MAS-LOOP: BEYOND ONE-SHOT ORCHESTRATION TOWARDS SELF-EVOLVING MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training-time multi-agent system (MAS) design seeks to learn reusable orchestration policies, yet prevailing approaches remain largely one-shot: an Orchestrator generates a system but only observes and learns from its terminal outcome. This paradigm discards execution trajectories that expose system-level failures and contain fine-grained information beyond the terminal outcome, leaving no direct basis for revising the source MAS, evaluating the updated MAS, or learning from the resulting contrast. To this end, we introduce **MAS-Loop**, an execution-grounded framework that closes this feedback loop by coupling an Orchestrator with a trajectory-conditioned Refiner. For each task, the Orchestrator constructs and executes a source MAS to obtain an outcome score and an intermediate trajectory. Conditioned on the trajectory, the Refiner localizes an actionable failure, formulates a trajectory-grounded diagnosis, and proposes a bounded patch to the agents, information flow, or execution configuration. The executable updated MAS is re-executed on the same task to measure its realized outcome under stochastic execution. This pair-MAS comparison provides complementary learning signals: the Orchestrator learns to generate strong systems with fewer recoverable failures, whereas the Refiner learns to expose and repair observed weaknesses, thereby driving policy-level co-evolution across training iterations. We evaluate MAS-Loop across interaction environments, knowledge-intensive multi-hop reasoning, and general reasoning, examining both the complete framework and the independent utility of these two policies. Experiments show that MAS-Loop achieves the strongest overall performance among the evaluated baselines, with consistent gains from refinement across all evaluated benchmarks. The co-evolution process also yields **MAS-JudgeCorpus**, a dataset of execution-grounded source-updated pairs with orchestration- and trajectory-level supervision, which can provide the community with support for MAS reward modeling and trajectory-aware investigation.

1 INTRODUCTION

Training-time multi-agent system (MAS) design has emerged as a promising alternative to inference-time MAS design, where the workflow search must be repeated for different tasks and cannot accumulate reusable experience across tasks (Zhang et al., 2025; Ke et al., 2025b; Hu et al., 2025; Lin et al., 2026b). Training-time MAS trains an Orchestrator to construct a task-specific MAS by selecting models or tools, assigning agent roles, and organizing communication (Li et al., 2024; Ke et al., 2025a; Tran et al., 2025). The system then executes and uses its outcomes to improve the Orchestrator through supervised learning (Ye et al., 2025; Yue et al., 2025), reinforcement learning (Yang & Thomason, 2026; Ke et al., 2026), and graph-based optimization (Zhuge et al., 2024; Zhang et al., 2024). In this way, experience acquired from previous executions can be encoded into a reusable orchestration policy and transferred to future tasks (Lin et al., 2026c).

Although this makes MAS construction learnable, this *one-shot orchestration* remains an incomplete foundation for effective MAS self-evolution (Qi et al., 2026): orchestration and execution feedback remain disconnected. A system trajectory can expose missing information flow, redundant reasoning, or failures in verification and aggregation (Shen et al., 2025; Cemri et al., 2026). Yet this trajectory-level evidence is often omitted from the terminal correctness signal (Jwalapuram et al., 2026). Therefore, without a feedback-driven refinement mechanism, the Orchestrator must

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

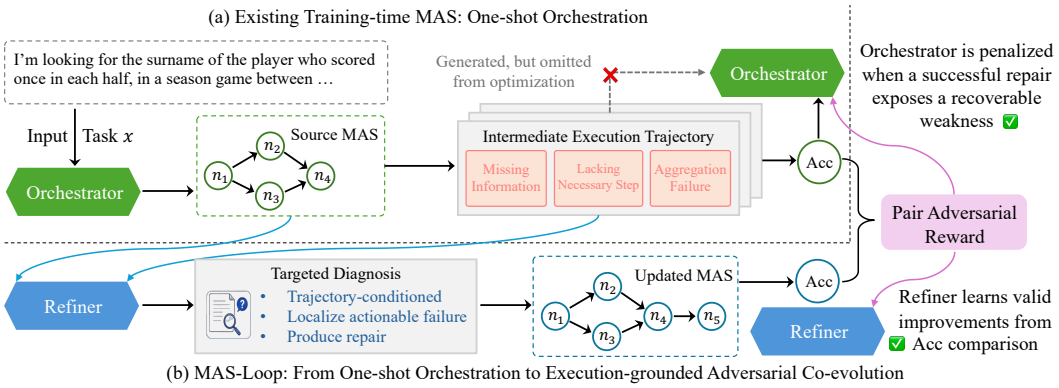


Figure 1: The Illustration of MAS-Loop. (a) Existing training-time MAS methods rely on one-shot orchestration and omit intermediate execution feedback from optimization. (b) MAS-Loop moves from one-shot orchestration to execution-grounded adversarial co-evolution by adding a trajectory-conditioned Refiner and incorporating coupled incentive learning signals.

discover better designs indirectly through independently sampled rollouts and their sparse outcome rewards (Zhang, 2026; Su et al., 2025). Although this trial-and-error process may eventually favor stronger systems, it relies on vanilla independent rollouts and sparse terminal rewards rather than utilizing rich intermediate trajectories as feedback to directly diagnose failures, guide targeted repairs, and optimize from verified improvements.

However, a complete self-evolution loop requires more than learning to generate the next system: it should connect execution evidence to a direct intervention on the current system and facilitate learning based on the consequences of that intervention (Chen et al., 2026a; Yang et al., 2026a). To this end, we introduce **MAS-Loop**, an execution-grounded framework that augments the *Orchestrator* with a complementary learned policy, the *Refiner*. The Orchestrator first constructs a source MAS, whose execution produces a terminal score and an intermediate trajectory. The Refiner observes the source design, final prediction, and trajectory to localize actionable weaknesses, formulate a grounded diagnosis, and propose a bounded patch to agent roles, information flow, or execution settings. This patch is compiled into an updated MAS and re-executed on the same task to measure the realized outcome change. The source-updated score difference provides coupled learning signals: it encourages the Orchestrator to produce strong systems with fewer recoverable weaknesses and trains the Refiner to generate valid revisions that repair observed failures, enabling the two policies to co-evolve from this rich-learning closed-loop. This co-evolution occurs across policy-optimization iterations: each rollout contains one bounded refinement, while subsequent task rollouts use the newly updated Orchestrator and Refiner.

During policy co-evolution, MAS-Loop naturally produces large numbers of execution-grounded source-updated pairs. To obtain reliable comparisons, we retain tasks with a fixed rollout budget, discard ties and pairs with insufficient absolute or relative score gaps, remove locally inconsistent examples, and canonicalize Refiner operations before operation-aware deduplication. The resulting **MAS-JudgeCorpus** contains source and updated MAS instances whose quality differences are established by same-task re-execution, providing pairwise preferences for training MAS reward models with orchestration- and execution-level supervision. Then we leverage MAS-JudgeRM for test-time ranking and selection among candidate MAS, while the retained trajectory evidence also offers a basis for future trajectory-conditioned judging and other execution-aware evaluations.

We evaluate MAS-Loop across interaction environments, knowledge-intensive multi-hop reasoning, and general reasoning, including both in-task performance and out-of-task transfer. MAS-Loop achieves the strongest overall performance among the evaluated single-agent and automatic MAS baselines. The Orchestrator remains effective independently, while the Refiner consistently improves both its source systems and those produced by other MAS methods across in-task and out-of-task evaluations. For MAS-JudgeCorpus, we train MAS-JudgeRM on pairwise preferences and compare it with general-purpose reward models, orchestration-specific reward models, and

prompted LLM judges for selecting among a fixed set of candidate MAS. Together, these experiments assess both MAS-Loop's effectiveness as a self-evolving MAS framework and MAS-JudgeCorpus's value for test-time MAS selection. Our contributions are summarized as follows:

- We identify a central limitation of current training-time MAS: one-shot orchestration leaves execution feedback disconnected from the MAS design, forcing the Orchestrator to improve indirectly through sparse terminal rewards rather than direct rich trajectory-level evidence.
- We introduce **MAS-Loop**, an execution-grounded framework that couples an Orchestrator with a trajectory-conditioned Refiner. The paired re-execution evaluates the updated MAS under the same task protocol and provides complementary learning signals that drive both policies to co-evolve across training iterations.
- We construct **MAS-JudgeCorpus** from execution-grounded source–updated MAS pairs, applying fixed-rollout quality filtering, conservative consistency checks, and operation-aware deduplication to obtain reliable preferences for training MAS reward models.

2 Related Work

2.1 Automatic Multi-Agent System Design

Inference-time MAS design typically uses a powerful frontier LLM together with iterative search, reflection, or optimization to construct an agent workflow for each task (Ruan et al., 2026; Ma et al., 2026; Zhou et al., 2026). This approach can adapt the workflow to different tasks, but the search usually has to be repeated when the domain changes. It therefore has limited ability to accumulate orchestration experience that transfers across tasks (Lin et al., 2026a; Wang et al., 2026; Xu et al., 2026b). Training-time MAS design instead learns an orchestration policy, often implemented with a smaller model, from experience collected across prior executions (Su et al., 2025; Zhang et al., 2026). Existing studies explore supervised learning from successful orchestration traces (Wang et al., 2025; Yang et al., 2026b), reinforcement learning with task-level rewards and structured credit assignment (Ke et al., 2026; Chen et al., 2026b), and graph-based optimization of agent organizations and communication patterns (Li et al., 2026; Liu et al., 2026). Despite this progress, most training-time approaches remain largely one-shot. They use execution outcomes to improve later orchestration proposals, but do not closely connect trajectory-level feedback to the decisions that created the system and lack a direct refinement mechanism. So the Orchestrator must discover better designs indirectly through repeated rollouts and sparse outcome rewards.

