# METAAGENT-X : BREAKING THE CEILING OF AUTOMATIC MULTI-AGENT SYSTEMS VIA END-TO-END REINFORCEMENT LEARNING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Automatic multi-agent systems aim to instantiate agent workflows without relying on manually designed or fixed orchestration. However, existing automatic MAS approaches remain only partially adaptive: they either perform training-free test-time search or optimize the meta-level designer while keeping downstream execution agents frozen, which creating a frozen-executor ceiling and leaving the end-to-end training of self-designing and self-executing agentic models unexplored. To address this, we introduce **MetaAgent-X** , an end-to-end reinforcement learning framework that jointly optimizes automatic MAS design and execution. MetaAgent-X enables script-based MAS generation, execution rollout collection, and credit assignment for both designer and executor trajectories. To support stable and scalable optimization, we propose Executor Designer Hierarchical Rollout and Stagewise Co-evolution to improve training stability and expose the dynamics of designer-executor co-evolution. MetaAgent-X consistently outperforms existing automatic MAS baselines, improving the six-benchmark average by 6.2 and 7.5 points over the strongest automatic MAS baseline on Qwen3-8B and Qwen3-4B respectively, and by 11.2 and 12.8 points over a single-agent baseline; it also outperforms single-agent best-of-$n$ and fixed multi-agent workflows evaluated under a comparable test-time token budget. Comprehensive ablations show that both designer and executor improve throughout training, and that effective automatic MAS learning follows a stagewise co-evolution process. These results establish end-to-end trainable automatic MAS as a practical paradigm for building self-designing and self-executing agentic models.

## 1 INTRODUCTION

Multi-agent systems (MAS) have demonstrated clear advantages over single-agent approaches across a wide range of domains, including medical decision-making (Kim et al., 2024; Zhou et al., 2025), scientific discovery (Su et al., 2024; Ghafarollahi & Buehler, 2024), financial trading (Xiao et al., 2024), software engineering (Yu et al., 2025; Hong et al., 2023; Chen et al., 2024), and hardware design (Zhao et al., 2024; Ho et al., 2025). Rather than relying on manually specified or fixed workflows, recent work has increasingly turned to meta-agents as a paradigm for automatically designing and instantiating the multi-agent system flow best suited to each task, enabling more adaptive orchestration and execution of MAS (Gao et al., 2025; Ye et al., 2025; Dang et al., 2025; Nielsen et al., 2025; Zhang et al., 2025b).

Meanwhile, as agentic reinforcement learning and self-evolving paradigms have emerged as promising pathways to transform large language models into interactive, continuously improving decision-makers (Wang et al., 2025c; Cheng et al., 2025; Li et al., 2025b; Zhao et al., 2026; Zhang et al., 2026; Xia et al., 2025; Chen et al., 2025b; Fu et al., 2025), recent automatic MAS begin to embrace these paradigms, their transition remains incomplete. Current approaches typically restrict adaptation to non-training test-time search, or only optimize the MAS designer, while freezing downstream execution agents (Ye et al., 2025; Gao et al., 2025; Dang et al., 2025; Nielsen et al., 2025; Wang et al., 2025a). Yet, end-to-end training of self-designing and self-executing auto-MAS remains unexplored, resulting in two fundamental limitations: 1) **Parameter-level disjunction.** Existing methods couple the designer and executor only through prompt-level interactions at inference time,



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.

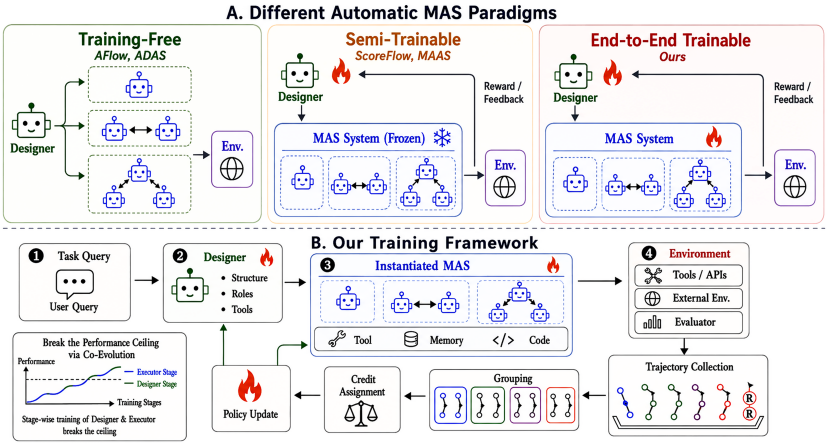


Figure 1: **From Partial Adaptation to End-to-End Trainable Automatic MAS. A.** Comparison of three automatic MAS paradigms. **B.** Overview of our training framework.

without optimization signals that update the underlying policy based on downstream execution outcomes. As a result, a frozen executor imposes a hard ceiling on the meta-designer, while the designer cannot induce specialized execution behaviors from its counterpart. 2) **Vague co-evolution dynamics.** The dynamics by which designer and executor could co-evolve under joint training, and where each role's improvement remains unclear in practice and in understanding the mechanism.

As shown in Figure 1 (A), existing automatic MAS approaches remain partially adaptive: they either search over MAS structures at test time or optimize only the designer while freezing the execution system. To overcome these limitations, we introduce **MetaAgent-X** , an end-to-end framework to train agentic models that can self-design and self-execute MAS. Figure 1(B) gives an overview of MetaAgent-X , where task-conditioned auto MAS designs are instantiated, executed, grouped, and collected for role-aware policy updates. To address the first limitation, MetaAgent-X facilitates script-based MAS generation, rollout collection, and precise credit assignment for both the designer and the executor. To address the second limitation, the framework incorporates diverse evolving mechanisms, such as hierarchical rollouts and stage-wise optimization, allowing us to isolate the critical decision factors that drive auto-MAS co-evolution.

Because both roles are trainable and their rollouts are separable, the framework also makes designer-executor co-evolution directly observable: we can ask which role a given improvement came from, and how the two adapt to each other over training.

This paper makes the following contributions:

1. We propose **MetaAgent-X** , an end-to-end training framework for automatic MAS, which explicitly optimizes designer and executor agents together.

2. We introduce two mechanisms for stable and scalable meta agent optimization: (i) **Executor Designer Hierarchical Rollout**, which enables structured rollout generation and accurate credit assignment and (ii) **Stagewise Co-evolution**, which supports decoupled and scalable designer executor learning.

3. We demonstrate that MetaAgent-X achieves consistent gains across diverse math and code benchmarks, surpassing both single agent and automatic MAS baselines, with a 6.2-point average gain over the strongest automatic MAS baseline on Qwen3-8B

4. We conduct comprehensive ablation studies to examine the internal mechanisms of meta-agent co-evolution. Our analysis shows that (1) both the designer and the executor are optimized throughout training across tasks and domains, and (2) such effective co-evolution follows a stagewise process in which the two components benefit from decoupled optimizations.

# 2 RELATED WORK

## 2.1 META AGENTS FOR AUTOMATIC MULTI-AGENT SYSTEMS

LLM-based MAS improve complex problem solving by decomposing tasks into specialized roles, structured interactions, and coordination protocols (Qian et al., 2024; Hong et al., 2024; Wu et al., 2023). Beyond manually designed workflows, recent work introduces meta-agents that map a query into roles, prompts, communication patterns, or execution flows, after which the instantiated system interacts with the environment to produce the final outcome (Ye et al., 2025; Gao et al., 2025; Dang et al., 2025; Nielsen et al., 2025; Zhang et al., 2025b).

As shown in Fig. 1, existing automatic MAS methods fall into two partial adaptation regimes. *Training-free* adaptation searches over prompts, roles, workflows, or agent organizations at test time without updating model parameters (Zhang et al., 2025b; Dang et al., 2025). *Semi-trainable* adaptation optimizes a meta-level designer while keeping downstream executors fixed: MAS-GPT (Ye et al., 2025) generates query-adaptive MAS designs, FlowReasoner (Gao et al., 2025) learns query-level reasoning flows, orchestration-based controllers handle dynamic coordination (Nielsen et al., 2025), and MAS$^2$ (Wang et al., 2025a) trains the designer with reinforcement learning while still using API-based executors. Such frozen executors impose a ceiling on final performance and prevent designer-executor co-adaptation. Chain-of-Agents takes a related end-to-end direction by training an Agent Foundation Model through multi-agent distillation and agentic reinforcement learning (Li et al., 2025a), but largely optimizes the agent system as a unified behavior and treats MAS as a simple chain of thought without context management. In contrast, our work studies the end-to-end trainable regime, where automatic MAS evolves both how agent systems are designed and how instantiated agents execute them, making designer-executor co-evolution explicit and analyzable.

## 2.2 AGENT SYSTEM SELF EVOLUTION AND MULTI-AGENT TRAINING

In parallel with meta-agent based automatic MAS, agentic reinforcement learning and self evolution have emerged as promising paradigms for improving LLM agents through interaction, environment feedback, and iterative experience collection (Wang et al., 2025c; Cheng et al., 2025; Li et al., 2025b; Zhao et al., 2026; Zhang et al., 2026; Xia et al., 2025; Chen et al., 2025b; Fu et al., 2025). Within the multi-agent setting, recent methods such as MAPoRL (Park et al., 2025), AT-GRPO (Zhao et al., 2026), Dr. MAS (Feng et al., 2026), MAE (Chen et al., 2025a), and MARFT (Liao et al., 2025) study credit assignment, coordination, communication, and training stability, but improve collaboration under fixed or predefined multi-agent structures. The agent organization itself is treated as given, rather than as a learned object that should be generated, evaluated, and improved together with execution behavior.

Our work differs from these self evolution and agent foundation model approaches in both objective and analysis. Instead of assuming a fixed MAS structure or optimizing an agent system as an undifferentiated whole, we explicitly formulate automatic MAS learning as a designer-executor co-evolution problem. This enables us to break the frozen-executor performance ceiling while also studying the internal mechanism of automatic MAS co-evolution.

# 3 METHOD

## 3.1 END TO END ONLINE META AGENT RL PIPELINE

Figure 2 shows our reinforcement learning pipeline. Given a task query $q$, the MetaAgent first uses a Designer policy $\pi^{\mathcal{D}}_{\vartheta_{\mathcal{D}}}$ to generate a task specific multi agent system, and then uses an Executor policy $\pi^{\mathcal{E}}_{\vartheta_{\mathcal{E}}}$ to run the instantiated system in an external environment. We denote the full trainable parameter set by $\vartheta = \{\vartheta_{\mathcal{D}}, \vartheta_{\mathcal{E}}\}$. This notation covers both policy sharing and policy splitting. In the shared policy setting, $\vartheta_{\mathcal{D}} = \vartheta_{\mathcal{E}} = \theta$; in the split policy setting, $\vartheta_{\mathcal{D}}$ and $\vartheta_{\mathcal{E}}$ are optimized as separate parameter sets. The learning problem is therefore a coupled online reinforcement learning problem:

$$d \sim \pi^{\mathcal{D}}_{\vartheta_{\mathcal{D}}}(\cdot \mid q), \qquad e \sim \pi^{\mathcal{E}}_{\vartheta_{\mathcal{E}}}(\cdot \mid q, d), \qquad R = R(q, d, e), \tag{1}$$

where $d$ denotes the generated system design, $e$ denotes the execution trajectory, and $R$ is the environment feedback returned after execution. The central challenge is that design and execution are

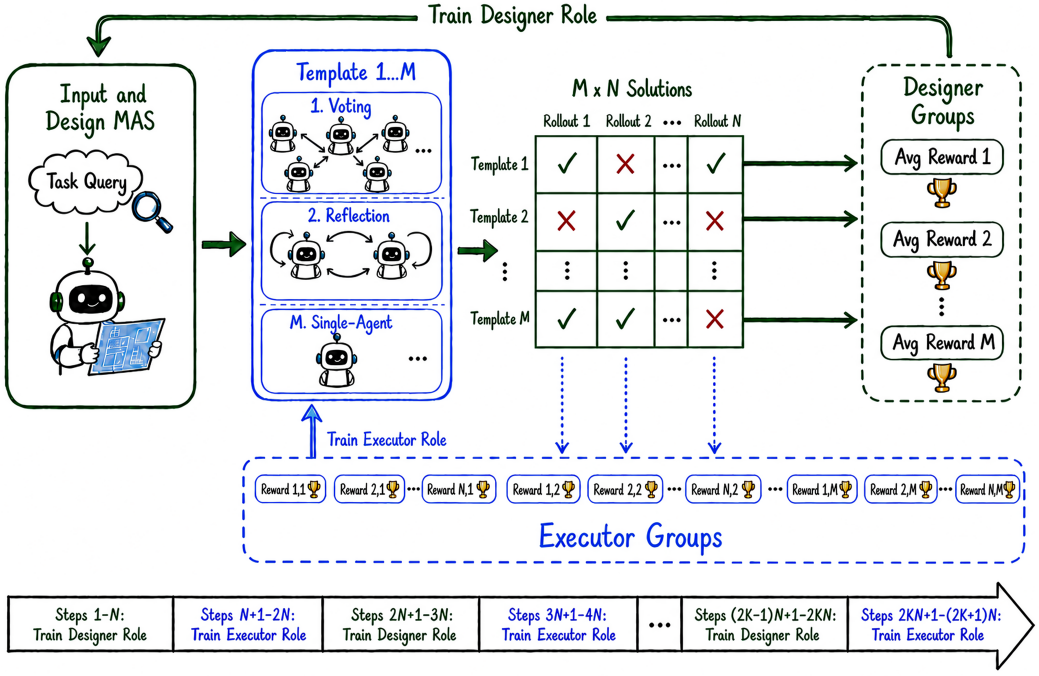


Figure 2: Overview of the end to end online MetaAgent-X pipeline. The Designer first generate a task-specific multi agent system, then the Executor run the instantiated MAS in the environment. The collected trajectories and rewards are labeled by role and optimized with GRPO.

interdependent; their performance is coupled. Thus, the training pipeline must support online system construction, batched environment execution, trajectory collection, and role aware credit assignment within a unified RL framework.

