arXiv:2611.20891v1 [cs.AI] 16 Nov 2026

From Recursion to Convolution: Parallel Self-Improvement for Multi-Agent Systems

openJiuwen Team
Huawei Technologies Co., Ltd.
Source Code: https://github.com/LYA2333/ConvRSI

Abstract

Multi-agent self-improvement enables agents to improve through interaction and feedback. However, sequential dependencies across agents and refinement rounds can propagate errors, increase computational costs, and limit parallelism. Moreover, refining individual features or agents in isolation can overlook their dependencies and restrict collective coordination. To address these challenges, we propose `ConvRSI`, a framework for parallelizing multi-agent self-improvement through convolution. `ConvRSI` couples intra-agent convolution, which jointly refines related agent features, with inter-agent graph convolution, which aggregates messages according to the collaboration topology. Both stages operate in latent space, enabling parallel updates without repeatedly decoding and re-encoding intermediate messages. Experiments on MATH-500 show that `ConvRSI` achieves 5.04% performance improvement while reducing 17.27% inference cost, and 45.11% latency compared with RecursiveMAS. It also supports diverse collaboration patterns by representing their topologies as adjacency matrices. These results demonstrate the potential of convolutional co-evolution for efficient multi-agent self-improvement.

Keywords: Recursive Self-Improvement, Parallel Multi-Agent Systems, Graph Convolutional Networks, Large Language Models

1. Introduction

Multi-agent systems (MAS) tackle complex tasks by coordinating agents with complementary roles, knowledge, and reasoning capabilities. Beyond collaboration among fixed agents, recent research explores how agents can improve through their interactions and experience. Prior work advances MAS through role-specific training feedback [Motwani et al., 2025] and the co-evolution of question generation, solving, and judging through self-play [Chen et al., 2026]. Recursive self-improvement (RSI) extends this direction by feeding improvements back into the system, enabling the updated system to guide subsequent rounds of improvement. For example, RecursiveMAS [Zou et al., 2026] iteratively refines shared latent states and jointly optimizes agents at the system level, while MASS [Lee et al., 2026] alternates workflow evolution with model training on self-generated trajectories. Despite this progress, scaling self-improvement to more agents and deeper iterations faces inherent challenges in optimization stability, computational efficiency, and the scope of collaboration, as discussed below.

First, deeper recursion propagates error. Each recursive round builds on earlier outputs or updates, allowing mistakes to influence subsequent decisions and become harder to correct. RecursiveMAS [Zou et al., 2026] jointly optimizes agents across rounds but retains the dependency chain through which errors can spread. Similar risks arise in other forms of agent RSI: inaccurate self-evaluation in MASS [Lee et al., 2026] can introduce flawed trajectories into subsequent training, while mistaken memory updates in Recuris [Yu et al., 2026] can affect future task execution. This raises a key question: can comparable improvement be achieved with fewer sequential updates?

Second, sequential multi-agent computation limits efficiency and parallelism. Multi-agent RSI repeatedly executes agents, exchanges information, and evaluates updates, increasing computational costs as agent count and iteration depth increase. Chain-like dependencies force later updates to wait for earlier results, limiting the use of available parallel computing resources. Such dependencies arise across recursive collaboration links in RecursiveMAS [Zou et al., 2026], the generation–verification–refinement pipeline in MALT [Motwani et al., 2025], and successive harness evolution cycles in ModularRSI [Wu et al., 2026]. This motivates restructuring agent updates to reduce the time and cost required to reach a target level of performance.

Third, individual feature updates and chain-like evolution limit global coordination. Each agent's behavior depends on related features, including task information, environmental observations, roles, experience, prompts, and messages. Refining one feature in isolation can overlook its dependencies on others. Existing methods address different aspects of improvement: Recuris [Yu et al., 2026] evolves memory, ModularRSI [Wu et al., 2026] evolves and integrates harness modules, MASS [Lee et al., 2026] improves workflows with model capabilities, and RecursiveMAS [Zou et al., 2026] refines hidden states across agents. These leave room for a complementary design that jointly models related features within each agent and aggregates information across agents to support global evolution. In particular, chain-like evolution passes information from one agent to the next, making distant agents interact through intermediate updates. A broader approach would aggregate agents for collective, rather than agent-by-agent evolution [Yu et al., 2026; Lee et al., 2026].

To address these challenges, we propose **ConvRSI**, a framework for *parallelizing multi-agent self-improvement through convolution*. The idea is to replace portions of sequential recursive computation with convolutional updates over agent features and multi-agent collaboration graphs. This introduces a *space–time scaling perspective*: improvement can be pursued not only by increasing temporal iteration depth, but also by expanding spatial computation within each iteration. Rather than simply adding more independent agents, ConvRSI uses structured convolutional operations to coordinate parallel updates within and across agents.

ConvRSI comprises two coupled stages of improvement. **Intra-agent convolution** applies one-dimensional convolution to hidden-state sequences representing an agent's task, role, experience, prompt, and messages. It jointly models local dependencies among these features, enabling each agent to independently refine its message state through the convolutional interactions. The features are *initialized once* and subsequently refined through convolution. **Inter-agent convolution** then applies graph convolution to the updated agent messages according to the collaboration topology. Through parallel matrix operations, it aggregates information across connected agents. The resulting messages are fed into the next intra-agent stage, coupling individual refinement with collective evolution. Both stages operate on latent representations, enabling efficient convolutional updates without repeatedly decoding messages into text and re-encoding them. This matrix-based design exposes parallelism across feature positions and agents throughout evolution. Moreover, ConvRSI supports diverse multi-agent collaboration patterns by representing their topologies as adjacency matrices for inter-agent graph convolution.

Empirically, we evaluate ConvRSI on 2 benchmarks spanning math and coding, using diverse model families and multi-agent collaboration patterns, including sequential style and mixture style. Compared with representative multi-agent and recursive baselines, ConvRSI improves task performance while reducing the cost of self-improvement. In particular, it achieves a 5.04% performance improvement, a 17.27% reduction in inference cost, and a 45.11% inference speedup. At comparable performance, ConvRSI achieves these gains with 1.82× and 1.80× inference speedups and generates 17.3% and

4.4% fewer tokens per problem, demonstrating the efficiency of convolutional co-evolution.

Our main contributions are:

- We introduce a **space–time perspective on multi-agent self-improvement**, exploring how spatially parallel computation can reduce reliance on deep sequential updates.
- We propose `ConvRSI`, which couples intra-agent convolution over related agent features with inter-agent graph convolution over collaboration topologies. Together, these stages support iterative individual refinement and collective evolution.
- We evaluate `ConvRSI` across diverse benchmarks, model families, and collaboration patterns, measuring performance gains under matched time and computational costs, as well as reductions in time and cost at comparable performance.

