EVOLVE AS A TEAM: COLLABORATIVE SELF-EVOLUTION FOR LLM-BASED MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based multi-agent systems (MAS) have emerged as an effective paradigm for complex and long-horizon tasks. However, in real-world tasks, MAS often exhibit various failures during execution and such failures are difficult to eliminate during design. This motivates experience-driven MAS evolution, where a system improves based on its own execution experience. Yet such evolution is challenging because MAS experience is prolonged and intricate, interleaving multiple agents' execution chains and communication messages, which makes it difficult to identify what should be improved. To address this challenge, we propose **Meta-Team**, an experience-driven MAS evolution framework based on collaborative self-evolution. Meta-Team preserves the execution context of each agent and coordinates post-task communication, enabling agents to exchange distributed evidence for evolution. Building on this design, Meta-Team conducts multi-scale self-evolution, transforming execution experience into reusable improvements to agent behaviors, inter-agent coordination, and team-level organization. Across six long-horizon agent benchmarks, Meta-Team consistently outperforms single-agent systems, hand-crafted MAS, and prior MAS evolution methods; further analyses demonstrate that Meta-Team enables more reliable and scalable MAS self-evolution.

1 INTRODUCTION

Recent advancements in Large Language Models (LLMs) [1, 2, 3, 4] have driven rapid progress in LLM agents [5, 6, 7]. However, as LLM agents are increasingly applied to complex, long-horizon real-world tasks, a fundamental bottleneck of single-agent systems has emerged: they struggle to handle long-context information within a limited context window [8, 9] and face cognitive burden during long-horizon execution [10, 11, 12].

To address this bottleneck, LLM-based Multi-Agent Systems (MAS) have become an effective paradigm for complex and long-horizon tasks [13, 14, 15]. MAS leverage the "wisdom of the crowd" and follow a divide-and-conquer principle — they decompose complex tasks into manageable subtasks, assign them to specialized agents, and coordinate the agents to complete the task collaboratively [16, 17, 18]. This paradigm effectively reduces per-agent context overload and makes complex tasks more tractable. MAS have developed from early workflow-based systems [19, 20, 21] to recent industry-scale agent teams [22, 23], demonstrating their growing utility in solving real-world tasks.

Despite this progress, building effective MAS still requires substantial human effort [24, 25]. Developers often need to manually design core MAS components, including agent prompts, interaction protocols, workflow structures, and coordination mechanisms, making such systems costly to build and difficult to adapt to diverse tasks. Moreover, practical deployments show that MAS often exhibit failure modes such as inter-agent miscommunication, role drift, and cascading errors [26, 27, 28, 29]. Since these failure modes arise from actual execution, they cannot be fully eliminated through design-time specifications alone.

These limitations have motivated recent studies on automated MAS generation and evolution. Early efforts primarily follow two lines: (1) Performance-driven strategies, which leverage a meta-level optimizer to generate, search, or refine MAS configurations based on performance metrics (e.g., pass rates or final rewards) [30, 31, 32, 33, 34]; (2) Memorization-driven strategies, which directly store MAS trajectories or interaction fragments for subsequent retrieval [35, 36, 37]. However, both

[‡]Code is available at `https://anonymous.4open.science/r/Meta-Team-CD4D`.

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

paradigms lack fine-grained attribution over MAS experience: they provide limited insight into why and how a system succeeds or fails, leaving evolution to rely on exhaustive trial-and-error or superficial trajectory reuse rather than targeted improvement.

More recently, to bridge this attribution gap, studies have employed a single centralized LLM analyzer to inspect full MAS trajectories, pinpoint failures, and propose revisions for MAS evolution [38, 39, 40]. While promising, this single-analyzer paradigm remains highly unreliable on long trajectories: existing failure-attribution benchmarks show that centralized analysis performs poorly in pinpointing the decisive failure step, and the performance declines as trajectories grow longer [41, 42, 43]. We argue that this limitation stems from a fundamental architectural mismatch between MAS execution and MAS evolution — while MAS mitigate context overload during execution by distributing task information across agents, existing evolution methods regress by flattening the entire team's trajectory into a single context for one analyzer. Consequently, they reintroduce the single-context bottleneck that MAS were originally designed to overcome. This bottleneck is even more pronounced during evolution, given that the full team trajectory is particularly prolonged and intricate [44, 45].

In this work, we advocate a simple principle for MAS evolution: **a MAS should not only execute as a team, but also evolve as a team**. Following this principle, we propose **Meta-Team**, a MAS evolution framework based on collaborative self-evolution. Embracing the philosophy of *team reflexivity* in organizational psychology [46, 47], Meta-Team preserves local execution contexts within the agents that produced them and coordinates inter-agent communication so that agents can exchange information required for evolution. This collaborative organization matches the distributed structure of MAS experience, avoiding the context and reasoning burden of flattening the full MAS trajectory into a single analyzer. Specifically, Meta-Team conducts self-evolution at three levels: (i) *Agent-level evolution*, where each agent reflects on its own execution behavior and updates its individual agent scaffold; (ii) *Interaction-level evolution*, where agents revisit their collaboration history to refine communication and teammate profiles; and (iii) *Team-level evolution*, where the team revises its organization and shared coordination rules via collective discussion. Instead of evaluating on saturated benchmarks, we conduct experiments on six challenging long-horizon agent benchmarks, where Meta-Team consistently improves over existing agents, hand-crafted MAS, and prior MAS evolution methods. For example, Meta-Team outperforms hand-crafted MAS by 6.6% on average across the nine benchmark columns in Table 1. These results demonstrate the effectiveness of collaborative self-evolution across diverse long-horizon tasks.

In summary, our main contributions are:

- We identify an architectural mismatch in experience-driven MAS evolution: MAS execute through distributed contexts, whereas existing evolution methods centralize the full experience, making it difficult to attribute failures and identify what to improve.
- We propose **Meta-Team**, a collaborative self-evolution framework that preserves agent-local contexts, coordinates post-task communication to exchange distributed evidence, and translates the distributed evidence into multi-scale evolution.
- We conduct extensive experiments and analyses. These results validate the effectiveness and scalability of Meta-Team in MAS self-evolution.

2 PRELIMINARIES

2.1 LLM-BASED MAS

An LLM-based Multi-Agent System (MAS) consists of multiple LLM agents that collaborate to solve a task. We define a MAS as

$$\mathcal{M} \equiv (\mathcal{A}, \mathcal{O}),$$

where $\mathcal{A} = \{a_i\}_{i=1}^{N}$ is a set of agents and $\mathcal{O}$ denotes the orchestration mechanism that specifies how the agents are organized and how they interact and coordinate during task execution. Each agent a_i consists of a backbone LLM ϕ and an agent scaffold s_i, including its role instruction, task strategy, tools, and memory mechanism [31, 48]. The orchestration mechanism $\mathcal{O}$ can take various forms in existing systems [19, 49, 14]. For clarity, we decompose the orchestration mechanism as $\mathcal{O} = (\mathcal{P}, \mathcal{S})$, where $\mathcal{P}$ denotes inter-agent protocols and $\mathcal{S}$ denotes the shared team scaffold. The protocol $\mathcal{P}$ specifies when and how agents exchange information and hand off intermediate results.

For instance, workflow-based MAS often follow predefined information flows, whereas flexible agent-team systems allow agents to reason about when to contact teammates and what information to share. The shared team scaffold $\mathcal{S}$ specifies team-level rules visible to all agents, such as the team roster, organization, and shared constitution.

2.2 Experience-Driven Self-Evolution of MAS

Given a task x, executing a MAS $\mathcal{M}$ yields an execution experience defined as

$$e \equiv (x, \tau, r),$$

where τ is the multi-agent trajectory and r is the task outcome, such as the final deliverable and evaluator score. Unlike a single-agent trajectory, τ is not a single execution chain, but an interleaving of multiple agents' local execution chains, communication messages, and intermediate artifacts. We denote the local execution chain of agent a_i as τ_i, and write the complete trajectory as

$$\tau = \mathrm{Interleave}(\tau_1, \ldots, \tau_N, \mathcal{B}),$$

where $\mathcal{B}$ denotes cross-agent events such as messages and shared artifacts. This formulation highlights the distributed structure of MAS experience: each agent holds a local execution context, while the complete team trajectory also depends on cross-agent interactions.

Experience-driven self-evolution aims to improve a MAS from its own execution experience across tasks [50, 48, 40]. Given a sequence of experiences $\mathcal{E}_{1:k} = \{e_1, \ldots, e_k\}$, the MAS is updated by

$$\mathcal{M}_{k+1} = \Omega(\mathcal{M}_k, \mathcal{E}_{1:k}).$$

Since task outcomes are usually sparse, self-evolution requires determining what should be improved from the trajectory and applying changes to the MAS.

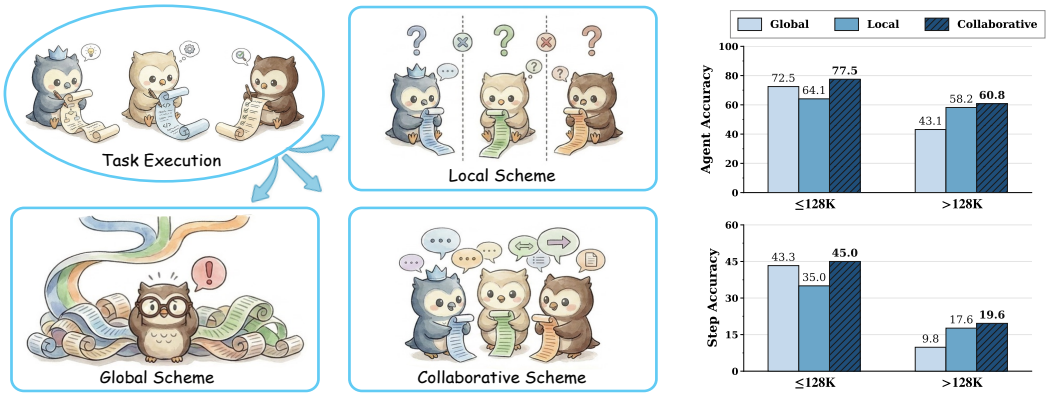
3 Method

3.1 Collaborative Scheme for MAS Self-Evolution

Real-world tasks often call for long-context understanding and long-horizon interaction [51, 52]. Solving them requires agents to perform hundreds of interaction steps and consume millions of context tokens, imposing a heavy burden of long-context understanding and long-horizon reasoning on a single agent. MAS alleviate this dual burden by distributing task information and execution responsibility across multiple agents during execution, allowing each agent to focus on more manageable subtasks within shorter and more coherent contexts [16, 17, 18, 11].

However, the distributed execution creates a new challenge for self-evolution: the resulting experience is longer and more complex than any single agent's local trajectory, making it difficult to identify what should be improved [53, 42, 45]. Existing analyzer-based evolution methods follow a *Global Scheme*: they flatten this joint experience and feed it to a single centralized LLM analyzer to attribute failures and propose revisions [39, 40]. While providing a global view, the global scheme concentrates the context and reasoning burden back on a single analyzer, reintroducing the bottleneck that MAS were originally designed to alleviate. A natural and naive counterpart is *Local Scheme*: decomposing the analysis by agent, with each analyzer reflecting on the corresponding local trajectory to identify local failures and propose agent-specific revisions, in the spirit of single-agent reflection [50, 54, 48]. Although the local scheme reduces the per-analyzer context length, it lacks a global view to understand how agents' local decisions, messages, and dependencies jointly shape the final outcome, making it difficult to identify what should truly be improved.

This reveals a global-local trade-off in organizing MAS experience for self-evolution. The global scheme preserves cross-agent coverage yet imposes a heavy centralized burden; the local scheme preserves local fidelity yet lacks cross-agent awareness. To address this trade-off, we propose a *Collaborative Scheme* for MAS self-evolution: each agent carries its local execution context into the evolution stage, while agents actively communicate to exchange processed local findings and trace how their outputs affected downstream execution. In this way, collaborative experience organization preserves the fidelity of agent-local contexts while recovering the cross-agent awareness needed to identify reliable targets for MAS evolution. Figure 1a illustrates how the global, local, and collaborative schemes organize MAS experience for self-evolution.



(a) Three schemes for organizing MAS experience during evolution. (b) Failure-attribution Comparison.

Figure 1: Three different schemes of experience-driven MAS self-evolution.

Empirical validation. To validate the global-local trade-off and the benefit of collaborative experience organization, we compare the three schemes in a failure-attribution setting.

