with LoRA adapters using the same training data as RecursiveMAS.
- **Single Agent (w/ Full-SFT)** further fine-tunes all parameters of the same single-agent backbone using fully supervised fine-tuning.

**Representative Multi-Agent Frameworks.**

- **Mixture-of-Agents (MoA)** (Wang et al., 2025b) organizes multiple LLM agents into a layered multi-agent system, where agents in each layer aggregate responses from the previous layer to produce refined outputs as the final answer.
- **TextGrad** (Yuksekgonul et al., 2025) optimizes multi-agent systems by propagating natural-language feedback as textual gradients. We use TextGrad as a baseline method for text-mediated system optimization.

**Recursion-based Methods.**

- **LoopLM** (Zhu et al., 2025) is a looped language model that performs recursive computation with shared transformations in latent space. In our evaluation, we mainly adopt the Ouro-2.6B model.
- **Recursive-TextMAS** uses the same recursive multi-agent collaboration structure as RecursiveMAS, but agents communicate through explicit text rather than latent representations.

### B.3. Additional Implementation Details

**Training Data Curation.** To optimize RecursiveMAS under our inner-outer training pipeline, we construct role-specific supervision targets for each agent across all collaboration patterns. We start by collecting question-answer pairs as raw training samples from four domains, including s1K (Muennighoff et al., 2025), m1K (Huang et al., 2025), OpenCodeReasoning (Ahmad et al., 2025), and ARPO-SFT (Dong et al., 2025). For each training sample, we rewrite the original answer into agent-level targets according to the role assignments of each collaboration pattern, as follows:

- For *Sequential-Style*, we use Qwen3.5-397B-A17B to rewrite the answers into an initial step-by-step plan and a refined critic-guided plan. During training, the initial plan is used as the supervision target for the Planner, the critic-guided plan is used for the Critic, and the original answer is retained for the Solver.
- For *Mixture-Style*, each specialist in the MAS first generates responses for questions from its specialized domain, and these responses are then used to supervise the corresponding specialist. The ground truth answers are used as targets for the Summarizer.
- For *Distillation-Style*, the Expert first generates guidance-style responses for each training sample, which are then used as supervision targets for the Expert. The Learner is supervised directly by the ground-truth answers.
- For *Deliberation-Style*, we use the ground truth answers as the supervision targets for both the Reflector and Tool-Caller agent.

After the role-specific data construction, each agent is assigned its own training pairs, where the input consists of the original question, and the output is the corresponding role-based response. These agent-level pairs are then used as the supervision data for subsequent training.

**Implementation Details.** During training, all base LLMs' parameters are frozen, and we only update the inner and outer RecursiveLink using AdamW with a batch size of 4 and a maximum sequence

length of 4096 tokens. During inference, the maximum generation length is set for 2000 tokens for MATH500, 4000 tokens for MedQA, GPQA-Diamond, LiveCodeBench, and MBPP Plus, and 16000 tokens for AIME2025/2026. For all Qwen models, we enables the official Instruct mode (Qwen Team, 2026) for more efficient and controllable answer generation. For the Deliberation-Style MAS, we provide a standard Python environment and a Tavily (Tavily, 2026) search API as external tools. We implement RecursiveMAS and baselines with both HuggingFace Transformer (Face, 2025) and vLLM backend (Kwon et al., 2023). All experiments are conducted on H100 and A100 GPUs.

**Evaluation Protocol.** Across all non-code generation tasks, we first normalize the extracted answer by removing extra whitespace, stripping punctuation, and converting text to lowercase before applying task-specific correctness checks.