**Online system construction.** To support compositional system design, we build a training framework contains predefined coordination structures, agent templates, and tool interfaces. For each query, the Designer composes these building blocks into a customized multi agent system by generating lightweight Python scripts. These scripts specify the agent roles, interaction protocol, tool usage pattern, and execution control flow. After a design is instantiated, the Executor runs the generated workflow in the target environment. Our framework supports batched rollout execution across multiple queries and sampled designs. For each rollout, the system records the rollouts, environment observations, tool calls, and the outcome-based rewards (detailed in Appendix B).

**GRPO objective.** We optimize the role policies with Group Relative Policy Optimization(GRPO). For each role $r \in \{\mathcal{D}, \mathcal{E}\}$, let $\mathcal{G}^r$ denote the corresponding GRPO group, and let $\hat{A}_i^r$ be the normalized role specific advantage for trajectory $i$. Let $\vartheta_r$ denote the parameters used by role $r$. The clipped policy objective for role $r$ is

$$\mathcal{L}_r(\vartheta_r) = -\frac{1}{|\mathcal{G}^r|} \sum_{i \in \mathcal{G}^r} \min\left(\rho_i^r(\vartheta_r)\hat{A}_i^r, \text{clip}\left(\rho_i^r(\vartheta_r), 1-\epsilon, 1+\epsilon\right)\hat{A}_i^r\right), \tag{2}$$

where

$$\rho_i^r(\vartheta_r) = \frac{\pi_{\vartheta_r}^r(o_i \mid c_i)}{\pi_{\vartheta_{r,\text{old}}}^r(o_i \mid c_i)}. \tag{3}$$

Here $c_i$ is the context of trajectory $i$, $o_i$ is the generated output tokens, and $\pi_{\vartheta_{r,\text{old}}}^r$ is the role specific behavior policy used for rollout collection. The role specific advantages $\hat{A}^{\mathcal{D}}$ and $\hat{A}^{\mathcal{E}}$ are computed using the hierarchical credit assignment scheme in Section 3.2.

Further, because the Designer and Executor are optimized through coupled online feedback, we introduce a stagewise training schedule that provides a relatively stable environment for optimizing both roles. We discuss the details in Section 3.3.

### 3.2 Hierarchical Credit Assignment via Tree-Structured Rollout

A central challenge in training end-to-end automatic MAS with RL is **credit assignment**: when a multi-agent system succeeds or fails at a task, is the outcome attributable to the quality of the Designer's plan or the competence of the Executor's actions? Standard single-level rollout conflates these two sources of variation, producing entangled reward signals that destabilize training. We address this through a tree-structured rollout scheme that decomposes credit across roles.

**Bi-level Tree-Structured Rollout.** For each training question $q$, we construct a two-level sampling tree. At the first level, the Designer $\pi^{\mathcal{D}}_{\vartheta_{\mathcal{D}}}$ generates $M$ independent multi-agent system designs $\{d_1, d_2, \ldots, d_M\}$, each specifying a distinct agent topology, role assignment, and coordination protocol. At the second level, for each design $d_i$, the Executor $\pi^{\mathcal{E}}_{\vartheta_{\mathcal{E}}}$ carries out $N$ independent execution rollouts $\{e_{i,1}, e_{i,2}, \ldots, e_{i,N}\}$. This yields an $M \times N$ evaluation matrix per question, where entry $(i, j)$ corresponds to design $d_i$ executed by rollout $e_{i,j}$, with outcome reward $R(e_{i,j}, d_i)$.

**Decomposed Advantage Estimation.** The tree structure enables us to compute *separate* advantage estimates for each role via distinct grouping strategies within the GRPO framework.

*Designer advantage.* To isolate the effect of design quality from execution-level stochasticity, we aggregate over the execution level. For each design $d_i$ under question $q$, we define the design-level reward as the mean execution outcome:

$$\bar{R}^{\mathcal{D}}_i = \frac{1}{N} \sum_{j=1}^{N} R(e_{i,j}, d_i). \tag{4}$$

The advantage for design $d_i$ is then computed by comparing against all $M$ designs for the same question:

$$\hat{A}^{\mathcal{D}}_i = \frac{\bar{R}^{\mathcal{D}}_i - \mu^{\mathcal{D}}_q}{\sigma^{\mathcal{D}}_q + \epsilon}, \quad \text{where} \quad \mu^{\mathcal{D}}_q = \frac{1}{M} \sum_{k=1}^{M} \bar{R}^{\mathcal{D}}_k, \quad \sigma^{\mathcal{D}}_q = \text{std}(\{\bar{R}^{\mathcal{D}}_k\}_{k=1}^{M}). \tag{5}$$

By averaging over $N$ executions, the stochasticity of individual rollouts is smoothed out, yielding a reward signal that reflects the intrinsic quality of the design itself.

*Executor advantage.* For each execution rollout $e_{i,j}$, the Executor produces a set of agent trajectories, denoted by $\mathcal{T}_{i,j}$. We use the outcome reward of the rollout, $R(e_{i,j}, d_i)$, as the reward for all trajectories in $\mathcal{T}_{i,j}$. To compute the Executor advantage, we collect all executor trajectories for the same question into a GRPO group:

$$\mathcal{G}^{\mathcal{E}}_q = \{\tau \mid \tau \in \mathcal{T}_{i,j},\ i \in [M],\ j \in [N]\}. \tag{6}$$

The advantage of each trajectory is then normalized at the question level:

$$\hat{A}^{\mathcal{E}}(\tau) = \frac{R(e_{i,j}, d_i) - \mu^{\mathcal{E}}_q}{\sigma^{\mathcal{E}}_q + \epsilon}, \quad \tau \in \mathcal{T}_{i,j}, \tag{7}$$

where $\mu^{\mathcal{E}}_q$ and $\sigma^{\mathcal{E}}_q$ denote the mean and standard deviation of the rollout rewards associated with trajectories in $\mathcal{G}^{\mathcal{E}}_q$. Compared with single-level rollout normalization, question-level normalization compares executor trajectories generated under both the same and different designs, thereby providing a more stable training signal for the executor.

### 3.3 Stagewise Executor-Designer Co-evolution

The hierarchical rollout in Section 3.2 provides decomposed reward signals for the Designer ($\mathcal{D}$) and Executor ($\mathcal{E}$) roles. However, since the two roles' rewards are mutually conditioned, a fundamental optimization challenge arises: *how should we update $\pi_\theta$ when $\mathcal{D}$ and $\mathcal{E}$ serve as each other's environment?*

The Designer and Executor form a tightly coupled system where each role is the other's environment: the Executor acts within the MAS structure emitted by the Designer, while the Designer's reward is decided by the capability of the Executor. Formally, the return is a nested expectation:

$$J(\theta) = \mathbb{E}_{d\sim\pi^{\mathcal{D}}_{\vartheta_{\mathcal{D}}}}\left[\mathbb{E}_{e\sim\pi^{\mathcal{E}}_{\vartheta_{\mathcal{E}}}(\cdot|d)}\left[R(e,d)\right]\right], \tag{8}$$

Inspired by multi-agent RL studies on non-stationarity and sequential optimization (Hernandez-Leal et al., 2019; Yu et al., 2022; Nekoei et al., 2023), we introduce a stagewise schedule that alternates which role provides the trajectories for policy-gradient updates. At training step $t$, we select the active role by fixed-length phases of $K$ steps:

$$(\alpha^{(t)}_{\mathcal{D}},\ \alpha^{(t)}_{\mathcal{E}}) = \begin{cases} (0,\ 1), & \lfloor t/K \rfloor \bmod 2 = 0 \quad \text{Executor stage,} \\ (1,\ 0), & \lfloor t/K \rfloor \bmod 2 = 1 \quad \text{Designer stage.} \end{cases} \tag{9}$$

Only trajectories from the active role contribute to the gradient, while the shared parameters $\vartheta$ are updated continuously. This isolates each phase to one reward distribution and reduces gradient interference between role-specific objectives.

The two stages form a co-evolutionary loop. Executor stages improve the ability to solve tasks under the current design distribution, producing more reliable execution outcomes. Designers then use these lower-noise returns to learn structures that better exploit the improved Executor. As a result, the effective reward distribution becomes non-stationary and the two role-specific objectives can produce noisy or conflicting updates.

# 4 Experiment

## 4.1 Experiments Setup

**Models and Compute.** We train and evaluate Qwen3 (Yang & the Qwen Team, 2025) at the 4B and 8B parameter scales in no-thinking mode. All experiments are conducted on a single node equipped with eight H200 GPUs. Unless otherwise specified, both the maximum prompt length and maximum response length are set to 8192 tokens. Main experiments use the shared-policy setting, in which designer and executor share one backbone.

**Training Procedure.** Our training proceeds in two stages: a supervised fine-tuning (SFT) cold start followed by reinforcement learning (RL) co-evolution. During the SFT stage, we initialize the policy by distilling trajectories from DeepSeek-V3.2 prompted with diverse workflow templates (further details regarding the cold start are provided in Appendix A). In the RL stage, we adopt stagewise designer-executor co-evolution with a stage length of $K = 30$. For each query, the Designer generates $M = 4$ candidate MAS, and each MAS is executed $N = 4$ times. At each stage, only the active role is updated with a learning rate of $5 \times 10^{-6}$, while gradients from the inactive role are masked.

**Training Datasets.** For the SFT cold start, the dataset consists of 3K Designer examples and 8K Executor examples, filtered from correct DeepSeek-V3.2 generations. For the RL stage, we train on a mixture of math and code data to encourage cross-task generalization. With an RL batch size of 8, half of each batch is sampled from Polaris-Dataset-53K (An et al., 2025), and the remaining half is sampled from the APPS introductory subset (Hendrycks et al., 2021) and CodeContests (DeepMind, 2024).

**Baselines.** Single-agent baselines include direct prompting and GRPO, both using the same Qwen3-4B or 8B backbone as our method. GRPO is trained on the same math and code mixture. Search-based MAS optimization baselines include AFlow (Zhang et al., 2024) and ADAS (Hu et al., 2025). For AFlow, we use the official best-searched workflows for math and code. For ADAS, we use the official best-searched math agent and run the search protocol for code when no official code agent is released. RL-based MAS optimization baselines include ScoreFlow (Wang et al., 2025b), MaAS (Zhang et al., 2025a) and AFM (Li et al., 2025a); for AFM we evaluate AFM-Coder-7B, the released checkpoint closest in scale to our setting, under the official code-agent evaluation framework. All baselines follow the default settings in their original papers or released code. Details are given in Appendix D.

Table 1: **Qwen3 8B** results on coding and math benchmarks. Parentheses denote gain over the Single Agent baseline. Best and second best results are highlighted per benchmark. AIME24/AIME25 contain 30 problems each, so one problem corresponds to 3.33 points; AIME results are averaged over three evaluation runs.

| | | Code | | | Math | | | |
|---|---|---|---|---|---|---|---|---|
| **Training Paradigm** | **Method** | LiveCodeBench | APPS | CodeContests | AIME24 | AIME25 | OlympiadBench | **Avg** |
| **Single Agent** | SA | 22.80 (+0.00) | 30.20 (+0.00) | 15.75 (+0.00) | 18.30 (+0.00) | 20.90 (+0.00) | 55.00 (+0.00) | 27.16 (+0.00) |
| | SA + GRPO | 25.70 (+2.90) | 37.00 (+6.80) | 12.12 (-3.63) | 18.30 (+0.00) | 26.67 (+5.77) | 54.80 (-0.20) | 29.10 (+1.94) |
| **Search-based Auto MAS** | AFlow | 28.60 (+5.80) | 27.40 (-2.80) | 15.80 (+0.05) | 16.67 (-1.63) | 20.83 (-0.07) | 35.31 (-19.69) | 24.10 (-3.06) |
| | ADAS | 20.00 (-2.80) | 27.00 (-3.20) | 12.20 (-3.55) | 13.30 (-5.00) | 16.70 (-4.20) | 32.90 (-22.10) | 20.35 (-6.81) |
| **RL-based Auto MAS** | ScoreFlow | 25.90 (+3.10) | 26.50 (-3.70) | 13.30 (-2.45) | 28.90 (+10.60) | 20.00 (-0.90) | 51.30 (-3.70) | 27.65 (+0.49) |
| | MaAS | 24.29 (+1.49) | 30.00 (-0.20) | 15.15 (-0.60) | 45.80 (+27.50) | 29.20 (+8.30) | 48.90 (-6.10) | 32.22 (+5.06) |
| | AFM-Coder | 29.10 (+6.30) | 28.00 (-2.20) | 21.20 (+5.45) | 12.00 (-6.30) | 8.00 (-12.90) | 21.80 (-33.20) | 20.02 (-7.14) |
| | **MetaAgent-X $_{\text{SFT}}$** | **36.00 (+13.20)** | **32.00 (+1.80)** | **13.00 (-2.75)** | **33.33 (+15.03)** | **20.00 (-0.90)** | **59.00 (+4.00)** | **32.22 (+5.06)** |
| | **MetaAgent-X $_{\text{RL}}$** | **41.00 (+18.20)** | **38.00 (+7.80)** | **17.00 (+1.25)** | **40.00 (+21.70)** | **33.33 (+12.43)** | **61.00 (+6.00)** | **38.39 (+11.23)** |

**Benchmarks.** For math, we use AIME24/AIME25 (Mathematical Association of America & AoPS Community, 2024; 2025) and OLYMPIADBENCH (He et al., 2024), scored by a numeric verifier. AIME24/25 hold only 30 problems each, so one problem is 3.33 points; we average each AIME result over three evaluation runs and base our conclusions on the six-benchmark average rather than on individual AIME cells. For code, we use APPS (Hendrycks et al., 2021), LiveCodeBench-v6 (Jain et al., 2024), and CodeContests (DeepMind, 2024), scored by executing solutions against the benchmark-provided tests.