Settings. We evaluate the global, local, and collaborative schemes on TraceElephant [45], a failure-attribution benchmark containing 220 real multi-agent failure traces from three public MAS frameworks. Each trace is annotated with a ground-truth (`mistake_agent`, `mistake_step`) tuple, and the goal is to identify the failure-causing agent (Agent Accuracy) and the decisive failure step (Step Accuracy). The three schemes differ only in how they organize the same MAS trajectory for attribution: global uses the full flattened trajectory, local uses isolated agent-wise trajectories, and collaborative uses agent-wise trajectories with cross-agent exchange. All schemes use the same backbone LLM, Claude Sonnet 4.6, and we report avg@3 Agent Accuracy and Step Accuracy. Implementation details are provided in Appendix C.1.

Findings. The results are shown in Figure 1, where the dataset is split by trajectory length into $\leq$ 128K and $>$ 128K regimes to examine how context length affects different schemes. Notably, all trajectories are within the 1M-token context limit of the backbone LLM, so the global scheme always has access to the full trajectory without truncation. On trajectories shorter than 128K, the global scheme outperforms the local scheme, suggesting that full-trajectory access is useful when the context remains manageable. However, on trajectories longer than 128K, the global scheme drops sharply and falls behind the local scheme on both metrics, indicating that long interleaved MAS traces impose a severe context and reasoning burden even without context overflow. By contrast, the collaborative scheme consistently achieves the best performance in both length regimes. These results show that, compared with purely global or local analysis, the collaborative scheme better aligns with the distributed structure of MAS experience and provides more reliable guidance for self-evolution.

Beyond empirical analysis, we give a theoretical analysis of the global, local, and collaborative schemes in Appendix C.2, which shows that the global–local ordering flips with trajectory length while the collaborative scheme dominates both regardless of length.

Discussion. Beyond its benefit for identifying failure causes through context decomposition, collaborative self-evolution reframes agents from passive targets of diagnosis into active contributors to system evolution. This collaborative philosophy brings two further advantages for MAS evolution.

Endogenous Feedback. Existing evolution methods often rely on final task outcomes or verifier scores as feedback [48, 37, 40], which are too sparse to specify how agents should improve their behavior. Collaborative self-evolution enriches this feedback through post-task communication: agents explain how others' outputs affected their decisions and receive feedback on how their own outputs affected downstream execution. In this way, the team's interactions become endogenous feedback signals for agent evolution.

Bottom-Up Self-Evolution. Collaborative self-evolution also enables agents to surface system-level improvements. Since agents observe different parts of the execution, they can identify team-level issues such as missing roles, redundant responsibilities, unclear handoffs, or ineffective coordination rules. Following the principle of team reflexivity [46, 47], the team can aggregate these observations

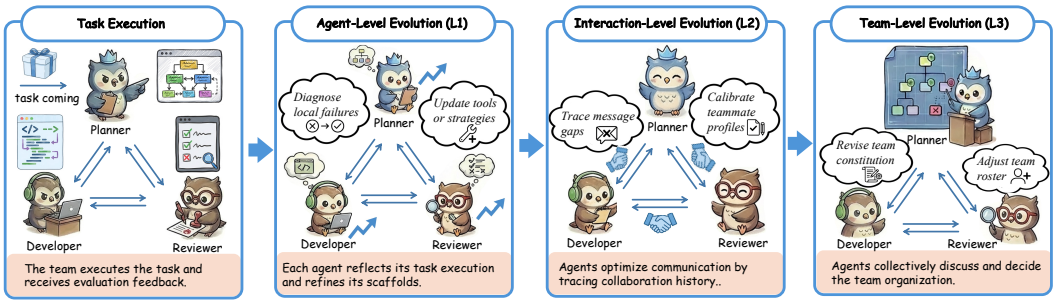


Figure 2: Overview of Meta-Team.

through collective discussion and revise its composition and organization. In this way, distributed execution experience becomes the basis for bottom-up self-evolution of the whole MAS.

3.2 META-TEAM: MULTI-SCALE COLLABORATIVE SELF-EVOLUTION

The previous subsection establishes how MAS experience should be organized for self-evolution: agent-local contexts are preserved and connected through post-task communication. Meta-Team instantiates this principle by aligning evolution with the compositional structure of MAS. After each task, agents receive the final deliverable and evaluation result, collaboratively revisit the execution process, and convert distributed evidence into updates at three complementary scopes: agent-level behavior, interaction-level collaboration, and team-level coordination. These scopes are complementary: a single failure may simultaneously reveal an agent-specific limitation, a cross-agent dependency issue, and a team-level coordination gap [55, 56]. The overall process is illustrated in Figure 2.

Agent-Level Evolution (L1). At the agent level, the evolution targets are individual agent scaffolds s_i. Although the update is applied to one agent a_i, it is derived from collaborative self-evolution rather than from that agent's local context alone. The agent reviews evidence from its own execution chain and actively solicits cross-agent evidence to verify how its assumptions, intermediate outputs, and decisions affected downstream execution. This enables agent-specific improvements with both local fidelity and cross-agent awareness, such as what the agent should observe, verify, report, or avoid in future tasks.

Interaction-Level Evolution (L2). At the interaction level, the evolution target is the collaboration mechanism $\mathcal{P}$ between agents. Through communication, agents revisit their previous collaboration to examine how information flowed and how agents relied on one another. Rather than optimizing a binary communication graph [37, 25], Meta-Team updates agents' teammate profiles, which capture how agents understand, query, and rely on one another. These profiles help agents calibrate their understanding of each teammate's strengths and limitations, enabling more effective collaboration in subsequent tasks.

Team-Level Evolution (L3). At the team level, the evolution target is the shared team scaffold $\mathcal{S}$, including the team roster, the organizational structure among agents, and the shared team constitution, i.e., the global prompt that defines common objectives, collaboration principles, and decision rules for all members. Through collective discussion and decision-making, Meta-Team examines whether the team has the appropriate composition, organization, and shared rules for the task. It then decides whether to introduce new roles, remove redundant ones, reorganize coordination, or revise the shared constitution. In this way, Team-Level Evolution improves the team's overall operating mechanism rather than only patching individual agents or pairwise interactions.

After the three-scale evolution, Meta-Team commits validated updates to the reusable team scaffold, including agent patches, teammate profiles, collaboration notes, and revisions to the shared constitution. An implementation pipeline is provided in Appendix D, including the prompts, tools, and per-level procedure of the collaboration mechanism and a worked example (Appendix D.2 and D.3).

4 EXPERIMENTS AND RESULTS

In this section, we conduct our experiments from the following perspectives. First, we compare Meta-Team against nine strong baselines on six agent benchmarks to examine its overall effectiveness (§4.2). Second, we conduct ablation studies to isolate the contribution of Meta-Team's core designs, including collaborative self-evolution and multi-scale evolution (§4.3). Third, we study scalability and generalization by evaluating whether Meta-Team remains effective across increasing context lengths and generalizes beyond the evolution distribution (§4.4). Finally, we analyze its efficiency under constrained budgets, testing whether Meta-Team achieves robust performance with limited time and cost (§4.5). Further experiments on cross-model generalization and the stability of the evolution process are reported in Appendix F.

4.1 EXPERIMENTAL SETUP

Benchmarks. Although prior MAS and MAS-evolution methods have been evaluated on various benchmarks, many of these benchmarks have become saturated under rapidly advancing LLM capabilities, detailed in Appendix E.1. Instead, we evaluate Meta-Team on six challenging benchmarks: **SWE-bench Pro** [57] and **BeyondSWE** [58] for realistic software-engineering tasks, **LOCA-Bench** [59] for long-context productivity assistants with MCP-style tools, **GAIA** [60] for multi-step open-web reasoning, **LoCoBench** [51] for long-context repository-level coding, and **ResearchRubrics** [61] for open-ended research evaluation. Appendix E.2 provides detailed benchmark descriptions, experimental settings, and initial configurations for each benchmark.

Baselines. We compare Meta-Team against three families of baselines; implementation details are in Appendix E.3. *(i) Hand-crafted agents and multi-agent systems:* **SA** (Single-Agent) [62], a ReAct executor under the same tools and budget as Meta-Team; **MAS**, our hand-designed scaffold identical to Meta-Team's initial configuration but without evolution; **AggAgent** [63], which aggregates K independent trajectories into a final answer; **OWL** [64], a fixed Planner–Coordinator–Worker hierarchy; and **AOrchestra** [65], which synthesizes sub-agents at runtime via a central orchestrator. *(ii) Performance-driven search:* **AgentSquare** [31], which searches a modular Planning/Reasoning/Tool-Use/Memory design space under a scalar reward. *(iii) Experience-based evolution* (our closest comparison points): **ReCreate** [48], which creates domain-specialized agents from past histories; **AgentNet** [37], a decentralized DAG of agents with retrieval-augmented, self-evolving edges; and **MASFly** [40], which adapts the MAS at test time using a distilled SOP and experience pool revised by a single analyzer.

Evolution and Evaluation Setup. Following [66, 48], we hold out around 20 instances per benchmark or subset as the evolution set. All evolution methods (AgentSquare, ReCreate, AgentNet, MASFly, Meta-Team) consume only this set during evolution and are evaluated on the disjoint held-out split with all learned artifacts frozen. Unless stated otherwise, all experiments use Claude Sonnet 4.6 [67] as the base LLM (`temperature` $= 0.2$, `max_tokens` $= 32768$). Full event traces (LLM calls, tool calls, inter-agent messages) are recorded for experience-driven evolution. To balance evaluation cost and variance, all results on all selected benchmarks are reported as avg@3 following [39, 40].

4.2 MAIN RESULTS

Table 1 reports the main results across six challenging agent benchmarks. Meta-Team achieves the best performance on all evaluated benchmark columns. This suggests that the benefit of Meta-Team is not tied to a single task format, but rather generalizes across heterogeneous agent settings.

We first compare Meta-Team with the single-agent SA baseline and the initial hand-designed MAS. The initial MAS uses the same starting team scaffold as Meta-Team, but does not evolve from execution experience. Notably, the initial MAS is not consistently better than the single-agent baseline: it underperforms single-agent on six out of nine benchmark columns, suggesting that a human-designed MAS may suffer from organizational or coordination issues. In contrast, after collaborative self-evolution, Meta-Team outperforms both single agent and the MAS on every evaluated column. The gains against MAS are particularly clear on long-horizon tasks, such as SWE-bench Pro Ansible with $+13.1$ points and LoCoBench Feat. with $+9.5$ points. This shows that

Table 1: Main results across diverse agent benchmarks. The best results are in **bold**.

Method	SWE-Pro		BeyondSWE		LOCA	GAIA	LoCoBench		ResRub.	Avg.
	Ansible	Qute.	DepMig.	CrossR.	Val	Val	Feat.	Refact.	All	
SA	44.7	62.7	43.5	41.7	78.3	70.0	64.5	62.0	43.9	56.8
MAS	40.8	61.0	37.6	40.4	82.1	74.7	57.6	60.9	49.5	56.1
AggAgent	49.6	57.7	42.8	40.6	80.4	70.3	60.2	54.7	47.2	55.9
OWL	39.5	60.1	36.7	36.9	72.9	64.0	53.8	56.2	47.9	52.0
AOrchestra	44.7	54.0	44.1	42.0	82.9	73.3	62.0	65.5	51.5	57.8
AgentSquare	39.0	56.3	38.4	37.4	68.3	65.0	56.2	55.9	41.9	50.9
ReCreate	42.5	59.6	41.6	39.1	74.6	69.0	58.9	62.0	45.9	54.8
AgentNet	40.8	58.7	44.3	39.6	69.6	67.3	57.0	56.9	40.3	52.7
MASFly	43.4	62.9	39.7	39.4	75.8	71.0	60.1	64.5	50.8	56.4
Meta-Team	**53.9**	**66.2**	**45.6**	**43.3**	**87.9**	**77.3**	**67.1**	**67.0**	**55.8**	**62.7**

Abbreviations. Ans. = ansible, Qute. = qutebrowser, DepMig. = DepMigrate, CrossR. = CrossRepo, Feat. = Feature Implementation, Refact. = Cross-File Refactoring, ResRub. = ResearchRubrics. **Avg.** is the unweighted arithmetic mean across the nine columns.

Table 2: Ablation on collaborative self-evolution.

Experience schemes	Ansible ↑	ResRub. ↑
No evolution (MAS)	40.8	49.5
Centralized experience	49.8	52.9
Partitioned experience	44.5	51.0
Collaborative experience	**53.9**	**55.8**

Table 3: Ablation on multi-scale evolution.

Variant	Ansible ↑	ResRub. ↑
Full Meta-Team	**53.9**	**55.8**
w/o L1 (agent)	48.5	47.9
w/o L2 (interaction)	50.7	54.4
w/o L3 (team)	51.9	52.1

Meta-Team does not merely rely on having multiple agents; rather, it turns an initially imperfect team into an adaptive system whose organization and coordination improve from execution experience.