- For *numerical problems (MATH500, AIME2025, AIME2026)*, we compare the numerical value of the extracted answer with the ground truth. The answer is considered correct if the two values are mathematically equivalent.
- For *multiple-choice questions (GPQA-Diamond MedQA)*, we directly compare the predicted choice with the ground truth letter, where an exact option match is counted as correct.
- For *code generation tasks (LiveCodeBench and MBPP Plus)*, we first extract the generated code block and then execute it with provided test cases in a sandboxed python environment. Each individual test case is assigned a timeout of 10 seconds to prevent non-terminating programs.
- For *search-based tasks (HotpotQA, Bamboogle)*, we follow the standard LLM-as-a-judge evaluation method (Li et al., 2025a) and use the Qwen3.5-397B-A17B model as a binary judge to determine whether the generated answer is correct with respect to the ground truth answer.

When an output reaches the maximum generation length without producing an extractable answer, we follow standard early-stopping evaluation methods (Muennighoff et al., 2025; Yang et al., 2025) by appending “Final Answer:” to the model output to elicit a final response.

Table 6 | Comparison of RecursiveMAS in Distillation-Style Multi-agent System. RecursiveMAS improves the Learner agent by 8.0% via distilling knowledge from the Expert agent while retaining a 1.5× end-to-end speed advantage over the Expert agent.

| Method (Distillation-Style) | Metric | AIME2026 | GPQA-D | LiveCodeBench | MBPP+ | MedQA |
|---|---|---|---|---|---|---|
| Expert Model | Acc. | 90.0 | 72.7 | 46.2 | 73.4 | 86.0 |
| | Time | 9473 | 2558 | 9352 | 2342 | 2124 |
| Learner Model | Acc. | 76.7 | 61.4 | 38.4 | 67.5 | 77.9 |
| | Time | 4495 | 1289 | 5396 | 1171 | 1183 |
| **RecursiveMAS** | Acc. | 83.3 | 70.0 | 40.1 | 71.9 | 83.0 |
| | Time | 5967 | 1671 | 6863 | 1516 | 1436 |

## C. Additional Related Work

**Latent-Space Collaboration.** Beyond text-based interaction, recent studies have explored leveraging the latent space as an alternative medium for LLM communication. One line of work studies transferring hidden embeddings for cross-model communication (Du et al., 2025; Yu et al., 2026), while other works investigate reusing internal states to share information across LLMs (Fu et al., 2025; Ye et al., 2025a). Recent studies extend this scheme to agentic settings, where latent interfaces are used to support coordination among multiple agents (Zheng et al., 2025; Zou et al., 2025). Different from these studies, RecursiveMAS treats latent information as part of a system-level recursive information flow, enabling heterogeneous agents to recursively collaborate and improve as a unified MAS.

## D. Additional Experiments

### D.1. Results on Different Collaboration Patterns

Table 7 | Comparison of RecursiveMAS in Mixture-Style Multi-agent System.

| Method (Mixture-Style) | AIME2026 | GPQA-Diamond | LiveCodeBench | MedQA |
|---|---|---|---|---|
| Math Specialist | 43.3 | 37.4 | 18.9 | 29.0 |
| Code Specialist | 13.3 | 26.2 | 21.5 | 43.3 |
| Science Specialist | 10.0 | 27.0 | 7.6 | 48.1 |
| **RecursiveMAS** | **46.7** | **43.0** | **23.8** | **61.7** |

Table 8 | Comparison of RecursiveMAS in Deliberation-Style Multi-agent System.

| Method (Deliberation-Style) | AIME2026 | GPQA-Diamond | HotpotQA | Bamboogle |
|---|---|---|---|---|
| Reflector | 76.7 | 61.2 | 27.5 | 40.9 |
| Tool-Caller | 86.7 | 63.1 | 39.6 | 49.8 |
| **RecursiveMAS** | 90.0 | 65.0 | 41.4 | 53.7 |

We report the detailed results of RecursiveMAS under three additional collaboration patterns in Tables 7, 6, and 8, corresponding to the summarized results in Figure 1 (Down). In both Mixture and

Deliberation settings, RecursiveMAS achieves consistent accuracy gains over the strongest individual agent in each setting. In Distillation Style, RecursiveMAS improves performance over the Learner while requiring substantially less inference time than the Expert. Overall, these results show that RecursiveMAS provides both performance gains and efficiency benefits across diverse MAS collaboration patterns, further demonstrating the generality of our method.

