# MULTI-AGENT SYSTEM SEARCH VIA ACTIVE SUBSTRUCTURE-AWARE POLICY OPTIMIZATION

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

## ABSTRACT

LLMs enable multi-agent systems (MAS) to tackle complex tasks, but manually designing agent roles, prompts, and communication structures requires substantial expertise and effort. This motivates learning policies that construct query-specific MAS from execution reward. Existing approaches typically train these policies by repeatedly traversing a fixed set of training queries and assigning rewards at the workflow level. However, this overlooks differences in queries' evolving learning potential and obscures which substructures improve solution quality. In this paper, we propose Active Substructure-aware Policy Optimization (ASPO), a RL framework for query-level MAS search. ASPO introduces an Adaptive Query-Selection Mechanism (AQSM) that focuses training on queries at the policy's competence boundary: those it can solve but not yet reliably. A complementary discovery mechanism widens architectural exploration for hard queries, helping distinguish insufficient exploration from operator capability limits. Beyond query selection, ASPO introduces substructure-level rewards that measure output-quality gains within each action's descendant subgraph. These rewards guide proximal policy optimization to reinforce useful architectural refinements and discourage redundant or harmful computation. Together, these mechanisms prioritize learnable queries and provide fine-grained feedback for learning effective MASs. Across six benchmarks spanning mathematical reasoning, general question answering, and code generation, ASPO ranks first on every benchmark against twelve baselines.

## 1 INTRODUCTION

Multi-agent systems (MAS) built on large language models (LLMs) combine complementary agent capabilities through communication (Hong et al., 2024). However, manually designing their roles, prompts, and communication structures requires substantial expertise and repeated trial and error (Hu et al., 2025; Zhang et al., 2025c; Shang et al., 2025). In response, researchers have explored automating individual aspects of agentic systems, including prompt optimization (Khattab et al., 2024; Yuksekgonul et al., 2025) and multi-agent collaboration (Chen et al., 2024b; Liu et al., 2024). More broadly, recent work has extended automation to the design of complete MASs. One prominent direction is **task-level MAS search**, which seeks an effective MAS that can be reused across queries within a target task domain (Zhuge et al., 2024; Hu et al., 2025; Shang et al., 2025; Xu et al., 2026). For example, AFlow (Zhang et al., 2025c) represents workflows using code, and applies Monte Carlo Tree Search to guide their optimization. MASS (Zhou et al., 2026) optimizes agent prompts and searches a pruned space of workflow topologies. EvoMAS (Wei et al., 2026) uses evolutionary search to optimize workflow structures, agent roles, and prompts. However, task-level search follows a largely "one-size-fits-all" design objective, seeking a single MAS for an entire task domain rather than tailoring the architecture to each query.

To address this limitation, **query-level MAS search** adapts architectural decisions to individual queries rather than deploying a single shared MAS. Existing approaches can be grouped by how they construct query-specific MASs: (i) The first category constructs MAS by selecting combinations of agents from a predefined pool (Yang et al., 2026). MaAS (Zhang et al., 2025a) uses an RL-trained controller to select operators layer by layer from an agentic supernet. Puppeteer (Dang et al., 2025) instead conditions each decision on the query and accumulated intermediate outputs. (ii) The second category adapts communication graphs over a predefined agent pool (Wu et al., 2026; Zhang et al., 2026; Tastan et al., 2026). G-Designer (Zhang et al., 2025b) uses a variational graph autoencoder to



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: Query-level outcomes and group-wise accuracy on MATH, evaluated on the full training set after each training epoch, with and without AQSM. Queries are grouped by their outcomes without AQSM during warm-up (epochs 1–3): easy (3/3 correct), medium (1–2/3), and hard (0/3).

generate communication graphs, whereas MAGE (Zhao et al., 2026) uses energy-guided diffusion to generate communication graphs. (iii) The third category uses meta-agents to generate MASs. Some approaches use prompted LLMs to construct and revise workflows, such as AutoAgents (Chen et al., 2024a), EvoMAS (Hu et al., 2026), and ASPEC (Vu et al., 2026). Others train MAS-generation policies through supervised fine-tuning (Ye et al., 2025), preference optimization (Wang et al., 2025; 2026), or reinforcement learning (RL) (Ke et al., 2026; Nielsen et al., 2026; Gao et al., 2025).

**Reward-driven policy learning.** In query-level MAS search, discrete architectural decisions require downstream execution to evaluate their utility, making direct end-to-end differentiation difficult. Accordingly, most query-adaptive methods discussed above learn MAS policies from execution feedback through policy-gradient RL or preference optimization. Specifically, these policies guide operator and agent selection (Zhang et al., 2025a; Yang et al., 2026; Dang et al., 2025), communication topology generation (Zhang et al., 2025b; 2026), complete workflow generation (Ke et al., 2026; Wang et al., 2026; Nielsen et al., 2026), or higher-level decisions about architecture reuse (Vu et al., 2026). However, this widespread adoption raises a fundamental question: *do these methods actually obtain informative execution feedback for learning a MAS policy?* We examine this question from two perspectives: identifying queries that provide useful learning signals and distinguishing MAS substructures that contribute to successful execution.

**Question #1.** ***Uneven query informativeness:*** *Does policy learning prioritize queries that provide informative execution feedback?* Existing reward-driven approaches typically revisit a fixed set of training queries over multiple epochs, without adapting query selection to their evolving learning value. To examine query-level learning dynamics, we train our RL-based MAS search method with GPT-4o-mini on 210 MATH training queries for six epochs. Figure 1(a) shows the results under the strategy of uniformly revisiting all queries. Across all six epochs, 90 queries (42.9%) are consistently answered correctly, 66 (31.4%) exhibit both correct and incorrect outcomes, and 54 (25.7%) are never answered correctly. These trajectories suggest that queries differ in their remaining learning potential. Consistently solved queries offer limited room for improvement, while persistently unsuccessful ones may exceed the capabilities of the current model and operator pool. Uniformly revisiting both groups can therefore consume training budget while providing little correctness-based feedback to guide architectural decisions. **Conclusion #1: These observations motivate adaptive training query selection: identifying queries that remain learnable and prioritizing those whose execution feedback can most effectively improve the policy.**

**Question #2.** ***Coarse-grained substructure attribution:*** *Does execution feedback reveal which MAS substructures improve answer quality?* Existing reward-driven approaches commonly optimize architectural policies using workflow-level feedback, which may reinforce unnecessary steps in suc-

cessful workflows or overlook useful steps in failed ones (Zhang et al., 2025a; Nielsen et al., 2026; Wang et al., 2025; 2026), thereby limiting the policy's ability to learn effective and efficient multi-agent systems. We compare our method with MaAS (Zhang et al., 2025a) on MATH after training, scoring each layer's answer as 1 if correct and otherwise by its BERT-based similarity to the reference answer. For our method, average answer quality increases by $+0.033$, $+0.030$, $+0.024$, and $+0.018$ across successive transitions from L1 to L5. By contrast, MaAS yields gains of $+0.026$, $+0.005$, and $+0.012$ over the first three transitions, but answer quality drops by $0.011$ from L4 to L5. **Conclusion #2: This motivates fine-grained credit assignment to reinforce MAS decisions that improve answer quality and discourage redundant or harmful computation.**

Building on these two conclusions, we propose Active Substructure-aware Policy Optimization (ASPO), a framework for query-level MAS search. To address Question #1, we introduce an Adaptive Query-Selection Mechanism (AQSM) that prioritizes queries with remaining learning potential. Specifically, after warm-up epochs, we maintain a success-probability posterior for each query and prioritize queries at the current policy's competence boundary, namely those it can solve but not yet reliably. This reduces training effort spent on queries that are already consistently solved or unsolvable with the available agents. For queries not yet solved, a discovery mechanism explores additional rollouts and retains useful trajectories to avoid mistaking insufficient exploration for inability. To evaluate AQSM, we group queries in Figure 1(a, b) by their outcomes without AQSM over the three warm-up epochs: easy ($n = 95$), medium ($n = 46$), and hard ($n = 69$). After identical warm-up procedures, we compare per-group accuracy from epoch 4 onward, when AQSM is enabled in one run. As shown in Figure 1(c–e), overall accuracy without AQSM barely improves after warm-up, rising from 59.0% to 59.5%. By contrast, AQSM raises overall accuracy to 74.3%. Its largest gains occur on medium and hard queries, where accuracy reaches 82.6% and 34.8%, compared with 60.9% and 8.7% without AQSM, while maintaining high accuracy on easy queries. To address Question #2, we define a substructure-level reward based on improvements in answer quality from parent agents to their children. We then optimize the MAS policy using Proximal Policy Optimization (PPO) with these localized rewards, reinforcing decisions that produce useful downstream refinements. Appendix A places these two components in the context of related work on adaptive data selection and credit assignment in other domains. Our contributions are as follows:

- We identify two limitations of execution feedback in query-level MAS policy learning: uneven query informativeness and coarse-grained substructure attribution.
- We propose an active, substructure-aware RL framework that combines AQSM for adaptive query selection with substructure-level rewards for policy optimization.
- We evaluate ASPO on six benchmarks spanning mathematical reasoning, general question answering, and code generation, demonstrating its consistent superiority over twelve baselines.

## 2 Problem Formulation

**Multi-agent systems.** Following MaAS (Zhang et al., 2025a), we define the search space through a pool of agentic operators $\mathcal{O}$. Each operator specifies an LLM-based operation for generating or refining a response. A MAS is represented as a directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node instantiates an operator from $\mathcal{O}$ and each edge specifies the flow of outputs between nodes. Given a training set $\mathcal{D}_{\text{train}} = \{(q_n, y_n)\}_{n=1}^{N}$ of queries and reference answers, executing a graph on query $q_n$ produces intermediate responses and a final answer $\hat{y}_n$.

**Query-level MAS search.** We aim to learn an architectural policy $\pi_\theta$ that sequentially selects operators and their connections conditioned on the query and the current partial graph. This policy induces a distribution $\pi_\theta(\mathcal{G} \mid q)$ over valid graphs in the search space. Let $U(\mathcal{G}; q)$ denote the task utility obtained by executing $\mathcal{G}$ on query $q$. We optimize the policy to maximize expected utility:

$$\max_{\theta} \ \mathbb{E}_{(q,y)\sim\mathcal{D}_{\text{train}}} \mathbb{E}_{\mathcal{G}\sim\pi_\theta(\cdot|q)} \left[ U(\mathcal{G}; q) \right]. \tag{1}$$

With a limited budget, we study how to extract more informative training feedback to learn $\pi_\theta$.