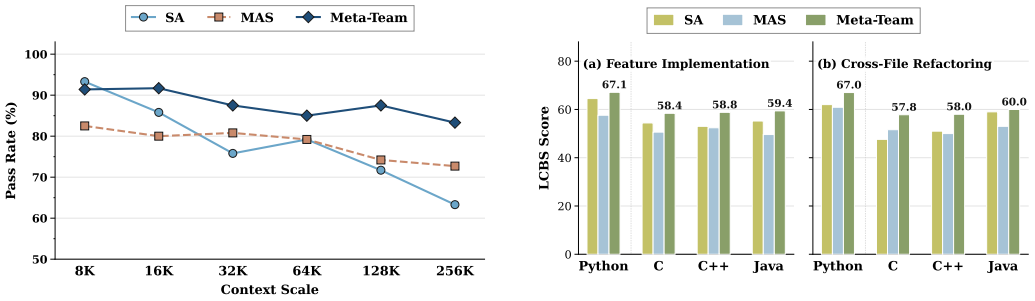
Meta-Team also outperforms prior automated and experience-driven evolution methods, which exploit task feedback or past trajectories through performance-driven search, memory retrieval, or centralized analyzer-based revision. Meta-Team obtains consistent gains over these baselines, including an average improvement of 6.3 points over MASFly, the strongest MAS evolution baseline. This comparison shows that simply using experience is not sufficient; how the experience is organized is critical. The consistent gains of Meta-Team indicate that preserving agent-local contexts and enabling post-task communication provides a more effective way to convert distributed execution experience into MAS improvements. All results are reported as avg@3; we verify the reliability of these margins with paired significance tests (Appendix F.1) and report their variance across independent evolution runs (Appendix F.5).

4.3 Ablation Study

We conduct ablations on SWE-bench Pro Ansible and ResearchRubrics to examine two core designs of Meta-Team: collaborative self-evolution and multi-scale evolution. All settings are kept identical to the main experiments, except for the experience scheme or the enabled evolution scales.

Table 2 compares different schemes of organizing execution experience for MAS evolution. Centralized experience refers to flattening the whole-team trajectory into a single LLM analyzer call for evolution. Partitioned experience refers to spawning one LLM analyzer for each agent on its local experience, without cross-agent information exchange. Collaborative experience refers to Meta-Team, which preserves the agent-wise partition of execution contexts and coordinates agents to exchange local information based on their reasoning abilities. Although centralized and partitioned experience improve over no evolution, they remain clearly below collaborative experience. This result is consistent with the attribution analysis in Figure 1: collaborative experience yields more reliable evolution targets by combining agent-local evidence with cross-agent discussion. The resulting improvement suggests that Meta-Team's gains come not merely from using past trajectories, but from organizing them in a form better suited to MAS failure attribution and self-evolution.

Table 3 evaluates the three evolution scales. Removing any scale hurts performance, indicating that the three evolution scales are complementary to the MAS design. L1 is the most influential scale: removing agent-level evolution drops performance by 5.4 points on Ansible and 7.9 points on ResearchRubrics, suggesting that improving individual agent scaffolds is the most direct lever for MAS improvement. L2 and L3 show benchmark-dependent effects: L2 contributes more on



(a) Experiments with different context scales.

(b) Experiments with various evaluation sets.

Figure 3: Experiments on scaling context and out-of-distribution evaluation sets.

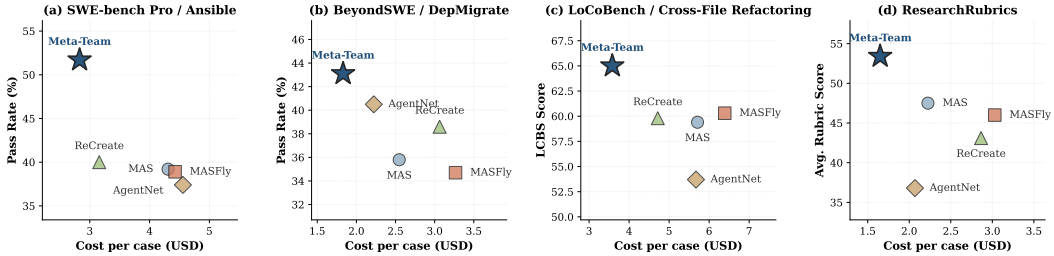


Figure 4: Experiments with constrained budget.

Ansible, where structured software tasks rely on precise coordination, while L3 contributes more on ResearchRubrics, where open-ended evaluation benefits from better team organization. Together, these ablation studies demonstrate that Meta-Team's gains come from both collaborative experience analysis and multi-scale evolution.

4.4 Scalability and Generalization of Meta-Team

We further examine whether Meta-Team remains effective beyond the distribution used for evolution. We study two axes: scalability across context length on LOCA-Bench and out-of-distribution generalization across programming languages on LoCoBench. In LOCA-Bench, Meta-Team is evolved only on 96K-token cases with held-out seeds, and is evaluated on standard seeds from 8K to 256K. In LoCoBench, Meta-Team is evolved on Python tasks and evaluated on C, C++, and Java tasks.

Figure 3a shows the scalability results on LOCA-Bench. As the context length increases, the single-agent baseline degrades substantially, while the fixed MAS is more stable but remains below Meta-Team. Meta-Team maintains the best performance across all context lengths except 8K context. This suggests that collaborative self-evolution does not merely improve performance at the evolution context length, but learns more robust coordination patterns that continue to help under extreme context growth. Interestingly, Meta-Team introduces two additional workers during evolution on 96K context, which explains why Meta-Team is especially effective in the longest-context setting (256K), where both the single-agent baseline and the fixed MAS face stronger context-management pressure.

Figure 3b reports the cross-language results on LoCoBench. Although evolution is performed on Python tasks, Meta-Team consistently outperforms both the single-agent baseline and the fixed MAS on C, C++, and Java for both Feature Implementation and Cross-File Refactoring. This indicates that the evolved improvements are not limited to language-specific shortcuts. Instead, Meta-Team learns reusable collaboration behaviors, such as task decomposition, handoff, and verification, that transfer to unseen programming languages.

4.5 Evolution under Constrained Budget

In real deployment, MAS are running with time, cost, and token limits. Therefore, we examine whether Meta-Team can evolve effective teams under a limited cost budget. For all evolution methods, we constrain the evolution budget to $1/3$ of the main setting, freeze the resulting artifacts, and evaluate

them under the same per-case deployment limits. We compare MAS, ReCreate, AgentNet, MASFly, and Meta-Team on four benchmarks. The evaluation results are shown in Figure 4, where the x-axis reports the average inference cost per evaluation case. The results show that Meta-Team achieves the best performance-cost trade-off across all four settings. It obtains the highest performance with relatively low average evaluation cost, outperforming the other MAS self-evolution methods. This advantage arises because Meta-Team can use timeout feedback on the evolution set to adjust the team system, allowing it to better adapt to the constrained-budget setting. These results suggest that collaborative self-evolution helps Meta-Team learn more cost-efficient team behaviors under constrained evolution budgets. A detailed accounting of the evolution cost, including a comparison with evolution baselines, is provided in Appendix F.2.

5 Related Work

5.1 LLM-based Multi-Agent Systems

MAS coordinate multiple language agents through role specialization, communication, and workflow-level organization. Early systems such as CAMEL [21], ChatDev [20], MetaGPT [19], and AutoGen [49] demonstrate that specified roles, protocols, and workflows can improve collaborative problem solving across tasks such as software development, reasoning, and tool use. More recent systems extend this paradigm to tool-rich and long-horizon settings [68, 5, 6]. For example, Magentic-One [69] coordinates specialized agents through a central orchestrator, while OWL [64] adopts a workforce-style organization with planning, coordination, and execution roles. Recent agent-team systems, including agent swarm [22], agent teams [23], and AOrchestra [65], further show the practical value of multi-agent coordination in real-world task automation. These works mainly study how agents should be organized during task execution, whereas our work studies how such organizations can be updated from execution experience.

5.2 Automated MAS Generation and Evolution

Although LLM-based MAS have shown promise, recent studies show that they remain brittle in realistic settings, with failures often stemming from poor specification, system-design flaws, inter-agent misalignment, and weak verification or termination mechanisms [70, 41]. Other studies further show that small errors at inter-agent handoffs can propagate into system-level failures [55], and that self-organizing agent teams may fail to effectively leverage their strongest members [71]. These challenges motivate automated methods for designing, adapting, and evolving MAS. Recent work has explored automated generation and optimization of agent systems. One line of work formulates MAS construction as performance-driven search over prompts, modules, workflows, communication graphs, or routing policies. Representative methods include ADAS [30], AgentSquare [31], GPTSwarm [32], AFlow [33], MASS [34]. Subsequent work trains workflow optimizers [34, 72, 73, 74, 75, 76, 77, 78]. These methods optimize agent-system configurations using task-level feedback and reducing human design effort. Inspired by experience learning in LLM and single-agent [50, 79, 54, 48], another line of work uses execution experience to support MAS self-evolution. Cross-task experiential learning methods store MAS trajectories or interaction fragments for later retrieval [35], while AgentNet [37] evolves decentralized agent connectivity and expertise through accumulated experience. ASpec [38] cultivates persistent specialist teams for adaptive agent systems, and MASFly [40] adapts MAS at test time by leveraging experience-guided revision. Meta-Team follows this experience-driven direction, but organizes evolution according to the distributed structure of MAS itself, enabling agents to preserve local execution contexts while collaboratively evolving the system.

6 Conclusion

In this paper, we study how LLM-based multi-agent systems can improve from their own execution experience. We advocate a simple principle for MAS evolution: a multi-agent system should not only execute as a team, but also evolve as a team. Building on this principle, we propose Meta-Team, which preserves agent-local contexts, enables post-task communication, and transforms distributed execution experience into targeted improvements for the team. We believe Meta-Team paves the way toward self-evolving agent teams that continuously improve their behaviors, communication, and coordination from their own distributed experience.

AI USE STATEMENT

We used generative AI tools only for language polishing. All such use took place after the research had been completed and the manuscript had been written by the authors, and was limited to improving the grammar, wording and readability of text that the authors had already drafted. No part of this paper was generated from scratch by an AI tool.

We did not use generative AI tools to formulate the research idea, to design Meta-Team or any of its components, to write or debug the code, or to run, analyse and interpret the experiments. Every number reported in this paper comes from our own implementation and our own evaluation pipeline.

The authors reviewed all AI-assisted edits. Suggested revisions were accepted or rejected sentence by sentence, and every accepted revision was checked to confirm that it left the technical content, the claims and the scope of those claims unchanged; edits that altered, weakened or overstated a claim were discarded. The final text was then re-read by the authors for factual accuracy and internal consistency. We take full responsibility for the entire content of this paper, including every sentence that was edited with the aid of generative AI.

REFERENCES

[1] Open AI. Introducing gpt-5.6. https://openai.com/index/gpt-5-6/, 2026.

[2] Google. A new era of intelligence with gemini 3. https://blog.google/products/gemini/gemini-3/, 2025.

[3] Anthropic. Introducing claude opus 5. https://www.anthropic.com/news/claude-opus-5, 2026.

[4] Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

[5] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[6] Xinbin Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, Sirui Hong, Sheng Fan, Xiao Tang, Bang Liu, Yuyu Luo, and Chenglin Wu. Openmanus: An open-source framework for building general ai agents, 2025. URL https://doi.org/10.5281/zenodo.15186407.

[7] Anthropic. Claude Code Overview. https://code.claude.com/docs/en/overview, 2026.

[8] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[9] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

[10] Deyu Zou, Yongqiang Chen, Jianxiang Wang, Garry YANG, Mufei Li, Qing Da, James Cheng, Pan Li, and Yu Gong. Reducing belief deviation in reinforcement learning for active reasoning of llm agents. In *ICLR*, 2026.

[11] HaoYang Shang, Xuan Liu, Zi Liang, Jie Zhang, Haibo Hu, and Song Guo. United minds or isolated agents? exploring coordination of llms under cognitive load theory. *arXiv preprint arXiv:2506.06843*, 2025.

[12] Chenxin An, Jun Zhang, Ming Zhong, Lei Li, Shansan Gong, Yao Luo, Jingjing Xu, and Lingpeng Kong. Why does the effective context length of llms fall short? In *ICLR*, 2025.

[13] T Guo, X Chen, Y Wang, R Chang, S Pei, NV Chawla, O Wiest, and X Zhang. Large language model based multi-agents: A survey of progress and challenges. In *33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*. IJCAI; Cornell arxiv, 2024.

[14] Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024.

[15] Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

[16] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arık. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

[17] Jun Zhao, Can Zu, Xu Hao, Yi Lu, Wei He, Yiwen Ding, Tao Gui, Qi Zhang, and Xuan-Jing Huang. Longagent: achieving question answering for 128k-token-long documents through multi-agent collaboration. In *EMNLP*, pages 16310–16324, 2024.