### D.2. Ablations on Latent Thoughts Lengths

Table 9 | Ablation Study on Length of Latent Thoughts $m$ transferred across agents on RecursiveMAS.

| **Latent Steps** | **0** | **16** | **32** | **48** | **64** | **80** | **96** | **112** | **128** |
|---|---|---|---|---|---|---|---|---|---|
| Math500 | 83.3 | 84.9 | 85.2 | 85.6 | 86.8 | 86.8 | 86.5 | 86.9 | 86.7 |
| GPQA-D | 61.4 | 62.0 | 62.8 | 63.6 | 64.1 | 64.2 | 64.5 | 64.3 | 64.4 |
| LiveCodeBench | 38.1 | 40.3 | 40.7 | 41.4 | 42.0 | 42.5 | 42.2 | 42.6 | 42.6 |

We provide detailed ablation results on the length of latent thoughts $m$ in Table 9, corresponding to the plot in Figure 8. As $m$ increases, RecursiveMAS consistently improves across all benchmarks, and the performance gradually saturates around $m = 80$, suggesting that a moderate latent thought budget is sufficient for effective latent collaboration.

# E. Prompt Template for RecursiveMAS

## Prompt Template for Sequential-Style RecursiveMAS

**System Prompt for All Agents:**
You are a helpful assistant.

**User Prompt for Planner Agent:**
You are a planner agent in a recursive multi-agent system. Here is the latent information from previous round: {Latent Thought Embeddings}. Given the latent information, you should output a step-by-step plan to solve the question: {Question}

**User Prompt for Critic Agent:**
You are a critic agent in a recursive multi-agent system. Here is the latent information from previous agent: {Latent Thought Embeddings}. Given the latent information, you should critique the initial plan and output an improved plan to solve the question: {Question}

**User Prompt for Solver Agent:**
You are a solver agent in a recursive multi-agent system. Here is the latent information from previous agent: {Latent Thought Embeddings} Given the latent information, you should solve the question and provide the final answer: {Question}
Solve the question and put the final answer inside \boxed{}.

## Prompt Template for Mixture-Style RecursiveMAS

**System Prompt for All Agents:**
You are a helpful assistant.

**User Prompt for Math Specialist Agent:**
You are a math specialist agent in a recursive multi-agent system. Here is the latent information from previous round: {Latent Thought Embeddings} Given the latent information, you should provide a domain-specific answer for the question: {Question}

**User Prompt for Science Specialist Agent:**
You are a science specialist agent in a recursive multi-agent system. Here is the latent information from previous round: {Latent Thought Embeddings} Given the latent information, you should provide a domain-specific answer for the question: {Question}

**User Prompt for Code Specialist Agent:**
You are a code specialist agent in a recursive multi-agent system. Here is the latent information from previous round: {Latent Thought Embeddings} Given the latent information, you should provide a domain-specific answer for the question: {Question}

**User Prompt for Summarizer Agent:**
You are a summarizer agent in a recursive multi-agent system. Here is the latent information from the math specialist: {Math Specialist Latent Thought Embeddings}. Here is the latent information from the code specialist: {Code Specialist Latent Thought Embeddings}. Here is the latent information from the science specialist: {Science Specialist Latent Thought Embeddings}. Given the latent information from all previous specialists, you should aggregate their reasoning and provide the final answer to the question: {Question}
Put the final answer inside \boxed{}.

## Prompt Template for Distillation-Style RecursiveMAS

**System Prompt for All Agents:**
You are a helpful assistant.

**User Prompt for Expert Agent:**
You are an expert agent in a recursive multi-agent system. Here is the latent information from previous round: {Latent Thought Embeddings}. Given the latent information, you should provide expert-level guidance to solve the question: {Question}

**User Prompt for Learner Agent:**
You are a learner agent in a recursive multi-agent system. Here is the latent information from previous agent: {Latent Thought Embeddings}. Given the latent information, you should use the expert guidance to solve the question and provide the final answer: {Question}
Put the final answer inside \boxed{}.