2. Related Work

Recursive Self-Improvement of Agents. Recursive self-improvement enables agents to improve their capabilities through feedback and accumulated experience. Existing work explores different targets of improvement, including model behavior, agent workflows, harness components, and memory. Specifically, RecursiveMAS [Zou et al., 2026] extends recursive computation to multi-agent collaboration by iteratively refining shared latent states and jointly optimizing agents through an inner–outer learning procedure. MASS [Lee et al., 2026] alternates multi-agent workflow optimization with model training on self-generated trajectories, using collective behavior as a source of self-supervision. MetaRSI [Tan et al., 2026] further explores meta-level self-improvement by coordinating data, harness, and model updates and adapting the strategy that schedules these updates. Beyond model-level improvement, ModularRSI [Wu et al., 2026] evolves agent harness modules using execution experience, Recuris [Yu et al., 2026] updates agent memory based on feedback, and Dream-RSI [Zheng et al., 2026] reuses discovery histories to improve exploration policies. Building on RSI, `ConvRSI` reorganizes the improvement process through convolutional updates, coupling individual agent refinement with collective evolution while reducing reliance on sequential recursive steps.

Multi-Agent Training and Co-Evolution. Multi-agent training methods improve collective performance by encouraging agents to develop complementary capabilities and learn from their interactions. MALT [Motwani et al., 2025] trains specialized agents for generation, verification, and refinement, while MAPoRL [Park et al., 2025] applies multi-agent reinforcement learning to improve collaboration among language models. Multi-Agent Evolve [Chen et al., 2026] uses interacting roles to generate training data and feedback for model self-improvement. Related approaches explore the evolution of agent populations, as in EvoAgent [Yuan et al., 2025], or the automatic optimization of agentic workflows, as in AFlow [Zhang et al., 2025]. `ConvRSI` jointly refines features within each agent and uses a graph to model collaboration among agents, enabling collective co-evolution.

Parallel and Efficient Multi-Agent Collaboration. Multi-agent systems improve performance through collaboration, but additional agents increase compute costs, while sequential interaction rounds increase latency and limit parallelism. Recent work [Wunderlich et al., 2026] compares the accuracy–compute trade-offs of self-consistency, self-refinement, multi-agent debate, and agent aggregation. Parallel self-refinement [Wang et al., 2026] generates multiple candidate solutions concurrently and synthesizes them into an improved response. Studies of multi-agent debate [Smit et al., 2024] further highlight the importance of comparison with strong alternatives under matched computational budgets. RecursiveMAS [Zou et al., 2026] improves the efficiency of recursive collaboration through latent-state communication. Complementing existing approaches to efficient collaboration, `ConvRSI` uses

matrix-based convolution to enable parallel updates across agent features and connected agents. Its latent-space operations avoid repeated text decoding and re-encoding during evolution.

3. Preliminary

3.1. Hidden States in Autoregressive Text Generation

An autoregressive language model generates a response one token at a time. Given an input context x and the previously generated tokens $y_{<t}$, a causal Transformer [Vaswani et al., 2017] produces a contextual hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^H$. An output head maps this state to the distribution of the next token:

$$p_\theta(y_t \mid x, y_{<t}) = \mathrm{softmax}\big(\mathbf{W}_{\mathrm{out}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{out}}\big)_{y_t}. \tag{1}$$

The hidden state is a continuous representation of the preceding context, whereas the output distribution is defined over discrete vocabulary tokens. During teacher forcing, the model conditions on a reference prefix $y^\star_{<t}$ and causal masking allows all reference positions in a given sequence to be processed in one forward pass. During free generation, the model instead conditions on its own preceding outputs.

Hidden states also provide a communication medium for a multi-agent system. Rather than decoding every intermediate thought into text, an agent can transform its hidden states into continuous input embeddings for another agent. Because agents may have different hidden widths and embedding spaces, these representations require learned mappings before they can be consumed by the receiving model.

3.2. Convolutional Computation

A one-dimensional convolution applies a shared local filter along an ordered axis [LeCun et al., 1998]. For feature vectors $\mathbf{x}_f$, its output at position f can be written as

$$\mathbf{z}_f = \sigma\left(\sum_{\delta \in \mathcal{K}} \mathbf{W}_\delta \mathbf{x}_{f+\delta} + \mathbf{b}\right), \tag{2}$$

where $\mathcal{K}$ denotes the filter offsets and σ is a nonlinearity. In our setting, the ordered axis contains different feature types for the agent; each position on this axis corresponds to a feature type.

A graph convolutional network (GCN) aggregates information in the neighborhoods of a graph [Kipf and Welling, 2017]. Let $\mathbf{A} \in \mathbb{R}^{N\times N}_{\geq 0}$ describe communication between N agents, using the convention that $A_{ij} > 0$ means that agent i receives information from agent j. We add a self-loop at each node and define

$$\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}, \qquad \widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}, \qquad \mathbf{S} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}. \tag{3}$$

Multiplying a matrix of agent representations by $\mathbf{S}$ aggregates the representations of each receiver's senders. In our implementation, this normalization can also be applied to a directed adjacency. In that case, $\mathbf{S}$ is not required to be symmetric.

3.3. Convolutional Multi-Agent Co-Evolution

We define **Convolutional Multi-Agent Co-Evolution** as a sequence of coupled representation updates. At the **intra-agent convolution** level, each agent combines its context features into a local message using a convolution over feature types. At the **inter-agent convolution** level, a graph convolution