[18] Zhen Xu, Shang Zhu, Jue Wang, Junlin Wang, Ben Athiwaratkun, Chi Wang, James Zou, and Ce Zhang. When does divide and conquer work for long context llm? a noise decomposition framework. In *ICLR*, 2026.

[19] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *ICLR*, 2024.

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

[21] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

[22] Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

[23] Anthropic. Orchestrate teams of Claude Code sessions. `https://code.claude.com/docs/en/agent-teams`, 2026.

[24] Anthropic. How we built our multi-agent research system, 2025. `https://www.anthropic.com/engineering/built-multi-agent-research-system`.

[25] Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Ö Arık. Multi-agent design: Optimizing agents with better prompts and topologies. *arXiv preprint arXiv:2502.02533*, 2025.

[26] Cara Phillips. Multi-agent coordination patterns: Five approaches and when to use them. `https://claude.com/blog/multi-agent-coordination-patterns`, April 2026. Anthropic Blog, Accessed: 2026-05-05.

[27] Dat Tran and Douwe Kiela. Single-agent llms outperform multi-agent systems on multi-hop reasoning under equal thinking token budgets. *arXiv preprint arXiv:2604.02460*, 2026.

[28] Walden Yan. Don't build multi-agents. `https://cognition.ai/blog/dont-build-multi-agents`, June 2025. Cognition AI Blog, Accessed: 2026-05-05.

[29] Vassilis Papadopoulos, McNair Shah, Sam Zimmerman, and Jack Lindsey. Mind viruses: Self-propagating ideas in multi-agent llm systems. *arXiv preprint arXiv:2608.10218*, 2026.

[30] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *ICLR*, 2025.

[31] Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *ICLR*, 2025.

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

[33] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *ICLR*, 2025.

[34] Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *International Conference on Machine Learning*, pages 75834–75852. PMLR, 2025.

[35] Yilong Li, Chen Qian, Yu Xia, Ruijie Shi, Yufan Dang, Zihao Xie, Ziming You, Weize Chen, Cheng Yang, Weichuan Liu, et al. Cross-task experiential learning on llm-based multi-agent collaboration. *arXiv preprint arXiv:2505.23187*, 2025.

[36] Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, et al. Se-agent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. *arXiv preprint arXiv:2508.02085*, 2025.

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

[38] Myan Vu, Harrish Ayyanar, PANG JIANG, Anwiketh Reddy, and Mayank Goel. Automated stateful specialization for adaptive agent systems. In *ICLR*, 2026.

[39] Yuntong Hu, Matthew Trager, Yuting Zhang, Yi Zhang, Shuo Yang, Wei Xia, and Stefano Soatto. Evolutionary generation of multi-agent systems. *arXiv preprint arXiv:2602.06511*, 2026.

[40] Guangyi Liu, Haojun Lin, Huan Zeng, Heng Wang, and Quanming Yao. Mas-on-the-fly: Dynamic adaptation of llm-based multi-agent systems at test time. *arXiv preprint arXiv:2602.13671*, 2026.

[41] Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *Proceedings of Machine Learning Research*, 267:76583–76599, 2025.

[42] Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems? *arXiv preprint arXiv:2509.03312*, 2025.

[43] Adi Banerjee, Anirudh Nair, and Tarik Borogovac. Where did it all go wrong? a hierarchical look into multi-agent error attribution. In *NeurIPS Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*, 2025.

[44] Jiazheng Li, Emine Yilmaz, Bei Chen, and Dieu-Thu Le. Towards self-improving error diagnosis in multi-agent systems. *arXiv preprint arXiv:2604.17658*, 2026.

[45] Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems. *arXiv preprint arXiv:2604.22708*, 2026.

[46] Michael West. Reflexivity and work group effectiveness: A conceptual integration. In *The handbook of work group psychology*, pages 555–579. John Wiley & Sons, Ltd, 1996.

[47] Michaéla C Schippers, Michael A West, and Jeremy F Dawson. Team reflexivity and innovation: The moderating role of team context. *Journal of Management*, 41(3):769–788, 2015.

[48] Zhezheng Hao, Hong Wang, Jian Luo, Jianqing Zhang, Yuyan Zhou, Qiang Lin, Can Wang, Hande Dong, and Jiawei Chen. Recreate: Reasoning and creating domain agents driven by experience. *arXiv preprint arXiv:2601.11100*, 2026.

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

[50] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[51] Jielin Qiu, Zuxin Liu, Zhiwei Liu, Rithesh Murthy, Jianguo Zhang, Haolin Chen, Shiyu Wang, Ming Zhu, Liangwei Yang, Juntao Tan, et al. Locobench: A benchmark for long-context large language models in complex software engineering. *arXiv preprint arXiv:2509.09614*, 2025.

[52] Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Dayuan Fu, Shijie Xia, Xiaojie Cai, Tianze Xu, et al. Agencybench: Benchmarking the frontiers of autonomous agents in 1m-token real-world contexts. *arXiv preprint arXiv:2601.11044*, 2026.

[53] Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. *arXiv preprint arXiv:2509.14295*, 2025.

[54] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

[55] Bohan Lin, Kuo Yang, Zelin Tan, Yingchuan Lai, Chen Zhang, Guibin Zhang, Xinlei Yu, Miao Yu, Xu Wang, Yudong Zhang, et al. Agentask: Multi-agent systems need to ask. *arXiv preprint arXiv:2510.07593*, 2025.

[56] Yeonjun In, Mehrab Tanjim, Jayakumar Subramanian, Sungchul Kim, Uttaran Bhattacharya, Wonjoong Kim, Sangwu Park, Somdeb Sarkhel, and Chanyoung Park. Rethinking failure attribution in multi-agent systems: A multi-perspective benchmark and evaluation. *arXiv preprint arXiv:2603.25001*, 2026.

[57] Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

[58] Guoxin Chen, Fanzhe Meng, Jiale Zhao, Minghao Li, Daixuan Cheng, Huatong Song, Jie Chen, Yuzhi Lin, Hui Chen, Xin Zhao, et al. Beyondswe: Can current code agent survive beyond single-repo bug fixing? *arXiv preprint arXiv:2603.03194*, 2026.

[59] Weihao Zeng, Yuzhen Huang, and Junxian He. Loca-bench: Benchmarking language agents under controllable and extreme context growth. *arXiv preprint arXiv:2602.07962*, 2026.

[60] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[61] Manasi Sharma, Chen Bo Calvin Zhang, Chaithanya Bandi, Clinton Wang, Ankit Aich, Huy Nghiem, Tahseen Rabbani, Ye Htet, Brian Jang, Sumana Basu, et al. Researchrubrics: A benchmark of prompts and rubrics for evaluating deep research agents. *arXiv preprint arXiv:2511.07685*, 2025.

[62] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[63] Yoonsang Lee, Howard Yen, Xi Ye, and Danqi Chen. Agentic aggregation for parallel scaling of long-horizon agentic tasks. *arXiv preprint arXiv:2604.11753*, 2026.

[64] Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Ziyu Ye, Bowei Xia, Tao Sun, Zhaoxuan Jin, Yingru Li, Zeyu Zhang, et al. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[65] Jianhao Ruan, Zhihao Xu, Yiran Peng, Fashen Ren, Zhaoyang Yu, Xinbing Liang, Jinyu Xiang, Yongru Chen, Bang Liu, Chenglin Wu, et al. Aorchestra: Automating sub-agent creation for agentic orchestration. *arXiv preprint arXiv:2602.03786*, 2026.

[66] Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. Prompt optimization with ease? efficient ordering-aware automated selection of exemplars. *Advances in Neural Information Processing Systems*, 37: 122706–122740, 2024.

[67] Anthropic. Introducing claude sonnet 4.6. `https://www.anthropic.com/news/claude-sonnet-4-6`, 2026.

[68] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[69] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

[70] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? In *NeurIPS Datasets and Benchmarks Track*, 2025.

[71] Aneesh Pappu, Batu El, Hancheng Cao, Carmelo di Nolfo, Yanchao Sun, Meng Cao, and James Zou. Multi-agent teams hold experts back. *arXiv preprint arXiv:2602.01011*, 2026.

[72] Yu Li, Lehui Li, Zhihao Wu, Qingmin Liao, Jianye Hao, Kun Shao, Fengli Xu, and Yong Li. Agentswift: Efficient llm agent design via value-guided hierarchical search. *arXiv preprint arXiv:2506.06017*, 2025.

[73] Hongcheng Gao, Yue Liu, Yufei He, Longxu Dou, Chao Du, Zhijie Deng, Bryan Hooi, Min Lin, and Tianyu Pang. Flowreasoner: Reinforcing query-level meta-agents. *arXiv preprint arXiv:2504.15257*, 2025.

[74] Yinjie Wang, Ling Yang, Guohao Li, Mengdi Wang, and Bryon Aragam. Scoreflow: Mastering llm agent workflows via score-based preference optimization. *arXiv preprint arXiv:2502.04306*, 2025.

[75] Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572, 2025.

[76] Zhijun Chen, Zeyu Ji, Qianren Mao, Hao Wu, Junhang Cheng, Bangjie Qin, Zhuoran Li, Jingzheng Li, Kai Sun, Zizhe Wang, et al. Scoring, reasoning, and selecting the best! ensembling large language models via a peer-review process. *arXiv preprint arXiv:2512.23213*, 2025.

[77] Mingda Zhang, Haoran Luo, Tiesunlong Shen, Qika Lin, Xiaoying Tang, Rui Mao, and Erik Cambria. Flowsteer: Interactive agentic workflow orchestration via end-to-end reinforcement learning. *arXiv preprint arXiv:2602.01664*, 2026.

[78] Mingze Kong, Zikun Qu, Zhongquan Zhou, Pengyu Liang, Xiang Li, Zhiwei Shang, Zhi Hong, Kaiyu Huang, Zhiyong Wang, and Zhongxiang Dai. Workflow-r1: Group sub-sequence policy optimization for multi-turn workflow construction. *arXiv preprint arXiv:2602.01202*, 2026.

[79] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[80] Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. Adaptive in-conversation team building for language model agents. *arXiv preprint arXiv:2405.19425*, 2024.

[81] Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 23142–23150, 2026.

[82] Hongjiang Chen, Xin Zheng, Yixin Liu, Pengfei Jiao, Shiyuan Li, Huan Liu, Zhidong Zhao, Ziqi Xu, Ibrahim Khalil, and Shirui Pan. Goagent: Group-of-agents communication topology generation for llm-based multi-agent systems. *arXiv preprint arXiv:2603.19677*, 2026.

[83] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[84] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[85] Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1743–1752, 2015.

[86] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[87] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[88] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

[89] Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, et al. Coact-1: Computer-using agents with coding as actions. In *ICLR*, 2026.

[90] Zihan Zhou, Chong Li, Xinyi Chen, Shuo Wang, Yu Chao, Zhili Li, Haoyu Wang, Qi Shi, Zhixing Tan, Xu Han, et al. Llm$\times$ mapreduce: Simplified long-sequence processing using large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27664–27678, 2025.

[91] Romain Froger et al. GAIA2: Benchmarking LLM agents on dynamic and asynchronous environments. In *International Conference on Learning Representations (ICLR)*, 2026.

CONTENTS

A Limitations **16**

B Broader Impact **16**

C Analysis of the Global-Local-Collaborative Schemes **17**

C.1 Experimental Comparison 17

C.2 Theoretical Analysis 17

D Pipeline and Implementation of Meta-Team **19**

D.1 System Overview and Execution 19

D.2 Details of the Collaboration Mechanism 20

D.3 A Worked Example 21

E Experiment Settings **22**

E.1 Saturation of Commonly Used Benchmarks 22

E.2 Experiment Settings on Different Benchmarks 23

E.3 Experiment Settings of Baseline Reproductions 27

F Further Experiment Results **28**

F.1 Statistical Reliability of Main Results 28

F.2 Cost of the Evolution Process 29

F.3 Generalization Across Base Models 29

F.4 Effect of Team Size 30

F.5 Stability Across Independent Evolution Runs 31

A LIMITATIONS

This work has two main limitations.

First, Meta-Team focuses on scaffold- and organization-level evolution, including agent behaviors, collaboration patterns, and team-level coordination rules. It does not adapt the underlying execution infrastructure, such as tool APIs, environment harnesses, verifiers, or memory backends, which typically require task-specific engineering.

Second, Meta-Team does not update the parameters of the underlying LLMs. Its evolution relies on post-task reflection and explicit scaffold updates, making it interpretable and model-agnostic, but limiting the extent to which recurring team behaviors can be internalized into the base models. Combining collaborative self-evolution with fine-tuning or reinforcement learning is a promising future direction.