## Prompt Template for Deliberation-Style RecursiveMAS

**System Prompt for All Agents (with Tool-Calling Instructions):**
You are a helpful assistant that can solve the given question with the help of external tools, including a web search tool and a Python interpreter. For each question, you should reason carefully and use tools when they are necessary. You can invoke the web search tool to retrieve factual information, and use the Python interpreter to perform calculations or execute code. The search query and search result should be enclosed within `<search> </search>` and `<result> </result>` tags respectively. The Python code and execution result should be enclosed within `<python> </python>` and `<result> </result>` tags respectively.

**User Prompt for Reflector Agent:**
You are a reflector agent in a recursive multi-agent system. Here is the latent information from previous round: {Latent Thought Embeddings}. Given the latent information, you should reflect on the current solution and provide an improved solution to the question: {Question}

**User Prompt for Tool-Caller Agent:**
You are a tool-caller agent in a recursive multi-agent system. Here is the latent information from the Reflector agent: {Latent Thought Embeddings}. Given the latent information, you should solve the question: {Question}
Put the final answer inside \boxed{}.

# F. Case Study on Different Recursion Rounds

## Output of RecursiveMAS in Recursion Round 1 (On MATH500)

**Question:**
For how many positive integers $n > 1$ is it true that $2^{24}$ is a perfect $n^{\text{th}}$ power?

---

**Final Answer of RecursiveMAS:**
First, let's express $k$ in terms of its prime factorization. Since $2^{24}$ is a power of 2, $k$ must also be a power of 2. Let $k = 2^m$ for some integer $m$. Then we have:

$$2^{24} = (2^m)^n = 2^{mn}$$

This equation tells us that $24 = mn$. We need to find all pairs of positive integers $(m, n)$ such that $mn = 24$ and $n > 1$. Let's list the factor pairs of 24:

$$(1, 24), (2, 12), (3, 8), (4, 6), (6, 4), (8, 3), (12, 2), (24, 1)$$

We discard the pairs where $n = 1$ because we are looking for $n > 1$. The remaining pairs are:

$$(2, 12), (3, 8), (4, 6), (6, 4), (8, 3), (12, 2)$$

Each pair $(m, n)$ corresponds to a valid $n$ greater than 1. Therefore, the possible values of $n$ are 2, 3, 4, 6, 8, and 12. Counting these values, we find there are 6 possible values for $n$.
Thus, the number of positive integers $n > 1$ for which $2^{24}$ is a perfect $n$-th power is $\boxed{6}$. *Incorrect Answer* ✗

## Output of RecursiveMAS in Recursion Round 2 (On MATH500)

**Question:**
For how many positive integers $n > 1$ is it true that $2^{24}$ is a perfect $n^{\text{th}}$ power?

---

**Final Answer of RecursiveMAS:**
First, let's express $2^{24}$ in terms of its prime factorization: $2^{24}$
For $2^{24}$ to be a perfect $n^{\text{th}}$ power, $n$ must be a divisor of 24. This is because if $2^{24} = k^n$, then $k$ must be a power of 2, say $k = 2^m$. Therefore, we have:

$$2^{24} = (2^m)^n = 2^{mn}$$

Equating the exponents, we get:

$$24 = mn$$

We need to find all pairs $(m, n)$ such that $mn = 24$ and $n > 1$. Let's list the divisors of 24:

$$1, 2, 3, 4, 6, 8, 12, 24$$

Since $n > 1$, we exclude 1 from the list of divisors. The remaining divisors are:

$$2, 3, 4, 6, 8, 12, 24$$

Thus, the number of positive integers $n > 1$ for which $2^{24}$ is a perfect $n^{\text{th}}$ power is: $\boxed{7}$ *Correct Answer* ✓