## 3 Method

In this section, we introduce ASPO, a framework for query-level MAS search through **A**ctive **S**ubstructure-aware **P**olicy **O**ptimization. As illustrated in Figure 2, ASPO comprises two main

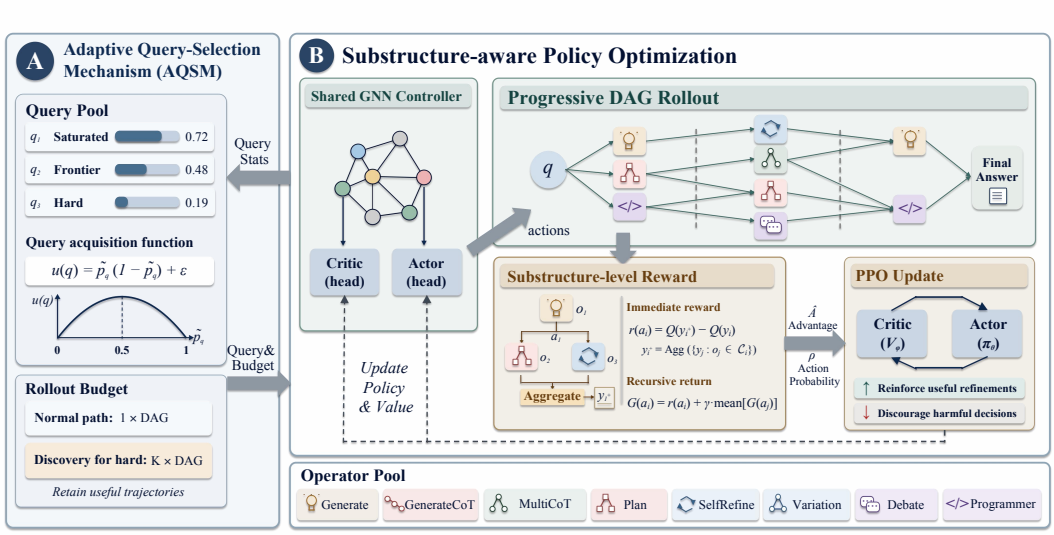



Figure 2: Overview of ASPO. (A) AQSM uses execution feedback to select queries and allocate rollouts. (B) The policy constructs query-specific MASs and learns from substructure-level rewards.

modules: (i) adaptive query selection (Section 3.3), which prioritizes training queries based on their evolving learning potential; and (ii) substructure-aware policy optimization (Section 3.2), which uses quality improvements within each action's descendant subgraph to compute localized returns for PPO updates. For each selected query, the architectural policy progressively constructs and executes a MAS (Section 3.1), providing execution feedback for both modules.

### 3.1 Policy-guided MAS Construction

We formulate MAS construction as a sequence of graph-expansion actions, starting from a root node representing the input query. A GCN-based state encoder captures the query and the current partial graph to guide these actions. Conditioned on this representation, the policy determines whether and how to expand each parent node using operators from $\mathcal{O}$.

**State representation.** To represent the query and the current partial graph, we use a state encoder based on a graph convolutional network (GCN) (Kipf & Welling, 2017). At expansion action $i$, let $\mathcal{G}_i$ denote the partial graph and $v_i$ the parent node to be expanded. We first project each node's operator embedding into the hidden space, then apply a GCN along parent-to-child edges with self-loops to obtain node representations. These representations are combined through attention pooling to form the graph representation $\mathbf{h}_{\mathcal{G}_i}$. We then concatenate $\mathbf{h}_{\mathcal{G}_i}$ with the parent representation $\mathbf{h}_{v_i}$ and query encoding $\mathbf{h}_q$ to form the state: $\mathbf{s}_i = [\mathbf{h}_{\mathcal{G}_i} \,\|\, \mathbf{h}_{v_i} \,\|\, \mathbf{h}_q]$, where $\mathbf{h}_q$ is obtained using a lightweight sentence encoder such as MiniLM (Wang et al., 2020) or Sentence-BERT (Reimers & Gurevych, 2019). The resulting state representation serves as a shared input to the actor and critic used in the policy optimization below. Implementation details are provided in Appendix C.1.

**Progressive DAG rollout.** Given the state representation $\mathbf{s}_i$, the policy expands the current parent node $v_i$ through the progressive DAG rollout shown in Figure 2. The actor implements this expansion through two action heads: a branching head and an operator-selection head. The branching head samples the number of children $k_i$ from $\pi_\theta^{\text{branch}}(k_i \mid \mathbf{s}_i)$, where $k_i = 0$ stops expansion of the current parent. When $k_i > 0$, the operator-selection head sequentially samples $k_i$ compatible operators without replacement from $\pi_\theta^{\text{op}}(o_{i,j} \mid \mathbf{s}_i, o_{i,<j})$, where $o_{i,<j}$ denotes the preceding selections. We denote the action for expanding $v_i$ by $a_i = (k_i, o_{i,1}, \ldots, o_{i,k_i})$, whose probability factorizes as

$$\pi_\theta(a_i \mid \mathbf{s}_i) = \pi_\theta^{\text{branch}}(k_i \mid \mathbf{s}_i) \prod_{j=1}^{k_i} \pi_\theta^{\text{op}}(o_{i,j} \mid \mathbf{s}_i, o_{i,<j}), \tag{2}$$

where $o_{i,j} \in \mathcal{O}$ denotes the operator selected for the $j$-th child. Each operator executes on the query and the outputs of its parent nodes. When parents in the same layer select the same operator, they share a single child node that receives all their outputs. Construction and execution proceed layer by layer until no operator node remains available for expansion or the construction budget is exhausted,

such as reaching the maximum depth. At that point, the outputs of all leaf operators are aggregated, for example through ensembling or voting, to produce the final answer.

### 3.2 Substructure-aware Policy Optimization

Building on the policy-guided MAS construction described above, we optimize the architectural policy using substructure-level execution feedback, as shown in Figure 2(B). For each action, we measure output-quality improvements within its descendant subgraph and aggregate them into an action-specific return. These returns guide PPO updates.

**Substructure-level rewards.** We assign rewards based on the quality improvement produced by each graph-expansion action. For node output $y_i$ and reference solution $y^*$, we define

$$Q(y_i) = \mathrm{sim}(y_i, y^*) + U_{\mathrm{task}}(y_i), \qquad U_{\mathrm{task}}(y_i) \in [0,1], \tag{3}$$

where $U_{\mathrm{task}}(y_i) \in [0,1]$ measures task completion using the benchmark evaluator, such as answer correctness or the fraction of test cases passed. However, this task-specific utility alone can provide sparse learning signals. To provide a finer-grained auxiliary signal, we introduce $\mathrm{sim}(y_i, y^*) \in [0,1]$, which measures alignment with the reference solution using a task-dependent similarity function (detailed in Appendix C.2). For action $a_i$ expanding node $v_i$, we then aggregate the outputs of its children $\mathcal{C}_i$ and measure their collective improvement over the parent:

$$y_i^+ = \mathrm{Agg}\big(\{y_j : j \in \mathcal{C}_i\}\big), \qquad r(a_i) = Q(y_i^+) - Q(y_i). \tag{4}$$

This local aggregation evaluates the collective outcome of an expansion rather than scoring each child independently. Finally, to account for downstream refinements and execution cost, we define the substructure-level rewards $G(a_i)$ over the descendant subgraph:

$$G(a_i) = r(a_i) + \frac{\gamma}{|\mathcal{C}_i|} \sum_{j \in \mathcal{C}_i} G(a_j) - \lambda\, \Delta C_i, \tag{5}$$

where $a_j$ denotes the expansion action at child node $v_j$, $\gamma \in [0,1]$ discounts more distant returns, and $\lambda \geq 0$ weights the incremental execution cost $\Delta C_i$, measured in FLOPs or tokens (Sardana et al., 2024). For stop actions ($k_i = 0$), we set $r(a_i) = G(a_i) = 0$. Each action thus receives a return determined by its own descendant subgraph rather than a shared workflow-level reward.

**PPO updates.** Using the substructure-level rewards, we now optimize the architectural policy $\pi_\theta$ as the actor in PPO (Schulman et al., 2017). To estimate the advantage of each expansion action, a critic $V_\phi(\mathbf{s}_i)$ first predicts the expected return from expanding the current parent node. For rollouts collected under $\pi_{\theta_{\mathrm{old}}}$, this gives the following advantage estimate and action probability ratio:

$$\hat{A}_i = G(a_i) - V_{\phi_{\mathrm{old}}}(\mathbf{s}_i), \qquad \rho_i(\theta) = \frac{\pi_\theta(a_i \mid \mathbf{s}_i)}{\pi_{\theta_{\mathrm{old}}}(a_i \mid \mathbf{s}_i)}, \tag{6}$$

where the action probability follows Equation 2 and $\phi_{\mathrm{old}}$ denotes the critic parameters used during rollout collection. The actor maximizes the clipped surrogate objective

$$\mathcal{L}_{\mathrm{policy}}(\theta) = \widehat{\mathbb{E}}_i \Big[ \min\Big( \rho_i(\theta)\hat{A}_i, \mathrm{clip}(\rho_i(\theta), 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}})\, \hat{A}_i \Big) \Big], \tag{7}$$

where $\widehat{\mathbb{E}}_i$ denotes the empirical average over collected expansion actions and $\epsilon_{\mathrm{clip}}$ controls clipping. This objective encourages positive-advantage actions and suppresses negative-advantage actions. On the other hand, the critic is trained to predict the expected substructure returns by minimizing

$$\mathcal{L}_{\mathrm{value}}(\phi) = \widehat{\mathbb{E}}_i \Big[ \big(V_\phi(\mathbf{s}_i) - G(a_i)\big)^2 \Big]. \tag{8}$$

The actor and critic share the state encoder, which receives gradients from both objectives.

### 3.3 Adaptive Query-Selection Mechanism

While substructure-level rewards provide localized feedback within each rollout, their usefulness also depends on the queries selected for rollouts. Instead of uniformly revisiting training queries as existing methods do, our Adaptive Query-Selection Mechanism (AQSM), shown in Figure 2(A), prioritizes queries based on their evolving learning potential and activates discovery for hard queries.

**Competence-boundary query selection.** We seek to allocate more training effort to queries at the current policy's competence boundary, rather than those it consistently solves or cannot yet solve with the available operators. Our insight is that queries the policy can solve but not yet reliably retain room for improvement and may provide informative execution feedback for policy learning. We therefore estimate each query's success probability under $\pi_\theta$ as a proxy for its current learning potential. For each query $q$, we maintain discounted success and failure counts, $S_q$ and $F_q$. Starting from a uniform prior, we represent its success probability $p_q$ using a Beta posterior:

$$p_q \sim \mathrm{Beta}(1 + S_q,\, 1 + F_q). \tag{9}$$

After each standard rollout, we update the counts using its binary success indicator $b_q \in \{0, 1\}$:

$$S_q \leftarrow \delta S_q + b_q, \qquad F_q \leftarrow \delta F_q + (1 - b_q), \tag{10}$$

where $\delta \in (0, 1]$ discounts earlier observations so that the estimate can track changes in policy capability. To prioritize queries at the competence boundary, we use binary outcome variance as a proxy for learning potential. For each query, we sample $\tilde{p}_q$ from its current posterior and compute:

$$w(q) = \tilde{p}_q(1 - \tilde{p}_q), \tag{11}$$

which peaks at $\tilde{p}_q = 0.5$, favoring queries that are solvable but not yet reliably solved. Appendix C.3 motivates this choice through an improvement bound proportional to $p_q(1-p_q)$ for a gradient step on query success probability, and explains how the discounted posterior sustains exploration throughout training. After $T_{\mathrm{w}}$ warm-up epochs, we activate **adaptive query selection** by redrawing $\tilde{p}_q$ before each batch and sampling queries in proportion to the resulting scores.

**Discovery for difficult queries.** However, queries downweighted for remaining consistently unsolved may fall into two cases: those beyond the current capabilities of the available models and operators, and those for which effective architectures have not yet been explored. We therefore introduce a **discovery mode** to distinguish insufficient exploration from capability limits. Specifically, after the warm-up epochs, a sampled query enters discovery mode if none of its previous rollouts has yielded a correct answer. For each query in discovery mode, we collect $K$ rollouts under $\pi_{\theta_{\mathrm{old}}}$, subject to the remaining per-epoch budget, compared with one for each other sampled query. To focus PPO updates on informative outcomes, we retain a discovery trajectory only if its final answer is correct or at least one architectural action has a positive substructure advantage, $\hat{A}_i > 0$. Discovery thus helps distinguish underexplored queries from those beyond current capabilities, helping them move toward the competence boundary and increase their future sampling weights.

### 3.4 Overall Algorithm

Algorithm 1 in Appendix C.4 summarizes the ASPO training procedure. After warm-up, AQSM selects queries for training and activates discovery for sampled queries that have never been solved. For each query, the policy progressively constructs and executes MAS, using substructure rewards for PPO updates. Once trained, the policy $\pi_\theta$ constructs a MAS tailored to each query at inference.