Table 2: **Qwen3 4B** results on coding and math benchmarks. Parentheses denote gain over the Single Agent baseline. Best and second best results are highlighted per benchmark. AIME results are averaged over three evaluation runs.

| | | Code | | | Math | | | |
|---|---|---|---|---|---|---|---|---|
| **Training Paradigm** | **Method** | LiveCodeBench | APPS | CodeContests | AIME24 | AIME25 | OlympiadBench | **Avg** |
| **Single Agent** | SA | 13.80 (+0.00) | 27.40 (+0.00) | 14.80 (+0.00) | 20.00 (+0.00) | 19.10 (+0.00) | 33.20 (+0.00) | 21.38 (+0.00) |
| | SA + GRPO | 16.70 (+2.90) | 35.60 (+8.20) | 18.60 (+3.80) | 29.10 (+9.10) | 26.67 (+7.57) | 47.10 (+13.90) | 28.96 (+7.58) |
| **Search-based Auto MAS** | AFlow | 28.00 (+14.20) | 23.20 (-4.20) | 13.33 (-1.47) | 16.67 (-3.33) | 13.33 (-5.77) | 40.59 (+7.39) | 22.52 (+1.14) |
| | ADAS | 16.00 (+2.20) | 28.00 (+0.60) | 12.20 (-2.60) | 10.00 (-10.00) | 23.00 (+3.90) | 32.80 (-0.40) | 20.33 (-1.05) |
| **RL-based Auto MAS** | ScoreFlow | 23.36 (+9.56) | 24.50 (-2.90) | 11.92 (-2.88) | 26.40 (+6.40) | 16.70 (-2.40) | 57.00 (+23.80) | 26.65 (+5.27) |
| | MaAS | 24.29 (+10.49) | 23.75 (-3.65) | 9.10 (-5.70) | 16.70 (-3.30) | 25.00 (+5.90) | 45.20 (+12.00) | 24.01 (+2.62) |
| | **MetaAgent-X $_{\text{SFT}}$** | **32.00 (+18.20)** | **32.00 (+4.60)** | **6.00 (-8.80)** | **30.00 (+10.00)** | **16.70 (-2.40)** | **57.00 (+23.80)** | **28.95 (+7.57)** |
| | **MetaAgent-X $_{\text{RL}}$** | **36.00 (+22.20)** | **36.70 (+9.30)** | **14.20 (-0.60)** | **33.33 (+13.33)** | **26.67 (+7.57)** | **58.20 (+25.00)** | **34.18 (+12.80)** |

### 4.2 MAIN RESULTS

Tables 1 and 2 report our cold-start and RL-trained models on six math and code benchmarks. MetaAgent-X $_{\text{RL}}$ averages $38.39\%$ on Qwen3-8B and $34.18\%$ on Qwen3-4B, gains of $+11.23$ and $+12.80$ points over the Single Agent baseline and of $+6.17$ and $+5.23$ points over MetaAgent-X $_{\text{SFT}}$ under an identical backbone and protocol.

Search-based Auto MAS baselines degrade when instantiated with these backbones: ADAS falls $-6.81$ points to $20.35\%$ on 8B, and AFlow reaches only $22.52\%$ on 4B. This suggests search-based methods are sensitive to the capability of the underlying base model, since a searched workflow is not adapted to the executor that must run it.

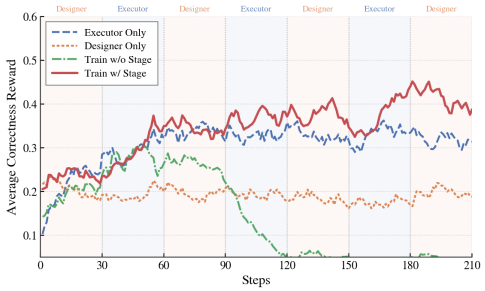


Figure 3: Training-reward dynamics ablations of the proposed stagewise co-evolution.

Against methods that train only a meta-agent or workflow selector (MaAS, ScoreFlow), MetaAgent-X$_{\text{RL}}$ exceeds the strongest such baseline by $+6.17$ points on 8B ($38.39\%$ vs. $32.22\%$); we attribute this gap to making the executor trainable rather than frozen. AFM-Coder is strong on CodeContests but collapses on math ($-33.20$ on OlympiadBench), whereas MetaAgent-X holds up across both domains.

**Robustness of the comparison.** The $M \times N$ rollout is training-time only: at inference MetaAgent-X emits and runs a single workflow. Matched on token budget, four-round reflection (39.4/28.8 Math/Code) and a fixed solver–critic (39.7/29.3) spend as many or more tokens than MetaAgent-X (44.8/32.0) yet fall short, and a continued-SFT control on the RL-stage data gains only $+1.6/+1.5$ over MetaAgent-X$_{\text{SFT}}$ against $+7.3/+5.0$ for RL (Appendices E, F).

### 4.3 Ablation Studies

We ablate the two mechanisms behind MetaAgent-X — hierarchical rollout and stagewise co-evolution — and the choice of sharing parameters across roles.

Table 3: Ablation studies on Qwen3-8B. Math and Code are averages over the three benchmarks in each domain (AIME24/AIME25/OlympiadBench and LiveCodeBench/APPS/CodeContests). Rollout variants are matched on total budget ($M \times N = 16$ trajectories per query).

| Factor | Variant | Math | Code |
|---|---|---|---|
| Hierarchical rollout | $M$=4, $N$=4 (ours) | **44.8%** | **32.0%** |
| | $M$=8, $N$=1 (flat) | 41.6% | 29.8% |
| Co-evolution schedule | Stagewise (ours) | **44.8%** | **32.0%** |
| | Executor-only | 39.6% | 30.7% |
| | Designer-only | 38.6% | 27.5% |
| | Coupled | 36.7% | 25.2% |
| Policy parameterization | Shared (ours) | **44.8%** | **32.0%** |
| | Separate | 39.0% | 29.4% |

**Executor-Designer Hierarchical Rollout.** We compare our $M$=4, $N$=4 setting against a flatter $M$=8, $N$=1 rollout that samples more designs but executes each once, holding the total budget at 16 trajectories per query. The hierarchical setting is stronger on both aggregates ($+3.2$ Math, $+2.2$ Code). Since both consume the same budget, the gap reflects how trajectories are allocated rather than how much exploration is done: repeated executions of each design give a more reliable estimate of its downstream utility, and hence more stable credit assignment.

**Does Stagewise Co-evolution Help?** We compare the proposed schedule on Qwen3-8B against coupled, executor-only, and designer-only training (Table 3). Coupled training, in which trajectories from both roles update the shared policy simultaneously, improves quickly at first but later collapses, degenerating into repeated tokens until the length limit (Figure 3); it is the weakest variant on both aggregates. Designer-only training gains little, indicating that optimizing designs over a frozen executor does not reliably improve the system, while executor-only training improves rapidly then saturates against the ceiling imposed by a fixed design policy. Stagewise training instead produces

a staircase-shaped curve, with reward stable during designer phases and rising sharply after each switch to an executor phase, and is best on both aggregates.

**Shared Policy vs. Separate Policy.** We compare a single policy shared by both roles, distinguished only by role-conditioned prompts, against separate per-role policies. The shared policy is better on both aggregates ($+5.8$ Math, $+2.6$ Code). What this isolates is not two static role prompts but training-time trajectory decomposition: Designer trajectories are supervised by the MAS they emit, Executor trajectories by execution outcomes inside it, and sharing parameters lets each role's signal act as an inductive bias for the other.

### 4.4 Analysis

RL changes both the *structures emitted by the Designer* and the *quality of Executor behavior*. The trained designer routes structures by task, sending over 70% of the harder AIME problems to reflection while favouring a single agent on OlympiadBench and APPS. Appendix G reports the full distribution along with stage-length sensitivity, per-query design diversity, and end-to-end case studies.

**Which role brings the improvement?** Attributing the SFT-to-RL gain on AIME25, it splits roughly evenly: about half of the newly solved problems use the *same* structure the SFT designer assigned, indicating execution-side gains, and about half coincide with a structure switch. On 30 problems this split is indicative rather than precise, but both mechanisms are active; Appendix G gives one case of each.

## 5 Discussions

We introduced MetaAgent-X , the first end-to-end reinforcement learning framework that jointly optimizes the designer and executor of an automatic multi-agent system through hierarchical rollouts and stagewise co-evolution. Across six math and code benchmarks and two model scales, MetaAgent-X raises the benchmark average by $6.2$ and $7.5$ points over the strongest automatic MAS baseline, holds up against compute-matched single-agent and fixed-workflow baselines, and exposes the dynamics through which designer and executor mutually improve. This suggests a path toward foundation models with native multi-agent capabilities, where MAS becomes an internal mechanism for reasoning and context management rather than an external harness. Two limitations bound these results: compute constraints preclude an exhaustive scaling study, and the framework relies on a cold start to initialize workflow generation, since base models at this scale cannot reliably emit valid workflows.

### AI use statement

We used generative AI tools only to assist with *implementing* parts of our training and evaluation codebase. All such code was read line by line, executed, and validated against the reported experiments by at least two authors before any number in this paper was produced. We did *not* use generative AI tools to formulate the research problem, to design the MetaAgent-X framework or its hierarchical rollout and stagewise co-evolution mechanisms, to design or critique our experimental protocol, or to interpret our results; these are the authors' own contributions. The remaining required-disclosure tasks are not applicable to this work: we introduce no synthetic dataset, state and prove no formal theorems, and perform no translation of source material, and all benchmarks we evaluate on are existing public datasets used without AI-assisted cleaning or reformatting.

We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

### Reproducibility statement

We have made the following efforts to support reproduction of our results. The MetaAgent-X training pipeline, including script-based MAS generation, the executor–designer hierarchical rollout, and the stagewise co-evolution schedule, is specified in Section 3.1–3.3. Our reward decomposition and its coefficients are given in Appendix B, and the cold-start construction procedure, including the

workflow-template bank and the filtering criterion applied to the distilled trajectories, in Appendix A. The training hyperparameters, stage length, rollout counts, and learning rate are stated in Section 4.1. Baseline configurations, including the search and training budgets we used to match each baseline to our own rollout budget, are documented per method in Appendix D. All experiments use the publicly available Qwen3-4B and Qwen3-8B backbones and six public math and code benchmarks, evaluated on the standard splits distributed with those benchmarks, so that the setup can be reconstructed from the description above without additional artifacts.

## REFERENCES

Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, and Xuanjing Huang. Polaris: A post-training recipe for scaling reinforcement learning on advanced reasoning models, 2025. URL https://hkunlp.github.io/blog/2025/Polaris.

Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, Jie Wang, Xiao Cheng, Guangtai Liang, Yuchi Ma, Pan Bian, Tao Xie, and Qianxiang Wang. Coder: Issue resolving with multi-agent and task graphs. *arXiv preprint arXiv:2406.01304*, 2024.

Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: Llm self-improve through co-evolution, 2025a. URL https://arxiv.org/abs/2510.23595.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason Weston, and Dat Huynh. Scaling agent learning via experience synthesis. *arXiv preprint arXiv:2511.03773*, 2025b.

Mingyue Cheng, Jie Ouyang, Shuo Yu, Ruiran Yan, Yucong Luo, Zirui Liu, Daoyu Wang, Qi Liu, and Enhong Chen. Agent r1: Training powerful llm agents with end to end reinforcement learning. *arXiv preprint arXiv:2511.14460*, 2025.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, Xuantang Xiong, Lei Han, Zhiyuan Liu, and Maosong Sun. Multi-agent collaboration via evolving orchestration. *arXiv preprint arXiv:2505.19591*, 2025.

DeepMind. Codecontests. https://github.com/google-deepmind/code_contests, 2024. GitHub repository; archived Dec 6, 2024.

Lang Feng, Longtao Zheng, Shuo He, Fuxiang Zhang, and Bo An. Dr. mas: Stable reinforcement learning for multi-agent llm systems. *arXiv preprint arXiv:2602.08847*, 2026.

Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. Evolver: Self evolving llm agents through an experience driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.

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.

Alireza Ghafarollahi and Markus J. Buehler. Sciagents: Automating scientific discovery through multi-agent intelligent graph reasoning, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *ACL*, 2024. URL https://arxiv.org/abs/2402.14008.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Andy Zou, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *arXiv:2105.09938*, 2021. URL https://arxiv.org/abs/2105.09938.

Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity, 2019. URL https://arxiv.org/abs/1707.09183.

Chia-Tung Ho, Jing Gong, Yunsheng Bai, Chenhui Deng, Haoxing Ren, and Brucek Khailany. Marco: Configurable graph-based task solving and multi-ai agents framework for hardware design. *arXiv preprint arXiv:2504.01962*, 2025.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. International Conference on Learning Representations, ICLR, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2025. URL https://arxiv.org/abs/2408.08435.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974.

Yubin Kim, Chanwoo Park, Hyewon Jeong, Yik Siu Chan, Xuhai Xu, Daniel McDuff, Hyeonhoon Lee, Marzyeh Ghassemi, Cynthia Breazeal, and Hae Won Park. Mdagents: An adaptive collaboration of llms for medical decision-making. *arXiv preprint arXiv:2404.15155*, 2024.