## Output of RecursiveMAS in Recursion Round 3 (On MATH500)

**Question:**
For how many positive integers $n > 1$ is it true that $2^{24}$ is a perfect $n^{\text{th}}$ power?

---

**Final Answer of RecursiveMAS:**
First, let's express $2^{24}$ in terms of its prime factorization:
For $2^{24}$ to be a perfect $n$-th power, $n$ must be a divisor of 24. This is because if $2^{24} = k^n$, then $k$ must be of the form $2^m$ for some integer $m$, and thus we have $(2^m)^n = 2^{mn} = 2^{24}$, this implies:

$$mn = 24$$

We need to find all positive divisors of 24. The divisors of 24 are:

$$1, 2, 3, 4, 6, 8, 12, 24$$

Since we are looking for $n > 1$, we exclude 1 from the list. The remaining divisors are:

$$2, 3, 4, 6, 8, 12, 24$$

Thus, the number of positive integers $n > 1$ for which $2^{24}$ is a perfect $n$-th power is: $\boxed{7}$ *Correct Answer* ✓

## Output of TextMAS (On MATH500)

**Question:**
For how many positive integers $n > 1$ is it true that $2^{24}$ is a perfect $n^{\text{th}}$ power?

---

**Output of Planner Agent:**
Ok, I need to provide a step-by-step plan to solve the problem.

- Step 1: Determine the prime factorization of $2^{24}$.
- Step 2: Recognize that $2^{24}$ can be expressed as a perfect $n^{\text{th}}$ power if the exponent 24 is divisible by $n$.
- Step 3: Find all positive integers $n > 1$ such that 24 is divisible by $n$.
- Step 4: Count the number of such integers $n$.
- Step 5: Conclude that the number of positive integers $n > 1$ for which $2^{24}$ is a perfect $n^{\text{th}}$ power is equal to the number of divisors of 24 that are greater than 1.
- Step 6: List the divisors of 24 greater than 1 and count them.
- Step 7: The final answer is the count of such divisors.

---

**Output of Critic Agent:**
Here is a detailed plan for the problem:

Step 1: Analyze the problem
We are asked to find how many positive integers $n > 1$ satisfy the condition that $2^{24}$ is a perfect $n^{\text{th}}$ power.

Step 2: Recall the definition of a perfect $n^{\text{th}}$ power
A number $x$ is a perfect $n^{\text{th}}$ power if there exists an integer $k$ such that $x = k^n$.

Step 3: Express $2^{24}$ as a power of some base
We can write $2^{24} = (2^m)^n$, where $m$ and $n$ are positive integers.

Step 4: Find the possible values of $n$
For $2^{24}$ to be a perfect $n^{\text{th}}$ power, $n$ must be a divisor of 24 (since $2^{24} = (2^m)^n$ implies $n$ divides 24).

Step 5: Count the number of valid divisors
Find the total number of positive divisors of 24, and then subtract 1 to account for the restriction that $n > 1$.

---

**Output of Solver Agent:**
To determine for how many positive integers $n > 1$ the number $2^{24}$ is a perfect $n$-th power, we need to find all the divisors of 24, since $2^{24}$ can be written as

$$(2^k)^n$$

, where $kn = 24$. First, we list all the divisors of 24. The divisors of 24 are:

$$1, 2, 3, 4, 6, 8, 12, 24$$

Each divisor $d$ of 24 corresponds to a possible value of $n$ such that $2^{24}$ is a perfect $n$-th power. Specifically, if $d$ is a divisor of 24, then $n = \frac{24}{d}$ will be an integer and $2^{24} = (2^d)^{24/d}$.
Let's list the corresponding $n$ values for each divisor $d$: - If $d = 1$, then $n = \frac{24}{1} = 24$ - If $d = 2$, then $n = \frac{24}{2} = 12$ - If $d = 3$, then $n = \frac{24}{3} = 8$ - If $d = 4$, then $n = \frac{24}{4} = 6$ - If $d = 6$, then $n = \frac{24}{6} = 4$ - If $d = 8$, then $n = \frac{24}{8} = 3$ - If $d = 12$, then $n = \frac{24}{12} = 2$ - If $d = 24$, then $n = \frac{24}{24} = 1$ (but since $n > 1$, we exclude this case)
Thus, the possible values of $n$ are $2, 3, 4, 6, 8, 12, 24$. There are 7 such values.
Therefore, the number of positive integers $n > 1$ for which $2^{24}$ is a perfect $n$-th power is: $\boxed{7}$ *Correct Answer* ✓

# G. Examples of RecursiveMAS Across Different Downstream Tasks

## Case Study of RecursiveMAS on Mathematical Reasoning Questions (AIME2026)

**Question:**
A plane contains points $A$ and $B$ with $AB = 1$. Point $A$ is rotated in the plane counterclockwise through an acute angle $\theta$ around point $B$ to point $A'$. Then $B$ is rotated in the plane clockwise through angle $\theta$ around point $A'$ to point $B'$. Suppose that $AB' = \frac{4}{3}$. The value of $\cos\theta$ can be written as $\frac{m}{n}$ , where $m$ and $n$ are relatively prime positive integers. Find $m+n$.

---

**Recursion Round 1 (in latent space):**
***Planner*** → ***Critic*** → ***Solver*** →

---

**Recursion Round 2 (in latent space):**
***Planner*** → ***Critic*** → ***Solver*** →

---

**Recursion Round 3 (with final output):**
***Planner*** → ***Critic*** → ***Solver*:**
To solve the problem, we need to analyze the geometric transformations and use the given distances to find the value of $\cos\theta$.

1. **Initial Setup:** Points $A$ and $B$ are in the plane with $AB = 1$. Point $A$ is rotated counterclockwise through an acute angle $\theta$ around point $B$ to point $A'$. Point $B$ is then rotated clockwise through the same angle $\theta$ around point $A'$ to point $B'$. We are given that $AB' = \frac{4}{3}$.
2. **Coordinate System:** Place point $B$ at the origin $(0,0)$ and point $A$ at $(1,0)$.
3. **Rotation of Point $A$:** When point $A$ is rotated counterclockwise by angle $\theta$ around point $B$, its new coordinates $A'$ are:
$$A' = (1\cos\theta, 1\sin\theta) = (\cos\theta, \sin\theta).$$
4. **Rotation of Point $B$:** Point $B$ is rotated clockwise by angle $\theta$ around point $A'$. The coordinates of $B'$ after this rotation can be found using:
$$B' = A' + R_{-\theta}(B - A'),$$
where $R_{-\theta}$ is the rotation matrix for clockwise rotation by $\theta$:
$$R_{-\theta} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$
The coordinates of $B - A'$ are $B - A' = (0 - \cos\theta, 0 - \sin\theta) = (-\cos\theta, -\sin\theta)$. Applying the rotation matrix:
$$R_{-\theta}(B - A') = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} -\cos\theta \\ -\sin\theta \end{pmatrix} = \begin{pmatrix} -\cos^2\theta - \sin^2\theta \\ \cos\theta\sin\theta - \sin\theta\cos\theta \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}.$$
Therefore, the coordinates of $B'$ are:
$$B' = (\cos\theta, \sin\theta) + (-1, 0) = (\cos\theta - 1, \sin\theta).$$
5. **Distance $AB'$:** The distance $AB'$ is given by:
$$AB' = \sqrt{(\cos\theta - 1 - 1)^2 + (\sin\theta - 0)^2} = \sqrt{(\cos\theta - 2)^2 + \sin^2\theta}.$$
Simplifying inside the square root:
$$AB' = \sqrt{\cos^2\theta - 4\cos\theta + 4 + \sin^2\theta} = \sqrt{1 - 4\cos\theta + 4} = \sqrt{5 - 4\cos\theta}.$$
We are given that $AB' = \frac{4}{3}$, so $\sqrt{5 - 4\cos\theta} = \frac{4}{3}$. Squaring both sides gives $5 - 4\cos\theta = \frac{16}{9}$. Solving for $\cos\theta$:
$$5 - \frac{16}{9} = 4\cos\theta \implies \frac{45}{9} - \frac{16}{9} = 4\cos\theta \implies \frac{29}{9} = 4\cos\theta \implies \cos\theta = \frac{29}{36}.$$
6. **Final Answer:** The value of $\cos\theta$ is $\frac{29}{36}$. The sum of the numerator and the denominator is $m + n = 29 + 36 = 65$.