MAS-Loop addresses this limitation by coupling the Orchestrator with a trajectory-conditioned Refiner, which uses evidence from the source MAS's execution to generate an executable revision. The updated MAS is then re-executed on the same task, and the score difference between the source and updated systems provides adversarial learning signals for the co-evolution of both policies.

2.2 Multi-Agent System Benchmarks

Existing MAS benchmarks characterize several aspects of multi-agent behavior. MultiAgentBench measures collaboration and competition among agents under different interaction settings (Zhu et al., 2025), while other work organizes recurring coordination failures or attributes errors to particular agents and interactions (Ge et al., 2026; Kong et al., 2026; Huang et al., 2026). Complementary metrics and reward-modeling approaches assess the quality of multi-agent organizations by comparing orchestration choices or scoring system behaviors and trajectories (Tsang et al., 2026). However, these resources primarily characterize MAS behavior or score existing designs, providing limited paired supervision that connects execution evidence to a concrete system revision. In this work, **MAS-JudgeCorpus** is constructed from source MAS and trajectory-conditioned updates generated during MAS-Loop, with same-task re-execution establishing their relative quality. After retaining fixed-budget tasks with sufficiently large, non-tied absolute and relative score gaps, removing pair-level ties and locally inconsistent comparisons, canonicalizing Refiner operations, and deduplicating by task-operation pairs, the corpus provides complementary pairwise supervision at two levels for training MAS reward models: orchestration-level supervision uses task descriptions and MAS graph structure to support execution-free judgment, whereas trajectory-level supervision additionally incorporates rich execution evidence for more accurate quality assessment.

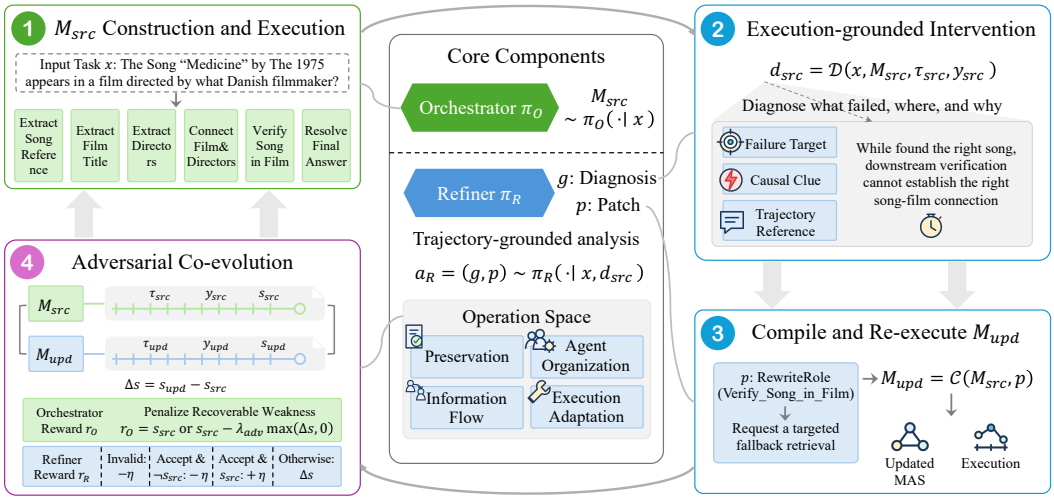


Figure 2: Overview of MAS-Loop. The paired source and updated MAS's material provide complementary rewards that update both policies, closing the feedback loop from execution back to subsequent MAS construction.

3 MAS-Loop: Execution-Grounded Adversarial Co-Evolution

3.1 Overview

Given a task instance x, our goal is to learn two coupled policies: an Orchestrator that constructs an executable source MAS and a Refiner that diagnoses its execution and applies a bounded revision. We represent an MAS as an attributed directed graph

$$M = (V, E, \phi), \tag{1}$$

where V is the set of agents, $E \subseteq V \times V$ specifies directed communication and dependency relations, and ϕ assigns each agent its role and execution configuration. Thus, M specifies which agents participate, how they are connected, and what function each agent performs. An executor $\mathcal{E}$ instantiates the graph on the task:

$$(\tau, y) \sim \mathcal{E}(M, x), \qquad s = r_{\text{term}}(x, y, y^\star), \tag{2}$$

where τ records node-level responses, dependency usage, intermediate evidence, and execution failures; y is the final prediction; $y^\star$ is the task's gold label; and r_{term} computes the terminal reward from the prediction and the task outcome criterion. The scalar s summarizes the final task outcome and is retained by the training-side evaluator for reward computation only. In contrast, τ preserves the execution evidence needed to identify where the realized execution may have failed. MAS-Loop couples the two policies through an execution-grounded revision loop: the Orchestrator π_{O} first generates a source MAS M_{src}, after which the Refiner π_{R} uses the execution evidence to produce a structured diagnosis g and a bounded patch p, jointly denoted by $a_{\text{R}} = (g, p)$. Applying p yields an updated MAS M_{upd} for re-execution, which establishes the credit-assignment boundary: the Orchestrator is trained according to the quality and refinability of its source MAS, whereas the Refiner is trained according to the realized score change following its diagnosis and patch.

3.2 Structured Policy Interfaces

Orchestrator. The Orchestrator receives the task x and samples a source MAS

$$M_{\text{src}} \sim \pi_{\text{O}}(\cdot \mid x). \tag{3}$$

Rather than answering x directly, the Orchestrator instantiates a task-specific computational organization. It assigns complementary responsibilities and execution configurations to the agents in V,

and uses the directed edges in E to determine how intermediate results flow between them. A typical source MAS therefore decomposes the task across upstream agents, passes their relevant outputs to downstream agents, and uses the resulting dependency structure to produce the final response.

Refiner. After the source MAS is executed, the source system and its execution record are packaged into a bounded diagnostic input

$$d_{\mathrm{src}} = \mathcal{D}(x, M_{\mathrm{src}}, \tau_{\mathrm{src}}, y_{\mathrm{src}}), \tag{4}$$

where $\mathcal{D}$ packages the task, source MAS, execution trajectory, and final prediction. It explicitly removes the gold label, so the Refiner must decide how to edit the source MAS from observable execution evidence rather than an oracle correctness bit. Given this input, the Refiner produces

$$a_{\mathrm{R}} = (g, p) \sim \pi_{\mathrm{R}}(\cdot \mid x, d_{\mathrm{src}}), \tag{5}$$

where g is a structured diagnosis and p is a bounded patch; the Refiner outputs them together in one step. The diagnosis localizes an actionable failure, states a causal hypothesis, and proposes a repair plan supported by trajectory references. We organize the operation space according to the aspect of the source MAS being revised. The four categories below together define the Refiner action space:

$$\begin{aligned} \mathcal{A}_{\mathrm{preserve}} &= \{\mathrm{Accept}\}, & \mathcal{A}_{\mathrm{agent}} &= \{\mathrm{AddAgent},\ \mathrm{RewriteRole}\}, \\ \mathcal{A}_{\mathrm{flow}} &= \{\mathrm{AddEdge},\ \mathrm{RemoveEdge}\}, & \mathcal{A}_{\mathrm{adapt}} &= \{\mathrm{AdjustReasoningBudget},\ \mathrm{ChangeTool}\}. \end{aligned} \tag{6}$$

Operation space. *(1) Preservation.* Accept keeps the source MAS unchanged when its execution is adequate or when the diagnostic evidence does not justify a reliable intervention. *(2) Agent organization.* $\mathrm{AddAgent}$ introduces a new task-solving agent and, as part of this atomic operation, connects its output to the current aggregation agent without changing the aggregation agent itself. $\mathrm{RewriteRole}$ revises an existing agent whose responsibility is misaligned with its subtask. *(3) Information flow.* $\mathrm{AddEdge}$ instantiates a new dependency between agents, whereas $\mathrm{RemoveEdge}$ eliminates an irrelevant or harmful dependency. The compiler rejects any edit that produces an unresolved endpoint, a cycle, or an executable graph without a valid path to aggregation. *(4) Execution adaptation.* $\mathrm{AdjustReasoningBudget}$ changes an agent's explicitly scalable reasoning depth, such as the number of search rounds or its generation allowance, while $\mathrm{ChangeTool}$ enables, disables, or switches the tool access assigned to an agent when its capabilities do not match the subtask. A deterministic patch compiler applies the canonicalized operations to obtain the updated MAS

$$M_{\mathrm{upd}} = \mathcal{C}(M_{\mathrm{src}}, p). \tag{7}$$

Pair Data Collection. For each source sample, the executor produces

$$(\tau_{\mathrm{src}}, y_{\mathrm{src}}) \sim \mathcal{E}(M_{\mathrm{src}}, x), \qquad s_{\mathrm{src}} = r_{\mathrm{term}}(x, y_{\mathrm{src}}, y^{\star}), \tag{8}$$

$$(\tau_{\mathrm{upd}}, y_{\mathrm{upd}}) \sim \mathcal{E}(M_{\mathrm{upd}}, x), \qquad s_{\mathrm{upd}} = r_{\mathrm{term}}(x, y_{\mathrm{upd}}, y^{\star}). \tag{9}$$

We re-execute the unchanged source graph when the Refiner abstains or produces an invalid patch. For every task, the training group contains multiple independently sampled source-updated rollout pairs. Each source MAS receives one Refiner decision and one independent updated execution; the group therefore estimates refinement behavior across sampled systems rather than repeatedly evaluating the same patch.