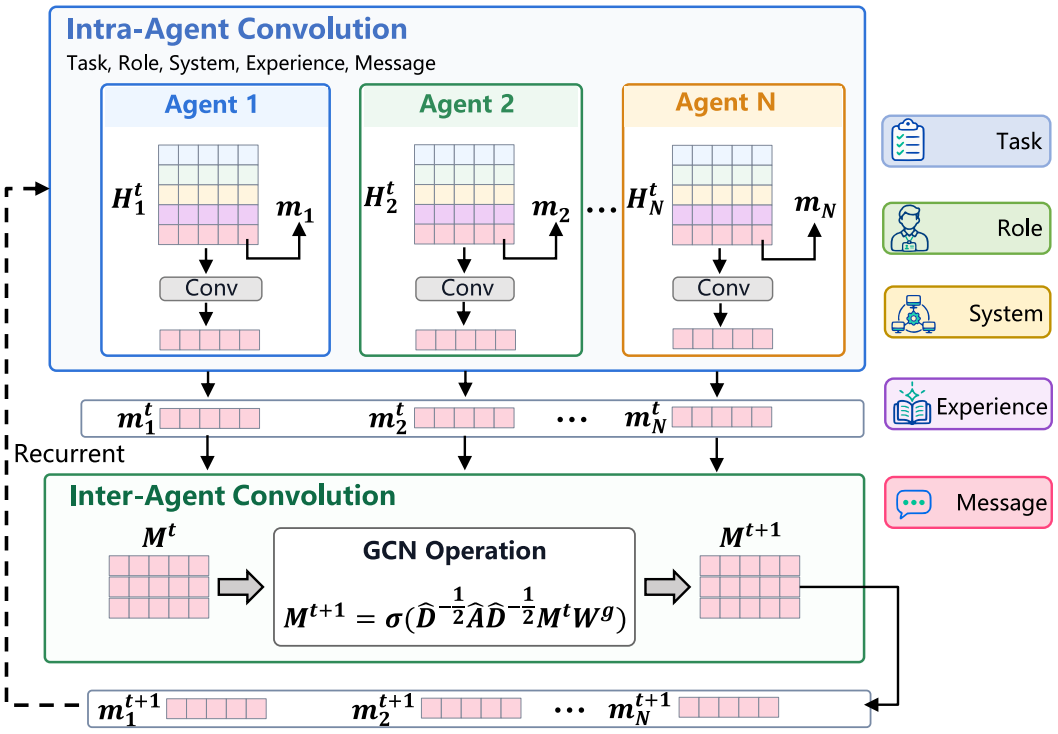


Figure 1: The `ConvRSI` framework shown for one slot across N agents. Intra-agent convolution produces a local message from each feature stack. The GCN aggregates the stacked local messages and returns one updated message per agent for the next round. The schematic uses M^t and M^{t+1} for the input and output of the graph layer, denoted by $\mathbf{U}_k$ and $\mathbf{M}_k$ in Eq. (7).

updates the messages jointly along the communication graph. The updated messages supply continuous prefixes for the current round and message features for the next round.

RecursiveMAS [Zou et al., 2026] performs multi-round collaboration in latent space, using InnerLinks to form continuous inputs and OuterLinks to transfer states between heterogeneous agents. `ConvRSI` augments this collaboration with a graph-message recurrence. The preceding GCN output supplies the recurrent message feature, and the underlying MAS routes agent execution states through its latent links.

4. Methodology

Consider a MAS with N agents and frozen language models $\{\mathcal{F}_i\}_{i=1}^{N}$ of native hidden widths H_i. The adjacency $\mathbf{A}$ specifies which agents exchange convolutional messages. The MAS supplies an execution schedule Π for model invocations and latent transfers, as well as a final output interface. Figure 1 shows a feature stack with $F = 5$ types ordered as *task*, *role*, *system*, *experience*, and *message*. The first $F - 1$ are static text features, with experience optional. The message feature is recurrent. In the full-sequence setting, each feature has K slots. The convolutional operators use a shared width D and run for R rounds.

4.1. Convolutional Co-Evolution

Intra-Agent Convolution Let $\mathbf{Z}_i \in \mathbb{R}^{F\times K\times H_i}$ contain the feature slots of agent i. The first round masks the message feature and initializes its slots to zero. Subsequent rounds obtain it from the preceding GCN output. An agent-specific projection maps each feature slot to the common space as

$$\mathbf{x}_{i,f,k} = \mathbf{W}_i^{\text{in}}\,\mathrm{LN}\big(\mathbf{z}_{i,f,k}\big) + \mathbf{b}_i \in \mathbb{R}^D. \tag{4}$$

We apply the feature mask after projection and after adding learned feature-type embeddings. At each slot k, we stack the resulting vectors in feature order to form the $F \times D$ convolution input.

We then apply a channel-wise one-dimensional convolution along the feature axis. For a tensor $\mathbf{V} \in \mathbb{R}^{F\times D}$, a convolutional block with the default residual connection is

$$\mathbf{V}' = \mathrm{LN}\big[\mathbf{V} + \mathrm{GELU}\big(\mathrm{Conv}(\mathbf{V})\big)\big]\,. \tag{5}$$

The operator Conv filters the feature axis independently in each channel, using a separate one-dimensional kernel per channel. The configured default has a kernel of size five, dilation one, and same padding. The convolution is applied independently to each slot and exchanges information across feature types while preserving the slot axis. In the optional no-residual variant, the block normalizes the activated convolution output without adding $\mathbf{V}$; masks follow the actual convolution windows.

Let $\mathbf{v}^{\text{conv}}_{i,f,k}$ denote a feature vector after the final convolutional block. We apply a softmax to the gate scores $g_i(\mathbf{v}^{\text{conv}}_{i,f,k})$ over the valid positions $\mathcal{A}_i$, yielding weights $\beta_{i,f,k}$. A residual multilayer perceptron and a normalized projection ψ_i transform the pooled vector $\overline{\mathbf{v}}_{i,k} = \sum_{f\in\mathcal{A}_i} \beta_{i,f,k}\mathbf{v}^{\text{conv}}_{i,f,k}$ into a local message

$$\mathbf{u}_{i,k} = \psi_i(\overline{\mathbf{v}}_{i,k} + \mathrm{MLP}_i(\overline{\mathbf{v}}_{i,k}))\,. \tag{6}$$

We denote the full local operator, from feature projection to message readout, by $\mathcal{C}_i$. Each agent has its own parameters and reuses them across rounds.

Inter-Agent Convolution For each slot, we stack the local messages into $\mathbf{U}_k = [\mathbf{u}_{1,k};\ldots;\mathbf{u}_{N,k}] \in \mathbb{R}^{N\times D}$. One GCN layer updates all nodes using the same pre-update matrix:

$$\mathbf{M}_k = \mathrm{LN}\Big[\mathrm{GELU}\Big(\mathbf{S}\mathbf{U}_k\mathbf{W}_g + \mathbf{1}\mathbf{b}_g^{\mathsf{T}}\Big)\Big]\,. \tag{7}$$

The i-th row, $\mathbf{m}_{i,k}$, is the updated graph message for agent i. The vector $\mathbf{1} \in \mathbb{R}^{N\times 1}$ broadcasts the zero-initialized learnable bias $\mathbf{b}_g$ to every agent. Layer normalization acts on the D channels of each row. The graph weights are shared among agents and rounds. Under Eq. (3), each agent aggregates the senders selected by $\mathbf{A}$ and its own message via a self-loop. When slots are padded, aggregation uses the active nodes at that slot. The GCN updates those nodes synchronously.