Thus, the final answer is $\boxed{65}$. *Correct Answer* ✓

## Case Study of RecursiveMAS on Scientific and Medical Questions (AIME2026)

**Question:**
A 45-year-old mechanic presents to the emergency department complaining of acute-onset shortness of breath while repairing a plowing tractor for his neighbor. The patient denies having any history of asthma or respiratory symptoms, and does not smoke. His temperature is 99.8°F (37.7°C), pulse is 65/min, blood pressure is 126/86 mmHg, and respirations are 20/min. His oxygen saturation is 97%. On exam, he is pale and diaphoretic. His pupils are contracted. Diffuse wheezes are noted in all lung fields. What is the best treatment for his condition?
A. Succinylcholine
B. Inhaled ipratropium and oxygen
C. Atropine and pralidoxime
D. Inhaled albuterol and oxygen

---

**Recursion Round 1 (in latent space):**
***Planner*** → ***Critic*** → ***Solver*** →

---

**Recursion Round 2 (in latent space):**
***Planner*** → ***Critic*** → ***Solver*** →

---

**Recursion Round 3 (with final output):**
***Planner*** → ***Critic*** → ***Solver*:**
The clinical presentation is highly suggestive of organophosphate poisoning. Clinical Signs:

- **Dizziness/Shortness of breath**: Suggests respiratory distress.
- **Pupils are contracted (Miosis)**: This is a classic sign of cholinergic excess (SLUDGE mnemonic: Salivation, Lacrimation, Urination, Defecation, GI upset, Emesis).
- **Diffuse wheezes**: Caused by bronchorrhea and bronchospasm due to overstimulation in the lungs.
- **Pale and diaphoretic**: Diaphoresis is a key sign of cholinergic crisis.
- **Vitals**: The patient is tachypneic (20/min is borderline but in context of distress, this is significant) and hypoxic (implied by distress, though 97% is maintained, the saturation will drop quickly without treatment).

Analysis of Options:

- **A. Succinylcholine**: Depolarizing neuromuscular blocker. Exacerbates or prolongs paralysis due to high endogenous acetylcholine; does not resolve the underlying cholinergic crisis.
- **B. Inhaled ipratropium and oxygen**: Anticholinergic. Relieves local respiratory symptoms but fails to address the root cause and systemic cholinergic toxicity.
- **C. Atropine and pralidoxime**: **Correct treatment**. Atropine (muscarinic antagonist) is life-saving, reversing severe secretions and bronchospasm. Pralidoxime reactivates acetylcholinesterase to resolve nicotinic effects.
- **D. Inhaled albuterol and oxygen**: Beta-2 agonist. May help bronchospasm but ineffective against massive bronchorrhea and systemic poisoning.

**Conclusion**: The patient is suffering from organophosphate poisoning (cholinergic crisis). The standard of care involves the administration of atropine to reverse muscarinic effects (especially respiratory secretions and bronchospasm) and pralidoxime to regenerate acetylcholinesterase. So the final answer is: $\boxed{C}$ *Correct Answer* ✓

## Case Study of RecursiveMAS on Code Generation Tasks (MBPP Plus)

**Question:**
Write a function to check if each element of second tuple is smaller than its corresponding element in the first tuple.