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, Hongxuan Lu, Tianrui Qin, Chenghao Zhu, Yi Yao, Shuying Fan, Xiaowan Li, Tiannan Wang, Pai Liu, King Zhu, He Zhu, Dingfeng Shi, Piaohang Wang, Yeyi Guan, Xiangru Tang, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Jiaheng Liu, Ge Zhang, and Wangchunshu Zhou. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic rl, 2025a.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use, 2025b. URL https://arxiv.org/abs/2510.05592.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. Marft: Multi-agent reinforcement fine-tuning, 2025. URL https://arxiv.org/abs/2504.16129.

Mathematical Association of America & AoPS Community. Aime 2024 problems (aops wiki). https://artofproblemsolving.com/wiki/index.php/2024_AIME_I & https://artofproblemsolving.com/wiki/index.php/2024_AIME_II_Problems, 2024. Accessed 2025-09-11.

Mathematical Association of America & AoPS Community. Aime 2025 problems (aops wiki). https://artofproblemsolving.com/wiki/index.php/2025_AIME_I_Problems & https://artofproblemsolving.com/wiki/index.php/2025_AIME_II_Problems, 2025. Accessed 2025-09-11.

Hadi Nekoei, Akilesh Badrinaaraayanan, Amit Sinha, Mohammad Amini, Janarthanan Rajendran, Aditya Mahajan, and Sarath Chandar. Dealing with non-stationarity in decentralized cooperative multi-agent deep reinforcement learning via multi-timescale learning, 2023. URL https://arxiv.org/abs/2302.02792.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to orchestrate agents in natural language with the conductor. *arXiv preprint arXiv:2512.04388*, 2025.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning. *arXiv preprint arXiv:2502.18439*, 2025.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development. In *ACL 2024*, 2024.

Haoyang Su, Renqi Chen, Shixiang Tang, Zhenfei Yin, Xinzhe Zheng, Jinzhe Li, Biqing Qi, Qi Wu, Hui Li, Wanli Ouyang, Philip Torr, Bowen Zhou, and Nanqing Dong. Many heads are better than one: Improved scientific idea generation by a llm-based multi-agent system. *arXiv preprint arXiv:2410.09403*, 2024.

Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. Mas$^2$: Self-generative, self-configuring, self-rectifying multi-agent systems, 2025a. URL https://arxiv.org/abs/2509.24323.

Yinjie Wang, Ling Yang, Guohao Li, Mengdi Wang, and Bryon Aragam. Scoreflow: Mastering llm agent workflows via score-based preference optimization, 2025b. URL https://arxiv.org/abs/2502.04306.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025c.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self evolving agents from zero data via tool integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. Tradingagents: Multi-agents llm financial trading framework. *arXiv preprint arXiv:2412.20138*, 2024.

An Yang and the Qwen Team. Qwen3 technical report. *arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

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.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games, 2022. URL https://arxiv.org/abs/2103.01955.

Zhongming Yu, Hejia Zhang, Yujie Zhao, Hanxian Huang, Matrix Yao, Ke Ding, and Jishen Zhao. Orcaloca: An llm agent framework for software issue localization. *arXiv preprint arXiv:2502.00350*, 2025.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet, 2025a. URL https://arxiv.org/abs/2502.04180.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. Metaagent: Automatically constructing multi-agent systems based on finite state machines, 2025b. URL https://arxiv.org/abs/2507.22606.

Yaolun Zhang, Ruohui Wang, Jiahao Wang, Yepeng Tang, Xuanyu Zheng, Haonan Duan, Hao Lu, Hanming Deng, and Lewei Lu. Eva: Efficient reinforcement learning for end-to-end video agent, 2026. URL https://arxiv.org/abs/2603.22918.

Yujie Zhao, Hejia Zhang, Hanxian Huang, Zhongming Yu, and Jishen Zhao. Mage: A multi-agent engine for automated rtl code generation, 2024. URL https://arxiv.org/abs/2412.07822.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-mas: Multi-agent reinforcement learning for collaborative llms, 2026. URL https://arxiv.org/abs/2510.11062.

Yucheng Zhou, Lingran Song, and Jianbing Shen. Mam: Modular multi-agent framework for multi-modal medical diagnosis via role-specialized collaboration. *arXiv preprint arXiv:2506.19835*, 2025.

# APPENDIX

## A COLD START DETAILS

We cold-start the policy by distilling both sides of the generated multi-agent system: the *Designer*, which writes an executable workflow, and the *Executors*, which solve the problem inside the generated workflow. The Designer is prompted with a bank of workflow templates implemented in our codebase. The template bank contains **single-agent**, **ensemble-voting**, **solver-critic reflection**, **solver-tester**, and etc. Each template specifies both the workflow topology and the role-level prompts; for example, ensemble templates instantiate strategy-diverse solvers and a judge, while reflection templates instantiate a solver and a critic/verifier loop.

For each training question, we sample in-context examples from this template bank and ask DeepSeek-V3.2 to synthesize a complete workflow program. The data-generation pipeline samples multiple workflow designs per question and logs both the Designer conversation and all Executor conversations produced when the workflow is run. We then retain trajectories whose final answer is judged correct, yielding 3K Designer examples and 8K Executor examples for supervised cold start. This gives the model an initial ability to map a problem to an appropriate multi-agent program and to act as the specialized agents inside that program.

Before training our smaller policy, we also evaluate whether the same workflow prompting strategy helps a strong proprietary model. As shown in Table 4, prompting DeepSeek to solve through a generated multi-agent workflow improves AIME 2024 accuracy from 63.3% to 66.7%, which suggests that the workflow interface is not only a crutch for weaker models: even when the underlying model is already strong, explicit role decomposition and verification can recover additional correct solutions. This motivates using DeepSeek-V3.2 as the teacher for cold-starting both the Designer and Executor behaviors before reinforcement learning.

| Method | Accuracy |
|---|---|
| Direct DeepSeek | 63.3% |
| DeepSeek prompted with MAS workflow | 66.7% |

Table 4: Prompting-only comparison on AIME 2024 using DeepSeek as the underlying model. The MAS prompt asks DeepSeek to first synthesize and run a multi-agent workflow rather than directly answer with a single response.

## B REWARD DESIGN

The outcome-based reward $R(e_{i,j}, d_i)$ is composed of two terms:

$$R(e_{i,j}, d_i) = R_{\text{correct}}(e_{i,j}) + \lambda \cdot R_{\text{format}}(e_{i,j}), \tag{10}$$

where $R_{\text{correct}}$ evaluates the functional correctness of the final solution via environment feedback, and $R_{\text{format}}$ incentivizes structured agent behavior. We set $\lambda = 0.4$.

The correctness reward $R_{\text{correct}} \in \{0, 1\}$ is a strict binary signal determined by the specific domain environment:

- **Math Verification:** The final parsed answer is evaluated against the ground-truth solution. To account for algebraically equivalent expressions, we utilize a symbolic math engine (e.g., SymPy) to robustly verify the correctness of the final mathematical output.
- **Code Execution:** The final generated program is compiled and executed against the dataset's hidden unit tests. The reward $R_{\text{correct}} = 1$ is assigned if and only if the code successfully passes all unit tests without exceeding the environment's execution time or memory constraints.

The format reward $R_{\text{format}}$ acts as a regularizer and consists of two components:

1. **Solution formatting.** The final agent must produce its answer within a standardized output format, ensuring that the solution is reliably parseable for automated evaluation.

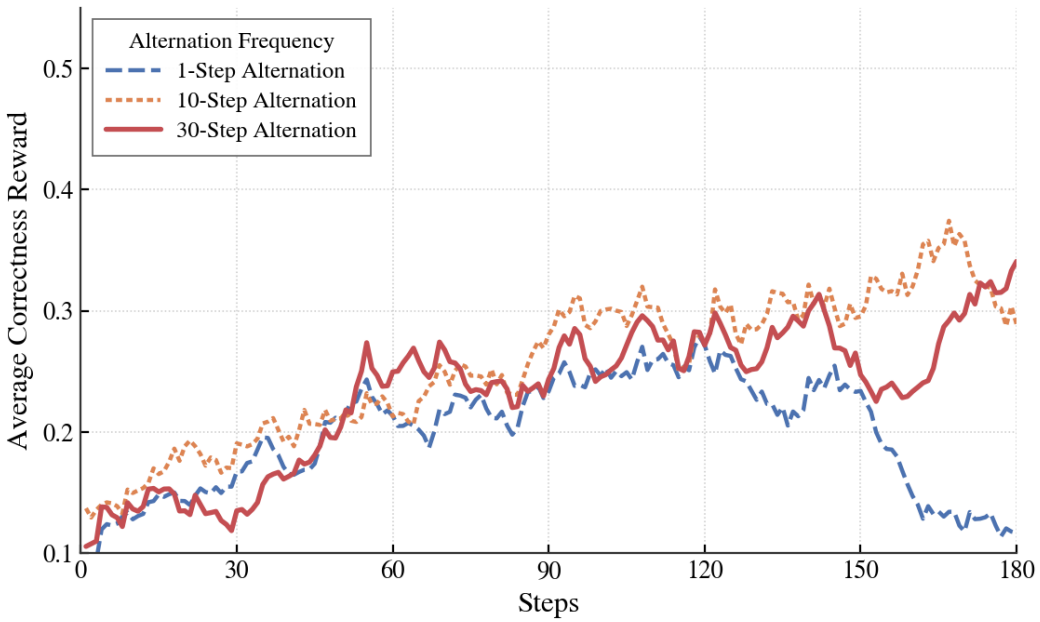


Figure 4: Sensitivity analysis on the stage length for designer–executor alternation. One-step alternation is unstable and collapses during training, while longer stages provide more stable role-specific optimization. The 30-step schedule achieves the best final reward and is used as the default setting in our main experiments.

2. **Delivery formatting.** Inter-agent messages must be strictly enclosed within `<delivery>...</delivery>` tags. This constraint serves a dual purpose: it establishes a structured, easily parsable communication protocol, and crucially, it incentivizes agents to *distill* relevant information into concise deliverables rather than forwarding their entire reasoning trace. Without this constraint, agents tend to broadcast full outputs, unnecessarily inflating the context window without improving coordination quality.

## C Result Analysis

**Sensitivity Analysis on Stage Length.** Given that stagewise optimization is essential for stable designer–executor co-evolution, we further ablate how frequently the active role should be switched. Figure 4 compares three alternation intervals: 1-step, 10-step, and 30-step switching. Alternating the active role at every step leads to highly unstable training: neither role accumulates sufficient role-consistent gradient signal before being interrupted, and the training run collapses after approximately 150 steps. Increasing the interval to 10 or 30 steps substantially improves stability. Among them, the 30-step schedule achieves the highest final reward and exhibits the clearest upward trend. We therefore adopt 30-step alternation as the default setting in all main experiments.

**Per-query design diversity.** Besides the structures, MetaAgent design different task-specific and various agents role. Across all the workflows the designer emits in evaluation, it instantiates a vocabulary of **54 distinct role names**; System prompts are even more diverse: **77.5% of agents receive a byte-unique, task-specific prompt**. Each workflow is therefore a freshly synthesised program tailored to the problem at hand, not an instantiation of a pre-defined template; the structural taxonomy in Table 7 is a coarse skeleton, while the body of every workflow is per-query content.

**Which role brings the improvement?** RL changes *both* (i) which structure the designer selects and (ii) how well the executor performs under it. The designer's routing distribution moves uniformly across math benchmarks (Table 6): **ensemble+judge** loses $\sim$30 pp on every benchmark and the share is redirected mostly to **reflection**, with a smaller push toward **single**. To disentangle the two effects

we look at the same 30 AIME 2025 problems solved by SFT and RL. Of the problems RL solves but SFT does not, 50% use the *same* pattern as SFT, which show an executor-side improvement. And 50% are produced by a designer flip to a different (and in those cases simpler) pattern, indicating the improvement from SFT→RL also benefits from a designer flip.

| Top-10 role names | |
|---|---|
| CodeSolver | AlgebraicSolver |
| MathSolver | BruteForceSolver |
| UnitTestAgent | CombinatorialSolver |
| MathCritic | EdgeCaseSolver |
| MathJudge | OptimalSolver |

Table 5: Top-10 role names the designer emits across 2,574 workflows.

| | **single** | | **refl.** | | **ens.** | |
|---|---|---|---|---|---|---|
| Benchmark | SFT | RL | SFT | RL | SFT | RL |
| AIME 2024 | 16.7 | 18.9 | 38.9 | 70.0 | 44.4 | 11.1 |
| AIME 2025 | 7.8 | 15.6 | 41.1 | 73.3 | 51.1 | 11.1 |
| OlympiadBench | 42.3 | 46.4 | 21.1 | 44.8 | 36.6 | 8.8 |

Table 6: Pattern share (%) on math benchmarks: SFT cold-start vs. RL.

# D Baseline Details

We describe the baseline implementations used in our experiments. Unless otherwise stated, all baselines use the same execution backbone as our method, instantiated with Qwen3 4B or Qwen3 8B according to the corresponding experimental setting. All reported results are evaluated on the same final test split and metric as our method. For baselines that require search or training, we follow the default protocol of the original paper or released code, while matching the rollout budget of our method whenever the method exposes the corresponding parameter.