## 4 Experiments

### 4.1 Experimental Setup

**Benchmarks and metrics.** We evaluate ASPO on six public benchmarks covering three domains: (i) **mathematical reasoning**, GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021); (ii) **general question answering**, HotpotQA (Yang et al., 2018) and MMLU-Pro (Wang et al., 2024); and (iii) **code generation**, MBPP+ and HumanEval+ (Liu et al., 2023). We report accuracy for GSM8K, MATH and MMLU-Pro, F1 for HotpotQA, and pass@1 for MBPP+ and HumanEval+. Data sources, selection procedures, and train/test splits are detailed in Appendix D.1.

**Baselines.** We compare ASPO with twelve baselines spanning four categories: (i) **single-agent methods**, Vanilla and CoT (Wei et al., 2022); (ii) **multi-agent collaboration**, AgentVerse (Chen et al., 2024b) and DyLAN (Liu et al., 2024); (iii) **task-level MAS search**, ADAS (Hu et al., 2025), AFlow (Zhang et al., 2025c) and EvoMAS (Wei et al., 2026); and (iv) **query-level MAS search**, which we group following Section 1: *agent selection*, MaAS (Zhang et al., 2025a), RAAS (Yang et al., 2026) and Puppeteer (Dang et al., 2025) (ASPO's category and its most direct baselines);

| Method | GSM8K | MATH | HotpotQA | MMLU-Pro | MBPP+ | HumanEval+ | Avg. | AVG-R |
|---|---|---|---|---|---|---|---|---|
| *Single-agent methods* | | | | | | | | |
| Vanilla | $48.58_{\pm 1.70}$ | $47.48_{\pm 2.51}$ | $76.53_{\pm 2.69}$ | $61.83_{\pm 1.87}$ | $75.40_{\pm 2.34}$ | $81.21_{\pm 2.27}$ | 65.17 | 10.67 |
| CoT | $71.58_{\pm 1.76}$ | $48.44_{\pm 1.27}$ | $76.32_{\pm 1.68}$ | $73.17_{\pm 1.81}$ | $76.32_{\pm 2.11}$ | $83.94_{\pm 1.13}$ | 71.63 | 8.17 |
| *Multi-agent collaboration* | | | | | | | | |
| AgentVerse | $73.08_{\pm 1.03}$ | $51.77_{\pm 1.00}$ | $77.36_{\pm 0.77}$ | $71.98_{\pm 1.76}$ | $74.60_{\pm 2.02}$ | $82.12_{\pm 1.13}$ | 71.82 | 8.00 |
| DyLAN | $\underline{75.83_{\pm 1.96}}$ | $52.93_{\pm 1.67}$ | $76.84_{\pm 2.14}$ | $\underline{75.48_{\pm 1.85}}$ | $77.51_{\pm 1.78}$ | $\underline{85.76_{\pm 0.86}}$ | $\underline{74.06}$ | $\underline{3.50}$ |
| *Task-level MAS search* | | | | | | | | |
| ADAS | $72.25_{\pm 2.55}$ | $40.00_{\pm 1.17}$ | $75.37_{\pm 2.50}$ | $71.90_{\pm 2.29}$ | $74.21_{\pm 3.09}$ | $77.27_{\pm 2.97}$ | 68.50 | 11.50 |
| AFlow | $70.58_{\pm 0.92}$ | $52.24_{\pm 1.59}$ | $67.50_{\pm 0.91}$ | $74.92_{\pm 1.56}$ | $76.06_{\pm 1.46}$ | $84.55_{\pm 0.74}$ | 70.98 | 7.50 |
| EvoMAS | $72.83_{\pm 1.33}$ | $49.46_{\pm 1.50}$ | $76.26_{\pm 0.98}$ | $74.29_{\pm 0.89}$ | $77.38_{\pm 2.12}$ | $84.55_{\pm 1.48}$ | 72.46 | 6.50 |
| *Query-level MAS search* | | | | | | | | |
| MaAS | $74.67_{\pm 1.45}$ | $\underline{53.67_{\pm 1.92}}$ | $78.69_{\pm 1.12}$ | $67.78_{\pm 1.75}$ | $75.53_{\pm 2.11}$ | $82.73_{\pm 0.74}$ | 72.18 | 6.00 |
| RAAS | $73.58_{\pm 1.45}$ | $53.61_{\pm 2.01}$ | $77.84_{\pm 1.18}$ | $69.21_{\pm 1.95}$ | $74.47_{\pm 1.46}$ | $84.24_{\pm 1.55}$ | 72.16 | 6.50 |
| Puppeteer | $73.42_{\pm 1.84}$ | $44.69_{\pm 2.00}$ | $77.31_{\pm 1.20}$ | $73.89_{\pm 0.98}$ | $78.44_{\pm 1.46}$ | $69.39_{\pm 1.87}$ | 69.52 | 7.67 |
| RADAR | $73.33_{\pm 1.84}$ | $47.28_{\pm 1.60}$ | $\underline{78.76_{\pm 0.70}}$ | $73.97_{\pm 1.76}$ | $\underline{82.80_{\pm 0.82}}$ | $83.64_{\pm 0.74}$ | 73.30 | 5.67 |
| ASPEC | $74.17_{\pm 1.96}$ | $52.38_{\pm 1.60}$ | $76.06_{\pm 1.94}$ | $67.22_{\pm 1.46}$ | $76.59_{\pm 1.80}$ | $80.30_{\pm 1.87}$ | 71.12 | 8.17 |
| **ASPO (Ours)** | $\mathbf{79.08_{\pm 1.25}}$ | $\mathbf{57.89_{\pm 1.11}}$ | $\mathbf{81.00_{\pm 1.17}}$ | $\mathbf{77.70_{\pm 0.88}}$ | $\mathbf{83.47_{\pm 0.82}}$ | $\mathbf{90.00_{\pm 1.48}}$ | **78.19** | **1.00** |

Table 1: Main results with `gpt-4o-mini`, reported as $\text{mean}_{\pm\text{std}}$ over three runs. **Bold** marks the best and underline the runner-up. AVG-R is the average rank (lower is better).

*communication topology*, RADAR (Zhang et al., 2026); and *meta-agents*, ASPEC (Vu et al., 2026). Each baseline is run with its official implementation, as detailed in Appendix D.3.

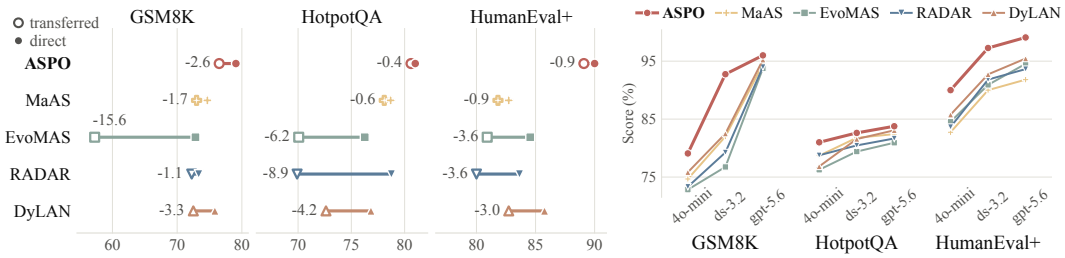
**Implementation details.** The search space of ASPO comprises ten agents described in Appendix D.7, which is also shared with the three agent-selection baselines. We use `gpt-4o-mini-2024-07-18` (OpenAI, 2024) as the LLM backbone for all methods, with temperature 1 for ASPO. We set the maximum number of layers to 5 and the maximum number of children per node to 2. For the substructure reward, we set the cost coefficient as $\lambda = 0.01$. For AQSM, we discount query statistics with $\delta = 0.9$ and use a discovery budget of $K = 3$ rollouts per query. We train for $T = 6$ epochs with $T_{\text{w}} = 3$ warm-up epochs, and cap the number of rollouts per epoch at the size of the training set. Appendix D.4 details PPO training and aggregation rules, and provides a summary table linking hyperparameter settings to their sensitivity analyses.

### 4.2 Main Results

Table 1 reports performance on the six benchmarks for ASPO and the twelve baselines. From these results we draw three conclusions. **(i) ASPO ranks first on every benchmark, while no baseline holds a stable position.** Its average of 78.19 exceeds the 74.06 of the strongest baseline, DyLAN, whose average rank is 3.50. The baselines are far less consistent: MaAS ranks second on MATH but eleventh on MMLU-Pro, Puppeteer third on MBPP+ but thirteenth on HumanEval+, and the runner-up position rotates among DyLAN, RADAR and MaAS. **(ii) The gain holds in every task domain.** Averaged within each domain, ASPO leads the strongest baseline for that domain by 4.1 points on mathematical reasoning, 3.5 on code generation and 3.0 on general question answering, and the identity of that baseline changes from domain to domain. **(iii) With the same agent pool and the same learning paradigm, the advantage comes from how the policy is trained.** MaAS, RAAS and Puppeteer also learn agent selection from execution feedback with RL, yet ASPO exceeds the best of the three by 2.3 to 5.8 points on every benchmark. We attribute this to the two components that distinguish our framework: AQSM focuses rollouts on queries with remaining learning potential, and substructure-level rewards credit individual expansion actions rather than scoring the MAS as a whole. Section 4.5 isolates the contribution of each component.

### 4.3 Transferability

In this section, we compare the top five methods in Table 1 to evaluate whether they transfer to task domains and LLM backbones that differ from those used during training. To limit experimental cost, this and subsequent analyses use three representative benchmarks: GSM8K, HotpotQA, and HumanEval+, selected by their agreement with the overall method ranking (Appendix D.2). (i)




(a) Task transfer: gap between transfer and direct training.

(b) LLM transfer: score per backbone.

Figure 3: Transferability of the learned policy, for the top-5 methods on representative benchmarks.

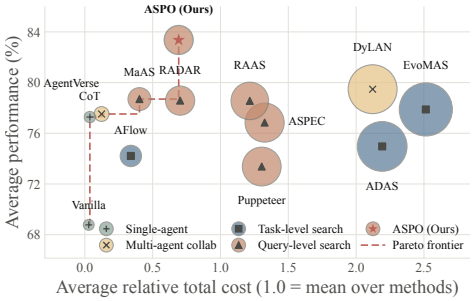



Figure 4: Cost-performance trade-off.

| | GSM8K | HotpotQA | HumanEval+ | Avg. |
|---|---|---|---|---|
| **ASPO** | **79.08** | **81.00** | **90.00** | **83.36** |
| *Adaptive query selection (Section 3.3)* | | | | |
| *w/o* discovery | $76.75_{-2.33}$ | $79.28_{-1.72}$ | $89.09_{-0.91}$ | $81.71_{-1.65}$ |
| *w/o* posterior | $75.75_{-3.33}$ | $79.04_{-1.96}$ | $87.27_{-2.73}$ | $80.69_{-2.67}$ |
| *w/o* AQSM | $75.25_{-3.83}$ | $78.57_{-2.43}$ | $88.18_{-1.82}$ | $80.67_{-2.69}$ |
| *Substructure-aware reward (Section 3.2)* | | | | |
| *w/o* similarity | $76.50_{-2.58}$ | $79.89_{-1.11}$ | $86.36_{-3.64}$ | $80.92_{-2.44}$ |
| *w/o* sub-reward | $75.50_{-3.58}$ | $78.83_{-2.17}$ | $86.36_{-3.64}$ | $80.23_{-3.13}$ |

Table 2: Ablation study: subscripts give the drop.