4.2. Parallel ConvRSI Training

Figure 2 shows how the operators in Fig. 1 are trained. Inner training learns to construct prefixes from each agent's local messages. Outer training jointly optimizes the graph-message recurrence and the latent transfer modules.

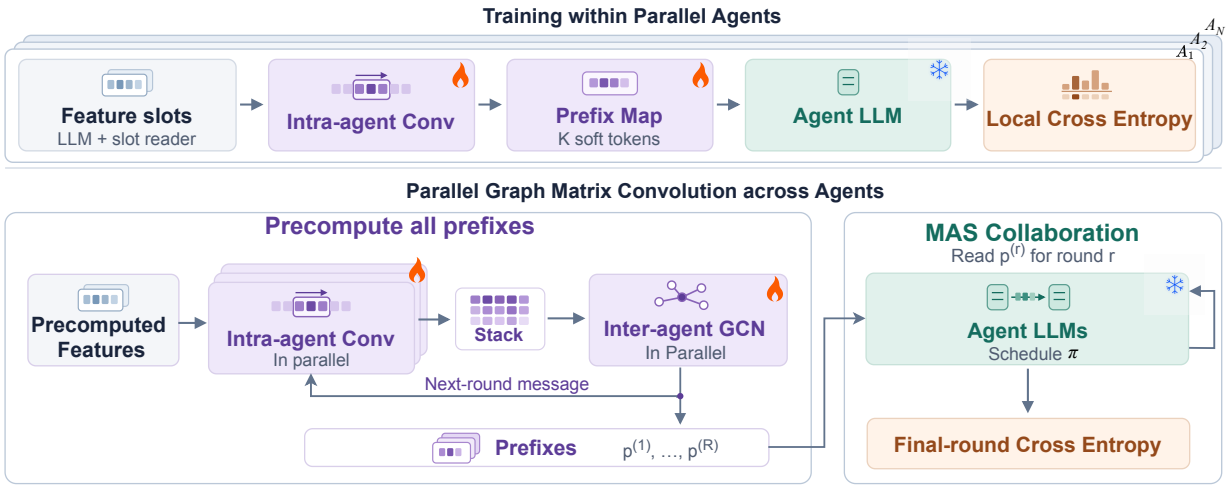


Figure 2: The `ConvRSI` training pipeline. Inner training uses independent agent objectives. Outer training can compute prefixes for all rounds through the convolutional recurrence before the collaboration forwards. The collaboration consumes the prefix for each round and computes cross-entropy at the final output. Flames highlight trainable components and snowflakes mark frozen language models.

Training within Parallel Agents Each static component $x_{i,f}$ is encoded independently by its agent's frozen language model. Let

$$\mathbf{H}_{i,f} = \mathrm{sg}\big[\mathcal{F}_i(\mathrm{Emb}_i(x_{i,f}))\big] \in \mathbb{R}^{L_{i,f}\times H_i},$$

where sg stops gradients through this static backbone computation. A trainable, agent-specific slot reader converts all valid hidden positions into K content-dependent slots:

$$\begin{aligned}\alpha_{i,f,\ell,k} &= \mathrm{softmax}_{\ell\in\mathcal{T}_{i,f}}\Big(\big[\mathbf{W}_i^{\mathrm{st}}\,\mathrm{LN}(\mathbf{h}_{i,f,\ell})\big]_k\Big),\\ \mathbf{z}_{i,f,k} &= \mathrm{LN}\left(\sum_{\ell\in\mathcal{T}_{i,f}} \alpha_{i,f,\ell,k}\mathbf{h}_{i,f,\ell} + \mathbf{e}^{\mathrm{st}}_{i,k}\right),\end{aligned} \tag{8}$$

where $\mathcal{T}_{i,f}$ contains the non-padding positions. Each agent shares its slot reader across its static text components. We compute these slots once per example in a system forward pass and reuse them across the convolutional rounds.

In Inner training, each agent applies $\mathcal{C}_i$ once with a masked message and maps the K resulting local messages to K soft prefix tokens. Its frozen LLM reads the prefix and predicts the local reference tokens under teacher forcing. The N losses in Eq. (11) are independent, so the agents can train concurrently.

Parallel Graph Matrix Convolution across Agents In Outer training, all agents can compute their local messages concurrently within a round. Stacking them yields a tensor of shape $[B, N, K, D]$ for a minibatch of B examples. We apply Eq. (7) as a batched graph operation on the BK matrices of shape $N \times D$. When the graph and node masks are fixed, the same normalization can be reused.

The superscript (r) now indexes states and prefixes that vary by round. Let $\mathcal{Q}_i : \mathbb{R}^D \to \mathbb{R}^{H_i}$ map a shared graph message to the input space of agent i. It gives the recurrent feature at $r > 1$ and the

prefix at every round as

$$\mathbf{z}_{i,\mathrm{msg},k}^{(r)} = \mathcal{Q}_i(\mathbf{m}_{i,k}^{(r-1)}), \qquad \mathbf{p}_{i,k}^{(r)} = \mathcal{Q}_i(\mathbf{m}_{i,k}^{(r)}). \tag{9}$$

Let $\mathbf{Z}_{\mathrm{static}}$ collect the static slots and $\mathbf{M}^{(r)}$ collect the updated graph messages over all slots. We write $\mathcal{C}$ for the local operators applied to their respective agents, $\mathcal{Q}$ for the corresponding $\mathcal{Q}_i$ applied to each graph row, and $\mathcal{G}_{\mathbf{A}}$ for the slot-wise graph update in Eq. (7). For $r > 1$, the messages evolve as

$$\mathbf{M}^{(r)} = \mathcal{G}_{\mathbf{A}}\Big(\mathcal{C}\Big(\mathbf{Z}_{\mathrm{static}}, \mathcal{Q}(\mathbf{M}^{(r-1)})\Big)\Big). \tag{10}$$

The first round uses the masked zero message. The recurrence requires only the cached static slots and the previous GCN output. As shown in the lower left of Fig. 2, we can run the convolutional rounds in order before the scheduled LLM forwards and store the resulting K-token prefix $\mathbf{p}_i^{(r)} = (\mathbf{p}_{i,k}^{(r)})_{k=1}^{K}$ for each agent and round.

Learning Convolution as a Whole When Π invokes agent i in round r, its LLM reads the prefix $\mathbf{p}_i^{(r)}$ before its chat input and consumes the latent states supplied by the schedule. For a transfer from i to j, the source InnerLink $\mathcal{I}_i$ and the edge-specific OuterLink $\mathcal{O}_{i\to j}$ map output hidden states to the receiver's input space, yielding $\mathbf{V}_{i\to j}^{(r)} = \mathcal{O}_{i\to j}(\mathcal{I}_i(\mathbf{H}_{i,\mathrm{out}}^{(r)}))$. The schedule governs which states are read within a round and which are carried forward. Invocations with independent inputs can run concurrently. The latent links share parameters across rounds.