B BROADER IMPACT

Meta-Team introduces a collaborative self-evolution paradigm for LLM-based multi-agent systems by enabling agent teams to improve from their own distributed execution experience. Instead of relying on static, manually designed architectures or centralized post-hoc analyzers, Meta-Team allows

agents to refine their roles, collaboration patterns, and team-level coordination through structured reflection. This approach can reduce the human effort required to design and maintain effective MAS, while improving their adaptability on long-horizon and tool-rich tasks. By making multi-agent systems more self-improving, inspectable, and reusable, Meta-Team has the potential to broaden access to intelligent automation across applications such as software engineering, research assistance, education, and enterprise workflows.

C Analysis of the Global-Local-Collaborative Schemes

C.1 Experimental Comparison

To examine this design choice, we compare three attribution paradigms on TraceElephant, a benchmark of 220 real MAS failure traces from captain-agent [80], magentic-one [69], and swe-agent [68].

Dataset. TraceElephant [45] provides 220 failed multi-agent traces collected from three open-source MAS frameworks: Captain-Agent [80] (n=85, short analytical tasks), Magentic-One [69] (n=91, Magentic-One-style orchestrated tasks) and SWE-Agent [68] (n=44, SWE-Agent traces on SWE-bench Verified). Each trace is a time-ordered list of step records (agent name, full input, full output), ranges from 2 to 94 steps (median 21) involving 2–4 agents, and is annotated with a ground-truth $(\texttt{mistake_agent}, \texttt{mistake_step})$ tuple. Our split yields 106 traces in the $\leq$128K split and 114 in the $>$128K split. Agent-Accuracy and Step-Accuracy are exact-match rates against the annotated tuple, averaged per split by simple arithmetic mean.

Settings. The three schemes are implemented as follows.

The global scheme issues a single LLM call on the flattened trace and reads the $(\texttt{agent}, \texttt{step})$ pair directly from its JSON response.

The local scheme spawns one analyzer per agent, each seeing only its own sub-trace and submitting $(\texttt{i_erred} \in \{0,1\}, \texttt{my_step}, c)$, where $c \in [0,1]$. The final $(\texttt{agent}, \texttt{step})$ is the submission of ($\texttt{i_erred}$=1) with the largest confidence c; if nobody self-accuses, we charge the first step of the least-confident denier, exposing the "no coordination" failure mode.

The collaborative scheme adds multiple rounds of communication: each analyzer first posts a short summary of its own findings, reads the summaries posted by the other analyzers, and then re-audits its sub-trace, submitting a tuple

$$(\texttt{i_erred}, \texttt{my_step}, c, r)$$

,where $c \in [0,1]$ is its self-confidence and $r=1$ if it disagrees with the global verdict and backs the disagreement with explicit counter-evidence from its own sub-trace. Each analyzer then votes for the pair it accuses with weight $w = c\,(1+\alpha r)$, where α=1 and the highest-scoring pair wins.

C.2 Theoretical Analysis

We now analyze the global, local, and collaborative schemes formally. Using a simple model of how an analyzer recovers evidence from a trajectory, we show why the collaborative scheme outperforms the global and local schemes across trajectory lengths, and why the ranking between the latter two flips as the trajectory grows.

Setup. Consider a MAS with N agents. Their steps are interleaved into one trajectory of total length L, and agent i's own part has length ℓ_i, so $L = \sum_{i=1}^{N} \ell_i$. To find the decisive cause of a failure, an analyzer reads a trajectory (or part of it) and looks for the relevant clues. We describe this with three quantities:

- *Local evidence* $S = \sum_{i=1}^{N} s_i$ (with $s_i \geq 0$): clues that sit inside a single agent's own part of the trajectory.
- *Interaction evidence* $B \geq 0$: clues that only appear when one agent hands off to another, i.e. they live between agents rather than inside any one of them.

- *Recovery fraction* $\rho(\ell) \in (0, 1]$: the share of the clues an analyzer actually picks up when it reads a context of length ℓ.

Getting the attribution right means recovering as many of the $S + B$ clues as possible. We write V for the *effective evidence* a scheme recovers — the amount of $S + B$ its analyzers actually pick up, after dilution — which is the quantity the schemes will differ in. The analysis rests on the following two assumptions.

Assumption 1 (distributed evidence)**.** Each agent's own part is much shorter than the whole trajectory, $\ell_i \ll L$, and the interaction evidence B comes from how different agents' parts fit together, so it cannot be found by looking at any single agent's part alone.

The short-part condition is just what dividing the work among agents gives us. The second half says B really is about the links between agents, which is why a scheme that only looks inside each agent's own part can never see it.

Assumption 2 (context dilution)**.** The recovery fraction ρ does not increase as the context gets longer: the more an analyzer has to read, the smaller the share of clues it actually picks up.

This is the well-documented effect that models use long contexts less effectively. It does not need any cutting-off of the input: even if the whole trajectory fits in the context window, a longer ℓ still lowers $\rho(\ell)$.

Evidence recovered by each scheme. We compare the schemes by their effective evidence V: since accuracy is non-decreasing in V ($\mathrm{Acc} = g(V)$), the scheme that recovers more evidence is at least as accurate. In the collaborative scheme an analyzer additionally reads a teammate's brief summary of length δ, so its context $\ell^\star = \max_i \ell_i + \delta$ is still short ($\ell^\star \ll L$). The schemes then obtain

$$V_{\mathrm{global}} = \rho(L)\,(S + B), \tag{1}$$

$$V_{\mathrm{local}} = \sum_{i=1}^{N} \rho(\ell_i)\, s_i, \tag{2}$$

$$V_{\mathrm{collab}} = \sum_{i=1}^{N} \rho(\ell_i)\, s_i + \rho(\ell^\star)\, B. \tag{3}$$

Intuitively, the global scheme sees B but dilutes everything by $\rho(L)$; the local scheme keeps its short-context fidelity but misses B entirely; the collaborative scheme keeps the local terms and adds B back at only the mild dilution $\rho(\ell^\star)$.

Proposition 1. *There is a critical length L_c such that $V_{\mathrm{global}} \geq V_{\mathrm{local}}$ for short trajectories ($L \leq L_c$) and $V_{\mathrm{global}} < V_{\mathrm{local}}$ for long ones ($L > L_c$).*

Proof. Subtracting (2) from (1),

$$f(L) \;:=\; V_{\mathrm{global}} - V_{\mathrm{local}} = \rho(L)\, B - \sum_{i=1}^{N} \big[\rho(\ell_i) - \rho(L)\big]\, s_i. \tag{4}$$

When the trajectory is short, every ℓ_i is short too, so each bracket $\rho(\ell_i) - \rho(L)$ is small and the subtracted sum is close to zero; the right-hand side is then dominated by $\rho(L)\, B \geq 0$, so $f(L) \geq 0$. As $L \to \infty$, $\rho(L) \to 0$: the first term vanishes while the subtracted sum stays bounded away from zero (each $\ell_i \ll L$ keeps $\rho(\ell_i)$ well above $\rho(L)$), so $f(L) < 0$. Since f is continuous in L, it changes sign at some critical length L_c, which gives the stated ordering. □

This exposes the limitation of the two single-analyzer schemes: neither is uniformly reliable, and which one is preferable depends on the trajectory length. The next two propositions show that the collaborative scheme is at least as good as *both* of them, regardless of length.

Proposition 2. *For any trajectory length, $V_{\mathrm{collab}} \geq V_{\mathrm{local}}$, with strict inequality whenever $B > 0$.*

Proof. Subtracting (2) from (3), the local terms cancel and

$$V_{\mathrm{collab}} - V_{\mathrm{local}} = \rho(\ell^\star)\, B \;\geq\; 0, \tag{5}$$

which is strict when $B > 0$ since $\rho(\ell^\star) > 0$. □

The gain is exactly the recovered interaction evidence and is independent of the total length L. Because the local and collaborative schemes use identical per-agent analyzers and differ only in whether cross-agent evidence is exchanged, the improvement is attributable to coordination structure rather than to additional computation.

Proposition 3. $V_{\text{collab}} \geq V_{\text{global}}$, *and the gap is non-decreasing in the total trajectory length* L.

Proof. Subtracting (1) from (3),

$$V_{\text{collab}} - V_{\text{global}} = \underbrace{\sum_{i=1}^{N} \big[\rho(\ell_i) - \rho(L)\big] s_i}_{\geq 0} + \underbrace{\big[\rho(\ell^\star) - \rho(L)\big] B}_{\geq 0}. \tag{6}$$

Since ρ is non-increasing (Assumption 2) and $\ell_i, \ell^\star \ll L$, each bracketed term is non-negative, so the difference is non-negative. As L grows, $\rho(L)$ does not increase while $\rho(\ell_i)$ and $\rho(\ell^\star)$ are unaffected, so neither term decreases and the gap does not shrink. □

Summary. The analysis identifies a length-dependent trade-off between the two single-analyzer schemes—the global scheme is preferable on short trajectories and the local scheme on long ones (Proposition 1)—and shows that the collaborative scheme is at least as good as both regardless of length (Propositions 2 and 3). This matches the experimental comparison in Appendix C.1, where the global–local crossover appears near 128K and the collaborative scheme is best in both length regimes.

D Pipeline and Implementation of Meta-Team

D.1 System Overview and Execution

Meta-Team uses an *open-roster* orchestration mechanism. It maintains a candidate pool of agents and recruits an active subset for each task. The shared team scaffold contains the candidate roster and a team constitution, which specifies common objectives, collaboration principles, and decision rules. Each agent is implemented as an editable directory, with `prompt.md` storing its role prompt and `config.yaml` specifying the backbone model and allowed tools. The evolving components are stored as plain-text files under `evolution/`, including behavioral patches, teammate profiles, and pairwise collaboration notes. Agent skills are kept under `skills/<name>/SKILL.md` and loaded by progressive disclosure: only skill names and descriptions are shown by default, while the full skill body is retrieved when needed.

During execution, the active roster is assembled dynamically. The orchestration protocol provides a small set of primitives, including ListPool, StartAgent, StopAgent, Finalize, and Terminate. Each episode starts with pool-level visibility through ListPool; agents are recruited through StartAgent when additional expertise is needed; and the episode ends when an agent calls Finalize to submit the final deliverable. Communication is handled by an append-only message bus. When an agent sends a message, the event is appended to the bus and delivered to the receiver's mailbox. Each agent runs its own loop: it either works by calling the backbone model and tools, or waits for new messages when idle. The recorded experience therefore contains each agent's local trace, together with cross-agent events such as messages, handoffs, and shared artifacts.

After each task, Meta-Team updates the system at three levels. Agent-level evolution revises an agent using its local trace and the cross-agent evidence directly related to it. Interaction-level evolution updates teammate profiles and collaboration notes for agents that actually interacted. Team-level evolution aggregates the short summaries and selected evidence from the previous two levels to revise the constitution, coordination rules, or candidate roster. This avoids sending the full flattened trajectory to a single analyzer: local traces are preserved, relevant evidence is exchanged, and only concise summaries are used for team-level revision. Before updates are committed, Meta-Team checks role consistency, tool availability, formatting validity, and budget constraints. If a retry is requested during evolution, it is allowed only when budget remains, and the retry cost is counted toward the evolution budget. Algorithm 1 summarizes the procedure.

Algorithm 1 Meta-Team: Collaborative Self-Evolution of an LLM-based MAS.

Require: Initial MAS $\mathcal{M}_0$ with pool $\mathcal{A}^\star$; task stream $\{x_k\}_{k=1}^{K}$; evaluator $\mathcal{V}$.
Ensure: Evolved MAS $\mathcal{M}_K$.

for $k = 1$ to K **do**
  $\mathcal{M} \leftarrow \mathcal{M}_{k-1}$
  $(\mathcal{A}, \tau_k, \hat{r}) \leftarrow \textsc{Run}(\mathcal{M}; x_k)$ ▷ recruit $\mathcal{A} \subseteq \mathcal{A}^\star$ and execute
  $r_k \leftarrow \mathcal{V}(\hat{r})$; $e_k \leftarrow (x_k, \tau_k, r_k)$ ▷ freeze the task experience
  for all $a_i \in \mathcal{A}$ **in parallel do**
    $(\Delta_i, \mathcal{K}_i, u_i) \leftarrow \Omega_{\mathrm{L1}}(s_i, \tau_i, r_k)$ ▷ agent-level reflection
  end for
  for all interacting pairs (a_i, a_j) in τ_k **do**
    $(\Phi_i, \mathrm{R}_i, \Phi_j, \mathrm{R}_j) \leftarrow \Omega_{\mathrm{L2}}(s_i, s_j, \tau_k, r_k)$ ▷ interaction-level reflection
  end for
  $\mathcal{S} \leftarrow \Omega_{\mathrm{L3}}\big(\mathcal{S}, \{u_i\}_{a_i \in \mathcal{A}}, \tau_k, r_k\big)$ ▷ team-level revision
  if the team requests retry **then**
    re-execute x_k with the updated $\mathcal{M}$
  end if
  $\mathcal{M}_k \leftarrow \mathcal{M}$ ▷ commit the updated MAS
end for
return $\mathcal{M}_K$

Meta-Team instantiates the MAS $\mathcal{M} = (\mathcal{A}, \mathcal{O})$ with an *open-roster* orchestration mechanism. The active agent set $\mathcal{A}$ is selected at run time from a fixed candidate pool $\mathcal{A}^\star$, and the shared scaffold is written as $\mathcal{S} = (\mathcal{A}^\star, \mathcal{C})$, where $\mathcal{C}$ denotes the team constitution. Each agent scaffold is decomposed as

$$s_i = (\pi_i,\ \Delta_i,\ \mathcal{K}_i,\ \Phi_i,\ \mathrm{R}_i),$$

where π_i is the base role prompt, Δ_i stores behavioral patches, $\mathcal{K}_i$ is the skill library, Φ_i contains teammate profiles, and R_i records pairwise correlation notes.