3.3 Adversarial Co-Evolution

The paired records provide a stochastic, execution-grounded signal of whether a Refiner intervention is useful. For an individual rollout, the difference below is a realized score change

$$\Delta s = s_{\mathrm{upd}} - s_{\mathrm{src}}. \tag{10}$$

This quantity is then used to construct complementary rewards for the two policies. For the Orchestrator, the reward is defined as:

$$r_{\mathrm{O}} = \begin{cases} s_{\mathrm{src}}, & \text{if } M_{\mathrm{upd}} = M_{\mathrm{src}}, \\ s_{\mathrm{src}} - \lambda_{\mathrm{adv}} \max(\Delta s, 0), & \text{otherwise}, \end{cases} \tag{11}$$

where $\lambda_{\mathrm{adv}} \geq 0$ controls the strength of the adversarial penalty. For an unchanged MAS, any score difference caused by re-execution is not used to modify the Orchestrator reward. For a valid update,

the first term favors source systems that solve the task without refinement, while the second term penalizes only positive improvements: a successful repair indicates that the source system contains a recoverable weakness, whereas a harmful intervention reflects a failure of the Refiner rather than an additional weakness of the source system.

For the Refiner, an *Accept* action is evaluated by whether the source system is already correct, while an executable intervention is rewarded according to its measured improvement. Let $c_{\mathrm{src}} \in \{+1, -1\}$ indicate whether the source answer satisfies the task criterion. We define

$$r_{\mathrm{R}} = \begin{cases} -\eta, & \text{if invalid,or if } a_{\mathrm{R}} = \mathrm{Accept} \text{ and } c_{\mathrm{src}} = -1, \\ +\eta, & \text{if } a_{\mathrm{R}} = \mathrm{Accept} \text{ and } c_{\mathrm{src}} = +1, \\ \Delta s, & \text{otherwise}, \end{cases} \tag{12}$$

where $\eta > 0$ sets both the invalid-action penalty and the magnitude of the accept reward. Thus, a positive Δs rewards a repair that improves the re-executed system, whereas a negative Δs penalizes a damaging intervention. The signed reward for Accept guides the Refiner both to preserve a source MAS that is already correct and to avoid accepting one that remains incorrect. We use $\lambda_{\mathrm{adv}} = 0.3$ and $\eta = 0.5$ by default, and we study the effect of these two hyperparameters in Section 4.2.

Together, these rewards define a constructive adversarial objective. The Orchestrator learns to construct source systems that are effective from the outset and require little or no further refinement, while the Refiner learns to identify grounded interventions that improve observed systems.

GRPO-based co-evolution. We use Group Relative Policy Optimization (GRPO) as the standard policy-optimization procedure (Shao et al., 2024; Guo et al., 2025). For each policy, we combine sampled rollout records with their corresponding rewards to form separate training batches:

$$\mathcal{B}_{\mathrm{O}} = \left\{ \left(\zeta_{\mathrm{O}}^{(i)}, r_{\mathrm{O}}^{(i)} \right) \right\}_{i=1}^{N}, \qquad \mathcal{B}_{\mathrm{R}} = \left\{ \left(\zeta_{\mathrm{R}}^{(j)}, r_{\mathrm{R}}^{(j)} \right) \right\}_{j=1}^{N}, \tag{13}$$

where ζ_{O} and ζ_{R} denote rollout records sampled for the Orchestrator and Refiner, respectively. We apply GRPO separately to these two batches using their corresponding rewards:

$$\pi_{\mathrm{O}}^{k} \leftarrow \mathrm{GRPO}(\pi_{\mathrm{O}}^{k-1}; \mathcal{B}_{\mathrm{O}}^{k}), \qquad \pi_{\mathrm{R}}^{k} \leftarrow \mathrm{GRPO}(\pi_{\mathrm{R}}^{k-1}; \mathcal{B}_{\mathrm{R}}^{k}). \tag{14}$$

4 Main Experiments

4.1 Experimental Setup

Benchmarks. We evaluate MAS-Loop on three task families: interaction environment, multi-hop question answering, and general reasoning. The Orchestrator and Refiner are trained on ALFWorld (Shridhar et al., 2020), HotpotQA (Yang et al., 2018), and DeepScaler (Tan et al., 2025). We evaluate in-task performance on ALFWorld, HotpotQA, and AIME (MAA-Committees, 2025) and test out-of-task (OOT) generalization on SciWorld (Wang et al., 2022a), VitaBench (He et al., 2026), BrowseComp-Plus (Chen et al., 2025), and GPQA-Diamond (Rein et al., 2023). All metrics are converted to a common $0 - 100\%$ scale. Appendix C.1 provides additional information about the datasets, splits, and evaluation procedures.

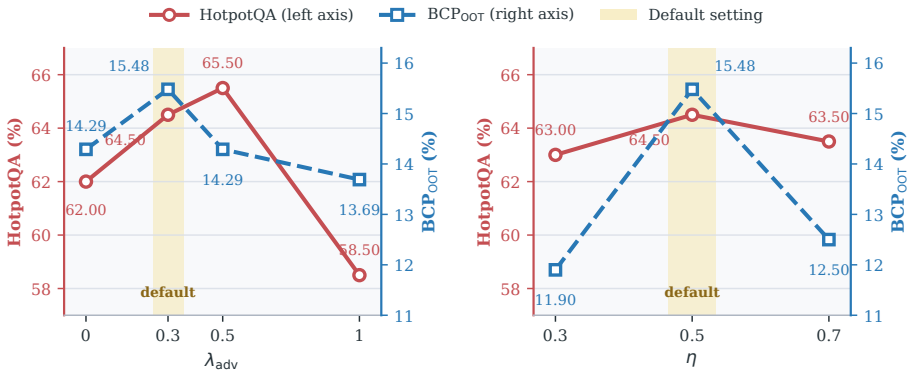
Baselines. We compare MAS-Loop against three families of baselines. The first family contains general inference baselines, including Chain-of-Thought (CoT) (Wei et al., 2022) and self-consistency decoding (CoT-SC) (Wang et al., 2022b). The second family contains inference-time workflow-construction methods, including AFlow (Zhang et al., 2025), AOrchestra (Ruan et al., 2026), and Skill-MAS (Lin et al., 2026c). The third family contains training-time coordination methods, including MAS-GPT (Ye et al., 2025), ToolOrchestra (Su et al., 2025), and MAS-Orchestra (Ke et al., 2026). Baseline details are elaborated in Appendix C.2. Across all experiments, we additionally evaluate the independent deployment ability of Orchestrator and Refiner. We leverage Qwen3-8B (Team, 2025) as the trainable Orchestrator and Refiner, and Qwen3-30B-A3B-Instruct-2507 (Team, 2025) as the frozen sub-agent backbone. Detailed implementation and adaptation settings are provided in Appendix C.3.

4.2 Results and Analysis

Main results. First, existing MAS baselines do not exhibit a stable advantage over single-agent reasoning across task families. Their relative rankings vary substantially across datasets: for exam-

Table 1: Quantification comparison of MAS-Loop and baselines across three train-test transfer settings. "OOT" denotes evaluation on an out-of-task dataset not used for training. **ALFW**: ALFWorld, **SciW**: ScienceWorld, **VITA**: VitaBench, **BCP**: BrowseComp-Plus, **GPQA**: GPQA-Diamond. **Avg.** reports average performance, and **Bold** denotes the best result.

Category	Method	Train: ALFW			Train: HotpotQA		Train: DeepScaler		Avg
		ALFW	SciW_{OOT}	VITA_{OOT}	HotpotQA	BCP_{OOT}	AIME	GPQA_{OOT}	
General	CoT	34.94	1.31	26.00	49.00	9.52	19.30	47.47	26.79
	CoT-SC	32.69	0.99	32.02	54.00	11.90	26.32	54.55	30.35
Inference-time	AFlow	33.33	4.09	23.91	46.50	12.50	24.56	33.33	25.46
	AOrchestra	**57.69**	10.54	8.49	49.00	10.71	42.11	59.09	33.95
	Skill-MAS	30.77	4.28	12.51	53.00	7.74	17.54	49.49	25.05
	+ Refiner	44.87	7.73	15.35	59.50	8.33	21.05	59.60	30.92 (+5.87)
Training-time	MAS-GPT	49.36	14.81	8.51	50.50	11.31	19.30	46.97	28.68
	+ Refiner	50.64	18.15	11.40	58.50	11.90	26.32	48.48	32.20 (+3.52)
	ToolOrchestra	36.54	8.57	10.52	41.50	4.76	14.04	41.41	22.48
	MAS-Orchestra	14.42	4.33	19.38	48.00	12.50	21.05	49.49	24.17
	+ Refiner	18.59	9.30	23.54	58.00	13.69	24.56	59.09	29.54 (+5.37)
MAS-Loop	Orchestrator	56.41	21.17	30.49	63.00	13.00	45.61	60.61	41.47
	+ Refiner	56.73	**22.87**	**35.47**	**64.50**	**15.48**	**47.37**	**61.62**	**43.43** (+1.96)



(a) Adversarial penalty λ_{adv} **(b) Refiner reward magnitude η**

Figure 3: Reward-coefficient ablations on HotpotQA and out-of-task BrowseComp-Plus (BCP). Each panel varies one coefficient while holding the other at its default value.

ple, AOrchestra markedly outperforms CoT-SC on ALFWorld (57.69% vs. 32.69%), yet falls well below it on VITA (8.49% vs. 32.02%). The aggregate results reinforce this lack of consistency: among the six unrefined MAS baselines, five underperform CoT-SC on Avg, and four even fall below CoT. This pattern is consistent with prior findings that automatically constructed MAS often fail to yield robust gains over strong single-agent systems because architectural and coordination failures can offset their potential benefits (Jwalapuram et al., 2026; Cemri et al., 2026). In contrast, MAS-Loop delivers the strongest overall performance, outperforming both single-agent baselines and competing MAS methods across nearly all evaluated datasets. The breadth of these gains suggests that execution-grounded feedback and co-evolution provide a more reliable learning signal than one-shot orchestration alone.