Each teacher-forced invocation obtains the reference-conditioned hidden states in one forward pass. A configurable token budget caps the length of each transferred sequence during training. The final output yields $\mathcal{L}_{\mathrm{out}}$ in Eq. (11). The stored prefixes retain their computation graph, so the final loss backpropagates through the graph-message recurrence, the frozen backbones, and the differentiable latent transfers.

Training Objectives The local and final token-level cross-entropy objectives are

$$\mathcal{L}_{\mathrm{in},i} = \mathrm{CE}(\mathbf{P}_i, \mathbf{y}_i^\star), \qquad \mathcal{L}_{\mathrm{out}} = \mathrm{CE}(\mathbf{P}_{\mathrm{out}}^{(R)}, \mathbf{y}^\star). \tag{11}$$

Here $\mathbf{P}_i$ and $\mathbf{P}_{\mathrm{out}}^{(R)}$ are predicted vocabulary distributions at the local and final output positions. The corresponding targets are the reference token IDs $\mathbf{y}_i^\star$ and $\mathbf{y}^\star$. We average CE over supervised output tokens and then over examples. The Inner loss updates each agent's slot reader, local operator, and prefix map. The Outer loss jointly updates these modules, the GCN, and the OuterLinks, with frozen backbones and InnerLinks.

Collaboration Patterns The same convolutional operators can be applied to sequential, mixture, distillation, and deliberation styles by constructing a communication graph appropriate to the underlying MAS. These collaboration styles also appear in RecursiveMAS [Zou et al., 2026]. Table 1 gives representative adjacency constructions without fixing agent roles or backbone models. In every case, the degree matrix follows from $\widetilde{D}_{ii} = 1 + \sum_j A_{ij}$; it is derived from the adjacency. The execution schedule, output interface, and any pattern-specific supervision are supplied by the MAS, while the intra/inter-convolution definitions remain the same.

Table 1: Representative communication graphs for different collaboration styles. Rows of $\mathbf{A}$ are receivers and columns are senders. Unspecified entries are zero. In the mixture construction, o is a designated receiving node, $\mathcal{B}$ contains the remaining nodes, and $w_j > 0$. These are abstract graph constructions.

Style	Adjacency construction	Degree after adding self-loops
Sequential	$A_{i+1,i} = 1,\ 1 \le i < N$	$\widetilde{D}_{11} = 1;\ \widetilde{D}_{ii} = 2$ for $i > 1$
Mixture	$A_{o,j} = w_j,\ j \in \mathcal{B}$	$\widetilde{D}_{oo} = 1 + \sum_{j\in\mathcal{B}} w_j;\ \widetilde{D}_{jj} = 1$ for $j \in \mathcal{B}$
Distillation	Asymmetric influence: $A_{ij} \neq A_{ji}$ on selected pairs	$\widetilde{D}_{ii} = 1 + \sum_j A_{ij}$
Deliberation	Reciprocal exchange: $A_{ij}, A_{ji} > 0$ on interacting pairs	$\widetilde{D}_{ii} = 1 + \sum_j A_{ij}$

Inference Stage At inference, we encode the static inputs once, initialize the graph message with a masked zero, and use the same recurrence to supply prefixes to Π. For each intermediate invocation, the InnerLink maps the last hidden state to the next continuous input. The OuterLinks route the resulting hidden states according to Π. At the final output interface, the model autoregressively decodes the answer.

4.3. Stability and Computational Complexity Analysis

Stability For a fixed graph on the active agents, $\mathbf{S}$ is similar to the row-stochastic matrix $\widetilde{\mathbf{D}}^{-1}\widetilde{\mathbf{A}}$, so its spectral radius is one. A directed graph can yield $\|\mathbf{S}\|_2 > 1$. For fixed static slots and feature masks after initialization, let $L_{\mathcal{C}}$, $L_{\mathcal{Q}}$, L_σ, and L_{LN} bound the Lipschitz constants of the local operator in its message input, the shared-to-native map, GELU, and LayerNorm. Two trajectories of Eq. (10) satisfy

$$\|\Delta\mathbf{M}^{(r)}\|_F \le \kappa\|\Delta\mathbf{M}^{(r-1)}\|_F, \qquad \kappa = L_{\mathrm{LN}}L_\sigma\|\mathbf{S}\|_2\|\mathbf{W}_g\|_2 L_{\mathcal{C}}L_{\mathcal{Q}}. \tag{12}$$

Graph-message perturbations contract when $\kappa < 1$. For fixed teacher-forced references and latent sequence shapes under Π, suppose the retained execution states $\mathbf{V}^{(r)}$ satisfy

$$\|\Delta\mathbf{V}^{(r)}\| \le a\|\Delta\mathbf{M}^{(r)}\|_F + b\|\Delta\mathbf{V}^{(r-1)}\| \tag{13}$$

for uniform constants $a, b \ge 0$. Then $\max(\kappa, b) < 1$ is sufficient for both state perturbations to vanish over successive rounds.

Computational Complexity With J depthwise blocks of kernel width q, the per-round forward costs of local message and prefix construction and dense graph propagation are

$$C_{\mathrm{local}} = \mathcal{O}\left(BK\sum_{i=1}^{N}\left[FH_iD + JFqD + D^2 + DH_i\right]\right), \tag{14}$$

$$C_{\mathrm{graph}} = \mathcal{O}\left(BK\left[ND^2 + N^2D\right]\right). \tag{15}$$

The local terms cover feature projection, depthwise convolution, pooled channel transforms, and mapping to the native space. The graph terms cover channel projection and dense aggregation. The R rounds cost $R(C_{\mathrm{local}} + C_{\mathrm{graph}})$.

Let $C_i(L;B)$ be the cost of an agent-i backbone forward at length L. Static encoding costs $\sum_i \sum_{f=1}^{F-1} C_i(L_{i,f};B)$ per system forward, and slot reading costs $\mathcal{O}(BK\sum_{i,f} L_{i,f}H_i)$ over the same features. Teacher-forced collaboration adds $\sum_a C_{i(a)}(L_a;B)$ over scheduled invocations a with agent $i(a)$ and input length L_a. At inference, Q_Π intermediate invocations of T latent steps each require $Q_\Pi T$ latent forwards before final text decoding. Storing all round prefixes takes $\mathcal{O}(BRK\sum_i H_i)$ entries, in addition to cached feature slots and training activations.