Given a task x_k, Meta-Team executes the current MAS to collect experience $e_k = (x_k, \tau_k, r_k)$. It then applies the update operator Ω at three scales: agent-level updates, interaction-level updates, and team-level updates. The resulting MAS is committed as $\mathcal{M}_{k+1}$. Algorithm 1 summarizes the procedure.

D.2 Details of the Collaboration Mechanism

The description above and Section 3.2 specify *what* each level evolves. Here we detail *how* the collaboration is realized during evolution, covering the prompts, the tools, and the per-level procedure, including what information is exchanged at each level and how the exchanged evidence is aggregated into committed updates. Evolution in Meta-Team is thus not parallel local reflection, but a sequence of three phases in which agents actively exchange evidence, corresponding to the operators Ω_{L1}, Ω_{L2}, and Ω_{L3} in Algorithm 1.

Prompts for each stage. After a task finishes, the shared outcome feedback (the final deliverable, the evaluation verdict, an error diagnosis, and the preserved local traces and pairwise communication summaries) is broadcast to the active agents, and the following reflection prompts operationalize L1–L3.

- **L1 (Agent-Level).** Each agent is asked four questions about its own trace — what it did well, where it went wrong, what was inefficient, and what it would do differently — and is instructed to distill the answers into reusable behavioral lessons rather than task-specific fixes.
- **L2 (Interaction-Level).** For each pair of agents that actually interacted, the prompt asks the two agents to revisit only the messages they exchanged, record critical collaboration patterns, and produce a structured teammate profile (reliability, strengths, weaknesses, style); pairs that did not interact are skipped.
- **L3 (Team-Level).** Each agent is first prompted to propose team-level improvements. The chairman then combines these proposals with its own assessment of the team — whether the team

Table 4: Tools available during execution and at each evolution level. Benchmark-specific MCP services are additionally exposed during execution depending on the task environment.

Phase	Category	Tools
Execution	Environment	`bash`, `read_file`, `write_file`, `str_replace_editor`
	Communication	`send_message`, `read_messages`, `wait_for_replies`
L1	Agent-level	`update_prompt_patch`, `update_skill`
L2	Interaction-level	`update_teammate_profile`, `update_correlation`
L3	Team-level	`suggest_team_improvement`, `apply_reflection`

is the right size, whether some work lacked verification, whether any agent was underused, and where the workflow stalled — and makes concrete changes to the team composition (adding or removing a role) and the shared constitution.

Tools for each stage. Beyond prompting and self-evaluation, agents use dedicated tools to carry out evolution. During execution, agents act on the environment and coordinate over the message bus; during evolution, each level exposes tools that turn reflection into concrete, checkable edits to the scaffold. Table 4 summarizes the tools available in each phase.

Per-level evolution procedure. We describe what information each level uses, how agents exchange evidence across their local contexts, and how that evidence is aggregated into committed updates.

- **L1 — assessing effects on downstream execution.** Each agent first reflects on its own execution trace and the final feedback. From this reflection it identifies where its own outputs or actions may have caused problems for downstream teammates. If needed, it then contacts those related teammates on demand to request the corresponding evidence, and finally turns the confirmed lessons into behavioral patches (Δ_i) or skills ($\mathcal{K}_i$) via `update_prompt_patch` and `update_skill`.
- **L2 — revisiting collaboration through specific messages.** Meta-Team uses the recorded message history to identify which pairs of agents actually communicated, and pairs them up for reflection. Each pair re-reads only the messages they exchanged and, if a real problem occurred, opens a short discussion. The exchange is aggregated into an updated teammate profile (Φ_i) and a pairwise collaboration note (R_i) through `update_teammate_profile` and `update_correlation`; smooth pairs record only brief observations.
- **L3 — examining composition, organization, and rules.** Rather than re-reading the entire trajectory, the chairman gathers the concise L1/L2 summaries and the improvement suggestions raised by teammates (via `suggest_team_improvement`), reviews the current team setup, and decides whether the roster or the shared rules should change. Validated decisions are committed to the constitution or roster through `apply_reflection`.

Across the three levels, each analyzer stays within its own local context and recovers cross-agent awareness through targeted exchange: local traces are preserved, only the relevant evidence is exchanged on demand, and only concise summaries are passed up for team-level revision, instead of flattening the whole trajectory into a single analyzer.

D.3 A Worked Example

We illustrate the three phases with a real GAIA sample (`da52d699`, difficulty Level 3) to show what cross-agent exchange recovers but independent per-agent reflection cannot.

Scenario. The question asks,"The attached spreadsheet lists the books I read in 2022; by words read per day, which one did I read the slowest?" (gold answer: *Out of the Silent Planet*). Solving it requires two halves of evidence: each book's reading days can be computed from the start/end dates in the attachment, but the word counts are not in the file and must be looked up online per book; the answer is the book with the smallest words-per-day.

Execution. The Planner assigns the FileAnalyzer to compute reading days, the WebSearcher to look up word counts, and the Summarizer to aggregate and format. For lesser-known books the WebSearcher only finds *estimated* word counts and returns them without flagging which are exact and which are estimates; the Planner treats them all as exact when ranking. The word count of the true answer happens to be over-estimated, so the team picks the wrong book.

Outcome feedback. After the task, the system injects a shared feedback to all members: the submitted answer, the evaluation verdict and the gold answer (here judged incorrect), an error diagnosis, plus the preserved execution trace (including the per-book word counts the WebSearcher reported) and a pairwise communication summary. This lets the team reason backward toward the source of the failure.

Evolution. Conditioned on this shared feedback, the three levels run in sequence.

- **L1 (agent level).** Each agent reflects on its own behavior and updates its own scaffold from its own trajectory. Since every local action was individually reasonable, L1 alone cannot localize the fault.
- **L2 (interaction level, one round per problematic pair).** During the task the Planner communicated with the FileAnalyzer, WebSearcher, and Summarizer, so L2 revisits each pair; smooth pairs exchange no messages and record only brief observations. Only Planner↔WebSearcher opens a discussion: checking the feedback, the Planner notes that the gold answer should have had the smallest words-per-day yet did not in the ranking, traces this to a suspiciously high word count the WebSearcher had reported for that book, and asks about it; the WebSearcher confirms the value was an estimate. This single exchange surfaces a precision gap invisible in either agent's own log. *Aggregation:* the pair records a correlation note ("externally looked-up values must be tagged with source and confidence, and low-confidence values must not be used directly in critical computation") and updates each other's teammate profile.
- **L3 (team level, one collect-and-decide round).** Members push observations via `suggest_team_improvement`; the Planner, acting as chairman, reads them, fuses them with the L2 finding, and commits a structural rule to the shared constitution: "any externally looked-up value must be tagged with its source and confidence, and low-confidence values must be re-verified before use in critical computation." *Aggregation:* the chairman commits the edit to the team constitution so the failure cannot recur.

Difference from independent local reflection. A purely local baseline sees only its own log, exchanges no messages, and commits only self-edits; it therefore cannot diagnose this failure, since every agent acted correctly in isolation. Only the L2 message exchange recovers the shared root cause, which L3 then aggregates into a team-level fix. This corresponds to the gap between the *local* and *collaborative* schemes in Figure 1, and is quantified by the drop when L2/L3 are ablated in Table 3.

E Experiment Settings

E.1 Saturation of Commonly Used Benchmarks

Prior MAS and MAS-evolution studies [34, 35, 75, 81, 82] have evaluated on a wide range of standard benchmarks. However, as LLM capabilities advance rapidly, many of these commonly used benchmarks have become increasingly saturated. As shown in Table 5, a state-of-the-art model already achieves near-perfect performance on several widely used datasets, including HumanEval [83], MBPP [84], MultiArith [85], GSM8K [86], MATH [87], and MMLU [88]. Such saturation reduces their discriminative power for evaluating new MAS evolution methods, since small performance differences near the ceiling may no longer reflect substantive improvements in reasoning, tool use, collaboration, or long-horizon execution.

Therefore, in our main experiments, we focus on more challenging, long-horizon, and verifiable benchmarks where current agents still exhibit substantial room for improvement. This design allows us to better assess whether an MAS evolution method can improve agent behavior beyond saturated short-form reasoning and coding tasks.

Table 5: Performance of Claude Sonnet 4.6 on commonly used benchmarks. We report our own reproduced scores under standard evaluation protocols.

Model	HumanEval	MBPP	MultiArith	GSM8K	MATH	MMLU
Claude Sonnet 4.6	99.4	99.6	100.0	100.0	98.7	97.2

Table 6: Unified runtime settings shared across all benchmarks.

Component	Setting	Value
Backbone LLM	Model	Claude Sonnet 4.6
	API layer	LiteLLM + OpenAI-compatible gateway
	Retry policy	5 attempts, 1.5–60 s exp. back-off
Decoding	Executor temperature	0.2
	Planner temperature	0.2
	Max output tokens	32768
Context window	Max history messages	150 (with pair-preserving trim)
	Tool schema refresh	every LLM call
Reflection	Per-phase step budget	50
	Per-phase timeout	300–900 s
	Per-reflection cost	$50
Force-finalize	Tools disabled	Yes
	Timeout	240 s

E.2 Experiment Settings on Different Benchmarks

SWE-bench Pro SWE-bench Pro [57] evaluates coding agents on real-world GitHub pull requests that require multi-file bug fixes or feature implementations. The public split contains 731 instances across 11 repositories, and each instance is evaluated by running the generated patch against Fail-to-Pass (F2P) and Pass-to-Pass (P2P) tests in an isolated Docker environment. An instance is counted as resolved only if all F2P tests pass and no P2P test regresses.

Experimental settings. Subject to budget constraints, we evaluate on the two largest Python repositories, shown in Table 8. For each repository, we select 20 random instances as the *Evolution set*; the remaining instances form the *Evaluation set*. Evolution is performed separately for each repository and evaluated on its corresponding held-out instances. The reported metric is *Pass Rate*.

Initial MAS configurations. SWE-bench Pro's official scaffold [57] is single-agent (SWE-Agent), but repository-level patch generation is a well-studied multi-agent task: prior work such as MetaGPT [19] and ChatDev [20] decomposes software development into specialized roles (product manager, architect, engineer). We adopt a minimal instantiation of this recipe—Planner + Developer + Reviewer—that retains the three functions most predictive of patch quality: task decomposition, code implementation, and independent verification [79]. The planner scopes the fix and dispatches, the developer edits inside the Docker workspace, and the reviewer inspects `git diff` and reruns tests before submission.

Single-agent baseline. Our SA replicates SWE-bench Pro's officially endorsed reproducibility scaffold [57], namely SWE-Agent (Scale) [68], which is the same single-agent recipe used in the leaderboard's reference Sonnet runs. It is a function-calling agent with the canonical `bash` + `str_replace_editor` + `finish` tool set, executed inside the per-instance Docker workspace and submitting via `git diff HEAD`. We keep all decoding and tool-filter hyperparameters at the SWE-Agent benchmark config and only swap in our shared backbone.

BeyondSWE BeyondSWE [58] extends software-engineering evaluation beyond single-repository bug fixing, covering tasks that require cross-repository knowledge, domain expertise, and whole-repository modifications. The full benchmark contains 500 instances from 246 Python repositories.

Table 7: Time and cost budget for evolution and evaluation for all benchmarks.

Benchmark	**max_seconds**	**max_messages**	**max_cost ($)**	**initial team size**
SWE-bench Pro	1800	600	50	3
BeyondSWE	1800	600	30	3
GAIA	600	300	30	4
LoCoBench	1200	400	30	4
LOCA-bench	3200	600	150	4
ResearchRubrics	1800	600	50	1

Table 8: SWE-bench Pro and BeyondSWE subsets used in our experiments. For each benchmark, we hold out 20 random instances per subset as the *Evolution set*; the remainder forms the *Evaluation set*.