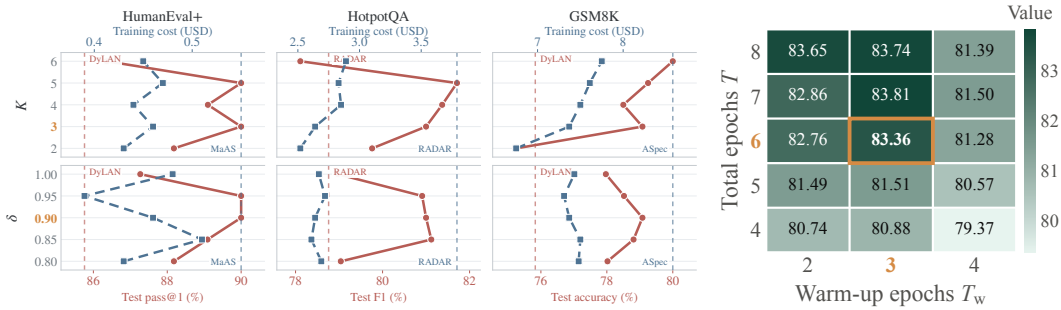
**Transfer across task domains.** We train each method on MATH and evaluate it on the three representative benchmarks without further adaptation, using `gpt-4o-mini` throughout. Figure 3(a) compares these results with training directly on each target benchmark. ASPO retains the highest score on all three benchmarks, while ASPO and MaAS show smaller average performance drops than the other methods. By contrast, EvoMAS transfers a fixed MAS optimized on MATH and loses 8.5 points on average. DyLAN and RADAR retain the agent team or role pool fixed during training, limiting their ability to adapt to the target task's role requirements. Their performance therefore drops to varying degrees depending on how well the retained roles suit the target task, with DyLAN losing 3.5 points on average and RADAR losing 8.9 points on HotpotQA. (ii) **Transfer across LLM backbones.** Here, training and inference use the same benchmark but different LLM backbones: each method is trained with `gpt-4o-mini` and evaluated with `deepseek-v3.2` (DeepSeek-AI, 2025) or `gpt-5.6-terra` (OpenAI, 2026). As shown in Figure 3(b), the ordering of the five methods remains largely consistent across different LLM backbones, with ASPO ranking first in all nine benchmark–backbone combinations. This suggests that the comparative advantage of the learned policy persists across execution LLMs.

### 4.4 Cost Analysis

We next examine the computational cost of all methods. Figure 4 compares average performance against total monetary cost on the three representative benchmarks, where costs are normalized by the mean across methods within each benchmark before averaging. We draw two conclusions. **(i) ASPO occupies the highest-performance end of the Pareto frontier.** ASPO achieves the highest average performance of 83.36 at a normalized cost of 0.69. MaAS costs less (0.40) but scores 4.66 points lower, while RADAR incurs a similar cost (0.70) yet scores 4.78 points lower. **(ii) ASPO achieves higher performance without relying on greater expenditure.** Every method more expensive than ASPO performs worse. In particular, compared with DyLAN, the strongest baseline, ASPO improves performance by 3.88 points while reducing cost by 67.4%. Appendix D.5 further reports per-benchmark training and inference costs and compares wall-clock times.

### 4.5 Ablation Study

In this section, we isolate the contribution of each component of ASPO by removing one at a time. Table 2 reports the results, where every variant loses accuracy on every benchmark. **(i) Ablation**




(a) Posterior discount $\delta$ and discovery budget $K$. (b) Epoch settings.

Figure 5: Sensitivity analysis of ASPO hyperparameters over three representative benchmarks.

**of adaptive query selection.** Removing discovery costs 1.65 points, highlighting the value of additional exploration for unsolved queries. Disabling AQSM entirely restores uniform query selection and increases the drop to 2.69 points. Moreover, the mechanism helps only if $p_q$ is estimated well: replacing it with the raw empirical success rate costs 2.67, almost as much. **(ii) Ablation of substructure-aware policy optimization.** The *w/o* sub-reward variant replaces substructure-level rewards with an end-to-end reward (as in MaAS) and costs 3.13 points on average, since no individual expansion receives credit of its own. Even with this attribution retained, removing semantic similarity causes a 2.44-point drop. Because the similarity term provides finer-grained feedback on intermediate refinements that may not change the task evaluator's score. **(iii) Component contributions vary across benchmarks.** Removing AQSM hurts GSM8K and HotpotQA more than HumanEval+, suggesting that query selection benefits from variation in query difficulty. Conversely, removing substructure-level rewards hurts GSM8K and HumanEval+ more than HotpotQA, possibly because derivations and programs offer richer signals of partial progress than short answers.

### 4.6 Sensitivity Analysis

In this section, we examine ASPO's hyperparameter sensitivity for its two main components, following the grouping in Table 5. **(i) Adaptive query selection.** Figure 5(a) examines $\delta$ and $K$, with red and blue curves showing test performance and training cost, respectively. The matching vertical references indicate the strongest baseline's performance and the baseline cost immediately above ASPO's default cost. Performance remains relatively stable for $\delta \in [0.85, 0.95]$, a range that includes our default $\delta = 0.9$. More aggressive forgetting ($\delta = 0.8$) or retaining all past observations ($\delta = 1$) reduces performance, suggesting the importance of balancing recent and historical feedback. Performance remains relatively stable for $K \in [3, 5]$, which includes our default $K = 3$, while smaller or larger budgets can cause substantial drops on some benchmarks. Across most settings, performance changes are smaller than ASPO's margin over the strongest baseline, preserving its advantage. Exceptions occur at the discovery-budget extremes: $K = 2$ on GSM8K and $K = 6$ on HotpotQA. **(ii) Substructure-aware optimization.** Figure 5(b) reports average performance for different training and warm-up epochs. Performance largely plateaus beyond our default training length of $T = 6$. A moderate warm-up of $T_w = 3$ generally performs better than shorter or longer warm-up periods. We further examine the architecture-related hyperparameters, including $\lambda$ and the maximum numbers of layers and children per node, in Appendix D.6.

## 5 Conclusion

In this paper, we presented ASPO, which combines adaptive query selection and discovery with substructure-level rewards to learn query-specific MAS policies. ASPO outperforms twelve baselines across six benchmarks and retains its advantage across task domains and LLM backbones. Ablations validate both components, while cost analyses show favorable performance–cost trade-offs. ASPO currently searches within a predefined operator pool and uses reference solutions to compute training rewards. Future work could explore applying ASPO to dynamically changing operator pools, and alternative training signals for tasks without reliable reference solutions (Appendix B).

## AI USE STATEMENT

We used generative AI tools to polish the manuscript's wording and assist with writing scripts for experimental analysis. Three authors have reviewed the language revisions to ensure that they preserved the intended meaning and technical accuracy. AI-assisted scripts were inspected and tested for correctness by three authors, and their outputs were checked against the experimental records. We take full responsibility for the final content of this work, including all AI-assisted text and code.

## ETHICS STATEMENT

Our work studies automated multi-agent system design through reinforcement learning. Given this, we foresee no issues regarding fairness, privacy, or security, or any other harmful societal or ethical implications outside broader considerations for the field itself. However, we do note that reliance on expensive LLMs may still create financial barriers to access.

## REPRODUCIBILITY STATEMENT

We provide ASPO code (`aspo/aspo_code/`), benchmark splits (`aspo/aspo_data/`), and installation, data preparation, training, and evaluation instructions in the supplementary material. Appendix D.1 documents dataset sources and curation, while Appendices D.3 and D.4 detail baseline configurations and ASPO implementation settings. The training procedure is summarized in Appendix C.4, with the operator pool specified in Appendix D.7. The assumptions and proofs underlying our query-selection analysis are provided in Appendix C.3.

## REFERENCES

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje Karlsson, Jie Fu, and Yemin Shi. Autoagents: a framework for automatic agent generation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024a. ISBN 978-1-956792-04-1. doi: 10.24963/ijcai.2024/3. URL https://doi.org/10.24963/ijcai.2024/3.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/578e65cdee35d00c708d4c64bce32971-Paper-Conference.pdf.

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

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. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 165025–165059. Curran Associates, Inc., 2025. doi: 10.52202/085713-5502. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f1320d2e2842169c6fc89dcbd80e94d0-Paper-Conference.pdf.

DeepSeek-AI. DeepSeek-V3.2 model card, 2025. URL https://huggingface.co/deepseek-ai/DeepSeek-V3.2.

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.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, zili wang, Steven Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/6507b115562bb0a305f1958ccc87355a-Paper-Conference.pdf.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Yuntong Hu, Yuting Zhang, Matthew Trager, Yi Zhang, Shuo Yang, Wei Xia, and Stefano Soatto. EvoMAS: Evolutionary generation of multi-agent systems. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ic0AGRIkmY.

Zixuan Ke, Yifei Ming, Austin Xu, Ryan Chin, Xuan-Phi Nguyen, Prathyusha Jwalapuram, Jiayu Wang, Semih Yavuz, Caiming Xiong, and Shafiq Joty. MAS-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=3fGXBm4c3S.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54928–54958, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to orchestrate agents in natural language with the conductor. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 135686–135724, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/dbb8193ad7e6fcbc7bb62ed9ee835110-Paper-Conference.pdf.

OpenAI. GPT-4o-mini model card, 2024. URL https://platform.openai.com/docs/models/gpt-4o-mini.

OpenAI. GPT-5.6 Terra model card, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.6-terra.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43445–43460. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/sardana24a.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/0ae94013da7cd459402fd77874e09ee3-Paper-Conference.pdf.

Nurbek Tastan, Samuel Horváth, and Karthik Nandakumar. Stochastic self-organization in multi-agent systems. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 40242–40271, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/43833a8a514392cb19ca09b52cf4b5b7-Paper-Conference.pdf.

Myan Vu, Harrish Ayyanar, PANG JIANG, Anwiketh Reddy, and Mayank Goel. Automated stateful specialization for adaptive agent systems. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 90290–90313, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/9230cf96fb02545f83e34dfe2810e204-Paper-Conference.pdf.

Kun Wang, Guibin Zhang, ManKit Ye, Xinyu Deng, Dongxia Wang, Xiaobin Hu, Jinyang Guo, Yang Liu, and Yufei Guo. Masˆ2: Self-generative, self-configuring, self-rectifying multi-agent systems. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 113586–113613, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/b88318174aad2cc174a4e05ab6bfad80-Paper-Conference.pdf.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5776–5788. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

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

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi

Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ad236edc564f3e3156e1b2feafb99a24-Paper-Datasets_and_Benchmarks_Track.pdf.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Yangbo Wei, Zhen Huang, Ronghao Xu, Hong Wang, and WEI W. XING. EvoMAS: Heuristics in the loop—evolving smarter agentic workflows. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=cdWnpXkcQo.

Tongtong Wu, Yanming Li, Ziye Tang, Chen Jiang, Linhao Luo, Guilin Qi, Shirui Pan, and Gholamreza Haffari. Card: Towards conditional design of multi-agent topological structures. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 156705–156729, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/fdc05db30de212a4137efd86208b6bf6-Paper-Conference.pdf.

Iris Xu, Guangtao Zeng, Zexue He, Charles Jin, Aldo Pareja, Dan Gutfreund, Chuang Gan, and Zhang-Wei Hong. Boad: Discovering hierarchical software engineering agents via bandit optimization. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 7991–8015, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/0d89023ee7f8728fb6018d5c84a720bf-Paper-Conference.pdf.

Jiayi Yang, Guancheng Wan, Man Zhang, and Mang Ye. Raas: Llm agentic system architecture search with grpo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 34470–34479, June 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

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. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 72063–72090. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/ye25g.html.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75834–75852. PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76678–76692. PMLR, 13–19 Jul 2025b. URL https://proceedings.mlr.press/v267/zhang25cu.html.

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. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025c. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5492ecbce4439401798dcd2c90be94cd-Paper-Conference.pdf.