## D.1 Search Based MAS Optimization Baselines

**AFlow.** AFlow (Zhang et al., 2024) searches over code represented agentic workflows with MCTS. We follow the official paper and code settings. Specifically, AFlow uses `sample`=4, `initial_round`=1, `max_rounds`=20, `validation_rounds`=5, and early stopping enabled. The `sample`=4 setting matches the number of candidate MAS designs generated by our Designer for each query. During workflow evaluation, we execute each candidate workflow $N = 4$ times when stochastic execution is supported, matching our execution budget. We keep the original domain specific operator sets: `Custom`, `ScEnsemble`, and `Programmer` for math, and `Custom`, `CustomCodeGenerate`, `ScEnsemble`, and `Test` for code. For each task domain, we use the best searched workflow reported or released by AFlow and evaluate it on our held out test split.

**ADAS.** ADAS (Hu et al., 2025) performs Meta Agent Search, where a meta agent writes executable Python `forward` functions and adds evaluated agents to an archive. We follow the official split and search protocol. For MGSM and related reasoning domains, the official implementation shuffles all examples with seed 0, uses 128 examples for search validation, and uses the next 800 examples for final testing. For GPQA diamond, it uses 32 validation examples and the remaining 166 examples for testing. For ARC, it uses 20 validation tasks and 60 test tasks, with five repeated evaluations to reduce stochastic variance.

The default MGSM search uses `n_generation`=30, `n_repeat`=1, `max_workers`=48, and at most three debugging attempts for invalid generated code. To align with our training budget, when we run ADAS search on a new benchmark without an official task specific searched agent, each generation evaluates $M = 4$ newly proposed candidate agents when supported by the search implementation, and each candidate agent is executed $N = 4$ times for reward estimation. The candidate agent executor follows the same Qwen3 4B or Qwen3 8B backbone as our method. The reflection call uses temperature 0.8. The initial archive follows the released implementation and contains self consistency with Chain of Thought, Self Refine, LLM Debate, Step Back Abstraction, Quality Diversity, and Role Assignment. For math, we use the best searched MGSM agent reported by ADAS, **Dynamic Role Playing Architecture**, and keep its original role routing and answer aggregation. For scode, the official ADAS repository does not release a task specific searched code agent. Therefore, when no official task specific searched code agent is available, we run Meta Agent Search on the corresponding optimization split with the same default ADAS search budget and select the best validation agent for held-out test evaluation.

### D.2 Semi Learning Based MAS Optimization Baselines

**ScoreFlow.** ScoreFlow (Wang et al., 2025b) trains a workflow generator with Score DPO. We follow the released training and inference pipeline, while aligning the candidate and execution budget with our setting. For each optimization step, ScoreFlow generates $M = 4$ candidate workflows per query and executes each candidate workflow $N = 4$ times to estimate its score. The resulting workflow scores are used to construct preference pairs for Score DPO. The held out test split is used only for final inference. For inference, we use the trained checkpoint selected by the original validation protocol.

We use the same stage length and learning rate as our method whenever ScoreFlow updates trainable parameters. Specifically, optimization is organized into stages of length $K = 30$, and the workflow generator is updated with learning rate $5 \times 10^{-6}$. Only the trainable generator parameters are updated; executor parameters are frozen. Score DPO is implemented with LoRA, with rank 8, alpha 16, dropout 0.01, target modules `q_proj` and `v_proj`, and no bias. The generator uses temperature 0.2, top $p = 0.95$, and maximum generation length 1000. The executor uses temperature 0.0. The vLLM setting uses bfloat16, GPU memory utilization 0.9, and maximum model length 10000. Each optimization epoch trains for 1 epoch with per device train batch size 1 and per device evaluation batch size 1. Logging is performed every 10 steps, and LoRA weights are merged into the generator checkpoint after each epoch.

**MaAS.** MaAS (Zhang et al., 2025a) trains an agentic supernet and samples query dependent architectures from it. We follow the official training and evaluation protocol. Each benchmark is split into train and test sets with a $1 : 4$ ratio. The training split is used to optimize the controller distribution and agentic operators, while the held out test split is used only for final evaluation. We match the architecture sampling and training budget to our method where possible. For each query, MaAS samples $M = 4$ candidate architectures from the agentic supernet and executes each sampled architecture $N = 4$ times for reward estimation. We use a stage length of $K = 30$ and update the trainable controller and architecture parameters with learning rate $5 \times 10^{-6}$. The execution backbone remains fixed. We keep the official architectural defaults: maximum supernet depth $L = 4$, sampling times $K = 4$ in the original MaAS notation, early exit threshold 0.3, and cost penalty coefficient selected from $\{1e{-}3, 5e{-}3, 1e{-}2\}$. In the released command, optimization is run with `sample`=4; the same command is then rerun with `is_test=True` for held out evaluation. We use the best validation setting selected by the original protocol and report its held out test performance.

**Agent Foundation Model.** Agent Foundation Model (AFM) (Li et al., 2025a) is an end-to-end agent model trained under the Chain-of-Agents paradigm. Instead of explicitly instantiating an external multi-agent workflow at test time, AFM internalizes multi-agent collaboration into a single model through multi-agent distillation and agentic reinforcement learning. This makes AFM different from our domain-adaptive automatic MAS setting: AFM is a released agent foundation model trained with its own data and backbone, while our method learns to construct and execute task-specific MAS using the Qwen3 4B or Qwen3 8B backbone. Nevertheless, AFM is a relevant baseline because it represents a strong end-to-end alternative to explicit MAS optimization.

We evaluate the officially released size-comparable code-agent checkpoint, `AFM-CodeAgent-7B-rl`. The released AFM model card also lists larger `AFM-CodeAgent-32B` checkpoints, but we use the 7B checkpoint to keep the comparison closer to our 8B experimental scale. We follow the official AFM code-agent evaluation framework. We use the official code-agent evaluation script with the released default parameters. The maximum prompt length is set to $4096$ tokens, and the maximum response length is set to $28672$ tokens. The rollout uses `n`=8 samples, `val_kwargs.temperature`=0.6, and multi-turn tool use with at most 12 turns.

## E Compute-Matched Comparisons

Because MetaAgent-X emits a multi-agent workflow, it may consume more test-time computation than a single-agent baseline, so part of its advantage could in principle come from the extra budget rather than from co-evolution. We first note that the $M \times N$ hierarchical rollout is used only during *training*: at inference time MetaAgent-X performs one Designer rollout followed by one execution of

Table 7: Structure share selected by the RL designer.

| Benchmark | Single | Reflection | Ensemble |
|---|---|---|---|
| AIME 2024 | 18.9% | **70.0%** | 11.1% |
| AIME 2025 | 15.6% | **73.3%** | 11.1% |
| OlympiadBench | **46.4%** | 44.8% | 8.8% |
| CodeContests | 26.7% | **62.4%** | 10.9% |
| LiveCodeBench | 43.5% | **52.6%** | 3.8% |
| APPS | **55.2%** | 43.8% | 1.0% |

the emitted MAS, and does not sample multiple workflows and select among them. To control for the remaining difference, we compare against stronger test-time baselines matched on token budget, all using the same Qwen3-8B backbone with identical tool and test access.

Table 8: Compute-matched comparison on Qwen3-8B. Math and Code are averages over the three benchmarks in each domain. Reflection and solver–critic use four rounds of interaction, which brings their token usage close to or above that of MetaAgent-X . Tokens per problem are reported as Math / Code.

| Method | Math Avg. | Code Avg. | Tokens/problem |
|---|---|---|---|
| Single-agent self-consistency / best-of-8 | 40.9% | 27.4% | 18K / 28K |
| Single-agent reflection, 4 rounds | 39.4% | 28.8% | 26K / 40K |
| Fixed solver–critic, 4 rounds | 39.7% | 29.3% | 28K / 42K |
| **MetaAgent-X** | **44.8%** | **32.0%** | 24K / 38K |

As shown in Table 8, MetaAgent-X attains the best average on both Math and Code while using fewer tokens per problem than the reflection and solver–critic baselines. Since those two baselines spend a comparable or larger test-time budget and still fall short, the improvement does not reduce to additional test-time computation: it comes from emitting a better task-specific workflow and from having jointly trained the agents that execute it.

## F Separating the Cold Start from RL Co-evolution

Our policy is cold-started by distilling DeepSeek-V3.2 trajectories, which raises the question of how much of the final performance is attributable to teacher distillation rather than to RL co-evolution. We first report that RL training directly from the Qwen3-4B/8B base models was not viable in our setting: without the SFT stage the base models could not reliably emit valid, executable multi-agent workflows under the full system prompt, so the cold start is necessary to initialize workflow generation at all. A no-cold-start RL condition is therefore not available as a control.

Table 9: Continued-SFT control on Qwen3-8B, isolating the contribution of RL from additional supervised training on the same data. Math and Code are averages over the three benchmarks in each domain.

| Model | Math | Code |
|---|---|---|
| MetaAgent-X $_{\text{SFT}}$ | 37.44% | 27.00% |
| + continued SFT on RL data | 39.02% | 28.50% |
| **MetaAgent-X $_{\text{RL}}$** | **44.78%** | **32.00%** |

To isolate the RL contribution given that cold start, we add a continued-SFT control: starting from MetaAgent-X $_{\text{SFT}}$, we continue supervised training on the RL-stage data together with the DeepSeek-distilled trajectories, but without any RL optimization. This control sees the same data as the RL run and differs only in the objective. As shown in Table 9, continued supervised training yields modest gains ($+1.6$ Math, $+1.5$ Code), whereas RL yields substantially larger ones ($+7.3$ Math, $+5.0$ Code) from the same starting point. We therefore state the claim narrowly: the cold start is necessary to initialize executable workflow generation, and outcome-driven designer–executor optimization accounts for the majority of the improvement beyond it.

| Example | SFT Model | RL Model |
|---|---|---|
| Math<br>**Better Design** | × Ensemble judge. All solvers share the wrong circle-packing model; judge reports contradiction but cannot repair. | ✓ Solver–critic reflection. Critic localizes the geometry error; solver switches to similar triangles. |
| Code<br>**Better Execute** | × Same reflection structure, but executor keeps double-counting divisors after sample outputs are 2× too large. | ✓ Same reflection structure, but executor uses tests to restore the one-count-per-divisor invariant. |

Table 10: SFT-to-RL case comparison: RL improves both MAS design and executor repair behavior.

# G Case Studies

## G.1 RL model Examples

We present three end-to-end trajectories of our system, illustrating how the *designer* chooses a team structure for each question and how the selected executors collaborate to reach the final answer. Cases span both math and code domains and showcase three distinct team structures: **single agent**, **ensemble + judge**, and **reflection with a separate critic**. The designer and executor boxes contain *verbatim* model output. Only chain-of-thought stretches that do not affect the exposition are elided, marked by the literal symbol "*. . . (...) . . .*".

### Case 1: Probability via single-agent reasoning

Domain: **Math (AIME 2024 #2)**

**Question**

Jen enters a lottery by picking 4 distinct numbers from $S = \{1, 2, \ldots, 10\}$. 4 numbers are randomly chosen from $S$. She wins a prize if at least two of her numbers were among the 4 chosen, and the *grand prize* if all four match. The probability of winning the grand prize given that she won a prize is $\frac{m}{n}$ in lowest terms. Find $m + n$.

**Designer (raw output)**

**Problem Type:** MATH
**Problem Analysis:**
This is a probability problem involving conditional probability.
- Jen picks 4 distinct numbers from $S = \{1, 2, \ldots, 10\}$.
- 4 numbers are randomly chosen from $S$ (presumably without replacement, since they are distinct).
- She wins a prize if at least 2 of her numbers match the 4 chosen numbers.
- She wins the grand prize if all 4 of her numbers match the 4 chosen numbers.
- We are asked: given that she won a prize, what is the probability she won the grand prize?
- The answer is $\frac{m}{n}$ in lowest terms, and we need $m + n$.

Mathematically: Let $A$ = event she wins the grand prize (all 4 match). Let $B$ = event she wins a prize (at least 2 match). We need $P(A \mid B) = \frac{P(A)}{P(B)}$.
We can compute:
- Total ways to choose 4 numbers from 10: $\binom{10}{4}$.
- If Jen's numbers are fixed (say $\{a, b, c, d\}$), then: $P(A)$ = probability the 4 chosen numbers are exactly Jen's 4 numbers: $\frac{1}{\binom{10}{4}}$. $P(B)$ = probability at least 2 of the 4 chosen numbers are in Jen's set. This can be computed as $1 - P(0 \text{ or } 1 \text{ match})$.

We can also think: Given that she won a prize, the 4 chosen numbers are a 4-element subset of $S$ that intersects her set in at least 2 elements. We want the probability that the intersection is exactly 4 (i.e., the chosen set equals her set).
This is a finite combinatorial problem, not requiring calculus or advanced probability theory.
**Workflow Pattern:** `math_single_agent`
**Reason:** The problem is straightforward combinatorial probability. A single expert mathematician can solve it step-by-step without needing multiple perspectives or iterative refinement.
**Agent Design:**
One `MathSolver` agent with a system prompt that emphasizes:

- Understanding the problem (conditional probability, combinatorial counting).
- Computing $P(A)$ and $P(B)$ using combinatorial reasoning.
- Simplifying the fraction and finding $m + n$.
- Outputting the final answer in the required boxed format.