Benchmark	**Selected Subset**	**Total**	**Evolution**	**Evaluation**
SWE-bench Pro	Ansible	96	20	76
	Qutebrowser	91	20	71
	Total	*187*	*40*	*147*
BeyondSWE	CrossRepo	200	20	180
	DepMigrate	178	20	158
	Total	*378*	*40*	*338*

Experimental settings. We evaluate on the two largest subsets, CROSSREPO (200) and DEPMIGRATE (178), which together cover cross-repository reasoning and whole-repository modification—the two capability dimensions most distinct from SWE-bench Pro, yielding 378 instances in total (Table 8). For each subset, we select 20 random instances as the *Evolution set*; the remaining instances form the *Evaluation set*. Evolution is performed separately for each subset and evaluated on its corresponding held-out instances. Evaluation follows the same F2P+P2P protocol as SWE-bench Pro. The reported metric is *Pass Rate*.

Initial MAS configurations. Since BeyondSWE instances share the repository-level patch-generation format with SWE-bench Pro, we reuse the same PLANNER + DEVELOPER + REVIEWER configuration, differing only in the constitution prompts that adapt the team to BeyondSWE's per-task working directory, cross-repository context, and dependency-migration conventions. Using an identical architecture across the two benchmarks isolates the effect of task distribution from that of scaffold design.

Single-agent baseline. BeyondSWE shares the same patch-submission protocol as SWE-bench Pro, so for parity we reuse the SWE-Agent (Scale) [68] SA recipe described above, only adjusting the per-task working directory and the cross-repository context fields prescribed by BeyondSWE [58]. Using identical SA machinery across the two SWE benchmarks isolates the effect of task distribution and matches the convention adopted by BeyondSWE's own evaluation.

LOCA-Bench LOCA-Bench [59] evaluates LLM agents under controllable context growth. It procedurally scales environment state size while keeping task semantics invariant, so the effective context grows smoothly from 8K to 256K tokens. The full benchmark contains 15 Short-to-Long (S2L) tasks $\times$ 5 seeds $\times$ 7 context lengths (8K/16K/32K/64K/96K/128K/256K) = 525 configurations. Evaluation is binary (0/1), computed by offline scripts comparing the agent's final environment state to ground truth.

Experimental settings. To control evaluation cost, we select *8 out of 15 tasks* covering e-commerce, academic, and education domains with 8 business MCP services (Table 9). Leveraging LOCA-Bench's procedural generation, we construct a *dually-isolated* split:

- **Evolution set** (16 cases): 8 tasks $\times$ 96K $\times$ 2 human-specified seeds `{101, 102}` (outside the original benchmark).

Table 9: The eight LOCA-Bench tasks used in our experiments.

#	Task	Total	Domain	Key MCP Services
1	WoocommerceNewWelcome	30	E-commerce CRM	woocommerce, google_cloud, email
2	WoocommerceStockAlert	30	E-commerce purchasing	woocommerce, google_sheet, email
3	FilterLowSellingProducts	30	E-commerce clearance	woocommerce, email
4	ApplyPhDEmail	30	Academic	email, pdf_tools
5	SetConfCrDdl	30	Academic	calendar, email
6	CourseAssistant	30	Education	excel, email
7	CanvasArrangeExam	30	Education	canvas, email, excel
8	CanvasListTest	30	Education	canvas
	Total	*240*		

- **Evaluation set** (240 cases): 8 tasks $\times$ 5 standard seeds `{42, 123, 456, 789, 2024}` $\times$ 6 context lengths `{8K, 16K, 32K, 64K, 128K, 256K}` (deliberately excluding 96K).

The `(seed, context_length)` tuples of the two sets are *fully disjoint*, guaranteeing no data leakage. This design jointly tests *cross-seed* and *cross-scale* generalization. All runs use LOCA-Bench's local mock MCP servers, requiring no Docker or external network. The reported metric is Pass Rate.

Initial MAS configurations. LOCA-Bench officially ships only single-agent scaffolds, so no multi-agent baseline is available for replication. We adopt the *orchestrator–worker* pattern popularized by recent work [24, 89], instantiating it as CHAIRMAN + 3 WORKERS. The pattern fits LOCA-Bench naturally: every task decomposes into *explore* $\rightarrow$ *bulk-process* $\rightarrow$ *verify*, where the bulk phase (80–300 items) exceeds a single agent's effective context [90]. The chairman additionally preserves recoverability when MCP back-ends crash—a known benchmark feature [59]—and the workers act without incurring excessive coordination overhead.

Single-agent baseline. Our SA is LOCA-Bench's officially shipped default scaffold, the `react` strategy [59]: a single ReAct agent [62] with direct access to the full local mock-MCP tool stack (WooCommerce, Google Sheets, Calendar, Email, Canvas, Excel, etc.). We do not modify any tool wiring or context-management policy beyond the shared budget table, so the SA column corresponds to the same single-agent baseline reported in the LOCA-Bench paper, only re-evaluated under our identical backbone and budget.

GAIA GAIA [60] is a knowledge-intensive question-answering benchmark that requires multi-step reasoning with tool use, such as web search, file reading, and code execution. Its public validation set contains 165 questions stratified into three difficulty levels.

Experimental settings. Following [32, 40], we evaluate on the 120-samples subset released by MASFly (Table 10); the 45 excluded samples either require multimodal tools outside our search setting or are marked by annotators as unsolvable under automated exact-match grading. We use Serper API* for Google Search. The returned results are capped at the top-5 entries per query and the page fetcher retrieves a specific URL and returns up to 8K characters of plain text. On these 120 questions we use 20 for evolution (training) and 100 as a disjoint held-out test set, so that every evolved configuration is evaluated on tasks it has never seen. Evaluation follows the official answer-matching protocol and the reported metric is *Pass Rate*.

Initial MAS configurations. We initialize Meta-Team with the four-agent configuration released in the MASFly codebase [40]: a Planner that holds no information-gathering tools and coordinates through message passing, a FileAnalyzer for attached documents, a WebSearcher equipped with Google Search and page fetching, and a Summarizer that enforces the strict GAIA exact-match answer format. Starting from this hand-crafted multi-agent baseline ensures that any improvement we observe is attributable to evolution rather than to the initial design.

*`https://serper.dev`

Table 10: GAIA validation subset used in our experiments. All instances in the three levels are shuffled for evolution and evaluation.

Level	Full (Official)	Our Subset	Train	Test
Level 1	53	44	6	38
Level 2	86	58	9	49
Level 3	26	18	5	13
Total	*165*	*120*	*20*	*100*

Table 11: LoCoBench subsets used in our experiments.

#	Task Category	Total	Evolution	Evaluation
1	Feature Implementation	100	20	80
2	Cross-File Refactoring	100	20	80
	Total	*200*	*40*	*160*

Single-agent baseline. The SA follows the GAIA single-agent recipe used by [32, 40]: a function-calling ReAct agent with {`web_search` (Serper API, top-5), `web_fetch`, `bash`, `read_file`, `submit`}, the same tool surface our MAS exposes to its Web/File workers. The system prompt enforces the official GAIA exact-match answer format, so SA and MAS share both tools and answer normalization—the comparison isolates collaboration, not tool access.

LoCoBench LoCoBench [51] is a long-context code-generation benchmark containing 8,000 scenarios across 10 programming languages, 100 synthetic projects, and 8 task categories. Each scenario provides an entire codebase ranging from 10K to 1M tokens and asks the agent to implement the given requirement. Performance is measured by the LoCoBench Score (*LCBS*), a weighted average of Software Engineering Excellence (40%), Functional Correctness (30%), Code Quality (20%), and Long-Context Utilization (10%), normalized to $[0, 100]$.

Experimental settings. We focus on two scenarios that are most relevant to collaborative coding: Feature Implementation and Cross-File Refactoring (Table 11). For each category, we focus on Python-based tasks to control cost. We use 20 cases for evolution and the remaining 80 cases for evaluation. In addition, we conduct out-of-domain evaluation on C, C++, and Java tasks. The reported metric is average LCBS (LoCoBench Score).

Initial MAS configurations. The defining challenge of LoCoBench is long-context comprehension: each scenario ships an entire codebase (10K–1M tokens, up to 100 files) that rarely fits in a single agent's working context. We therefore adopt a read–write separation inspired by divide-and-conquer long-text processing [18], instantiated as a PLANNER, a DEVELOPER, and two parallel READERS. The planner partitions the file list and dispatches disjoint subsets to readers; each reader extracts structured notes (APIs, call sites, coding conventions) from its assigned files and sends them directly to the developer, which synthesizes the notes into the final patch. The readers provide enough parallelism to absorb 15–50-file scenarios while keeping each reader's context well under the model's window.

Single-agent baseline. LoCoBench's official harness [51] evaluates a single LLM with full-codebase context as the reference scaffold. To keep the comparison fair under our shared cost cap (full-codebase prompting alone exhausts the per-case budget on 100K–1M-token scenarios), our SA mirrors the SWE-Agent (Scale) [68] agentic recipe—`bash` + `str_replace_editor` + `finish` on the same checked-out workspace—which is the standard single-agent baseline used by every recent long-context coding evaluation. This grants the SA the same file-reading bandwidth as our MAS readers, so any gap reflects coordination, not tool starvation.

ResearchRubrics ResearchRubrics [61] evaluates deep-research agents on 101 open-ended research questions spanning ten domains, including AI&ML, historical analysis, STEM, and creative writing. Each question is paired with a human-authored rubric covering explicit requirements, im-

plicit requirements, synthesis, communication quality, instruction following, and citation quality. Responses are scored by an LLM judge, and the final score is a rubric-weighted average.

Experimental settings. We use 20 questions as the *Evolution set* and the rest of questions as *Evaluation set*. The reported metric is *Average Score* on the 0–100 scale.

Initial MAS configurations. Since ResearchRubrics grades responses along six orthogonal axes and the most useful specialists are not obvious a priori, we start from a single LEAD RESEARCHER equipped with web search, page fetching, and file I/O, following the solo-start pattern of [24]. During self-evolution, when reflection identifies a persistent weakness on a specific axis, the Lead Researcher can introduce new specialists, e.g., a citation verifier when citation scores are low, a requirement tracker when explicit requirements are frequently missed, or a report writer when synthesis and communication scores lag.

Single-agent baseline. The SA is the deep-research single-agent recipe from AggAgent [63]: a ReAct loop with `search` (Serper) and `visit` (page fetch with goal-conditioned extraction), and the verbatim SYSTEM PROMPT + SYSTEM-PROMPT-DR from the AggAgent rollout codebase, which encodes the citation, structure, and confidence requirements that ResearchRubrics' rubric grades along. We use AggAgent's reference single-agent recipe rather than a custom one so that the SA column corresponds to a published, reproducible deep-research baseline.

E.3 EXPERIMENT SETTINGS OF BASELINE REPRODUCTIONS

All baselines share the same executor LLM (Claude Sonnet 4.6 via an OpenAI-compatible gateway), tool bundle, evaluation splits, and per-case time / message / cost limits as Meta-Team. All MAS baselines also start from the same benchmark-specific initial team as Meta-Team (the "MAS" row in Table 1), mapped into each baseline as closely as its native parameterization allows. We only describe the method-specific adaptations below.

SA. Vanilla ReAct [62] with the same system prompt template, tool bundle, step cap (50), and context-trimming rule (150 retained messages, pair-preserving) as Meta-Team's executor.

MAS. Meta-Team's initial configuration on each benchmark with reflection and the commit gate disabled. Concretely, we zero out the L1/L2/L3 analyzers and skip the verifier call, so the team never mutates after task k.

In AggAgent [63], we use the authors' implementation of `aggagent` package (`rollout/` + `aggregation/`). Per case we sample $K{=}5$ single-agent rollouts at temperature 0.2, then run the default `CROSS_VALIDATION` aggregation strategy.

In OWL [64], we keep the released PLANNER–COORDINATOR–WORKER hierarchy unchanged with minor adaptations on each benchmark.

AOrchestra (Agent-Orchestra). The `MainAgent` runner is used unchanged; the orchestrator is allowed to synthesize up to 4 sub-agents per task, each inheriting $M{=}$Claude Sonnet 4.6. Because AOrchestra hard-codes `config/example/model_config.yaml`, our adapter `chdirs` into the AOrchestra directory before import and redirects LLM calls through our gateway in `LLMsConfig.default()`.

For AgentSquare [31], we run the search loop over {PLANNING, REASONING, TOOLUSE, MEMORY} for 30 trials on the evolution set of each benchmark. After search, we freeze the best-performing configuration and evaluate it on the held-out split. For each benchmark, we construct a carefully curated search pool by prompting Claude Opus 4.6 to augment the baseline MAS with more than 60 scaffold templates. Each template is further decomposed into modular components, which serve as the candidate units for AgentSquare's search procedure.