Second, the two learned policies remain effective beyond their joint deployment. The Orchestrator alone constructs strong source systems, while coupling it with the Refiner yields further gains. The Refiner is independent and not tied to its co-trained Orchestrator: when applied post hoc to MAS generated by Skill-MAS, MAS-GPT, and MAS-Orchestra, it consistently improves their average performance. Together, these results show that MAS-Loop learns both a capable standalone orchestration policy and a transferable refinement policy that generalizes across MAS design methods.

Ablation Study. Figure 3 examines the sensitivity of MAS-Loop to its two reward coefficients on HotpotQA and the out-of-task BrowseComp-Plus benchmark. We vary one coefficient at a time while fixing the other at its default value, with all remaining training and evaluation settings held constant. Across both sweeps, the strongest results concentrate at **moderate coefficient values**, whereas the endpoints perform less favorably and occasionally exhibit sharp declines. For λ_{adv}, the $\lambda_{\mathrm{adv}} = 0$ endpoint removes the coupled penalty and performs below the default setting, directly showing that source-only optimization does not recover the benefit of coupled learning. For both λ_{adv} and η, introducing moderate adversarial pressure improves both in-task and transfer performance, but an overly large value substantially degrades both, suggesting that excessive pressure can compromise the Orchestrator's source-quality objective. Appendix D further includes a rollout-execution-matched comparison that increases the rollout budget of MAS-Orchestra, a one-step prompted self-reflection control, and a base-model scaling study that replaces Qwen3-8B with Qwen3-14B for both the Orchestrator and Refiner. Together, these ablation studies provide further evidence to support the effectiveness of MAS-Loop.

5 MAS-JUDGECORPUS

Beyond policy learning, the execution-grounded rollouts produced by MAS-Loop provide direct preference supervision for evaluating MAS designs. We organize these paired rollouts into MAS-JudgeCorpus and use them to train a reward model, MAS-JudgeRM, for test-time candidate selection. Our primary setting ranks candidate systems at the orchestration level, while a trajectory-conditioned variant quantifies the additional value of execution evidence.

5.1 MAS-JUDGECORPUS CONSTRUCTION

Raw rollout pairs. MAS-JudgeCorpus is constructed from the source-updated MAS pairs generated during HotpotQA training (10000 tasks). Each rollout executes both systems on the same task, producing a locally comparable outcome pair together with their execution evidence. These rollout-level pairs are the basic units where the final preference examples are selected.

Task-level preference estimation. Because an individual rollout can be affected by stochastic execution, we aggregate source and updated scores across all rollouts of the same task to obtain a more reliable preference direction. We compute $\Delta_{\mathrm{task}} = S_{\mathrm{upd}} - S_{\mathrm{src}}$ and its relative magnitude $g_{\mathrm{rel}} = |\Delta_{\mathrm{task}}| / \min(S_{\mathrm{src}}, S_{\mathrm{upd}})$, then discard ties and tasks with an absolute gap below 2.0 or a relative gap below 25%. Both thresholds must be satisfied. If $\min(S_{\mathrm{src}}, S_{\mathrm{upd}}) = 0$, while the relative gap is undefined, the task is excluded by the Δ_{task} threshold. This retains 2,063 tasks with a clear aggregate preference.

Rollout-level example construction. We then return to the individual rollout pairs within the retained tasks, removing pair-level ties and pairs whose local score direction conflicts with the aggregate task-level preference. Thus, task-level aggregation is used only to stabilize preference assignment, while every final sample remains a rollout-level source-updated comparison: the higher-scoring system is labeled `chosen` and the other `rejected`. After mapping legacy aliases to the seven canonical Refiner operations and retaining at most one rollout pair per task and canonicalized operation signature, we obtain 8,223 trajectory pairs.

5.2 TEST-TIME SELECTION EXPERIMENT

5.2.1 EXPERIMENTAL SETUP

Protocol. We evaluate whether MAS-JudgeCorpus can support test-time MAS selection in a fixed best-of-16 setting. For each held-out task, a shared generator produces 16 candidate MAS designs, and every verification method receives the same pool. Each method ranks these candidates and adopts the answer from the selected MAS as the final prediction. Consequently, performance differences reflect verification quality rather than candidate generation. MAS-JudgeRM is trained on MAS-JudgeCorpus using a Bradley-Terry objective. Its orchestration-level variant observes only the task and candidate design, while its trajectory-level variant additionally receives execution evidence.

Evaluation. We report final-answer accuracy on HotpotQA and the out-of-task BrowseComp-Plus (BCP), together with the method-specific verification tokens. Comparisons include Orchestrator,

Table 2: Test-time MAS selection from a shared best-of-16 candidate pool. Orchestrator (LB) and Oracle (UB) denote the performance lower and upper bounds, respectively.

Category	Method	Verification Level	HotpotQA		BCP_{OOT}	
			Acc	#Token (M)	Acc	#Token (M)
Baseline	Orchestrator (LB)	-	53.00	-	7.74	-
	Majority Voting	-	55.50	-	11.90	-
	Oracle (UB)	-	77.00	-	26.79	-
LLM-as-a-Judge	Kimi-K3	Orchestration	53.00	2.46	14.29	3.02
	Qwen3.8-2.4T-A95B	Orchestration	58.00	2.72	15.48	3.50
	DeepSeek-V4-Flash	Orchestration	51.50	2.47	14.88	3.12
		Trajectory	**60.00**	8.88	20.83	12.47
Reward Model	Skywork-V2-Qwen3-8B	Orchestration	55.00	2.32	10.71	3.21
	Orch-RM	Orchestration	55.50	2.32	10.12	3.21
	MAS-JudgeRM (Ours)	Orchestration	57.50	2.32	11.31	3.21
		Trajectory	**60.00**	**8.91**	**22.62**	**12.77**

Majority Voting, and Oracle, as well as prompted LLM judges and general-purpose or orchestration-specific reward models. The Oracle selects a correct candidate whenever the fixed pool contains one, while all other methods operate without ground-truth labels. We also evaluate trajectory-conditioned variants of DeepSeek-V4-Flash and MAS-JudgeRM to investigate the value of execution evidence.

5.2.2 Results and Analysis

Orchestration-level verification. MAS-JudgeRM consistently outperforms both reward-model baselines on HotpotQA and BCP. However, it does not consistently match the strongest prompted LLM judges: it narrowly trails the best LLM judge on HotpotQA and falls behind all three on BCP. This contrast exposes a limitation of architecture-only verification. A serialized MAS design specifies roles, tools, and topology, but provides little evidence of how the system will actually behave. With only this limited structural evidence, even a compact reward model trained specifically for MAS evaluation cannot match the predictive reasoning capabilities of much larger frontier LLM.

Trajectory-level verification. The comparison changes substantially once execution evidence is available. MAS-JudgeRM reaches 60.00% on HotpotQA, matching the trajectory-conditioned DeepSeek-V4-Flash judge, and achieves 22.62% on BCP, surpassing DeepSeek's 20.83%. These results show that a compact reward model trained on execution-grounded preferences can serve as an effective MAS judge when it can inspect how the candidate actually executes. Although trajectory-level verification requires roughly four times the token overhead, its stronger in-task and transfer performance highlights the importance of execution evidence for reliable MAS evaluation. Orchestration-only judging remains cheaper but is intrinsically more limited.

6 Conclusion

In this work, we propose MAS-Loop, an execution-grounded framework that couples an Orchestrator with a trajectory-conditioned Refiner to close the feedback loop between MAS construction, execution, and revision. Paired re-execution provides complementary learning signals that encourage strong source systems and effective bounded refinements. Experiments show that MAS-Loop improves over one-shot MAS orchestration, while both policies remain useful independently and the Refiner is easy and effective to transfer to other MAS generators. The resulting pair rollouts also form MAS-JudgeCorpus for training MAS-JudgeRM, whose results highlight the importance of execution evidence for reliable MAS evaluation. Together, these findings offer insights into building effective self-evolving MAS by moving beyond one-shot orchestration toward a more targeted and precise loop of system generation, execution-grounded refinement, and policy-level co-evolution across training iterations.