Zhen Zhang, Wanjing Zhou, Juncheng Li, Hao Fei, Jun Wen, and Wei Ji. RADAR: Redundancy-aware diffusion for multi-agent communication structure generation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=GtpiqFaJtZ.

GuanHao Zhao, Wenbo Lu, Cheng Cheng, Zhenya Huang, Wei Song, Zhiding Liu, Runze Wu, and Enhong Chen. Navigating the energy landscape of collaboration: Multi-agent communication graph generation via score-based diffusion. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=lhSg9cJUKY.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Arik. Multi-agent design: Optimizing agents with better prompts and topologies. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 15844–15872, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/1ab4e0e8f35078109bc78d7b465d306f-Paper-Conference.pdf.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

# A Related Work

The Introduction (Section 1) reviews the background of MAS search and identifies two limitations of existing query-level methods that motivate ASPO: uniform allocation of training effort across queries and coarse workflow-level rewards. Although these ideas remain underexplored in MAS search, they connect to broader research on adaptive data selection and credit assignment. We therefore review work on adaptive training data selection and credit assignment, highlighting their connections to ASPO and the differences in their objectives and mechanisms.

**Adaptive Training Data Selection.** Early work on active learning sought to improve label efficiency by selecting informative examples for annotation (Settles, 2009). Representative strategies include uncertainty sampling (Lewis & Gale, 1994) and expected-gradient-length criteria that estimate an example's influence on model updates (Settles et al., 2007). Related ideas extend to reinforcement learning, where the budget is spent on task interactions and rollout execution. Goal GAN generates goals within an intermediate success-rate range to build curricula that adapt to policy capability (Florensa et al., 2018). In LLM reinforcement learning, DAPO filters prompt groups with uniformly correct or incorrect outcomes (Yu et al., 2025), while GRESO uses historical feedback to skip uninformative prompts before rollout generation (Zheng et al., 2025). MoPPS uses discounted Beta posteriors and posterior sampling to select prompts near a target success rate (Qu et al., 2026). ASPO extends this line of work in two ways. First, it brings adaptive training data selection to query-specific MAS architecture policy learning. Second, while existing selection criteria often downweight or filter queries at both extremes of the success-rate spectrum, ASPO treats these extremes asymmetrically: consistently solved queries receive less training effort, whereas sampled queries that have never been solved receive additional rollouts through discovery. This discovery mode helps distinguish queries that remain unsolved due to insufficient exploration from those beyond the capabilities of the current policy and operator pool.

**Credit Assignment in Multi-Agent Systems.** Shared global rewards make it difficult to distinguish individual agents' contributions, motivating more localized credit assignment. COMA, for example, uses a counterfactual baseline to estimate an agent's action contribution while holding other agents' actions fixed (Foerster et al., 2018). In LLM reasoning, process supervision provides feedback on intermediate steps (Lightman et al., 2024). Subsequent work uses Monte Carlo value estimation or implicit reward models trained on outcome labels to obtain fine-grained signals without manual step-level annotations (Kazemnejad et al., 2025; Yuan et al., 2025). For interactive agents, GraphGPO assigns transition credit using a state-transition graph assembled from multiple rollouts (Cheng et al., 2026). SHARP addresses credit assignment among LLM agents through Shapley-based marginal rewards (Li et al., 2026). ASPO differs from these methods in both its optimization target and credit assignment mechanism. First, these methods primarily assign credit to reasoning steps or agent actions within an execution framework, whereas ASPO uses localized feedback to optimize the MAS architecture policy, guiding operator selection, branching, and termination. Second, rather than scoring individual reasoning steps or estimating agents' marginal contributions, ASPO assigns credit to each graph expansion by measuring the quality improvement of its aggregated child outputs over the parent output and incorporating discounted returns from its descendant subgraph together with execution costs.

# B Limitations and Future Work

**Predefined operator pool.** ASPO composes operators from a predefined pool, so its capabilities remain bounded by the available operators and underlying LLMs. Discovery explores additional compositions but cannot introduce missing capabilities. Future work could extend ASPO to dynamically changing operator pools and jointly learn operator design and composition.

**Dependence on reference-based training feedback.** ASPO relies on reference solutions and task evaluators for training feedback, limiting its direct application to tasks without reliable references or success criteria. Moreover, semantic similarity may also overlook valid alternative solutions or reward superficially similar but incorrect responses. Future work could explore executable verification, human preferences, or learned evaluators as alternative signals for query selection and architectural credit assignment.

# C TECHNICAL DETAILS

## C.1 STATE ENCODER IMPLEMENTATION

**Node initialization.** We use a hidden dimension of $d = 128$ throughout the graph encoder. Each operator, including the root, has an embedding initialized from its natural-language description using the frozen sentence encoder `all-MiniLM-L6-v2` (Wang et al., 2020)[1]. The resulting 384-dimensional embeddings are projected to $d$ dimensions to initialize an operator embedding table. This initialization projection is applied only once; the table is initially frozen and subsequently optimized during policy training. At expansion decision $i$, the input feature of node $v$ combines its operator embedding $\mathbf{e}_{o_v}$ with an indicator identifying the current parent $v_i$:

$$\mathbf{x}_v = \mathbf{W}_{\text{in}}[\mathbf{e}_{o_v} \| \mathbb{I}(v = v_i)] + \mathbf{b}_{\text{in}}, \qquad \mathbf{x}_v \in \mathbb{R}^d. \tag{12}$$

Nodes instantiating the same operator share its embedding, while their structural contexts are captured by graph convolution.

**Graph convolution.** We encode the current partial graph using two GCN layers (Kipf & Welling, 2017), implemented with normalized `GCNConv` layers in PyTorch Geometric. Edges retain the parent-to-child direction, and every node receives a self-loop. Both layers preserve the hidden dimension $d$. We apply ReLU after the first layer, omit the activation after the second layer, and use no dropout. The resulting representation $\mathbf{h}_v$ incorporates information from the node itself and upstream nodes reachable within two message-passing steps.

**Graph-level pooling.** A single-head attention pooling module summarizes the node representations into $\mathbf{h}_{\mathcal{G}_i}$. Let $\mathcal{V}_i$ denote the nodes of the current partial graph and $\mathbf{u} \in \mathbb{R}^d$ a trainable pooling vector. We compute

$$\alpha_v = \frac{\exp(\mathbf{u}^\top \mathbf{h}_v)}{\sum_{z \in \mathcal{V}_i} \exp(\mathbf{u}^\top \mathbf{h}_z)}, \qquad \mathbf{h}_{\mathcal{G}_i} = \sum_{v \in \mathcal{V}_i} \alpha_v \mathbf{h}_v. \tag{13}$$

The pooling vector is shared across queries, while the weights depend on the encoded partial graph and the current-parent indicator.

**State fusion and output heads.** We encode the query with the same frozen sentence encoder and apply a separate trainable linear projection to obtain $\mathbf{h}_q \in \mathbb{R}^d$. Concatenating $\mathbf{h}_{\mathcal{G}_i}$, $\mathbf{h}_{v_i}$, and $\mathbf{h}_q$ yields the 384-dimensional state $\mathbf{s}_i$. The branching head is a two-layer MLP with a $d$-dimensional hidden layer and a Tanh activation. The operator-selection head linearly projects $\mathbf{s}_i$ into the operator embedding space and computes dot-product scores against the operator embedding table. The critic uses a linear layer to map the same state to a scalar value. The graph encoder, pooling module, and state projections are shared by the actor and critic.

## C.2 SEMANTIC SIMILARITY

The similarity function is task-dependent. For mathematical reasoning and question answering, we match segments of the generated output $y_i$ and the reference solution $y^*$ in both directions. Let $\{g_i\}_{i=1}^m$ and $\{r_j\}_{j=1}^n$ denote the generated and reference segments, respectively, and let $e(\cdot)$ be a text encoder. We instantiate $e(\cdot)$ with the pretrained ROSCOE encoder (Golovneva et al., 2022)[2]. We use its final-layer `[CLS]` representation with $\ell_2$ normalization and keep the encoder fixed during policy training. Pairwise cosine similarities are $c_{ij} = \cos(e(g_i), e(r_j))$. We compute the average best-match scores in both directions and combine them using their harmonic mean:

$$P = \frac{1}{m}\sum_{i=1}^{m} \max_j c_{ij}, \qquad R = \frac{1}{n}\sum_{j=1}^{n} \max_i c_{ij}, \qquad \text{sim}_{\text{text}}(y_i, y^*) = \begin{cases} \frac{2PR}{P+R}, & P, R > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

Here, $P$ measures alignment with the reference, while $R$ measures coverage of the reference content. We define this similarity as zero if either solution is empty.

For code generation, we use a model-free AST-based structural similarity. We parse each Python program into an abstract syntax tree and count the occurrences of each node-type label, producing

[1] `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

[2] `https://huggingface.co/facebook/roscoe-512-roberta-base`

count vectors $\mathbf{v}(y_i)$ and $\mathbf{v}(y^*)$. Operator names are included in the labels: for example, binary addition is represented as `BinOp:Add`, and comparisons as `Compare:Lt` or `Compare:Eq`. Identifiers and literal values are not included, so programs such as $a + b$ and $x + y$ receive the same counts. The code similarity is the cosine similarity of these vectors, clipped to $[0, 1]$:

$$\mathrm{sim}_{\mathrm{code}}(y_i, y^*) = \mathrm{clip}_{[0,1]}\left(\frac{\mathbf{v}(y_i)^\top \mathbf{v}(y^*)}{\|\mathbf{v}(y_i)\|_2 \, \|\mathbf{v}(y^*)\|_2}\right). \tag{15}$$

The score is set to zero if parsing fails for either program, or if the reference program is empty. Thus, the overall $\mathrm{sim}(y_i, y^*)$ uses $\mathrm{sim}_{\mathrm{code}}$ for code tasks and $\mathrm{sim}_{\mathrm{text}}$ for mathematical and question-answering tasks.

### C.3 Why the Competence-Boundary Score

This section explains why the acquisition score of Section 3.3 takes the form $w(q) = \tilde{p}_q(1 - \tilde{p}_q)$. We first show that $p_q(1 - p_q)$ upper-bounds how much a single policy update can improve a query's success probability, and then characterize the effect of drawing $\tilde{p}_q$ from the posterior rather than using its mean.

**Setup.** Fix a query $q$ and let $\tau = (\mathcal{G}, \xi)$ denote a rollout, where $\mathcal{G} \sim \pi_\theta(\cdot \mid q)$ is the sampled architecture and $\xi$ collects the execution randomness of the backbone LLM, which does not depend on $\theta$. Let $b(\tau) \in \{0, 1\}$ indicate whether the final answer satisfies the task's success criterion, and write $p_q(\theta) = \mathbb{E}_\tau[b(\tau)]$ for the success probability tracked by Equation 10. The construction budget bounds the depth and branching of $\mathcal{G}$ and the operator pool $\mathcal{O}$ is finite, so the trajectory space is finite; since $\mathcal{D}_{\mathrm{train}}$ is finite as well, we may set $L^2(\theta) = \max_{q' \in \mathcal{D}_{\mathrm{train}}} \mathbb{E}_\tau \|\nabla_\theta \log \pi_\theta(\mathcal{G} \mid q')\|^2 < \infty$, a quantity shared by all queries at the current $\theta$.