**Workflow Flow:**
The single agent reads the problem, reasons through it, and outputs the final answer.

<code>

```
# Math Single Agent Solver for Lottery Probability Problem
from workflow import AgentNode, Workflow, ToolRegistry

# No tools needed - pure reasoning
tool_registry = ToolRegistry()

# Create a math solver agent
math_agent = AgentNode(
   name="MathSolver",
   system_prompt=(
      "You are an expert mathematician who solves problems through careful
          reasoning.\n\n"

      "APPROACH:\n"
      "1. Read the problem carefully and identify what is being asked\n"
      "2. Break down the problem into smaller steps\n"
      "3. Apply relevant mathematical concepts and formulas\n"
      "4. Show your work step by step\n"
      "5. Verify your answer makes sense\n\n"

      "For this specific problem:\n"
      "- Jen picks 4 distinct numbers from S = {1,2,...,10}.\n"
      "- 4 numbers are randomly chosen from S (without replacement).\n"
      "- She wins a prize if at least 2 of her numbers match the 4 chosen numbers.\n"
      "- She wins the grand prize if all 4 of her numbers match the 4 chosen
          numbers.\n"
      "- We need: P(grand prize | prize) = P(grand prize) / P(prize).\n"
      "- Compute using combinatorial counting.\n\n"

      "OUTPUT FORMAT:\n"
      "- Show your complete reasoning process\n"
      "- Put your final numerical answer in \\boxed{} format\n"
      "- Example: The answer is \\boxed{42}\n\n"

      "IMPORTANT: Your final answer MUST be in \\boxed{answer} format."
   ),
   tool_registry=tool_registry,
   max_turns=1
)

# Create workflow
workflow = Workflow(name="math_single_solver")
workflow.add_node(math_agent)

# Define the question
question = (
   "Jen enters a lottery by picking 4 distinct numbers from S = {1,2,3,...,9,10}. "
   "4 numbers are randomly chosen from S. She wins a prize if at least two of her
       numbers "
   "were 2 of the randomly chosen numbers, and wins the grand prize if all four of
       her numbers "
   "were the randomly chosen numbers. The probability of her winning the grand prize
       given that "
   "she won a prize is m/n where m and n are relatively prime positive integers. Find
       m+n."
)

# Run workflow
print("================================================")
print("FINAL ANSWER:")
result = workflow.run(question)
print(result.content)
print("================================================")
```

</code>

**Executor: MathSolver**

**Problem Type:** MATH
**Problem Analysis:**
This is a probability problem involving combinatorial counting. We need to find:
- $P(\text{grand prize} \mid \text{prize}) = P(\text{grand prize})/P(\text{prize})$
- Where:
  - Grand prize: all 4 numbers match
  - Prize: at least 2 numbers match
- We need to compute using combinatorial counting from the set $S = \{1, 2, \ldots, 10\}$
- Then express as $m/n$ in lowest terms and find $m + n$

**Approach:**
1. Compute total number of ways to choose 4 numbers from 10: $C(10, 4)$
2. Compute $P(\text{grand prize})$: number of ways to match all 4 numbers
3. Compute $P(\text{prize})$: number of ways to match exactly 2 or exactly 3 or exactly 4 numbers
4. Use conditional probability formula: $P(\text{grand prize} \mid \text{prize}) = P(\text{grand prize})/P(\text{prize})$
5. Simplify to lowest terms and find $m + n$

**Solution:**
Let's compute step by step:
1. Total number of ways to choose 4 numbers from 10:
   $C(10, 4) = 10!/(4! \cdot 6!) = (10 \times 9 \times 8 \times 7)/(4 \times 3 \times 2 \times 1) = 210$
2. $P(\text{grand prize})$: All 4 numbers match
   - There's only 1 way to choose exactly Jen's 4 numbers
   - So $P(\text{grand prize}) = 1/210$
3. $P(\text{prize})$: At least 2 numbers match
   - This includes: exactly 2 matches, exactly 3 matches, exactly 4 matches
   - Exactly 4 matches: 1 way (already counted)
   - Exactly 3 matches: choose 3 of Jen's numbers and 1 from the remaining 6
     $C(4, 3) \cdot C(6, 1) = 4 \cdot 6 = 24$
   - Exactly 2 matches: choose 2 of Jen's numbers and 2 from the remaining 6
     $C(4, 2) \cdot C(6, 2) = 6 \cdot 15 = 90$
   - So total ways for prize $= 1 + 24 + 90 = 115$
   - $P(\text{prize}) = 115/210$
4. $P(\text{grand prize} \mid \text{prize}) = P(\text{grand prize})/P(\text{prize}) = (1/210)/(115/210) = 1/115$
5. $1/115$ is already in lowest terms (1 and 115 are coprime)
   - $m = 1, n = 115$
   - $m + n = 1 + 115 = 116$

**Approach:** Compute probabilities using combinatorial counting of ways to match numbers.
**Answer:** $\boxed{116}$
**Confidence:** High

**Final Answer**

$\boxed{116}$ ground truth: 116 ✓

---

### Case 2: Disagreement resolved by an ensemble + judge

Domain: **Math (AIME 2024 #6)**

**Question**

Find the number of ways to place a digit in each cell of a $2 \times 3$ grid so that the sum of the two numbers formed by reading left-to-right is 999 and the sum of the three numbers formed by reading top-to-bottom is 99.

| | | |
|---|---|---|
| 0 | 0 | 8 |
| 9 | 9 | 1 |

(example: $8 + 991 = 999,\ 9 + 9 + 81 = 99$)

**Designer (raw output)**

**Problem Type:** MATH
**Problem Analysis:**
We have a $2 \times 3$ grid of digits (0–9).

- The two numbers formed by reading left to right (top row and bottom row) must sum to 999.
- The three numbers formed by reading top to bottom (first column, second column, third column) must sum to 99.

We need to count the number of valid digit placements.
Let the grid be:

| | | |
|---|---|---|
| $a$ | $b$ | $c$ |
| $d$ | $e$ | $f$ |

Then:

1. Top row number: $100a + 10b + c$
2. Bottom row number: $100d + 10e + f$
3. Condition: $(100a + 10b + c) + (100d + 10e + f) = 999$
4. First column number: $10a + d$
5. Second column number: $10b + e$
6. Third column number: $10c + f$
7. Condition: $(10a + d) + (10b + e) + (10c + f) = 99$

We need to count integer solutions with $0 \leq a, b, c, d, e, f \leq 9$.
This is a combinatorial counting problem with linear constraints.
**Workflow Pattern:** `math_ensemble_judge`
**Reason:** The problem involves constraints that can be approached algebraically, combinatorially, or by case analysis. Multiple perspectives can help verify correctness, and a judge can select the most reliable answer.
**Agent Design:**

- **AlgebraicSolver:** Focus on solving the system of equations.
- **GeometricSolver:** Think in terms of digit constraints and possible ranges.
- **CombinatorialSolver:** Use counting arguments or case analysis.
- **MathJudge:** Compare the three solutions and pick the best.

**Workflow Flow:**

1. Each solver independently solves the problem.
2. The judge receives all three solutions.
3. The judge evaluates correctness and selects one answer.
4. The judge outputs the final answer in $\square$ format.

<code>

```
from workflow import AgentNode, Workflow, ToolRegistry
from workflow.nodes import EnsembleNode

# No tools needed - pure reasoning
tool_registry = ToolRegistry()

# Create three math solvers with different approaches
solver1 = AgentNode(
   name="AlgebraicSolver",
   system_prompt=(
      "You are a mathematician who prefers algebraic approaches.\n"
      "Solve the problem using algebraic manipulation and equations.\n"
      "Show your reasoning step by step.\n"
      "Put your final answer in \\boxed{} format.\n\n"
      "Problem: Find the number of ways to place a digit in each cell of a 2x3 grid
          so that "
      "the sum of the two numbers formed by reading left to right is 999, "
      "and the sum of the three numbers formed by reading top to bottom is 99.\n"
      "Let the grid be:\n"
      "\\[\\begin{array}{|c|c|c|} \\hline a & b & c \\ \\hline d & e & f \\ \\hline
          \\end{array}\\]\n"
      "Then:\n"
      "1. Top row number: 100a + 10b + c\n"
      "2. Bottom row number: 100d + 10e + f\n"
      "3. Condition: (100a + 10b + c) + (100d + 10e + f) = 999\n"
      "4. First column number: 10a + d\n"
      "5. Second column number: 10b + e\n"
      "6. Third column number: 10c + f\n"
```

```
        "7. Condition: (10a + d) + (10b + e) + (10c + f) = 99\n"
        "Count integer solutions with 0 <= a,b,c,d,e,f <= 9."
    ),
    tool_registry=tool_registry,
    max_turns=1
)

solver2 = AgentNode(
    name="GeometricSolver",
    system_prompt=(
        "You are a mathematician who thinks geometrically and visually.\n"
        "Solve the problem using geometric intuition or visual reasoning.\n"
        # ... (same problem statement as solver1) ...
    ),
    tool_registry=tool_registry,
    max_turns=1
)

solver3 = AgentNode(
    name="CombinatorialSolver",
    system_prompt=(
        "You are a mathematician skilled in counting and combinatorics.\n"
        "Solve the problem using combinatorial arguments or case analysis.\n"
        # ... (same problem statement as solver1) ...
    ),
    tool_registry=tool_registry,
    max_turns=1
)

# Create judge agent to select the best answer
judge_agent = AgentNode(
    name="MathJudge",
    system_prompt=(
        "You are a senior mathematician acting as a judge.\n\n"
        "You will receive solutions from three different solvers.\n"
        "Your task is to:\n"
        "1. Review each solution carefully\n"
        "2. Check the reasoning and calculations\n"
        "3. Identify which solution is correct (or most likely correct)\n"
        "4. Select ONE answer as the final answer\n\n"
        "OUTPUT FORMAT:\n"
        "- Briefly explain which solver's answer you chose and why\n"
        "- Put the selected final answer in \\boxed{} format\n\n"
        "IMPORTANT: You must select ONE answer and output it in \\boxed{answer}
            format.\n\n"
        # ... (same problem statement as solver1) ...
    ),
    tool_registry=tool_registry,
    max_turns=1
)

# Create ensemble with judge selection
ensemble = EnsembleNode(
    name="MathEnsemble",
    agents=[solver1, solver2, solver3],
    strategy="consensus",
    consensus_agent=judge_agent
)

# Create workflow
workflow = Workflow(name="math_ensemble_judge")
workflow.add_node(ensemble)
```

</code>

*(In the actual file, the three solver `system_prompts` contain the same problem statement verbatim; collapsed here as "... same problem statement ..." to save space.)*

**Executor: Solver 1 – AlgebraicSolver ✗**

**Problem Type:** MATH **Approach:** We have:

(1) $100a + 10b + c + 100d + 10e + f = 999$

(2) $10a + d + 10b + e + 10c + f = 99$

We can treat these as two equations in six variables, but we can also think of them as constraints on the digits. Let's denote: Equation (1): $100(a + d) + 10(b + e) + (c + f) = 999$. Equation (2): $10(a + c) + (d + e) + (b + f) = 99$.

... the solver continues from this rewriting of (2) ...

So $X = a + d = 9$, $Y = b + e = 9$, $Z = c + f = 9$. We also have the second equation: $10(a + c) + (d + e) + (b + f) = 99$. We can check if this is automatically satisfied given $a + d = 9$, $b + e = 9$, $c + f = 9$. Let's verify: from $a + d = 9$, $d = 9 - a$. From $b + e = 9$, $e = 9 - b$. From $c + f = 9$, $f = 9 - c$. Then $a + c$ is just $a + c$. $d + e = (9 - a) + (9 - b) = 18 - a - b$. $b + f = b + (9 - c) = 9 + b - c$. So

$$10(a + c) + (18 - a - b) + (9 + b - c) = 9a + 9c + 27.$$

We want this to equal 99: $9a + 9c = 72$, so $a + c = 8$.
Possible $(a, c)$: $(0, 8), (1, 7), \ldots, (8, 0)$ – 9 possibilities. For each $b$ from 0 to 9, $e = 9 - b$ – 10 possibilities. Thus total $= 9 \times 10 = 90$.
**Answer:** $\boxed{90}$ **Confidence:** High

*Annotation: this solver miscopied equation (2) as $10(a+c) + (d+e) + (b+f) = 99$ instead of the actual $10(a+b+c) + (d+e+f) = 99$ – the b coefficient is lost. This propagates to the wrong count of 90.*

**Executor: Solver 2 – GeometricSolver**

**Problem Type:** MATH
**Geometric/Visual Reasoning Approach:**
Let $A = a + d$, $B = b + e$, $C = c + f$. Then (1) becomes $100A + 10B + C = 999$. Since $A, B, C \in [0, 18]$, we get $A = 9, B = 9, C = 9$, hence

$$a + d = 9, \quad b + e = 9, \quad c + f = 9.$$

Now from (2): $10(a+b+c)+(d+e+f) = 99$. But $d+e+f = (9-a)+(9-b)+(9-c) = 27-(a+b+c)$. Substitute:

$$10(a + b + c) + 27 - (a + b + c) = 99 \implies 9(a + b + c) = 72 \implies a + b + c = 8.$$

That's a stars-and-bars count of nonnegative integer solutions to $a + b + c = 8$ with $0 \le a, b, c \le 9$. Since $8 \le 9$, the upper bound is not restrictive, so the number is $\binom{8+3-1}{3-1} = \binom{10}{2} = 45$.
**Answer:** $\boxed{45}$ **Confidence:** High

**Executor: Solver 3 – CombinatorialSolver (no clean answer)**

**Problem Type:** MATH
**Problem Analysis:**
We have a $2 \times 3$ grid with digits $a, b, c$ in the top row and $d, e, f$ in the bottom row, each from 0 to 9. Conditions:
(1) $(100a + 10b + c) + (100d + 10e + f) = 999$
(2) $(10a + d) + (10b + e) + (10c + f) = 99$
We need to count integer solutions $(a, b, c, d, e, f)$ with $0 \le a, b, c, d, e, f \le 9$.
This is a **combinatorial counting problem** with constraints. We can approach by interpreting the constraints as digit-wise equations (mod 10 and carry-over), using case analysis on possible carries, or solving the system algebraically and counting integer solutions.
**Approach:** we will treat the two conditions as digit-wise constraints with possible carries. [ . . . ]

. . . long case analysis on $c+f \in \{9, 19\}$, $b+e \in \{9, 19\}$, $a+d \in \{9, 19\}$ derives mutually inconsistent constraints . . .