5. Experiments

5.1. Experimental Setup

Datasets and metrics. We use the Sequential-Math and Sequential-Code training datasets released with RecursiveMAS [Zou et al., 2026] for mathematical reasoning and code generation, respectively. We evaluate mathematical reasoning on MATH-500 [Lightman et al., 2024; HuggingFaceH4, 2024] using final-answer accuracy and code generation on MBPP+ [Liu et al., 2023] using pass@1, the fraction of problems for which a single generated solution passes all benchmark tests. We report both metrics as percentages averaged over four random seeds.

Baselines and backbones. We compare `ConvRSI` with five baselines. The first two use the Sequential-Light configuration of RecursiveMAS [Zou et al., 2026]: Recursive-TextMAS exchanges intermediate outputs as text, while RecursiveMAS passes latent states between agents. Both use Qwen3-1.7B as the planner, Llama-3.2-1B-Instruct as the critic, and Qwen2.5-Math-1.5B-Instruct as the solver. The remaining baselines are ModularRSI [Wu et al., 2026], Hermes+ [Xue et al., 2026], and MASS [Lee et al., 2026]. We adapt all three to use the Sequential-Light solver backbone, Qwen2.5-Math-1.5B-Instruct [Yang et al., 2025b], in place of the DeepSeek backbones used by ModularRSI and Hermes+ and the 27B Qwen backbone used by MASS.

Inference and evaluation. We evaluate `ConvRSI`, Recursive-TextMAS, and RecursiveMAS at recursion depths $r \in \{1,2,3\}$. Following RecursiveMAS [Zou et al., 2026], we use top-p sampling with $p = 0.95$ on both benchmarks. For MATH-500, we set the temperature to 0.6 and the maximum generation length to 2,000 tokens. For MBPP+, we use a temperature of 0.2 and a maximum of 4,000 tokens. Within each benchmark, all methods receive the same problem statement in the input format required by each method and use the same decoding settings and scoring procedure. We measure inference efficiency on NVIDIA RTX 4090D GPUs and report total runtime (seconds) and the mean number of generated tokens per problem.

5.2. Main Results: Performance Across Recursive Depths

We evaluate `ConvRSI` on MATH-500 and MBPP+ at recursion depths $r \in \{1,2,3\}$. Table 2 reports MATH-500 accuracy, MBPP+ pass@1, total inference time on NVIDIA RTX 4090D GPUs, and generated-token counts. For the adapted ModularRSI, Hermes+, and MASS baselines, we use Qwen2.5-Math-1.5B-Instruct [Yang et al., 2025b], the solver backbone in the Sequential-Light configuration. It replaces the DeepSeek backbones in ModularRSI and Hermes+ and the 27B Qwen backbone in MASS.

Performance across recursion depths. On MATH-500, `ConvRSI` improves from 71.6% accuracy at $r = 1$ to 72.4% at $r = 2$ and 75.0% at $r = 3$. Its lead over the better of Recursive-TextMAS and RecursiveMAS grows from 0.2 to 1.4 to 3.6 percentage points across the three depths. Over the same range, RecursiveMAS improves from 70.1% to 71.4%, while Recursive-TextMAS declines from 71.4% to

69.6%. On MBPP+, Recursive-TextMAS leads at $r = 1$ with 30.4% pass@1, 0.4 points above ConvRSI. ConvRSI overtakes both recursive baselines at $r = 2$ and reaches 32.9% at $r = 3$, 0.8 points above RecursiveMAS and 3.8 points above Recursive-TextMAS.

Inference efficiency. ConvRSI completes both benchmarks in less time and generates fewer tokens than either recursive baseline at every depth. At $r = 3$, it completes MATH-500 in 1,459 s, compared with 2,658 s for RecursiveMAS and 4,721 s for Recursive-TextMAS, yielding 1.82× and 3.24× speedups, respectively. On MBPP+, the corresponding runtimes are 695 s, 1,249 s, and 3,575 s, yielding 1.80× and 5.14× speedups. At this depth, ConvRSI generates 17.3% fewer tokens on MATH-500 and 4.4% fewer on MBPP+ than RecursiveMAS.

Table 2: Performance and inference efficiency across recursion depths on MATH-500 and MBPP+. All results are reproduced using NVIDIA RTX 4090D GPUs. Accuracy is reported as a percentage (%), time in seconds (s), and token usage as generated tokens.

Method	MATH-500			MBPP+		
	Acc. (%) ↑	Time (s) ↓	Tokens ↓	Pass@1 (%) ↑	Time (s) ↓	Tokens ↓
Recursion round $r = 1$						
Recursive-TextMAS (text-only)	71.4	2254	1195	**30.4**	1489	1163
RecursiveMAS (latent)	70.1	1409	558	29.1	776	603
ConvRSI (Ours)	**71.6**	**1162**	**471**	30.0	**431**	**559**
Recursion round $r = 2$						
Recursive-TextMAS (text-only)	71.0	3761	2201	30.0	3030	1879
RecursiveMAS (latent)	70.9	1799	482	29.5	982	566
ConvRSI (Ours)	**72.4**	**1255**	**464**	**30.3**	**549**	**508**
Recursion round $r = 3$						
Recursive-TextMAS (text-only)	69.6	4721	3088	29.1	3575	2548
RecursiveMAS (latent)	71.4	2658	556	32.1	1249	611
ConvRSI (Ours)	**75.0**	**1459**	**460**	**32.9**	**695**	**584**

5.3. Generalization Across Collaboration Patterns

We evaluate whether the gains from ConvRSI extend beyond the sequential planner–critic–solver configuration to a mixture in which a summarizer combines the outputs of code, science, and mathematics specialists. Table 3 lists the agents in each configuration. Within each configuration, we compare ConvRSI with the released RecursiveMAS checkpoints [Zou et al., 2026].

Table 4 shows that ConvRSI improves on RecursiveMAS under both collaboration patterns. In the sequential configuration, it gains 3.6 percentage points on MATH-500 and 0.8 points on MBPP+. The gains are larger in the mixture: MATH-500 accuracy rises from 60.6% to 65.8% (+5.2 points), and MBPP+ pass@1 rises from 47.4% to 50.3% (+2.9 points). Thus, the improvement observed with sequential collaboration also holds when a summarizer combines specialist outputs.

Table 3: Agent roles and model assignments for the two collaboration patterns evaluated with ConvRSI. Sequential-Light uses a planner, critic, and solver in sequence. In the mixture, a summarizer combines outputs from code, science, and mathematics specialists.