**Proposition 1** (Improvement bound)**.** *Under the setup above,*

$$\|\nabla_\theta \, p_q(\theta)\| \;\le\; L(\theta)\sqrt{p_q(\theta)\big(1 - p_q(\theta)\big)}. \tag{16}$$

*If in addition $p_q(\cdot)$ is $\beta$-smooth, then a gradient step $\theta' = \theta + \eta \nabla_\theta p_q(\theta)$ satisfies*

$$p_q(\theta') - p_q(\theta) \;\le\; \eta\Big(1 + \tfrac{\beta\eta}{2}\Big) L^2(\theta)\, p_q(\theta)\big(1 - p_q(\theta)\big). \tag{17}$$

*Proof.* By the score-function identity, $\nabla_\theta p_q = \mathbb{E}_\tau[b(\tau) \nabla_\theta \log \pi_\theta(\mathcal{G} \mid q)]$, where the score involves only $\mathcal{G}$ because $\xi$ does not depend on $\theta$. Since $\mathbb{E}_\tau[\nabla_\theta \log \pi_\theta(\mathcal{G} \mid q)] = \nabla_\theta \sum_{\mathcal{G}} \pi_\theta(\mathcal{G} \mid q) = 0$, subtracting the constant baseline $p_q$ leaves the gradient unchanged, giving $\nabla_\theta p_q = \mathbb{E}_\tau[(b(\tau) - p_q)\nabla_\theta \log \pi_\theta(\mathcal{G} \mid q)]$. Applying the Cauchy–Schwarz inequality and using $\mathbb{E}_\tau[(b(\tau) - p_q)^2] = \mathrm{Var}[b(\tau)] = p_q(1 - p_q)$ for a Bernoulli outcome yields Equation 16. For the second claim, $\beta$-smoothness gives $p_q(\theta') \le p_q(\theta) + \langle \nabla_\theta p_q, \theta' - \theta \rangle + \frac{\beta}{2}\|\theta' - \theta\|^2$; substituting $\theta' - \theta = \eta \nabla_\theta p_q$ and then Equation 16 yields Equation 17. □

**Interpretation.** The right-hand side of Equation 17 is the quantity $w(q)$ estimates, up to a factor shared by all queries at the current $\theta$. The bound therefore states that $p_q(1 - p_q)$ caps how much one update can raise a query's success probability. A query the policy solves almost always ($p_q \to 1$) or almost never ($p_q \to 0$) cannot be improved much by any single update, regardless of the direction the gradient takes. Sampling queries in proportion to $w(q)$ thus withholds budget from queries whose improvement is provably capped, which is the sense in which the competence boundary is the informative region. We sample in proportion to $w(q)$ rather than taking its maximizer because $p_q$ is estimated rather than known, and proportional sampling retains support on every training query.

**Proposition 2** (Effect of posterior sampling)**.** *Let $n_q = S_q + F_q$ and let $\hat{p}_q = (1 + S_q)/(2 + n_q)$ denote the mean of the posterior in Equation 9. Then*

$$\mathbb{E}_{\tilde{p}_q}\big[w(q)\big] = \frac{(1 + S_q)(1 + F_q)}{(2 + n_q)(3 + n_q)} = \frac{n_q + 2}{n_q + 3}\, \hat{p}_q(1 - \hat{p}_q). \tag{18}$$

*Moreover, under the update of Equation 10, the effective count satisfies $n_q < 1/(1 - \delta)$ for every query at every iteration.*

*Proof.* For $X \sim \mathrm{Beta}(\alpha, \beta)$ we have $\mathbb{E}[X(1-X)] = \mathbb{E}[X] - \mathbb{E}[X^2] = \alpha\beta/[(\alpha+\beta)(\alpha+\beta+1)]$. Setting $\alpha = 1 + S_q$ and $\beta = 1 + F_q$ gives the first equality, and dividing by $\hat{p}_q(1-\hat{p}_q) = (1+S_q)(1+F_q)/(2+n_q)^2$ gives the second. Summing Equation 10 over the two counts yields $n_q \leftarrow \delta n_q + 1$, so starting from $n_q = 0$ we obtain $n_q = (1-\delta^t)/(1-\delta) < 1/(1-\delta)$ after $t$ updates. □

**Interpretation.** Equation 18 shows that drawing $\tilde{p}_q$ from the posterior reproduces the plug-in score $\hat{p}_q(1-\hat{p}_q)$ in expectation, shrunk by a factor that approaches one as evidence accumulates. What posterior sampling adds is dispersion: the score is a random variable whose spread reflects how well $p_q$ is currently estimated, so a query with little recent evidence is not deterministically excluded. The second claim shows that the discount $\delta$ caps the effective evidence at $1/(1-\delta)$, which is 10 at our setting $\delta = 0.9$, so the posterior never collapses to a point mass and this dispersion persists throughout training. Beyond tracking changes in policy capability, $\delta$ thus also controls how much exploration the acquisition score retains. For illustration, at $\delta = 0.9$ a competence-boundary query with substantial evidence ($S_q = F_q = 4.5$) attains $\mathbb{E}[w(q)] \approx 0.229$, an unobserved query attains $1/6 \approx 0.167$, and a query observed at length but never or always solved attains $\approx 0.076$.

### C.4 ASPO Training Algorithm

**Algorithm 1** ASPO training

**Require:** Training set $\mathcal{D}_{\mathrm{train}}$, operator pool $\mathcal{O}$, epochs $T$, warm-up epochs $T_{\mathrm{w}}$, discovery budget $K$
**Ensure:** Learned architectural policy $\pi_\theta$

Initialize actor and critic parameters $\theta, \phi$
Set $S_q = F_q = 0$ and $\mathrm{solved}[q] = \mathrm{false}$ for every training query $q$
**for** epoch $e = 1, \ldots, T$ **do**
  $B_{\mathrm{rem}} \leftarrow |\mathcal{D}_{\mathrm{train}}|$
  **while** $B_{\mathrm{rem}} > 0$ **do**
    **if** $e \le T_{\mathrm{w}}$ **then**
      Take the next batch from a traversal of $\mathcal{D}_{\mathrm{train}}$
    **else**
      Draw $\tilde{p}_q$ from posteriors, refresh $w(q)$ (Equation 11), and sample a batch with weights $w(q)$
    **end if**
    Limit the batch size to at most $B_{\mathrm{rem}}$ queries
    Freeze rollout parameters $\theta_{\mathrm{old}} \leftarrow \theta$, $\phi_{\mathrm{old}} \leftarrow \phi$; clear buffer $\mathcal{B}$
    **for** each $(q, y^*)$ in the batch, while $B_{\mathrm{rem}} > 0$ **do**
      $d_q \leftarrow (e > T_{\mathrm{w}}) \wedge \neg\mathrm{solved}[q]$ ▷ Discovery flag
      $m_q \leftarrow K$ if $d_q$, otherwise $m_q \leftarrow 1$
      Collect $m_q$ DAG rollouts using $\pi_{\theta_{\mathrm{old}}}$ and $\mathcal{O}$ (Section 3.1)
      $B_{\mathrm{rem}} \leftarrow B_{\mathrm{rem}} - m_q$ ▷ Count all executed rollouts
      **for** each collected trajectory **do**
        Evaluate node outputs and final-answer correctness $b_q \in \{0, 1\}$
        Compute $r(a_i)$, $G(a_i)$ and $\hat{A}_i$ (Equations 4–6)
        **if** $\neg d_q$ or $b_q = 1$ or some $\hat{A}_i > 0$ **then**
          Add the trajectory, fixed PPO targets and old action probabilities to $\mathcal{B}$
        **end if**
        If $\neg d_q$, update $S_q, F_q$ using $b_q$ (Equation 10)
        $\mathrm{solved}[q] \leftarrow \mathrm{solved}[q] \vee (b_q = 1)$
      **end for**
    **end for**
    If $\mathcal{B} \neq \emptyset$, update actor, critic and shared encoder with PPO
  **end while**
**end for**
**return** $\pi_\theta$

# D EXPERIMENTAL SETUP

## D.1 BENCHMARKS AND DATA CURATION

We evaluate on six benchmarks covering mathematical reasoning (GSM8K, MATH), general question answering (HotpotQA, MMLU-Pro), and code generation (MBPP+, HumanEval+). For each benchmark, we select examples from publicly available datasets and split them into training and test sets at an approximately 1:2 ratio, using the same split for all compared methods. Sources and selection procedures are detailed below; Table 3 summarizes the sample counts.

**Selection toward harder problems.** For GSM8K and MATH we bias selection toward the harder end of each dataset, so that the comparison is not dominated by queries that a single LLM call already answers: we retain the lowest-solve-rate GSM8K problems and restrict MATH to Lv.5, as specified below. Absolute scores on these two splits are therefore well below those reported on the full benchmarks and should not be compared with them — in Table 1, Vanilla, a single LLM call, reaches only $48.6\%$ on the resulting GSM8K split. The remaining four benchmarks are not difficulty-filtered: HotpotQA and MMLU-Pro are sampled uniformly over their annotation grids, and MBPP+ and HumanEval+ are used in full.

**Mathematical reasoning.**

- **GSM8K** (Cobbe et al., 2021) — grade-school math word problems requiring multi-step arithmetic reasoning.[3] *Selection:* we select the $600$ problems with the lowest solve rates from the publicly available GSM8K difficulty annotations[4]. *Split:* $200$ train / $400$ test.
- **MATH** (Hendrycks et al., 2021) — competition mathematics problems, annotated with 7 subject types and $5$ difficulty levels (Lv.1–Lv.5).[5] *Selection:* we restrict to the hardest tier (Lv.5) and sample uniformly across the 7 subject types, keeping difficulty fixed at the highest level while balancing subject representation. *Split:* $210$ train / $490$ test.

**General question answering.**

- **HotpotQA** (Yang et al., 2018) — multi-hop QA, with two question types (*comparison* / *bridge*) and three difficulty labels (*easy* / *medium* / *hard*).[6] We use the distractor setting, in which each question is accompanied by supporting and distractor paragraphs, so answers are judged against the provided passages and no additional retrieval is required. *Selection:* we sample uniformly over the type $\times$ difficulty grid. *Split:* $210$ train / $420$ test.
- **MMLU-Pro** (Wang et al., 2024) — 10-option multiple-choice questions spanning $14$ subject categories (no difficulty labeling).[7] *Selection:* uniform sampling across the $14$ categories for both splits. *Split:* $210$ train / $420$ test.

**Code generation.**

- **MBPP+** (Liu et al., 2023) — the EvalPlus extension of MBPP (Austin et al., 2021), which augments each Python programming task with a substantially larger, higher-coverage test suite ($378$ problems; no difficulty/type labeling).[8] *Split:* $126$ train / $252$ test.
- **HumanEval+** (Liu et al., 2023) — the EvalPlus extension of HumanEval (Chen et al., 2021), with strengthened test cases ($164$ problems).[9] *Split:* $54$ train / $110$ test.

## D.2 REPRESENTATIVE BENCHMARK SELECTION

The analyses in Sections 4.3, 4.4, 4.5, and 4.6 repeat training and evaluation under many configurations: different execution backbones, ablated variants of the framework, and sweeps over individual

[3] https://huggingface.co/datasets/openai/gsm8k
[4] https://huggingface.co/datasets/lime-nlp/GSM8K_Difficulty
[5] https://huggingface.co/datasets/EleutherAI/hendrycks_math
[6] https://huggingface.co/datasets/hotpotqa/hotpot_qa
[7] https://huggingface.co/datasets/TIGER-Lab/MMLU-Pro
[8] https://huggingface.co/datasets/evalplus/mbppplus
[9] https://huggingface.co/datasets/evalplus/humanevalplus

| Benchmark | Task family | Train | Test | Total | Metric |
|---|---|---|---|---|---|
| GSM8K | Math | 200 | 400 | 600 | Accuracy |
| MATH | Math | 210 | 490 | 700 | Accuracy |
| HotpotQA | General QA | 210 | 420 | 630 | F1 |
| MMLU-Pro | General QA | 210 | 420 | 630 | Accuracy |
| MBPP+ | Coding | 126 | 252 | 378 | pass@1 |
| HumanEval+ | Coding | 54 | 110 | 164 | pass@1 |

Table 3: Curated train/test sample counts and evaluation metrics across benchmarks.

| Domain | Benchmark | Spearman | Kendall $\tau_b$ |
|---|---|---|---|
| Mathematical reasoning | **GSM8K** | **0.692** | **0.545** |
| | MATH | 0.609 | 0.494 |
| General question answering | **HotpotQA** | **0.656** | 0.520 |
| | MMLU-Pro | 0.647 | 0.520 |
| Code generation | **HumanEval+** | **0.727** | **0.562** |
| | MBPP+ | 0.617 | 0.468 |

Table 4: Rank correlation between each benchmark's method ordering and the overall ordering across all six benchmarks. **Bold** marks the benchmark selected for each domain.

hyperparameters. Running all six benchmarks for every configuration is expensive, so these analyses use one representative benchmark per domain, chosen so that the method ordering it induces best reproduces the ordering obtained from all six.