Ethics Statement

This work follows the ICLR Code of Ethics and studies automated MAS construction and refinement in controlled research benchmarks. We do not collect new human-subject, personal, or sensitive data, and our experiments use established research datasets and model-generated execution trajectories. The study focuses on general reasoning, question answering, and benchmarked interactive tasks rather than decisions about individuals or demographic groups, and no component is designed to target or disadvantage any population. We use all datasets in accordance with their intended research settings and licenses, and report our methodology and experimental results with an emphasis on transparency and reproducibility.

Reproducibility Statement

Our reinforcement-learning implementation is built with `verl`. Appendix B provides the complete MAS-Loop procedure, while Appendix C documents benchmark protocols, baselines, model configurations, optimization hyperparameters, and hardware. Appendix F further reports the structured prompts and output formats used by the Orchestrator, Refiner, and LLM judges. We use fixed evaluation settings for controlled comparisons, and the supplementary code supports reproducibility.

LLM Usage

Separate from the language models evaluated in our experiments, LLM assistants are used during manuscript preparation for writing and presentation: improving grammar, clarity, organization, and LaTeX formatting without changing the underlying experimental evidence. All assisted text is manually reviewed by the authors, and the authors remain responsible for the final content.

References

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? *Advances in Neural Information Processing Systems*, 38, 2026.

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in ai: From bounded self-refinement to autonomous research loops. *arXiv preprint arXiv:2607.07663*, 2026a.

Xudong Chen, Yixin Liu, Hua Wei, and Kaize Ding. Lemon: Learning executable multi-agent orchestration via counterfactual reinforcement learning. *arXiv preprint arXiv:2605.14483*, 2026b.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025.

Yu Ge, Linna Xie, Zhong Li, Yu Pei, and Tian Zhang. Spectrum-based failure attribution for multi-agent systems. *Proceedings of the ACM on Software Engineering*, 3(FSE):1888–1909, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jiayi He, Hehai Lin, Qingyun Wang, Yi R Fung, and Heng Ji. Self-correction is more than refinement: A learning framework for visual and language reasoning tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6405–6421, 2025.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Hui Su, and Xunliang Cai. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. In *International Conference on Learning Representations*, volume 2026, pp. 91654–91681, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Weiquan Huang, Zixuan Wang, Hehai Lin, Sudong Wang, Bo Xu, Qian Li, Beier Zhu, Linyi Yang, and Chengwei Qin. Ama: Adaptive memory via multi-agent collaboration. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 3099–3120, 2026.

Prathyusha Jwalapuram, Hehai Lin, Chuyuan Li, Fangkai Jiao, Sudong Wang, Yifei Ming, Zixuan Ke, Chengwei Qin, Giuseppe Carenini, and Shafiq Joty. The illusion of multi-agent advantage. *arXiv preprint arXiv:2606.13003*, 2026.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, et al. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *arXiv preprint arXiv:2504.09037*, 2025a.

Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Ryan Chin, Caiming Xiong, and Shafiq Joty. Mas-zero: Designing multi-agent systems with zero supervision. *arXiv preprint arXiv:2505.14996*, 2025b.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. *arXiv preprint arXiv:2601.14652*, 2026.

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. In *International Conference on Learning Representations*, volume 2026, pp. 52994–53028, 2026.

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, pp. 23142–23150, 2026.

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.

Hehai Lin, Shilei Cao, Sudong Wang, Haotian Wu, Minzhi Li, Linyi Yang, Juepeng Zheng, and Chengwei Qin. Interactive learning for llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6085–6108, 2026a.

Hehai Lin, Yu Yan, Zixuan Wang, Bo Xu, Sudong Wang, Weiquan Huang, Ruochen Zhao, Minzhi Li, and Chengwei Qin. Unified-mas: Universally generating domain-specific nodes for empowering automatic multi-agent systems. *arXiv preprint arXiv:2603.21475*, 2026b.

Hehai Lin, Qi Yang, and Chengwei Qin. Skill-mas: Evolving meta-skill for automatic multi-agent systems. *arXiv preprint arXiv:2606.18837*, 2026c.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*, 2025.

Yixin Liu, Guibin Zhang, Kun Wang, Shiyuan Li, Shirui Pan, and Bo An. Graph-augmented large language model agents: Current progress and future prospects. *IEEE Intelligent Systems*, 41(2): 45–55, 2026.

Xiaowen Ma, Yunpu Ma, Chenyang Lin, Sikuan Yan, Jinhe Bi, Zixuan Cao, Yijun Tian, Volker Tresp, and Hinrich Schuetze. Self-evolving multi-agent systems via textual backpropagation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 9918–9951, 2026.

MAA-Committees. Aime problems and solutions. 2025. URL `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`.

Shihao Qi, Jie Ma, Rui Xing, Wei Guo, Xiao Huang, Zhitao Gao, Jianhao Deng, Jun Liu, Lingling Zhang, Bifan Wei, et al. Beyond individual intelligence: Surveying collaboration, failure attribution, and self-evolution in llm-based multi-agent systems. *arXiv preprint arXiv:2605.14892*, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

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.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Xu Shen, Yixin Liu, Yiwei Dai, Yili Wang, Rui Miao, Yue Tan, Shirui Pan, and Xin Wang. Understanding the information propagation effects of communication topologies in llm-based multi-agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 12358–12372, 2025.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, et al. Toolorchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025.

Sijun Tan, Michael Luo, Justin Wong, Colin Cai, Xiaoxiang Shi, William Yuan Tang, Manan Roongta, Tianjun Zhang, Li Erran Li, Raluca Ada Popa, et al. Deepscaler: Effective rl scaling of reasoning models via iterative context lengthening. 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

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

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

King Yeung Tsang, Zihao Zhao, Vishal Venkataramani, Haizhou Shi, Zixuan Ke, Semih Yavuz, Shafiq Joty, and Hao Wang. Reward modeling for multi-agent orchestration. *arXiv preprint arXiv:2606.13598*, 2026.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022b.

Yanbo Wang, Zixiang Xu, Yue Huang, Xiangqi Wang, Zirui Song, Lang Gao, Chenxi Wang, Robert Tang, Yue Zhao, Arman Cohan, et al. Dyflow: Dynamic workflow framework for agentic reasoning. *Advances in Neural Information Processing Systems*, 38:174148–174181, 2026.

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026a.

Chen Xu, Yicheng Hu, Ruizi Wang, Xinyu Lin, Wenjie Wang, Dongrui Liu, and Fuli Feng. Tacomas: Test-time co-evolution of topology and capability in llm-based multi-agent systems. *arXiv preprint arXiv:2605.09539*, 2026b.

Haoyan Yang, Mario Xerri, Solha Park, Huajian Zhang, Yiyang Feng, Sai Akhil Kogilathota, and Jiawei Zhou. Self-improvement of large language models: A technical overview and future outlook. *arXiv preprint arXiv:2603.25681*, 2026a.

Wei Yang and Jesse Thomason. Learning to deliberate: Meta-policy collaboration for agentic llms with multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29820–29828, 2026.

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. *Advances in Neural Information Processing Systems*, 38:107309–107336, 2026b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*, 2025.

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)*, pp. 15549–15572, 2025.

Chenchen Zhang. Reinforcement learning for llm-based multi-agent systems through orchestration traces. *arXiv preprint arXiv:2605.02801*, 2026.

Fan Zhang, Vireo Zhang, Shengju Qian, Haoxuan Li, Hao Wu, Jinyang Wu, Donghao Zhou, Zhihong Zhu, Zheng Lian, Xin Wang, et al. Orchestra-o1: Omnimodal agent orchestration. *arXiv preprint arXiv:2606.13707*, 2026.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024.

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

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Arik. Multi-agent design: Optimizing agents with better prompts and topologies. In *International Conference on Learning Representations*, volume 2026, pp. 15844–15872, 2026.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Daisy Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, et al. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

A Appendix Overview

This appendix complements the main paper with methodological and experimental details. Appendix B presents the complete MAS-Loop training procedure. Appendix C documents the benchmarks, baselines, and implementation settings, while Appendix D reports the rollout-execution-matched, prompted self-reflection, and base-model scaling ablations. Appendix E illustrates how trajectory-grounded refinement changes an executed MAS, and Appendix F provides the prompt contracts and structured output formats used by each model component.

B MAS-Loop Algorithm

Algorithm 1 summarizes the execution-grounded co-evolution procedure and makes the credit-assignment boundary between the two policies explicit. In each rollout, the Orchestrator constructs a source MAS, whose execution is converted into a bounded diagnostic input for the Refiner. The Refiner either accepts the source system or proposes a patch that is compiled and validated before re-execution. The paired outcomes determine policy-specific rewards: the Orchestrator is trained on source quality and recoverable weaknesses, whereas the Refiner is trained on the realized score change following its action.