Although ReCreate [48] was originally proposed for single-agent evolution, its scaffold-level optimization interface makes it naturally extensible to experience-driven MAS evolution. We therefore adapt ReCreate as a MAS evolution baseline. Specifically, we use the default hyperparameter with `BATCH_SIZE`$=4$ and `N_REPEAT`$=2$. We use Claude Sonnet 4.6 as the execution agent model, replacing the default `gpt-5-mini`, and Claude Sonnet 4.6 as the *agent-as-optimizer* model. We

wrap Meta-Team's configuration through its `adapters/` entry points on benchmarks not shipped in the original repo and reuse the released adapters on SWE-bench-like domains.

AgentNet [37]. We instantiate one AgentNet per benchmark with 3 seed agents and the GAIA ability vector from the paper (`reasoning`/`knowledge`/`language` each at 0.6), and pre-populate the memory store with our evolution set. Official defaults are kept: router temperature 0.7, DAG pruning threshold 0.3, memory retrieval top-k=5. The episodic memory is persisted within a benchmark but cleared between benchmarks.

Evolution / held-out protocol across baselines. We evaluate every method under a strict evolution/test split. Methods that include a learning phase (Meta-Team, ReCreate, MASFly, AgentNet, AgentSquare) consume only the benchmark's designated evolution set, freeze all learned artifacts, and are then evaluated on the disjoint held-out set. ReCreate and AgentSquare support this protocol natively through `-skip-instances` / train–test file flags and are used as-is. For MASFly, we rebuild the SOP Repository and Personalised Experience Pool on our own evolution set (`go run . -eval 0`) rather than using the shipped repository, so that no released SOP can have seen our held-out instances. AgentNet's released runner performs *online* router/memory updates during evaluation; we extend it with a snapshot step that freezes the routing weights, ability vectors, and episodic memory after the evolution set, disabling further updates during held-out evaluation. Methods without a learning phase (SA, MAS, AggAgent, OWL, AOrchestra) are evaluated on the held-out set directly.

MASFly [40]. MASFly's reference Go implementation is driven by `go run . -eval 0` to build the SOP Repository and Personalised Experience Pool from the evolution set, then `-eval 2` for held-out evaluation. We expose Meta-Team's executor through MASFly's OpenAI-compatible environment variables (`OPENAI_BASE_URL`, `OPENAI_MODEL`); feedback style, analyzer prompt, and SOP top-k=3 retrieval are kept at the paper's defaults.

Shared harness notes. Three harness-level rules apply uniformly to every baseline: (a) SWE-bench Pro / BeyondSWE patches are scored through the official harness container with empty-patch = unresolved; (b) LOCA-Bench task setup and evaluation are wrapped in a process-level lock to avoid the `os.environ` race at workers ≥ 8; (c) all ResearchRubrics submissions are graded by the rubric judge (Claude Opus 4.6, temperature 0.2).

F Further Experiment Results

F.1 Statistical Reliability of Main Results

We further assess the statistical reliability of the main-result margins in Table 1. Since all methods are evaluated on the same held-out instances, each comparison forms a paired sample. We use the instance as the unit of analysis: for each method and instance, the three rollouts are aggregated into a single avg@3 score, and a two-sided paired t-test is applied to the per-instance differences. As a robustness check, the Wilcoxon signed-rank test gives the same significance label in all checked cases; we therefore report the paired t-test results.

Table 12 summarizes Meta-Team's margins over the initial MAS and over MASFly, together with the corresponding p-values. The most reliable gains appear on the longer and more coordination-intensive benchmarks: SWE-Pro Ansible, BeyondSWE DepMigrate, LOCA-Bench, both LoCoBench slices, and ResearchRubrics are all significant against both baselines at $p < 0.05$ or better.

The remaining columns, SWE-Pro Qute., BeyondSWE CrossRepo, and GAIA, show positive but not statistically significant margins. These columns have smaller effect sizes and many paired instances on which both methods obtain the same outcome, leaving limited paired variation for the test. We therefore interpret them as directionally positive rather than statistically conclusive.

Overall, the paired tests support the qualitative reading of Table 1: Meta-Team improves over the baselines across all benchmark columns, with statistically reliable gains on the settings that most benefit from collaborative self-evolution, while smaller-margin columns remain positive but inconclusive at the current sample size.

Table 12: Significance of Meta-Team's gains over the initial MAS and MASFly. Each p-value is from a two-sided paired t-test over per-instance avg@3 scores.

Benchmark	**Δ vs. MAS**	p_{MAS}	**Δ vs. MASFly**	p_{MASFly}
SWE-Pro (Ansible)	$+13.1$	< 0.05	$+10.5$	< 0.05
BeyondSWE (DepMig.)	$+8.0$	< 0.05	$+5.9$	< 0.05
LOCA-Bench	$+5.8$	< 0.001	$+12.1$	< 0.05
LoCoBench (Feat.)	$+9.5$	< 0.001	$+7.0$	< 0.05
LoCoBench (Refact.)	$+6.1$	< 0.01	$+2.5$	< 0.05
ResearchRubrics	$+6.3$	< 0.001	$+5.0$	< 0.05

F.2 Cost of the Evolution Process

We analyze the cost of Meta-Team's evolution, first for Meta-Team itself and then in comparison with the evolution baselines. All numbers are actual USD costs on Claude Sonnet 4.6.

Cost accounting of Meta-Team. In Table 13, *Evolution Run Cost* is the full cost of one evolution run (task execution plus L1–L3 evolution); *Evolution Overhead* is the extra cost added by the evolution on top of task execution; *Overhead Ratio* = *Evolution Overhead* / *Evolution Run Cost*; and *Evaluation Cost* is one avg@3 evaluation pass on the held-out set. The evolution overhead totals about \$172 (40–63% of an evolution run), while the remainder is the task-execution cost shared by any experience-driven method. Evolution is also a one-time cost: the total (\$348) is about 12.5% of a single avg@3 evaluation pass (\$2,790), after which the team is frozen and reused across downstream tasks. The overhead is largest on the long-context, repository-level benchmarks (e.g., LoCoBench Feature, \$60), tracking the length and difficulty of the underlying tasks.

Table 13: Evolution cost statistics of Meta-Team (USD) on each benchmark. All values are rounded to integers.

Benchmark	**Evolution Run Cost**	**Evolution Overhead**	**Overhead Ratio**	**Evaluation Cost (avg@3)**
SWE-Pro Ansible	30	13	43%	249
SWE-Pro Qutebrowser	30	12	40%	170
BeyondSWE CrossRepo	18	8	44%	378
BeyondSWE DepMigrate	25	10	40%	412
GAIA	28	15	54%	75
LoCoBench Feature	60	38	63%	394
LoCoBench Refactoring	44	20	45%	353
LOCA-Bench	68	30	44%	540
ResearchRubrics	44	25	57%	221
Total	**348**	**172**	**49%**	**2,790**

Cost comparison with the evolution baselines. We compare Meta-Team's evolution cost with two representative evolution baselines, ReCreate and MASFly, under the same setting as the main results. As shown in Table 14, Meta-Team's total evolution cost (\$348) is on the same order as the baselines (\$216–\$268), about 30% above MASFly and 60% above ReCreate. This difference stays bounded because agents in Meta-Team communicate on demand, based on their own reasoning about whether an exchange is needed, rather than exchanging all-to-all. For reference, this level of evolution cost accompanies average gains of $+6.3$ over MASFly and $+7.9$ over ReCreate, with statistically significant margins (Table 1 and Appendix F.1).

F.3 Generalization Across Base Models

The main experiments use Claude Sonnet 4.6 as the backbone. To examine whether the gains depend on a specific model, we re-run the full evolution and evaluation pipeline with GPT-5.5 as the backbone on three representative benchmarks that span our task categories: SWE-Pro-Ansible

Table 14: The *Evolution Run Cost* of Meta-Team versus the evolution baselines (USD). All values are rounded to integers.

Benchmark	Meta-Team	ReCreate	MASFly
SWE-Pro Ansible	30	19	21
SWE-Pro Qutebrowser	30	17	21
BeyondSWE CrossRepo	18	14	13
BeyondSWE DepMigrate	25	19	18
GAIA	28	18	30
LoCoBench Feature	60	33	42
LoCoBench Refactoring	44	25	39
LOCA-Bench	68	47	53
ResearchRubrics	44	24	31
Total	**348**	**216**	**268**

(coding, long-horizon), ResearchRubrics (non-coding, open-ended research), and GAIA2-mini (non-coding, tool-agentic).[†] We compare against the initial hand-crafted MAS (the starting point of evolution) and MASFly (the strongest MAS-evolution baseline in Table 1), keeping all other settings fixed. We report pass rate (%) for SWE-Pro-Ansible and GAIA2-mini and the average rubric score (0–100) for ResearchRubrics. As shown in Figure 5, Meta-Team remains the best method on all three benchmarks under GPT-5.5, indicating that its benefits carry over across frontier model families rather than being tied to a single model.

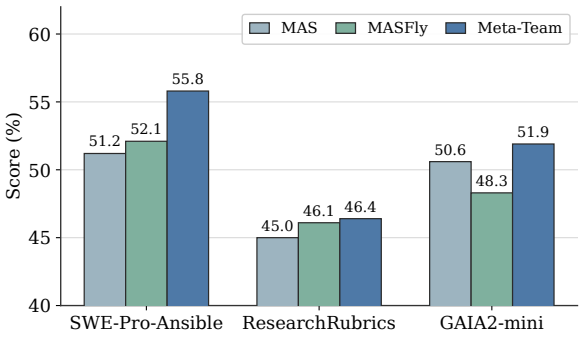


Figure 5: Cross-model validation with GPT-5.5.

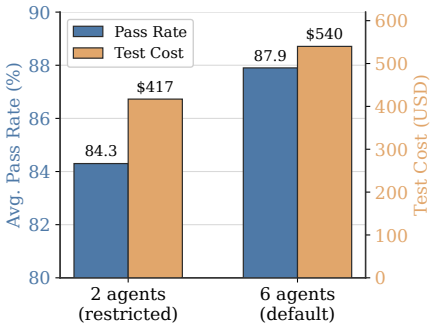


Figure 6: Effect of team size.

F.4 Effect of Team Size

Team size in Meta-Team is not a fixed hyperparameter but an outcome of evolution: the open-roster mechanism (Appendix D) can dynamically recruit new members, and team-level evolution (L3) can introduce new roles from accumulated experience. This adaptation is already visible on LOCA-Bench (§4.4), where evolving only on 96K-token cases leads Meta-Team to add two extra workers, which is what keeps it effective at the longest context (256K).

To directly examine the effect of varying the number of agents, we take the same evolved team and, at test time, restrict its active roster to 2 agents (1 Chairman + 1 Worker), comparing against the default evolved team (6 agents); all other settings follow Figure 3a, and both pass rate and test cost are averaged over the six context scales (8K–256K). As shown in Figure 6, allowing the team to grow improves the average pass rate by $+3.6$ while the test cost rises only moderately ($\sim$30%), suggesting that Meta-Team scales with team size in both accuracy and cost because the open-roster protocol recruits only the agents a task needs.

[†]GAIA2 [91] is a recent tool-agentic benchmark over a simulated tool-and-app ecosystem; we use its official mini subset with the same evolution/evaluation protocol as the main paper.

F.5 Stability Across Independent Evolution Runs

All results in the main paper are reported as avg@3 (three independent rollouts per instance), and the main-result margins are backed by paired significance tests (Appendix F.1). These characterize evaluation variance, but not the variance of the evolution process itself, which is driven by LLM reflection. To characterize the latter, we repeat the entire self-evolution pipeline from scratch under three independent runs on the main backbone (Claude Sonnet 4.6); each run re-executes the full L1/L2/L3 evolution and produces its own evolved team, which is then evaluated. We use the same three representative benchmarks and baselines as in Appendix F.3, reporting pass rate (%) for SWE-Pro-Ansible and GAIA2-mini and the average rubric score (0–100) for ResearchRubrics. Table 15 reports the mean and standard deviation across the three runs. The small deviations indicate that Meta-Team's gains are stable across independent evolution runs rather than the product of a single evolution trajectory. (MAS does not evolve, so its deviation reflects avg@3 evaluation variance only.)

Table 15: Results across three independent evolution runs on the main backbone (mean $\pm$ std).

Method	**SWE-Pro-Ansible**	**ResearchRubrics**	**GAIA2-mini**
MAS	42.7 ± 1.8	49.3 ± 2.1	43.8 ± 1.9
MASFly	44.8 ± 1.9	50.1 ± 2.2	44.2 ± 1.7
Meta-Team	$\mathbf{53.1 \pm 1.6}$	$\mathbf{53.7 \pm 1.5}$	$\mathbf{46.9 \pm 2.4}$