**Procedure.** For each benchmark we rank the thirteen methods of Table 1 by mean performance. Averaging the six ranks of a method gives its overall rank, and the vector of overall ranks across methods is the ordering we wish to preserve. For every benchmark we then compute the Spearman correlation between its own rank vector and this target vector, and within each domain we keep the benchmark with the higher correlation. Table 4 reports the results, selecting GSM8K for mathematical reasoning, HotpotQA for general question answering, and HumanEval+ for code generation.

**Validation.** Taking the best benchmark within each domain is a greedy rule, so we also enumerate all $2^3 = 8$ ways of picking one benchmark per domain and correlate the average rank over the selected three with the overall rank over all six. The greedy choice is also the best of the eight: GSM8K, HotpotQA and HumanEval+ reach a Spearman correlation of $0.907$, against $0.778$ to $0.891$ for the remaining seven combinations.

### D.3 Baseline Setups

This section describes the twelve baselines of Section 4.1: for each, the search space it optimizes, how that space is optimized, and the configuration we run. We group them as in the main table — single-agent, multi-agent collaboration, task-level MAS search, and query-level MAS search.

**Vanilla.** A single LLM call that answers the query directly, with no scaffold; it lower-bounds the benefit of any multi-agent structure. No training.

**CoT (Wei et al., 2022).** Chain-of-thought prompting elicits intermediate reasoning steps before the final answer rather than answering directly. No training; the method is purely a prompt. Our setup is identical to Vanilla except for the reasoning instruction, so the two differ only in the prompt.

**AgentVerse (Chen et al., 2024).** A hand-designed four-stage loop — expert recruitment, collaborative decision-making, action execution, and evaluation — in which a recruiter assembles a task-specific team whose proposal is revised over several rounds of feedback. The topology is fixed by design and is never searched, and there is no gradient training: all adaptation happens through inference-time LLM interaction. We use the official `tasksolving` pipeline with its default `cnt_agents`$=2$, `max_turn`$=3$, and `max_criticizing_rounds`$=3$, with the external executor disabled.

**DyLAN (Liu et al., 2024).** Agents form a layered feed-forward network and run in two stages. *Training* consists of *team optimization*: a label-free trial over a pool of candidate roles on the training split, in which each node rates its predecessors on a 1–5 scale, the ratings propagate backward as edge weights into an Agent Importance Score, and the top-scoring agents form one fixed team per domain. No gradients are involved, and this team is the only artifact carried forward. *Testing* runs that team alone, adding two query-dependent mechanisms: a listwise ranker that keeps the two best answers and deactivates the rest, and early stopping once more than $2/3$ of active agents agree. The official pools cover three of our benchmarks — 12 code writers and reviewers for HumanEval, 7 domain experts for MMLU, and 4 role-free agents for MATH — so we pair each remaining benchmark with the pool of matching task type: MBPP+, HotpotQA, and GSM8K respectively.

**ADAS (Hu et al., 2025).** The search space is an *entire agentic system expressed as Python code* — a `forward` function spanning prompts, tool calls, and control flow. A meta-agent, conditioned on a growing archive of previously discovered agents, programs the next candidate; the paper likens archive recombination to LLM-implemented crossover but defines no explicit population or selection operator. There is no gradient training and no learned model: each iteration designs a candidate (with two self-reflection rounds), evaluates it on a validation split, repairs runtime errors for up to five rounds, and appends it to the archive; validation performance is the only signal. The output is one fixed program per dataset. We use the official implementation with 30 generations.

**AFlow (Zhang et al., 2025b).** The search space is a *code-represented workflow*: LLM-invoking nodes connected by Python control flow, plus a pool of predefined operators. Because the model, temperature, and output format of each node are frozen, the effective space is prompts, edges, and operator choices. The search is a Monte-Carlo tree search whose expansion operator is an LLM, with each tree node holding a complete workflow. Optimization is again gradient-free: an LLM optimizer rewrites code and prompts, guided by the validation score and by textual "experience" propagated back up the tree, and each candidate is executed five times on validation. The result is one fixed workflow per dataset. We run the official implementation with 20 rounds, 4 samples, and one validation round. Following the official test convention, the best-validation round's workflow is copied into the test slot before evaluation.

**EvoMAS (Wei et al., 2026).** The search space is the graph topology together with agent roles and prompts, evolved by seven strategies: three exploratory (diversity expansion, conceptual recombination, cross-domain hybridization), three exploitative (fine optimization, best-practice synthesis, role specialization), and one custom operator. A meta-controller fires every $K$ generations to prune and induce rules and to update the strategy distribution by multiplicative weights. Evolution is LLM-driven with no gradients; the fitness signal is training-split accuracy minus a cost term. We run 20 generations with a population of 8 and a reflection interval of $K = 5$, since generated workflows can otherwise contain unbounded loops.

**MaAS (Zhang et al., 2025a).** MaAS replaces the search for a single workflow with an *agentic supernet*: a distribution over layer-wise operator configurations, conditioned on the query. Two mechanisms make the sampled architecture query-dependent — a mixture-of-experts top-$p$ router that activates operators until their cumulative score exceeds a threshold (varying the *width* of each layer), and an early-exit operator that truncates sampling (varying the *depth*). Training is two-sided: the controller is updated by a policy-gradient estimator with cost-aware importance weights, while the operators themselves, which admit no numerical gradient, are revised by agent-generated *textual* gradients over their prompts, temperatures, and node structure. We use the official implementation with its default of 4 samples. MaAS shares ASPO's operator-selection space.

**RAAS (Yang et al., 2026).** RAAS adopts the MaAS agentic supernet verbatim; its contribution is the *evaluation signal* rather than a new search space. For each query it samples a cohort of $N$ workflows, scores each by a trimmed mean over $K$ executions, and uses the cohort mean as a group baseline, yielding a zero-centered advantage in the style of GRPO — so the supernet is trained by policy gradient without a learned critic. We run the official implementation with its defaults of $N = 5$ and $K = 5$. The upstream code required several corrections before it produced a valid run; the two consequential ones are a reward bug (candidates were graded against a field absent from our data, so every sample scored zero and the controller optimized cost alone) and a grouping bug (a missing key collapsed each batch into a single advantage group, computing advantages across different problems). We verified on the first batch of every run that rewards are non-zero and that the number of per-query groups is correct. RAAS also shares ASPO's operator-selection space.

| Hyperparameter | Value | Sensitivity Analysis |
|---|---|---|
| *Adaptive Query-Selection Mechanism (AQSM)* | | |
| Posterior discount $\delta$ | 0.9 | Section 4.6, Figure 5(a) |
| Discovery samples $K$ | 3 | Section 4.6, Figure 5(a) |
| *Substructure-aware Optimization* | | |
| Maximum number of layers | 5 | Appendix D.6, Figure 8 |
| Maximum number of children per node | 2 | Appendix D.6, Figure 8 |
| Cost coefficient $\lambda$ | 0.01 | Appendix D.6, Figure 8 |
| Warm-up epochs $T_{\mathrm{w}}$ | 3 | Section 4.6, Figure 5(b) |
| Training epochs $T$ | 6 | Section 4.6, Figure 5(b) |

Table 5: ASPO hyperparameter settings grouped by component, with references to the corresponding sensitivity analyses.

**Puppeteer (Dang et al., 2025).** Puppeteer searches neither prompts nor code but the *sequence of agent activations*: a centralized orchestrator selects one agent per reasoning step, so the topology is never represented explicitly and instead emerges incrementally, recoverable as a directed graph only after the fact. The orchestrator is a genuinely learned neural policy trained online with REINFORCE, rewarded by terminal task success minus a compute cost. We use the official implementation with its defaults (episode length 4, up to 3 parallel explorations, $\lambda = 0.1$, $\gamma = 0.99$). Puppeteer also shares ASPO's operator-selection space.

**RADAR (Zhang et al., 2026).** RADAR generates the *communication topology* together with each node's role, using a conditional discrete graph diffusion model: a redundancy-aware ordering network masks nodes and edges, and a denoiser regenerates the graph node by node conditioned on the query, with redundancy measured by effective size. Both networks are gradient-trained — the denoiser on a variational bound and the ordering network by REINFORCE, using the denoiser's reconstruction likelihood as its reward — and the topology is generated per query at inference. We use the official implementation with adapters for our benchmarks, ordering the data so that the train/test boundary falls exactly at the intended index.

**ASPEC (Vu et al., 2026).** ASPEC is an LLM meta-agent method in which a meta-optimizer evolves specialist prompts alongside the controller's architectural choices over training epochs, with specialists subject to pruning. We use the official implementation with 2 samples, 6 debate rounds, 5 self-consistency paths, and 10 mutator turns.

### D.4 ASPO Implementation Details and Hyperparameter Reference

This section describes ASPO's PPO implementation and aggregation rules, and provides a reference table linking its hyperparameter settings to the corresponding sensitivity analyses.

**PPO implementation.** We adopt the standard clipped PPO objective (Schulman et al., 2017) with conventional training settings, rather than introducing a new policy-update rule. We use Adam with a learning rate of $10^{-4}$ for the actor, critic, and shared GNN. Trainable query- and operator-embedding parameters use a learning rate of $10^{-5}$ with no weight decay. The clipping threshold is $\epsilon_{\mathrm{clip}} = 0.2$, corresponding to the probability-ratio clipping interval $[0.8, 1.2]$. Each batch contains eight queries and is used for at most four PPO updates. We stop further updates on the batch if the absolute value of the approximate KL divergence exceeds $0.02$. We set the subgraph discount factor in Equation 5 to $\gamma = 0.95$: MAS construction spans at most five layers, so a value close to one keeps the reward of a distant descendant nearly undiscounted while still preferring earlier refinements.

**Aggregation.** We distinguish the local aggregation used to compute substructure rewards from the final aggregation used to produce the system's answer. For $\mathrm{Agg}$ in Equation 4, we directly use the `ScEnsemble` operator from our operator pool (Appendix D.7) to combine the child outputs into $y_i^+$. For the final aggregation in the progressive DAG rollout (Section 3.1), we collect the outputs of all leaf operators and apply a task-dependent rule. For open-ended generation tasks, such as code generation, we use the same ensemble operator. For multiple-choice and fill-in-the-blank questions, we vote over the leaf answers and select the most frequent answer.