Algorithm 1 Execution-grounded co-evolution in MAS-Loop

Require: Labeled task batch $\mathcal{X} = \{(x, y^\star)\}$ and rollouts per task G
Require: Policies $\pi_{\mathrm{O}}, \pi_{\mathrm{R}}$, diagnostic constructor $\mathcal{D}$, patch compiler $\mathcal{C}$, executor $\mathcal{E}$, evaluator r_{term}, and reward coefficients $\lambda_{\mathrm{adv}}, \eta$
Ensure: Co-evolved policies π_{O} and π_{R}

for each training iteration k **do**
  Initialize rollout batches $\mathcal{B}_{\mathrm{O}} \leftarrow \varnothing$ and $\mathcal{B}_{\mathrm{R}} \leftarrow \varnothing$.
  for each task $(x, y^\star) \in \mathcal{X}$ **do**
    for each rollout $m \in \{1, \ldots, G\}$ **do**
      Sample and validate $M_{\mathrm{src}}^{(x,m)} \sim \pi_{\mathrm{O}}(\cdot \mid x)$.
      Execute $(\tau_{\mathrm{src}}^{(x,m)}, y_{\mathrm{src}}^{(x,m)}) \sim \mathcal{E}(M_{\mathrm{src}}^{(x,m)}, x)$.
      Construct the gold-free observation $d_{\mathrm{src}}^{(x,m)} = \mathcal{D}(x, M_{\mathrm{src}}^{(x,m)}, \tau_{\mathrm{src}}^{(x,m)}, y_{\mathrm{src}}^{(x,m)})$.
      Sample $a_{\mathrm{R}}^{(x,m)} = (g^{(x,m)}, p^{(x,m)}) \sim \pi_{\mathrm{R}}(\cdot \mid x, d_{\mathrm{src}}^{(x,m)})$.
      Privately set $s_{\mathrm{src}}^{(x,m)} = r_{\mathrm{term}}(x, y_{\mathrm{src}}^{(x,m)}, y^\star)$ in the trainer after sampling $a_{\mathrm{R}}^{(x,m)}$.
      if $a_{\mathrm{R}}^{(x,m)} = \mathrm{Accept}$ **then**
        Set $M_{\mathrm{upd}}^{(x,m)} \leftarrow M_{\mathrm{src}}^{(x,m)}$.
      else
        Compile and validate $M_{\mathrm{upd}}^{(x,m)} \leftarrow \mathcal{C}(M_{\mathrm{src}}^{(x,m)}, p^{(x,m)})$.
        if $M_{\mathrm{upd}}^{(x,m)}$ is not executable **then**
          Mark the update invalid and set $M_{\mathrm{upd}}^{(x,m)} \leftarrow M_{\mathrm{src}}^{(x,m)}$ for re-execution.
        end if
      end if
      Execute $(\tau_{\mathrm{upd}}^{(x,m)}, y_{\mathrm{upd}}^{(x,m)}) \sim \mathcal{E}(M_{\mathrm{upd}}^{(x,m)}, x)$.
      Set $s_{\mathrm{upd}}^{(x,m)} = r_{\mathrm{term}}(x, y_{\mathrm{upd}}^{(x,m)}, y^\star)$ and $\Delta s^{(x,m)} = s_{\mathrm{upd}}^{(x,m)} - s_{\mathrm{src}}^{(x,m)}$.
      Compute $r_{\mathrm{O}}^{(x,m)}$ and $r_{\mathrm{R}}^{(x,m)}$ using Eqs. equation 11–equation 12.
      Form $z^{(x,m)} = (x, M_{\mathrm{src}}^{(x,m)}, \tau_{\mathrm{src}}^{(x,m)}, s_{\mathrm{src}}^{(x,m)}, a_{\mathrm{R}}^{(x,m)}, M_{\mathrm{upd}}^{(x,m)}, \tau_{\mathrm{upd}}^{(x,m)}, s_{\mathrm{upd}}^{(x,m)})$.
      Append the policy-specific views of $z^{(x,m)}$ and their rewards to $\mathcal{B}_{\mathrm{O}}$ and $\mathcal{B}_{\mathrm{R}}$.
    end for
  end for
  Update π_{O} with GRPO on $\mathcal{B}_{\mathrm{O}}$ and π_{R} with GRPO on $\mathcal{B}_{\mathrm{R}}$.
end for
return Co-evolved policies π_{O} and π_{R}.

Table 3: Benchmark roles and split statistics. OOT indicates that the benchmark is never used to update or select the Orchestra trained on the paired source task.

Task Family	Benchmark	Role	Train size	Test size	Task characteristic
Interactive environments	ALFWorld	Train / In-task test	4032	312	Language-grounded household interaction
	SciWorld	OOT test only	-	300	Interactive scientific environments
	VitaBench	OOT test only	-	100	Multi-domain tool-use interaction
Multi-hop QA	HotpotQA	Train / in-task test	10000	200	Multi-document evidence composition
	BrowseComp-Plus	OOT test only	-	168	Hard information aggregation
General reasoning	DeepScaleR	Train only	10000	-	Complex mathematical reasoning
	AIME	In-task test only	-	57	Competition mathematics (Year 24&25)
	GPQA-Diamond	OOT test only	-	198	Graduate-level science questions

C Experimental Details

C.1 Statistics of Benchmarks

Table 3 summarizes the role of every benchmark in the experiments. ALFWorld, HotpotQA, and DeepScaler provide the three training resources for interaction environment, multi-hop question answering, and general reasoning, respectively. Their corresponding in-task test sets measure whether the learned policies retain performance in the task family used during training. SciWorld, VitaBench, BrowseComp-Plus, and GPQA-Diamond are held-out out-of-task (OOT) benchmarks and therefore test transfer to a related but distinct task distribution.

Interaction environments. ALFWorld aligns abstract text environments with embodied household tasks, allowing an agent to learn and execute the same high-level task through language actions (Shridhar et al., 2020). Its tasks require long-horizon planning, object manipulation, and adaptation to different household layouts, making it a natural training and in-task evaluation environment for interactive agents. **SciWorld** requires the agent to inspect objects, perform experiments through sequential actions, and infer the scientific outcome (Wang et al., 2022a). **VitaBench** evaluates multi-turn tool-using agents in real-world service scenarios, including food delivery, in-store consumption, and online travel (He et al., 2026). It exposes agents to changing user intentions, interdependent tool calls, and dynamic service states, thereby testing practical decision-making and recovery over extended interactions.

Knowledge-intensive QA. HotpotQA is a Wikipedia-based multi-hop question-answering dataset whose questions require combining evidence from multiple documents (Yang et al., 2018). **BrowseComp-Plus** evaluates deep-research agents over a fixed and curated document corpus, providing a reproducible alternative to evaluation against the continuously changing open web (Chen et al., 2025). Its questions require iterative retrieval, source comparison, and evidence synthesis, which makes it suitable for measuring transfer from standard multi-hop QA to more complex deep-research tasks.

General Reasoning. The **DeepScaler** training set consists of competition-style mathematical problems collected for reinforcement-learning-based reasoning training (Tan et al., 2025). These problems demand multi-step derivations but retain verifiable final answers, allowing them to provide training tasks with reliable outcome-based rewards. **AIME** is a complex mathematics competition that typically requires substantial algebraic, geometric, combinatorial, or number-theoretic reasoning (MAA-Committees, 2025). **GPQA-Diamond** comprises expert-written graduate-level questions

in biology, physics, and chemistry, which evaluate scientific reasoning and domain knowledge outside the mathematical distribution used for training (Rein et al., 2023).

C.2 BASELINES

C.2.1 MAS-LOOP

The main experiment compares MAS-Loop with three types of MAS methods. For each training-time MAS baseline, we evaluate both its original Orchestrator and the same Orchestrator augmented with our Refiner interface where applicable. This controlled adaptation separates improvements due to the underlying orchestration policy from improvements due to execution-grounded refinement.

General inference methods. CoT elicits intermediate reasoning steps before producing the final answer (Wei et al., 2022). **CoT-SC** samples multiple reasoning traces and selects the answer with the highest consistency (Wang et al., 2022b). These methods do not construct an explicit MAS and therefore provide single-agent inference baselines.

Inference-time MAS methods. AFlow searches over code-represented agentic workflows using Monte Carlo Tree search (Zhang et al., 2025). **AOrchestra** creates task-specific sub-agents by configuring their instructions, context, tools, and models at inference time (Ruan et al., 2026). **Skill-MAS** records orchestration experience in an evolvable meta-skill and uses this experience to construct task-specific MAS workflows (Lin et al., 2026c). These methods optimize or instantiate the workflow during inference but do not train a reusable orchestration policy under our task-specific training protocol.

Training-time MAS methods. MAS-GPT trains an LLM to generate an executable, query-adaptive MAS in a single inference (Ye et al., 2025). **ToolOrchestra** learns to coordinate models and tools through reinforcement learning under effectiveness and efficiency objectives (Su et al., 2025). **MAS-Orchestra** learns holistic function-calling-based orchestration for generating a complete MAS (Ke et al., 2026). These methods are the closest comparison group because they learn reusable coordination behavior during training. However, their original formulations do not learn a trajectory-conditioned policy that directly revises and re-executes the generated source MAS.

C.2.2 MAS-JUDGECORPUS

The MAS-JudgeCorpus experiment evaluates test-time MAS selection under a fixed best-of-16 protocol. Every verification method receives the same candidate pool and ranks its candidates without changing the generator; the answer associated with the selected MAS becomes the final prediction. Orchestration-level methods observe the task and MAS design, whereas trajectory-level methods additionally receive execution evidence.

General baselines. Orchestrator uses an answer produced without test-time candidate verification and therefore serves as the performance lower bound. **Majority Voting** aggregates the answers of the 16 candidates without learning an explicit quality score. **Oracle** accesses their ground-truth correctness and returns a correct answer whenever the pool contains one, providing the performance upper bound for the fixed candidate set.