Collaboration Pattern	Role	Model Size & Version
Sequential Style (Light)	Planner	Qwen3-1.7B [Yang et al., 2025a]
	Critic	Llama-3.2-1B-Instruct [Grattafiori et al., 2024]
	Solver	Qwen2.5-Math-1.5B-Instruct [Yang et al., 2025b]
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 [Yang et al., 2025b]
	Summarizer	Qwen3.5-2B [Yang et al., 2025a]

Table 4: MATH-500 accuracy and MBPP+ pass@1 (%) by collaboration pattern. Bold marks the higher score in each column.

	Sequential (Light)		Mixture	
Method	MATH-500	MBPP+	MATH-500	MBPP+
RecursiveMAS	71.4	32.1	60.6	47.4
ConvRSI	**75.0**	**32.9**	**65.8**	**50.3**

5.4. Comparison with Alternative Architectures

We compare ConvRSI with three alternative approaches to self-improvement. ModularRSI [Wu et al., 2026] uses successful and failed execution trajectories to improve individual modules of an agent's harness, such as tool use and context management. Hermes+ [Xue et al., 2026] retains and updates memories and procedural skills for reuse on later tasks. MASS [Lee et al., 2026] searches over multi-agent workflows and fine-tunes its model on trajectories generated by those workflows. We also include Recursive-TextMAS [Zou et al., 2026] as a text-based comparison within the Sequential-Light configuration.

Table 5 shows that ConvRSI achieves the highest scores among the configurations evaluated with Qwen solvers, reaching 75.0% accuracy on MATH-500 and 32.9% pass@1 on MBPP+. The closest comparison is Recursive-TextMAS: with the same Sequential-Light team and three recursion rounds, ConvRSI gains 5.4 percentage points on MATH-500 and 3.8 points on MBPP+. MASS also improves from Level-0 to Level-1 on both benchmarks, gaining 5.9 and 5.2 points, respectively. At Level-1, it trails ConvRSI by 6.3 points on MATH-500 and 4.2 points on MBPP+.

5.5. Efficiency Analysis

Performance at a comparable runtime. Figure 3 shows how much additional time each method spends as recursion depth increases and how its score changes. On MATH-500, three-round ConvRSI reaches 75.0% accuracy in 1,459 s. This is 4.9 percentage points above one-round RecursiveMAS (70.1%) for only 50 s of additional runtime over the full benchmark. On MBPP+, three-round ConvRSI reaches 32.9% pass@1 in 695 s, finishing sooner than either one-round baseline: RecursiveMAS reaches 29.1% in 776 s and Recursive-TextMAS reaches 30.4% in 1,489 s.

Table 5: Comparison on mathematical reasoning and code generation, grouped by method family. The second column identifies the solver backbone or agent configuration used in our experiments. Bold marks the highest score in each benchmark column.

Method	Solver or agent configuration	MATH-500 accuracy (%)	MBPP+ pass@1 (%)
Modular harness optimization			
ModularRSI	Qwen3.5-4B	67.1	32.1
Persistent memory and skills			
Hermes+	Sequential-Light solver	64.6	28.6
Hermes+	Qwen3.5-4B	67.2	27.3
Multi-agent workflow optimization and self-training			
MASS (Level-0)	Qwen2.5-Math-1.5B-Instruct	62.8	23.5
MASS (Level-1)	Qwen2.5-Math-1.5B-Instruct	68.7	28.7
Sequential multi-agent collaboration			
Recursive-TextMAS ($r = 3$)	Sequential-Light team	69.6	29.1
`ConvRSI` ($r = 3$)	Sequential-Light team	**75.0**	**32.9**

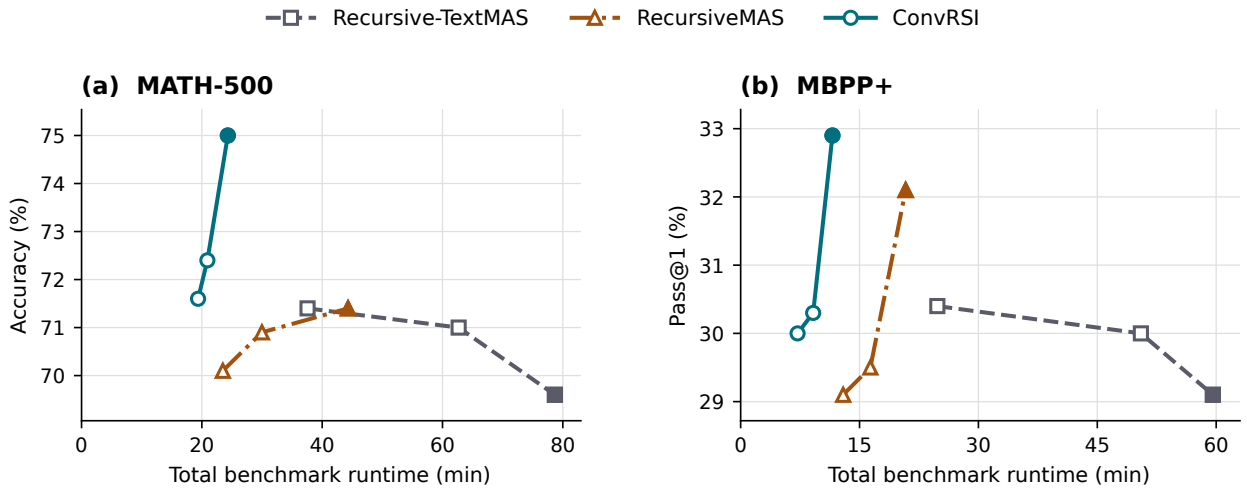


Figure 3: Performance versus total benchmark runtime. Along each line, points from left to right denote recursion depths $r = 1, 2, 3$, with a filled marker at $r = 3$. Points higher and further to the left indicate higher scores and shorter runtimes.

How inference cost grows with depth. Figure 4 compares each method with its own one-round configuration. On MATH-500, increasing the depth from one to three rounds raises the runtime of `ConvRSI` by 25.6%, compared with 88.6% for RecursiveMAS and 109.4% for Recursive-TextMAS. On MBPP+, the relative runtime increases for `ConvRSI` and RecursiveMAS are nearly identical (61.3% and 61.0%), but `ConvRSI` starts from a shorter one-round runtime and remains faster at three rounds. Generated-token counts change little for `ConvRSI` across these depths (-2.3% on MATH-500 and $+4.5\%$ on MBPP+), whereas they more than double for Recursive-TextMAS on both benchmarks.