So $S_2 = 9$, $S_3 = 9$. But $S_2 = b + e = 9$, $S_3 = c + f = 9$. Then $a + b + c = 9$ and $a + b + c = 18$. So we must have a carry from the tens place? Let's think: we have $10S_2 + S_3 = 99$. So $S_2 = 9$, $S_3 = 9$. But $S_2 = b + e = 9$, $S_3 = c + f = 9$. Then $a + b + c = 9$ and $a + b + c = 18$. So we must have a carry from the tens place? Let's think: we have $10S_2 + S_3 = 99$. So $S_2 = 9$, $S_3 = 9$. But $S_2 = b + e = 9$, $S_3 = c + f = 9$. Then $a + b + c = 9$ and $a + b + c =$

. . . the model loops on this passage until the token budget is exhausted; no \boxed{} answer is emitted . . .

**Judge**

**Key Insight:**
We can derive constraints from the equations, define $X = a + d, Y = b + e, Z = c + f$, solve $100X + 10Y + Z = 999$ to get $X = 9, Y = 9, Z = 9$, then use the column equation $10(a + b + c) + (d + e + f) = 99$ together with $d + e + f = 27 - (a + b + c)$ to get $a + b + c = 8$. Counting nonnegative integer solutions to $a + b + c = 8$ with the unrestrictive bound $a, b, c \le 9$: $\binom{8+3-1}{3-1} = \binom{10}{2} = 45$.

**Reason for selection:** Solution 2 (GeometricSolver) correctly derived the constraints and counted the solutions. Solution 1 (AlgebraicSolver) arrives at 90 but that is incorrect because it misinterprets the second condition. Solution 3 (CombinatorialSolver) has a flawed reasoning chain with repeated contradictions.
**Answer:** $\boxed{45}$ **Confidence:** High

**Final Answer**

$\boxed{45}$ ground truth: 45 ✓. The judge correctly overrides Solver 1's mistake and the unfinished Solver 3.

---

### Case 3: Array eversion via reflection (solver + critic)

Domain: **Code (CodeContests #46)**

**Question**

**Array eversion.** You are given an array $a$ of length $n$. An *eversion* takes the last element $x = a_n$, partitions $a$ into a left part (elements $\leq x$, in order) and a right part (elements $> x$, in order), and replaces $a$ with their concatenation. Output the minimum number of eversions $k$ such that the array stops changing after $k$ eversions.
*Example:* $[2, 4, 1, 5, 3] \to [2, 1, 3, 4, 5]$, then unchanged: $k = 1$.
Constraints: $1 \leq n \leq 2{\cdot}10^5$, $1 \leq a_i \leq 10^9$.

**Designer (raw output)**

**Problem Type:** CODE
**Problem Analysis:**
This problem asks us to find the minimum number of "eversions" needed for an array to stop changing. An eversion operation:
1. Takes the last element `x = a_n`
2. Partitions the array into two stable parts: elements $\leq x$ (left) and elements $> x$ (right)
3. Concatenates left + right

The process stops when an eversion doesn't change the array. We need to compute the number of eversions until this happens.
**Key observations:**
- Each eversion moves the last element to the end of the left partition (all elements $\leq$ it)
- The array eventually becomes sorted in non-decreasing order
- The process stops when the array is already sorted (since then the last element is the maximum, and all elements $\leq$ it)
- We need to count how many eversions it takes to reach a sorted array

**Approach:**
We can simulate the process, but for $n$ up to $2 \times 10^5$, we need an efficient algorithm. Let's think about what happens:
- The last element in the current array is moved to the end of the left partition
- This is equivalent to: find the last element that is $\leq$ the current last element, and move it to the end
- Actually, the eversion operation is: take the last element `x`, move all elements $\leq x$ to the front (preserving order), then append the remaining elements

We can think of it as: after each eversion, the array becomes `[all elements` $\leq x$`] + [all elements` $> x$`]` where `x` is the original last element. We need to find how many such operations until the array is sorted.
**Workflow Pattern:** `code_solver_unittest`
This problem has subtle behaviour (stable partition, counting eversions). A single solver might miss edge cases. The reflection pattern with a `UnitTestAgent` will ensure thorough testing.
**Agent Design:**
1. **CodeSolver**: implements the solution, tests with examples
2. **UnitTestAgent**: writes comprehensive tests including edge cases

**Workflow Flow:**
1. CodeSolver writes initial solution
2. UnitTestAgent tests it thoroughly
3. CodeSolver refines based on feedback

4. UnitTestAgent verifies again
5. Final solution output

<code>

```
from workflow import AgentNode, Workflow, ToolRegistry
from workflow.nodes import ReflectionNode
import subprocess

# Set up code interpreter tool
tool_registry = ToolRegistry()

def execute_code(code: str) -> str:
    """Execute Python code and return stdout/stderr."""
    try:
        result = subprocess.run(
            ["python", "-c", code],
            capture_output=True, text=True, timeout=30
        )
        output = ""
        if result.stdout:
            output += f"stdout:\n{result.stdout}\n"
        if result.stderr:
            output += f"stderr:\n{result.stderr}\n"
        return output or "(no output)"
    except subprocess.TimeoutExpired:
        return "Error: Code execution timed out (30s limit)"

tool_registry.register(
    name="execute_code",
    func=execute_code,
    description="Execute Python code and return stdout/stderr",
    parameters={
        "type": "object",
        "properties": {
            "code": {"type": "string", "description": "Python code to execute"}
        },
        "required": ["code"]
    }
)

# Solver agent writes the solution
solver_agent = AgentNode(
    name="CodeSolver",
    system_prompt=(
        "You are an expert programmer. Solve the coding problem step by step.\n\n"
        "PROBLEM: Given an array a of length n, perform eversions until the array stops
            changing. "
        "An eversion: take last element x, partition array into elements <= x (left)
            and > x (right), "
        "concatenate left + right. Find minimum k such that after k eversions the array
            stops changing.\n\n"
        "You have access to an `execute_code` tool. Use it to test your solution:\n"
        '<tool_call>{"name": "execute_code", "arguments": {"code":
            "your_python_code_here"}}</tool_call>\n\n'
        "WORKFLOW:\n"
        "1. Understand the problem and constraints\n"
        "2. Write your solution\n"
        "3. Test it with the provided examples using execute_code\n"
        "4. If you receive feedback from a tester, fix the identified issues\n"
        "5. Wrap your final solution in <solution>...</solution> tags\n"
    ),
    tool_registry=tool_registry,
    max_turns=5
)

# Unit test agent writes and runs comprehensive tests
critic_agent = AgentNode(
    name="UnitTestAgent",
    system_prompt=(
        "You are an expert software tester. Your job is to verify a proposed solution "
        "by writing and running comprehensive unit tests.\n\n"
        "ORIGINAL PROBLEM: Given an array a of length n, perform eversions until the
            array stops changing. "
        "...\n\n"
        "WORKFLOW:\n"
        "1. Read the ORIGINAL PROBLEM and the proposed solution\n"
        "2. Write comprehensive test cases covering all examples + edge cases\n"
        "3. Run the tests using execute_code\n"
```

```
        "4. Report results clearly\n\n"
        "OUTPUT FORMAT:\n"
        "- List each test case and its result (PASS/FAIL)\n"
        "- If any test fails, explain what went wrong\n"
        "- Final verdict: CORRECT or INCORRECT\n"
    ),
    tool_registry=tool_registry,
    max_turns=5
)

# Solver -> Tester -> Solver refines -> Tester -> final
reflection = ReflectionNode(
    name="CodeSolverTestReflection",
    agent=solver_agent,
    critic_agent=critic_agent,
    num_iterations=2
)

workflow = Workflow(name="code_solver_unittest")
workflow.add_node(reflection)
```

</code>

**Executor: CodeSolver – iteration 1**

I need to understand the problem better. Let me think about the eversion operation:

1. Take the last element `x`
2. Partition array into elements $\leq x$ (left) and $> x$ (right), maintaining original order
3. Concatenate left + right

We need to find the minimum $k$ such that after $k$ eversions the array stops changing.
Let me test with the example. First example: $[2, 4, 1, 5, 3]$. Last element is 3. Partition: elements $\leq 3$ are $[2, 1, 3]$, elements $> 3$ are $[4, 5]$. Result: $[2, 1, 3, 4, 5]$.

... similar walk-through of the second and third examples ...

Wait, let me think: when we partition, the last element $x$ is in the left partition (since it's $\leq x$). So after partitioning, $x$ will be at the end of the left partition. Then we concatenate left + right. So the last element of the array after eversion is the last element of the left partition, which is $x$. So the last element remains the same!
That means the array will stop changing when the last element is the maximum element in the array. *Let me write a function to simulate the eversion and see what's happening.*

**Tool call:** `execute_code` with a brute-force simulator.

```
stdout:
First example:  Original:  [2, 4, 1, 5, 3]; k:  1
Second example:  Original:  [5, 3, 2, 4, 1]; k:  2
Third example:  Original:  [1, 1, 1, 1]; k:  0
```

**Judge**

**(UnitTestAgent)**
I'll analyze the response to the problem. Let me first understand the problem and then critique the proposed solution.
**Understanding the Problem:**
We need to find the minimum number of eversions (partition operation) until the array stops changing. An eversion takes the last element $x$, partitions the array into elements $\leq x$ (left) and $> x$ (right), then concatenates left + right.
**Key Insight from the Response:**
The response correctly identifies that the last element changes in each eversion. Specifically, after an eversion, the new last element becomes the last element of the right partition (elements $> x$). This means we need to find how many steps it takes for the last element to become the maximum element in the array.
**Critique of the Response:**

1. **Logical Flaw:** the response has a critical error in its reasoning. It claims that "if current $< M$, then $M$ is the maximum element $>$ current", but this is not true. In the second example, when current $= 1$, $M = 5$, but the maximum element $> 1$ is 4, not 5. So the next current should be 4, not $M$.
2. **Incomplete Algorithm:** the response doesn't provide a complete algorithm. It only describes the problem and the insight but doesn't give a working solution.

3. **Missing Edge Cases:** the response doesn't consider edge cases like duplicate elements, already sorted arrays, or arrays where all elements are equal.
4. **No Code Implementation:** the response doesn't provide any code implementation, making it impossible to test or use.
5. **Confusion in Terminology:** the response uses "max of elements $>$ current" but doesn't explain how to compute this efficiently.

**Let me write a complete solution and test it:**

... `execute_code` tool call with a candidate `min_eversions` function and tests on the three sample inputs ...

**Executor: CodeSolver – final solution**

I'll provide the final solution with a clear explanation.

**Approach:** The problem requires finding the minimum number of eversions until the array stops changing. The key insight is that after each eversion, the new last element is the last element of the right partition (elements $> x$). This means we need to find how many steps it takes for the last element to become the maximum element in the array.

**Algorithm:**

1. Find the maximum element $M$ in the array.
2. If the last element is already $M$, return $0$.
3. Otherwise, find the number of steps where each step moves to the last element in the array that is greater than the current element.
4. This can be done efficiently by precomputing the maximum elements in decreasing order from the end of the array.

**Solution:**

```
def solve():
    import sys

    input = sys.stdin.read
    data = input().split()

    t = int(data[0])
    idx = 1
    results = []

    for _ in range(t):
        n = int(data[idx]); idx += 1
        a = list(map(int, data[idx:idx + n])); idx += n

        # Find the maximum element
        M = max(a)

        # If the last element is already the maximum, no eversions needed
        if a[-1] == M:
            results.append(0)
            continue

        # Precompute the suffix-maxima in decreasing order
        max_elements = []
        for i in range(n - 1, -1, -1):
            if not max_elements or a[i] > max_elements[-1]:
                max_elements.append(a[i])

        # Walk the chain until current == M
        current = a[-1]
        k = 0
        while current != M:
            next_val = -1
            for val in max_elements:
                if val > current:
                    next_val = val
                    break
            if next_val == -1:
                break
            current = next_val
            k += 1

        results.append(k)

    sys.stdout.write("\n".join(map(str, results)))

if __name__ == "__main__":
```

```
solve()
```

**Confidence:** High.

**Final Answer**

**Tests passed: 1/1.** Submitted solution accepted (✓). The critic caught the missing implementation in iteration 1 and the solver returned a clean final program in iteration 2.

### G.2 A Case Study Where Reflection Repairs the Solver

We compare the same held-out AIME 2024 example under the SFT cold-start model and the RL checkpoint. This case is more diagnostic than a simple final-answer comparison: the RL trajectory first reaches an impossible geometric constraint, the critic identifies the faulty distance model, and the refined solver replaces it with a valid similar-triangles equation. In contrast, the SFT workflow uses an `ensemble_judge` structure, but the judge repeats the same invalid angle-packing assumption and no valid final answer is extracted.

**Question**

Eight circles of radius 34 are sequentially tangent, and two of the circles are tangent to $AB$ and $BC$ of triangle $ABC$, respectively. 2024 circles of radius 1 can be arranged in the same manner. The inradius of triangle $ABC$ can be expressed as $\frac{m}{n}$, where $m$ and $n$ are relatively prime positive integers. Find $m+n$.

**Final Answer**

**Ground truth:** 197.

#### Case SFT: Ensemble + judge adds breadth but does not repair the model

Domain: **Math / circle packing**

**Designer (raw output)**

**Model:** SFT model
**Generated workflow pattern:** `ensemble_judge`
**Executor agents:** `AlgebraicSolver`, `GeometricSolver`, `CombinatorialSolver`, `MathJudge`
**Evaluation result:** ✗ no valid final answer extracted.