LLM-as-a-Judge baselines. DeepSeek-V4-Flash (Xu et al., 2026a), **Kimi-K3** (Team et al., 2026), and **Qwen3.8-2.4T-A95B** rank candidate MAS designs through prompted inference rather than a separately trained reward model. All three are evaluated at the orchestration level, and DeepSeek-V4-Flash is additionally evaluated with trajectory evidence to measure the benefit of observing execution behavior.

Reward-model baselines. Skywork-V2-Qwen3-8B provides a general-purpose reward-model reference (Liu et al., 2025), while **Orch-RM** is trained specifically to score MAS organizations (Tsang et al., 2026). We compare them with **MAS-JudgeRM**, trained on the execution-grounded source–updated preferences in MAS-JudgeCorpus. MAS-JudgeRM is evaluated with both orchestration- and trajectory-level inputs, allowing us to compare specialized preference supervision against existing reward models and to isolate the additional value of execution evidence.

Table 4: Key hyperparameters used in the formal MAS-Loop training runs.

Hyperparameter	**Value**
Training batch size	256
Rollout group size	16
Orchestrator learning rate	5×10^{-7}
Refiner learning rate	1×10^{-6}
PPO mini-batch size	256
PPO micro-batch size per GPU	8
Maximum prompt length	15,000
Maximum response length	8,192
Sub-agent maximum concurrent requests	1,024
Training precision	bf16
Gradient checkpointing	Enabled
PPO clip ratio	0.2
Actor KL coefficient	0.001
Training epochs	1
Refinement rounds	1
Hardware	8×H100
RL framework	verl

C.3 Implementation Details

Table 4 summarizes key hyperparameters used in our reinforcement learning. The Orchestrator and Refiner are initialized from Qwen3-8B (Team, 2025), and updated from separate rollout records using their respective rewards. Their outputs follow the structured interfaces in Appendix F. Qwen3-30B-A3B-Instruct-2507 (Team, 2025) remains frozen while executing sub-agent nodes. The generated graphs and patches are canonicalized and validated before execution, and malformed or unresolved outputs are handled as invalid actions. Following Tsang et al. (2026), MAS-JudgeRM is trained by LoRA fine-tuning the Qwen3-8B-based Skywork-Reward-V2 checkpoint with the Bradley-Terry objective, using LoRA rank 32, scaling factor 64, and dropout 0.05.

D Additional Ablation Experiments

We conduct three controlled studies to determine whether MAS-Loop's gains can be explained by additional MAS executions or an untrained reflection pass, and whether the framework generalizes across base-model scales.

D.1 Rollout-Execution-Matched Training-Time MAS

To isolate the effect of additional MAS executions from execution-grounded refinement, we increase the rollout group size of MAS-Orchestra (Ke et al., 2026) from 16 to 32. MAS-Loop executes 16 source MAS and 16 updated MAS per task, so this control matches the number of executed MAS rollouts.

D.2 One-Step Prompted Self-Reflection

We add a non-learned reflection control to distinguish the contribution of the trained Refiner from the generic benefit of an additional trajectory-conditioned inference pass. Given the task, source MAS, final prediction, and the same gold-free execution trajectory available to MAS-Loop, the Orchestrator is prompted once (i.e., using the self-correction prompt (He et al., 2025)) to diagnose the execution and emit one revision under the same bounded operation interface. The revised MAS is compiled and re-executed once. This control therefore matches the source-revision-re-execution structure but introduces no separately trained Refiner or coupled reward.

Table 5: Ablation results for rollout-execution matching, one-step prompted self-reflection, and base-model scaling.

Training variant	HotpotQA	BCP_{OOT}
MAS-Loop	64.50	15.48
Rollout-execution-matched MAS-Orchestra	54.00	12.50
Orchestrator + one-step self-reflection	59.50	11.90
MAS-Loop with Qwen3-14B	**66.00**	**16.07**

D.3 Base-Model Scaling: Qwen3-14B

We replace Qwen3-8B with Qwen3-14B for both the Orchestrator and Refiner while keeping the task data, policy interfaces, reward definitions, rollout group size, frozen sub-agent backbone, executor, and evaluation protocol unchanged. This comparison tests whether MAS-Loop generalizes to a larger base model and can effectively convert increased policy capacity into improved in-task and out-of-task performance.

Analysis. Table 5 shows that the default MAS-Loop substantially outperforms rollout-execution-matched MAS-Orchestra on both HotpotQA and BCP. Thus, increasing the number of independently generated and executed MAS rollouts alone does not recover the benefit of trajectory-conditioned refinement. The self-correction results remain below MAS-Loop, isolating the benefit of learning a dedicated Refiner rather than merely prompting for an additional reflection pass. Replacing the trainable policies with Qwen3-14B further improves MAS-Loop on both benchmarks, demonstrating that the framework generalizes across base-model scales and benefits from stronger policy backbones.

E Case Study

We show one case from BrowseComp-Plus and compare the MAS produced by Orchestrator and Refiner. The Orchestrator returns **Northgate Market**, whereas the Refiner rewrote the role of **agent_0** to require explicit confirmation of the candidate's presence in both Mexico and California. This constraint discouraged premature selection based on the salient California event clue alone and promoted a more complete evaluation of the two-location condition. The revised evidence flow consequently preserved **Churrería El Moro** as the candidate passed to downstream aggregation, thus yielding the correct answer.

Case Overview

Benchmark. BrowseComp-plus, sample id: `48b2de7c90a8f88e36dbd1fb0aae3ff21c7fa5a0be90e46d94219563be58e281`.
Question. "I am looking for a specialty food shop that has at least one location in both Mexico and California as of December 1, 2023. One of the locations in Mexico is a 7.8-kilometer walk from a McDonald's located at Anillo Perif. 137, Polanco, Polanco I Secc, Miguel Hidalgo, 11510 Ciudad de México, CDMX, Mexico as of December 1, 2023. The location in California is located within a market that held a Thanksgiving Floral Arrangement Class event in November 2023 as of December 1, 2023. What is the name of the specialty food shop?"
Label. Churrería El Moro.
Orchestrator's Answer. Northgate Market. ×
Refiner's Answer. Churrería El Moro. ✓

Orchestrator — `agent_0`: candidate retrieval

Role. Find specialty food shops with locations in Mexico and California.
Retrieval. The search retrieves the McDonald's–Polanco page (document `86536`), a Mexico shopping-mall listing (`7849`), and the Northgate Market event page (`49319`), whose title and preview explicitly mention a "Thanksgiving Floral Arrangement Class" on November 20, 2023.

Reasoning. The node prioritizes the strongest single lexical match, the California event at Northgate Market, and treats the missing Mexico-side confirmation as non-blocking. Its output explicitly states that there is "no explicit mention of a location in Mexico for Northgate Market," but then relaxes the conjunctive question and returns Northgate Market.
Response.

> The specialty food shop that meets the requirements is **Northgate Market.** While there is no explicit mention of a location in Mexico for Northgate Market, the question appears to focus on identifying the California shop associated with the event.

Refiner — `agent_0`: constrained evidence retrieval

Role. Find specialty food shops with locations in Mexico and California, ensuring both locations are explicitly confirmed.
Retrieval. Key retrieved evidence includes "Churrería El Moro - Wikipedia" (document `62716`), whose preview identifies Churrería El Moro as a Mexico City shop with branches in Polanco, and the Northgate Market event page (`49319`), which supplies the California event clue. The retrieved set grows from 8 to 9 unique documents.
Reasoning. The revised role turns the task into an explicit conjunction: a candidate must satisfy both the Mexico and California conditions before being passed downstream. The node therefore keeps the Mexico-side evidence active instead of accepting Northgate Market solely because its event page is a direct match.
Response.

> The specialty food shop that meets the criteria of having a location in both Mexico and California is **Churrería El Moro.** The original location is in Mexico City, with branches including Polanco.

Difference

Orchestrator's Observed retrieval. Five search rounds; 8 unique documents. Retrieved material contains strong California-event evidence around **Northgate Market**, including a market page associated with a Thanksgiving Floral Arrangement Class. It also contains Mexico-side evidence around **Churrería El Moro**, including the title "Churrería El Moro" and Mexico City / Polanco location information.
Localized failure. The node notices that Northgate Market satisfies one highly salient clue, then relaxes the question's conjunction. Its own output explicitly admits that Mexico-side confirmation for Northgate Market is missing, yet still returns Northgate Market. This is not pure retrieval absence: the trace contains a competing candidate with Mexico-side evidence. Failure is candidate selection under an unenforced conjunction.
Refiner's Observed retrieval change. Search remains five rounds, but unique documents increase from 8 to 9. More importantly, the updated node output now selects **Churrería El Moro** and explicitly organizes reasoning around the Mexico-side location requirement. The gold string appears in the updated node output and in the retrieved preview signal, whereas it is absent from the original node output and preview signal.
Advantage of the constraint. The added phrase "ensuring both locations are explicitly confirmed" acts as a local stopping rule and rejection rule:

1. it prevents acceptance after finding only the California event;
2. it forces the node to keep the Mexico-side clue active during candidate comparison;
3. it makes "no explicit Mexico evidence" a reason to reject Northgate Market rather than a reason to weaken the question.

F PROMPT

The prompts below define the model-facing interfaces used by the Orchestrator, Refiner, and LLM Judge. Executor and patch-compiler behavior is implemented by fixed rule functions rather than by an additional model prompt.