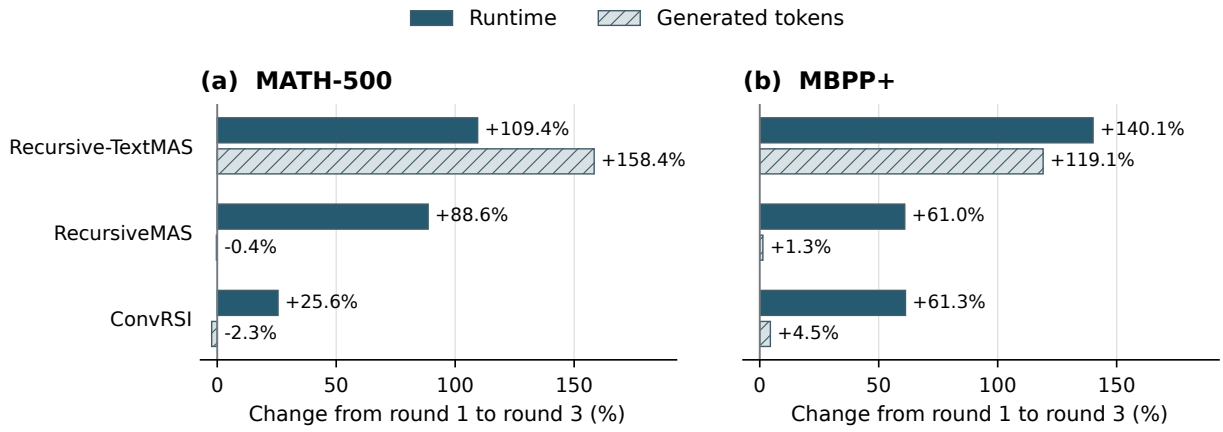


Figure 4: Change in total runtime and mean generated tokens per problem from $r = 1$ to $r = 3$. Each percentage is relative to the same method's one-round value on the same benchmark.

6. Conclusion

We presented `ConvRSI`, a framework that restructures multi-agent self-improvement through coupled intra-agent and inter-agent convolution. By jointly refining agent features and aggregating messages across a collaboration graph, `ConvRSI` supports individual refinement and collective evolution in latent space. Its matrix-based operations expose opportunities for parallel computation while accommodating different multi-agent collaboration patterns. Experiments show that `ConvRSI` improves task performance while reducing computational costs and latency across different multi-agent collaboration patterns. Future work will explore more diverse agent collaboration patterns and extend `ConvRSI` to large-scale agent swarms.

References

Yixing Chen, Yiding Wang, Siqi Zhu, Weixian Xu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: Llm self-improve through co-evolution. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, pages 39563–39593, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

HuggingFaceH4. Math-500 dataset. https://huggingface.co/datasets/HuggingFaceH4/MATH-500, 2024.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1609.02907.

Yanis Labrak, Adrien Bazoge, Emmanuel Morin, Pierre-Antoine Gourraud, Mickael Rouvier, and Richard

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.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. URL https://bottou.org/papers/lecun-98h.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Somayeh Sojoudi, Matei Zaharia, and Yujin Tang. Recursive self-improvement through multi-agent self-supervision, 2026. URL https://arxiv.org/abs/2610.12176.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, pages 21558–21572, 2023. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract.html.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=jXP9bgFack.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. MAPoRL2: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30215–30248. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1459. URL https://aclanthology.org/2025.acl-long.1459/.

Andries Petrus Smit, Nathan Grinsztajn, Paul Duckworth, Thomas D. Barrett, and Arnu Pretorius. Should we be going MAD? A look at multi-agent debate strategies for LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 45883–45905. PMLR, 2024. URL https://proceedings.mlr.press/v235/smit24a.html.

Zihan Tan, Leixin Sun, Zitong Shi, Yitao Liu, Jiajun Wu, Nathaniel Brooks, Jiaru Qian, Xiaoran Shang, Suyuan Huang, Yi Ding, et al. Metarsi/rsi2: A meta-recursive self-improving system for recursive self-improving systems themselves. *arXiv preprint arXiv:2609.06396*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/abs/1706.03762.

Qibin Wang, Pu Zhao, Shaohan Huang, Fangkai Yang, Lu Wang, Furu Wei, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Learning to refine: Self-refinement of parallel reasoning in llms. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 25904–25921, 2026.

Siwei Wu, Jincheng Ren, Yizhi Li, Haau-Sing Li, Chengran Yang, Xudong Ma, Yuxuan Zhang, Weicheng

Gu, Jian Yang, Riza Batista-Navarro, Chuanyi Zhang, Xianglong Liu, Ming Zhou, Bryan Dai, and Chenghua Lin. Modularrsi: Modular and generalizable recursive harness self-improvement. *arXiv preprint arXiv:2609.14857*, 2026.

Florian Valentin Wunderlich, Lars Benedikt Kaesberg, Jan Philip Wahle, Terry Ruas, and Bela Gipp. Multi-agent reasoning improves compute efficiency: Pareto-optimal test-time scaling. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 1–14. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-srw.1. URL https://aclanthology.org/2026.acl-srw.1/.

Shuhan Xue, Zixin Ding, Yichen Shen, Yinjie Wang, Zhenfei Yin, Yingcheng Wu, Yuxin Chen, Mengdi Wang, and Ling Yang. Past-bench: Benchmarking the foundations of recursive self-improvement in personal agents. *arXiv preprint arXiv:2608.04003*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Qwen: An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report, 2025b. URL https://arxiv.org/abs/2412.15115.

Zhaochen Yu, Yingcheng Wu, Zhenfei Yin, Kaiyuan Chen, Zhe Zhao, Mengdi Wang, Shuicheng Yan, and Ling Yang. Recursive experiential-working memory evolution for long-horizon agent harnesses. *arXiv preprint arXiv:2608.24876*, 2026.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. 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 6192–6217, 2025.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pages 34040–34077, 2025.

Tong Zheng, Xidong Wu, Zheng Zhang, Zhankui He, Chaoyi Zhang, Benjamin Coleman, Ruoqiao Wei, Di Bai, Haolin Liu, Rui Liu, et al. Dream-rsi: Recursive self-improvement through evolving worlds. *arXiv preprint arXiv:2609.14858*, 2026.

Jiaru Zou, Rui Pan, Ruizhong Qiu, Pan Lu, Shizhe Diao, Jindong Jiang, Hanghang Tong, Tong Zhang, Markus J. Buehler, Jingrui He, and James Zou. Recursive multi-agent systems. In *The Fortieth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=ML0KFwGcPG.

A. Author List

Core Contributors. Zheng Wang, Yuang Liu, Sachith Abeywickrama, Yangkai Ding, Tao Yu, Xinyu Zhang, Hongbo Wang, Qi Ye, Xuefeng Jin, Zhangchun Zhao