---

**Recursion Round 1 (in latent space):**
***Planner*** → ***Critic*** → ***Solver*** →

---

**Recursion Round 2 (in latent space):**
***Planner*** → ***Critic*** → ***Solver*** →

---

**Recursion Round 3 (with final output):**
***Planner*** → ***Critic*** → ***Solver*:**
To solve this problem, we need to write a function that takes two tuples as input and checks if each element of the second tuple is smaller than its corresponding element in the first tuple. If all elements satisfy this condition, the function should return True; otherwise, it should return False.
Here is the code logic:

1. Define the function ‘check_smaller‘ that takes two tuples as input.
2. Check if the lengths of the two tuples are equal. If they are not, return False.
3. Iterate over the elements of the tuples and check if each element of the second tuple is smaller than its corresponding element in the first tuple.
4. If all elements satisfy the condition, return True; otherwise, return False.

Here is the Python code for the solution:

```
def check_smaller(tuple1, tuple2):
    # Check if the lengths of the tuples are equal
    if len(tuple1) != len(tuple2):
        return False

    # Iterate over the elements of the tuples
    for i in range(len(tuple1)):
        if tuple1[i] <= tuple2[i]:
            return False

    # If all elements satisfy the condition, return True
    return True
```

*Correct Answer* ✓

## Case Study of RecursiveMAS on Search Based Tasks (Bamboogle)

**Question:**
Who was president of the United States in the year that Citibank was founded?

---

**Recursion Round 1 (in latent space):**
***Planner*** → ***Critic*** → ***Solver*** →

---

**Recursion Round 2 (in latent space):**
***Planner*** → ***Critic*** → ***Solver*** →

---

**Recursion Round 3 (with final output):**
***Planner*** → ***Critic*** → ***Solver*:**
To answer this question, I need to find out the year Citibank was founded and then identify the President of the United States in that year.
First, I will search for the founding year of Citibank.

<search>When was Citibank founded?</search>
<result>
[1] Citigroup - Wikipedia
Citibank was chartered by the State of New York on June 16, 1812, with $2 million of capital. · · ·
[2] What is Brief History of Citi Company? - Matrix BCG
The genesis of Citi, originally named the City Bank of New York, dates back to June 16, 1812, when it received its charter in New York City. The bank commenced operations on September 14, 1812. · · ·
[3] Citibank - Students | Britannica Kids | Homework Help
Second largest bank in U.S., with about 270 branch banks in New York City area. Present name since 1976. · · ·
[4] A bank with history and tradition - Citi.com
Citibank opened for business in 1812 in New York under its original name City Bank of New York. Since then, Citi has been a major player in some of the world's most innovative breakthroughs in banking. · · ·
</result>

Based on the search results, Citibank (originally City Bank of New York) was founded in **1812**. Specifically, it was chartered on June 16, 1812, and opened for business on September 14, 1812. Now I need to find out who was the President of the United States in 1812.

<search>President of the United States 1812</search>
<result>
[1] The Election of 1812 | American Battlefield Trust
On October 14, 1812, incumbent president, James Madison, wrote to his friend and former president, · · ·
[2] The War of 1812: The Nation Braces for War
James Madison (1751-1836) James Madison, the Father of the Constitution and the Bill of Rights, was President of the United States during the War of 1812. · · ·
[3] Key American Personalities | 1812 Virtual Exhibition
As secretary of state under President Thomas Jefferson, future president James Madison had been deeply involved in the events leading to the war. · · ·
[4] James Madison - Wikipedia
Madison was elected president in 1808. Motivated by a desire to acquire land held by Britain, Spain, and Native Americans, Madison led the United States into the War of 1812. · · ·
</result>

The search results indicate that James Madison was the President of the United States in 1812. So, the answer is *James Madison* *Correct Answer* ✓