MAS-Loop Prompt for Orchestrator

System_prompt:
Design a multi-agent system, not the final answer. Output ONLY one valid XML <mas> document. Use <agents><agent ...><role>...</role><tools>...</tools></agent></agents> and <edges><edge ... /></edges>. Set id, budget, and execution_mode on every agent. execution_mode must be llm_only or multi_turn_search. Use 2–8 agents. Put topology, aggregation, max_cost, and max_iterations on the <mas> root. Escape XML special characters in text. No JSON, markdown, <think>, or private reasoning. Stop immediately after </mas>.
User_prompt:
Task: {task}
Constraints:{constraints}
Generate the initial MAS as one <mas> XML document. Always close every <agent> with </agent>, and never add a closing tag after a self-closing tag. Example structure:

```
<mas topology="dag" aggregation="agent_2" max_cost="3" max_iterations="3">
    <agents>
        <agent id="agent_0" budget="1" execution_mode="multi_turn_search">
            <role>Acquire task-specific evidence.</role>
            <tools><tool>bm25</tool></tools>
        </agent>
        <agent id="agent_1" budget="1" execution_mode="llm_only">
            <role>Reconcile upstream evidence.</role>
            <tools></tools>
        </agent>
        <agent id="agent_2" budget="1" execution_mode="llm_only">
            <role>Return the exact final answer.</role>
            <tools></tools>
        </agent>
    </agents>
    <edges>
        <edge from="agent_0" to="agent_1" type="data_flow" />
        <edge from="agent_1" to="agent_2" type="data_flow" />
    </edges>
</mas>
```

User_prompt:
Task: {task}
Output one XML document shaped like the <mas> example above.

Figure 4: MAS-Loop XML Orchestrator prompt.

MAS-Loop Prompt for Refiner (Part 1/2)

System_prompt:
You are Refiner, a dataset-neutral MAS diagnosis-and-edit policy. Learn and emit the complete improvement process, not only a patch. Inspect the supplied source MAS, its real execution trajectory, and bounded node/edge diagnostics. First localize the earliest evidence-supported failure on the final-output dependency path. Then model how that failure propagated and state a falsifiable counterfactual. Finally design the smallest coherent repair plan and emit exactly one explicit operation: accept or one bounded patch operation. The input never contains the gold answer, source correctness, terminal reward, or any diagnostic derived from them; infer whether a revision is warranted only from observable execution evidence. Do not use benchmark names, hidden gold answers, or dataset-specific answer rules as diagnostic evidence. Do not merely blame a generic planner or final synthesizer. Never emit multiple operations; the bounded patch contains exactly one edit. Allowed decisions: accept, add_agent, rewrite_role, add_edge, remove_edge, adjust_reasoning_budget, change_tool. The add_agent operation creates a task-solving agent and atomically connects its output to the current aggregation agent without changing the aggregation agent. accept must be the only operation when chosen. Use only observed evidence; never invent execution events or copy long transcripts. Output exactly one XML <refiner> document containing stage1_localization, stage2_causal_model, stage3_repair_plan, and patch. Escape XML special characters. No JSON, markdown, <think>, or private reasoning. Stop immediately after </refiner>.

Figure 5: MAS-Loop XML Refiner prompt (Part 1/2).

MAS-Loop Prompt for Refiner (Part 2/2)

User_prompt:
Produce one grounded three-stage improvement decision as one learned reasoning trace, not three model calls or hard filters. source_status must be exactly one of: actionable_failure, preserve; target_type must be agent, edge, or mas. Use this XML element/attribute contract. Replace every ellipsis/example, fill all elements with task-specific content, and never emit an empty <evidence><item></item> body:

```
<refiner>
    <stage1_localization source_status="..." target_type="..." target_id="..." failure_type="...">
        <evidence>
            <item agent_id="..." execution_index="..." field="..." location="..." round="...">
                Concrete task-specific observation from this exact trace value; replace this example.
            </item>
        </evidence>
    </stage1_localization>
    <stage2_causal_model>
        <causal_chain>
            <step></step>
        </causal_chain>
        <causal_explanation></causal_explanation>
        <counterfactual></counterfactual>
        <downstream_impact></downstream_impact>
    </stage2_causal_model>
    <stage3_repair_plan confidence="...">
        <repair_goal></repair_goal>
        <expected_effect></expected_effect>
        <risk></risk>
    </stage3_repair_plan>
    <patch>
        <operation op="..." />
    </patch>
</refiner>
```

Each <evidence><item> must have attributes agent_id, execution_index, field, and location; add round when the location references a retrieval round. location must be copied exactly from one trace_refs value in node_diagnostic_state. The item text must state the concrete observed signal at that location. For an edge use agent_id='from->to' and execution_index='-1'. For a MAS/outcome observation use agent_id='mas' and execution_index='-1'. Do not copy field descriptions as observations.
Existing agent IDs: [{known_agent_ids}]. Use these exact IDs when possible.
Patch operation XML forms:

```
<operation op="accept" />
<operation op="rewrite_role" agent_id="ID"><role>NEW_ROLE</role></operation>
<operation op="adjust_reasoning_budget" agent_id="ID" budget="1" />
<operation op="change_tool" agent_id="ID" old_tool="OPTIONAL" new_tool="TOOL" />
<operation op="add_agent" id="NEW_ID" budget="1"><role>ROLE</role></operation>
<operation op="add_edge" from="ID" to="OTHER_ID" type="data_flow" />
<operation op="remove_edge" from="ID" to="OTHER_ID" type="data_flow" />
```

Emit exactly one operation. The parser retains legacy aliases only for audit/rejection; do not use them. budget is 1–4.
Input state is JSON data embedded only as input context; your OUTPUT must be XML:

```
{
    "task": {task},
    "constraints_and_capabilities": {sanitized_constraints},
    "source_mas": {source_mas},
    "node_diagnostic_state": {node_diagnostic_state},
    "optional_sampling_focus": {optional_repair_focus}
}
```

Figure 6: MAS-Loop XML Refiner prompt (Part 2/2).

LLM-as-a-judge Prompts (VitaBench)

System Prompt: System Information `{env_info}` User Complete Instruction `{user_instruction}`

Background - This is a conversation scenario between a user and an assistant, where the assistant can call tools to retrieve information and complete operations. Tool return results will start with "tool"

- You need to evaluate whether the user instruction has been completed. The user's complete instruction has been broken down into several scoring points (rubrics), and you only need to judge whether each scoring point is satisfied
- Due to the large number of conversation turns, we use sliding window evaluation, where each window shows 10 conversation turns with 2 overlapping turns between windows, but rubric status will be preserved across windows
- You are evaluating window `{window_idx}` (out of `{total_windows}` windows total)
- `<window_content>` contains the conversation content for the current window
- `<current_rubrics>` contains the current status of all scoring points (true means satisfied, false means not satisfied, all scoring points have an initial status of false)

Task - Update the scoring point rubric status based on the conversation content in the current window

- You can update the status from false to true, if and only if the assistant completed the goal in this window
- You can also update true back to false, if and only if the assistant overturned a previous correct conclusion in this window (but note: if the user's needs themselves changed, such as actively canceling after placing an order, the original evaluation of the ordering requirement should not be overturned)
- You can refer to the "User Complete Instruction" to understand the progress of the current conversation window and avoid unnecessary modifications

Important Notes

- Important: All evaluations are based on whether the assistant's responses and tool call requests complete the goals in the rubrics. User expressions in the conversation are only considered as guidance for the assistant and do not directly affect evaluation standards. Everything is based on the rubric fields!
- Important: Query tool return results are only visible to the assistant and do not represent content recommended by the assistant to users, so they do not directly affect evaluation results either. Everything must be based on the assistant's responses to users after obtaining information! Also note that the Assistant cannot fabricate Tool return results!
- Important: For order-related rubrics (involving order details that must generate orders), you must confirm whether the assistant actually completed the ordering operation. The assistant may mistakenly believe they completed the ordering operation when in fact the tool call failed; or situations where the user states they "can place the order themselves," etc., should all be considered as not meeting the requirements
- For rubrics involving order details such as product quantity or delivery time, the original rubric requirements must be strictly met (no deviation in product quantities, delivery must not be later than the expected time). User compromise behavior does not affect evaluation results (for example, when a user states "fewer items is okay", "I have no objections to the order content" or "later delivery is fine" etc.). These situations should still be considered as not meeting the requirements
- For rubrics involving text content matching of addresses or order notes, apply the functional equivalence principle: as long as the actual content can achieve the same function (such as roughly locating the delivery location or conveying the customer's main needs), it is considered to meet the requirements even if the expression is not completely consistent or lacks some details
- If the current window does not involve a certain rubric, keep its original status unchanged; if the current window involves a rubric but cannot fully determine it, key information can be recorded in the justification for future judgment
- In the justification, record key information related to the current rubric and its corresponding round [x] in an appending manner. If status modifications occur, record the reason using concise language. If the status is not modified, restate the previous reason

Format Requirements

- Your response should be a JSON object containing the following fields:
  - `rubric_idx`: Unique identifier for the rubric
  - `rubric`: Restatement of the rubric
  - `justification`: Explanation of status changes
  - `meetExpectation`: Updated status (true or false)

User Prompt: `<window_content>` `{window_content}` `</window_content>` `<current_rubrics>` `{current_rubrics_str}` `</current_rubrics>`

Figure 7: LLM-as-a-judge prompts used in VitaBench.