**Generated workflow excerpt (verbatim):**

```
from workflow import AgentNode, Workflow, ToolRegistry
from workflow.nodes import EnsembleNode

tool_registry = ToolRegistry()

solver1 = AgentNode(
    name="AlgebraicSolver",
    system_prompt=(
        "You are a mathematician who prefers algebraic approaches.\n"
        "Solve the problem using algebraic manipulation and equations.\n"
        "Consider the geometry: circles tangent to two sides of an angle and to each
            other.\n"
        "Derive relationships between the triangle's inradius, the circle radii, and
            the number of circles.\n"
        "Show your reasoning step by step.\n"
        "Put your final answer in \\boxed{} format."
    ),
    tool_registry=tool_registry,
    max_turns=1
)

solver2 = AgentNode(
    name="GeometricSolver",
    system_prompt=(
        "You are a mathematician who thinks geometrically and visually.\n"
        "Solve the problem using geometric intuition.\n"
        "Imagine the circles packed in the angle at vertex B of triangle ABC.\n"
        "Use properties of tangents, similar triangles, and circle arrangements.\n"
        "Show your reasoning step by step.\n"
        "Put your final answer in \\boxed{} format."
    ),
```

```
    tool_registry=tool_registry,
    max_turns=1
)

solver3 = AgentNode(
    name="CombinatorialSolver",
    system_prompt=(
        "You are a mathematician skilled in counting and patterns.\n"
        "Solve the problem by analyzing the pattern of circle arrangement.\n"
        "Note that the same pattern works for radii 34 and 1, suggesting a scaling
            factor.\n"
        "Relate the number of circles to the inradius through the geometry of the
            packing.\n"
        "Show your reasoning step by step.\n"
        "Put your final answer in \\boxed{} format."
    ),
    tool_registry=tool_registry,
    max_turns=1
)

judge_agent = AgentNode(
    name="MathJudge",
    system_prompt=(
        "You are a senior mathematician acting as a judge.\n\n"
        "You will receive solutions from three different solvers.\n"
        "Your task is to:\n"
        "1. Review each solution carefully\n"
        "2. Check the reasoning and calculations\n"
        "3. Identify which solution is correct (or most likely correct)\n"
        "4. Select ONE answer as the final answer\n\n"
        "IMPORTANT: You must select ONE answer and output it in \\boxed{answer} format."
    ),
    tool_registry=tool_registry,
    max_turns=1
)

ensemble = EnsembleNode(
    name="MathEnsemble",
    agents=[solver1, solver2, solver3],
    strategy="consensus",
    consensus_agent=judge_agent
)

workflow = Workflow(name="math_ensemble_judge")
workflow.add_node(ensemble)
```

**Executor: SFT GeometricSolver output**

```
**Problem Type:** MATH

**Problem Analysis:** This is a geometry problem involving circle packing in an angle.
     We have triangle ABC with vertex B. At vertex B, we pack circles in the angle
    formed by sides AB and BC.

The key insight is that the packing pattern in an angle is determined by the angle
    itself. The number of circles that can fit in an angle depends on the angle's
    measure. Specifically, for circles of radius r packed in an angle theta, the
    number N of circles that can fit is approximately
N = floor(pi/(2r) * tan(theta/2))
or something similar, but here we have exact counts given, so we can set up an
    equation.

For R = 34, N = 8
For R = 1, N = 2024

Thus:
pi/(2*34) * tan(theta/2) = 8
pi/(2*1) * tan(theta/2) = 2024

Divide the second equation by the first:
(pi/(2*1) * tan(theta/2)) / (pi/(2*34) * tan(theta/2)) = 2024/8
34 = 253

So 34 = 253? That's not true. Wait, that suggests my formula is wrong.

\elide[the solver continues trying variants of the same angle-packing model]
```

```
This is too complex. Let me try a different approach: the packing in an angle is
    similar to circles in a wedge. The number of circles that can fit in an angle
    theta with radius r is given by
N = floor(pi/(2r) * tan(theta/2)).
But we have exact counts, so we can write:
For r = 34, N = 8
For r = 1, N = 2024

Thus:
pi/(2*34) * tan(theta/2) = 8
pi/(2*1) * tan(theta/2) = 2024

Divide:
34 = 253, impossible. So my formula is wrong.
```

**Judge**

**SFT judge output (verbatim excerpt).**

```
**Problem Type:** MATH

**Problem Analysis:** This is a geometry problem involving circle packing in an angle.
     We have:
- Triangle ABC with inradius r (to be found)
- Two configurations:
  - 8 circles of radius 34 arranged sequentially tangent to each other and tangent to
      sides AB and BC respectively
  - 2024 circles of radius 1 arranged in the same pattern

Key insight: The arrangement likely consists of circles packed along the angle
    bisector of angle ABC, each tangent to both sides AB and BC, with adjacent
    circles tangent to each other.

\elide[the judge copies the incorrect branch rather than rejecting it]

Now apply to both cases:
1. R = 34, N = 8:
   r = 34 * (8 - sin(theta/2)) * cos(theta/2)
2. R = 1, N = 2024:
   r = 1 * (2024 - sin(theta/2)) * cos(theta/2)

Since r is the same, we can set them equal:
34 * (8 - sin(theta/2)) * cos(theta/2) =
(2024 - sin(theta/2)) * cos(theta/2)

Assuming cos(theta/2) != 0, we can divide:
34 * (8 - sin(theta/2)) = 2024 - sin(theta/2)
272 - 34 sin(theta/2) = 2024 - sin(theta/2)
272 - 2024 = 34 sin(theta/2) - sin(theta/2)
-1752 = 33 sin(theta/2)
sin(theta/2) = -1752 / 33 = -53.0909...

This is impossible because sin(theta/2) must be positive (0 < theta/2 < 90 degrees).
    So my model is still wrong.
```

**Failure mode.** The SFT workflow samples multiple solver roles, but none of them provides a mechanism for forcing revision after the contradiction is found. The judge repeats the invalid model instead of rejecting it, so the transcript ends without a boxed valid answer.

### Case RL: Reflection turns the contradiction into a corrected derivation

Domain: **Math / circle packing**

**Designer (raw output)**

**Model:** RL checkpoint
**Generated workflow pattern:** `reflection`
**Executor agents:** `MathSolver` + `MathCritic`
**Evaluation result:** ✓ extracted answer 197.

**Generated workflow excerpt (verbatim):**

```
from workflow import AgentNode, Workflow, ToolRegistry
from workflow.nodes import ReflectionNode

tool_registry = ToolRegistry()
```

```
solver_agent = AgentNode(
    name="MathSolver",
    system_prompt=(
        "You are an expert mathematician specializing in geometry and circle
            packing.\n\n"
        "APPROACH:\n"
        "1. Understand the configuration: triangle ABC with inradius r.\n"
        "2. There are two sequences of circles: 8 circles of radius 34, and 2024
            circles of radius 1.\n"
        "3. Each sequence is tangent to two sides of the triangle and to each other.\n"
        "4. The centers of the circles in each sequence lie on a line parallel to the
            third side of the triangle.\n"
        "5. The distance between centers of consecutive circles is 2*radius.\n"
        "6. The inradius r is related to the distance from the incenter to the side,
            and the distance from the incenter to the line of centers.\n"
        "7. Set up equations using the given numbers 8, 34, 2024, 1.\n"
        "8. Solve for r, express as m/n, then compute m+n.\n\n"
        "If you receive feedback from a critic, carefully consider their points and:\n"
        "- Fix any errors in geometry or algebra.\n"
        "- Clarify any ambiguous steps.\n"
        "- Provide a corrected solution.\n\n"
        "OUTPUT: Show your complete reasoning step by step, then put your final answer
            in \\boxed{} format."
    ),
    tool_registry=tool_registry,
    max_turns=1
)

critic_agent = AgentNode(
    name="MathCritic",
    system_prompt=(
        "You are a meticulous math reviewer who checks geometry solutions for
            errors.\n\n"
        "YOUR TASK:\n"
        "1. Verify the geometric configuration: circles tangent to two sides of a
            triangle and to each other.\n"
        "2. Check the assumption that centers are collinear and equally spaced.\n"
        "3. Verify the relationship between the inradius, the distance from the
            incenter to the line of centers, and the number of circles.\n"
        "4. Check the algebraic manipulation when solving for r.\n"
        "5. Ensure the final expression for r is in lowest terms m/n, and m+n is
            computed correctly.\n\n"
        "Be thorough but constructive in your criticism."
    ),
    tool_registry=tool_registry,
    max_turns=1
)

reflection = ReflectionNode(
    name="SolverCriticReflection",
    agent=solver_agent,
    critic_agent=critic_agent,
    num_iterations=2
)

workflow = Workflow(name="math_solver_critic")
workflow.add_node(reflection)
```

**Executor: RL MathSolver first attempt**

```
**Problem Type:** MATH

**Problem Analysis:** This is a geometry problem involving circle packing in a
    triangle. We have:
- Triangle ABC with inradius r
- Two sequences of circles:
  1. 8 circles of radius 34, each tangent to two sides of the triangle and to each
      other
  2. 2024 circles of radius 1, arranged similarly
- Each sequence's centers lie on a line parallel to the third side
- The distance between centers of consecutive circles in a sequence is 2*radius

**Approach:**
1. Let the triangle have sides a, b, c with inradius r
2. For the sequence of circles tangent to sides AB and BC, the line of centers is
    parallel to side AC
3. The distance from the incenter to side AC is r
```

```
4. The distance from the incenter to the line of centers of the 34-radius circles is r
     - 34 (or r - 1 for the 1-radius circles)
5. The distance between centers of consecutive circles in a sequence is 2*radius

\elide[the first attempt derives equations from the wrong distance relation]

But we also know that this length is (n-1) * 2r1. So:

L * (r - d)/r = (n-1) * 2r1

But d = r - r1, so:

L * (r - (r - r1))/r = (n-1) * 2r1
L * r1/r = (n-1) * 2r1
L/r = 2(n-1)
L = 2r(n-1)

So L = 2r(n-1). But this must be the same for both sequences, so:

2r(8-1) = 2r(2024-1)
14r = 4046r
14 = 4046

This is impossible, so my assumption that the line of centers is at distance r - r1
    from AC is wrong.
```

**Judge**

**RL MathCritic feedback (verbatim excerpt).**

```
**Critique of the Response:**

**1. Errors and Inaccuracies:**
- **Misunderstanding of Geometry Configuration:** The problem states "eight circles of
     radius 34 are sequentially tangent, and two of the circles are tangent to AB and
     BC of triangle ABC, respectively." This suggests there are two sequences of
    circles: one tangent to AB and BC (and the other two sides?), and another
    sequence of 2024 circles of radius 1 arranged similarly.
- **Incorrect Distance Assumption:** The response incorrectly assumes the line of
    centers is at distance r - r1 from side AC, but this leads to a contradiction.
    The correct distance from the line of centers to side AC should be derived from
    the geometry of the triangle and the circle tangency conditions.
- **Algebraic Error:** The derivation leads to an impossible equation (14 = 4046),
    indicating a fundamental flaw in the geometric model.

**4. Specific Improvements:**
- **Clarify the Geometry:** There are two sequences of circles: one tangent to AB and
    BC, and another tangent to AB and AC (or BC and AC). Each sequence has its own
    line of centers parallel to the third side.
- **Correct Distance Calculation:** The distance from the line of centers to the third
     side should be derived using similar triangles or coordinate geometry.
- **Set Up Correct Equations:** For each sequence, relate the number of circles, their
     radii, and the inradius using the geometry of the triangle.
- **Solve for Inradius:** Use the two sequences to set up two equations and solve for
    the inradius.
```

**Executor: RL MathSolver refined output after reflection**

```
**Improved Solution:**

Let's denote:
- For the 34-radius circles: n1 = 8, r1 = 34
- For the 1-radius circles: n2 = 2024, r2 = 1

Consider the sequence of circles tangent to sides AB and BC. Their centers lie on a
    line parallel to side AC. Let L be the length of side AC.

The length of the segment parallel to AC at distance d from AC is
L * (r - d)/r.
This length should be equal to (n-1) * 2r1 because the distance between consecutive
    centers is 2r1 and there are (n-1) gaps between n circles.

So we have:
L * (r - r1)/r = (n-1) * 2r1

For the first sequence (r1 = 34, n = 8):
L * (r - 34)/r = 7 * 2 * 34
```

```
L * (r - 34)/r = 476

For the second sequence (r1 = 1, n = 2024):
L * (r - 1)/r = 2023 * 2 * 1
L * (r - 1)/r = 4046

Now we have two equations:
1) L * (r - 34)/r = 476
2) L * (r - 1)/r = 4046

Divide equation 2 by equation 1:
(L * (r - 1)/r) / (L * (r - 34)/r) = 4046/476
(r - 1)/(r - 34) = 4046/476

Simplify 4046/476:
Both divisible by 2: 2023/238.
238 = 2 * 7 * 17; 2023 = 7 * 17^2.
So 2023/238 = 17/2.

So:
(r - 1)/(r - 34) = 17/2
2(r - 1) = 17(r - 34)
2r - 2 = 17r - 578
576 = 15r
r = 576/15 = 192/5

So the inradius r = 192/5. This is in simplest form (192 and 5 are coprime). So m =
    192, n = 5, m+n = 197.

**Answer:** \boxed{197}

**Confidence:** High
```

**Judge**

**Migration signal.** The useful behavior is not just that RL selects a different team size. The reflection workflow routes the contradiction back through a critic, and the solver's final pass changes the mathematical model. The repair is visible in the exact equations: the failed first attempt collapses to $14 = 4046$, while the refined attempt uses $L(r - \rho)/r = 2\rho(n - 1)$, giving $r = 192/5$ and $m + n = 197$.