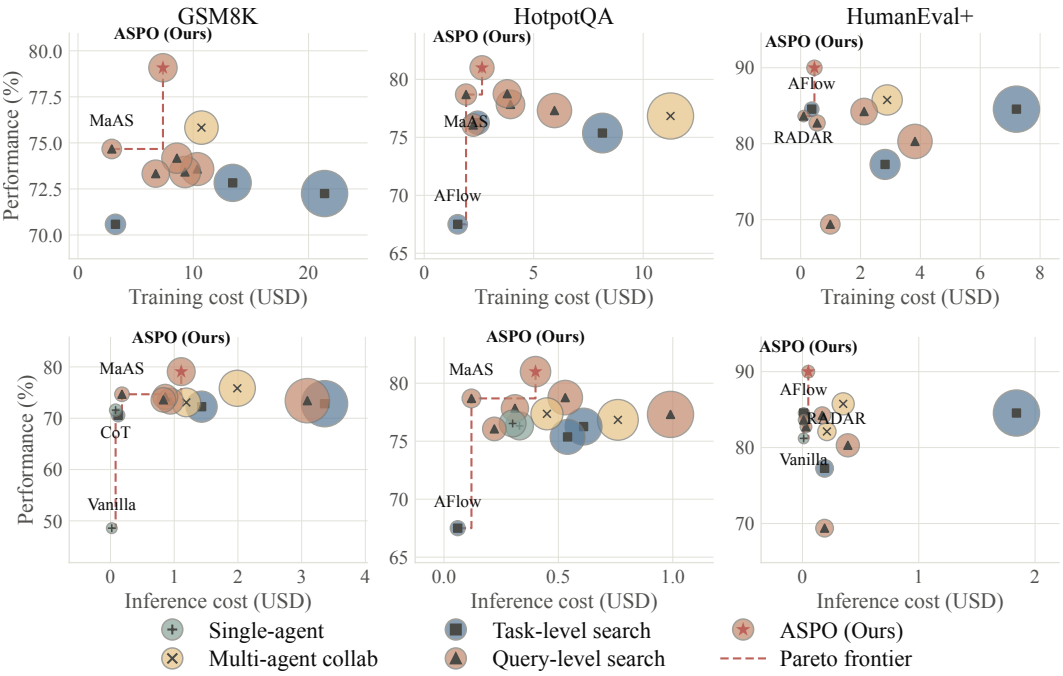



Figure 6: Cost–performance trade-offs on GSM8K, HotpotQA, and HumanEval+. The top and bottom rows show training and inference costs, respectively, against the same final test performance: accuracy, F1, and pass@1 for the three columns. Costs are in USD, with separate axis ranges and bubble-size scales for each panel. Dashed lines indicate the observed Pareto frontiers. Vanilla, CoT, and AgentVerse incur no training cost and are omitted from the top row.

**Hyperparameter reference.** Table 5 summarizes the ASPO-specific settings reported in Section 4.1 and links each to its corresponding sensitivity analysis. Section 4.6 examines the posterior discount $\delta$, discovery budget $K$, and training and warm-up durations. Appendix D.6 examines the cost coefficient $\lambda$ and the maximum numbers of layers and children per node.

### D.5 Cost–Performance Trade-offs by Benchmark and Phase

This appendix extends the cost–performance analysis in Section 4.4 with two complementary views: training and inference expenditure on GSM8K, HotpotQA, and HumanEval+, and wall-clock time on HumanEval+. Both are compared with final test performance to examine the monetary and time costs of the resulting systems.

**Monetary cost.** Figure 6 separates the aggregate comparison in Section 4.4 by benchmark and execution phase. Unlike the normalized costs in Figure 4, each panel reports cumulative expenditure in USD for the corresponding benchmark and phase. **(i) For training**, ASPO achieves the highest test performance on all three benchmarks with costs of \$7.37, \$2.64, and \$0.46, respectively. These are lower than DyLAN's corresponding costs of \$10.71, \$11.25, and \$2.89. The comparison with MaAS is more task-dependent: ASPO costs more to train on GSM8K and HotpotQA, but less on HumanEval+ (\$0.46 versus \$0.55), where it also improves pass@1 by 7.27 points. Thus, ASPO's higher performance does not require a consistently larger training expenditure. **(ii) At inference**, ASPO costs \$1.11, \$0.40, and \$0.05 on the three benchmarks, below those of DyLAN and EvoMAS in every case. However, MaAS is cheaper at inference on all three, costing \$0.18, \$0.12, and \$0.03, while trailing ASPO by 4.41, 2.31, and 7.27 performance points. A possible explanation is that uniform training gives substantial weight to queries already solvable by simple workflows, biasing the policy toward inexpensive architectures. The policy may consequently under-allocate computation to harder queries, reducing inference cost at the expense of answer quality. In conclusion, these results place ASPO at the highest-performance end of all six observed Pareto frontiers.

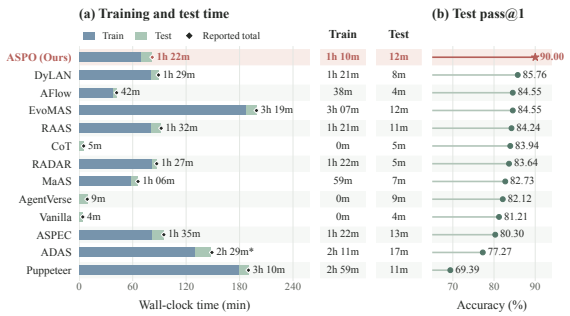



Figure 7: Wall-clock times on HumanEval+.

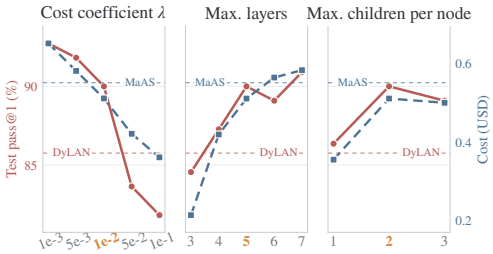



Figure 8: Sensitivity of pass@1 and total cost to the three hyperparameters that affect the searched MAS architecture, on HumanEval+.

**Wall-clock time.** Figure 7 complements the monetary-cost comparison with training, test, and total wall-clock times on HumanEval+. ASPO achieves 90.00% pass@1 with 70 minutes of training and 12 minutes of testing (82 minutes in total). This is shorter than DyLAN's 89 minutes and EvoMAS's 199 minutes, while improving pass@1 by 4.24 and 5.45 percentage points, respectively. Faster alternatives remain available: MaAS and AFlow take 66 and 42 minutes in total, respectively, but fall behind ASPO by 7.27 and 5.45 pass@1 points. To sum up, ASPO achieves the highest pass@1 with a total wall-clock time below the median across all evaluated methods.

### D.6 Sensitivity to Architecture Hyperparameters

Figure 8 examines sensitivity to three architecture-related hyperparameters on HumanEval+: **the cost coefficient** $\lambda$ in Equation 5, **the maximum number of layers**, and **the maximum number of children per node**. The defaults ($\lambda = 0.01$, five layers, and two children per node) were fixed before these sensitivity experiments. For $\lambda$, increasing the coefficient from 0.001 to 0.01 reduces accuracy by 2.7 points, with a further 6.4-point drop at 0.05, while cost decreases from \$0.65 to \$0.36 over the full range. This shows the performance–cost trade-off induced by stronger cost penalties. For the layer budget, accuracy increases from 84.6 to 90.0 as the limit grows from three to five, then largely plateaus. Extending the limit from five to seven layers adds 14% to cost for a gain of 0.9 points. For the branching limit, increasing the maximum from two to three children yields similar accuracy and cost (90.0 at \$0.51 versus 89.1 at \$0.50), suggesting limited sensitivity to this increase in branching capacity.

### D.7 Operator Space

We instantiate the operator pool $\mathcal{O}$ with the ten operators below. Each operator is an LLM-based operation that either *generates* a candidate solution, *aggregates* several candidates, *refines* a parent solution, or supplies *reference knowledge*. Unless noted otherwise, every solver/refiner operator emits a structured lane containing a `reasoning_process` and a `final_answer`; an optional {`background`} note (produced by the Background operator) is prepended to the prompt to aid reasoning and passed through unchanged, and parent-to-child information is threaded as separate {`parent_reasoning`} and {`parent_answer`} fields.

1. **Generate.** Produces a solution to the query following a given free-form instruction. It is the most basic generate-class operator and takes only the query (plus optional background).
2. **GenerateCoT.** Solves the query with explicit chain-of-thought reasoning (Wei et al., 2022), making every intermediate deduction explicit before stating the final answer.
3. **MultiGenerateCoT.** Samples several independent chain-of-thought solutions to the same query, producing a diverse pool of candidate lanes for downstream aggregation (Wang et al., 2023b).
4. **Plan.** A generate-class operator that first designs an explicit step-by-step plan (Wang et al., 2023a) (without solving) and then executes that plan step by step to obtain the answer.
5. **Background.** A retrieval / tool-use operator that reads only the query, extracts its core concepts, retrieves encyclopedia knowledge for them, and synthesizes solving-useful reference knowledge (definitions, formulas, identities). It does not solve the problem.

6. **ScEnsemble.** Aggregates several candidate solutions (Wang et al., 2023b) by grouping mathematically/semantically equivalent final answers, selecting the plurality answer (breaking ties by reasoning quality), and synthesizing the selected reasoning.
7. **SelfRefine.** Takes a parent solution and refines it (Madaan et al., 2023), finding and fixing errors or gaps and improving rigor while preserving a correct result.
8. **Variation.** Produces a new solution that deliberately follows a different line of reasoning from the parent, exploring an alternative approach to diversify the candidate pool rather than critiquing the parent.
9. **Debate.** Runs a sequential two-agent debate (Du et al., 2024) (a careful defender and a skeptical challenger) over two rounds with no external judge, returning the challenger's final solution.
10. **Programmer.** Solves the query by writing and executing Python code in a sandbox (Gao et al., 2023; Chen et al., 2023) (retrying with error feedback), then synthesizes the final answer from the code and its execution result.

Each operator's prompt template is shown below; $\{\cdot\}$ denotes an input placeholder filled at execution time.

**Operator 1: Generate**

**[Background]** {`background`}

**[Instruction]** {`instruction`}
**Problem:** {`query`}

**[Output]** `reasoning_process`; `final_answer`.

**Operator 2: GenerateCoT**

**[Background]** {`background`}

**[Instruction]** {`instruction`}
Solve this using EXPLICIT CHAIN-OF-THOUGHT reasoning: think step by step, making every intermediate deduction explicit and in order, and state the final answer ONLY after the full reasoning chain.
**Problem:** {`query`}

**[Output]** `reasoning_process`: the complete, self-contained step-by-step chain of thought; `final_answer`: the final answer only.

**Operator 3: MultiGenerateCoT**

Uses the same prompt as **GenerateCoT** (Operator 2), executed as $N{=}3$ independent samples. The three resulting candidate lanes are passed on together for downstream aggregation (e.g., by ScEnsemble or Debate).

**Operator 4: Plan (two calls: design then execute)**

**[Call 1 — Design]**
Read the problem and design a STEP-BY-STEP PLAN to solve it. Do NOT solve the problem yet — only lay out the steps. Output EXACTLY this format: "`Total N steps.`" followed by one line per step ("`Step k: ...`"). Each step states WHAT to do (the method/action), not the computation. Make the FIRST step search for the simplest route (identify any reusable structure or shortcut that shortens the problem); make the LAST step a verification step (check the result against the problem's conditions and constraints).
**Problem:** {`query`}
**Output:** `plan`.

**[Call 2 — Execute]**
You are given the problem and a step-by-step PLAN. EXECUTE THE PLAN: carry out each step in order, showing the work for every step, until you reach the answer. Stay faithful to the plan's steps.
**Problem:** {`query`} **Plan:** {`plan`}

**[Output]** `reasoning_process`: the full step-by-step execution that follows the plan; `final_answer`: the final answer only.

**Operator 5: Background (three steps: keyword, retrieval, synthesize)**

**[Step 1 — Keyword extraction]**
Read the problem and identify its 1–3 CORE concepts/topics — the things that, if looked up in an encyclopedia, would supply useful definitions and formulas for solving it. Output them as short search terms (matching an encyclopedia article title), comma-separated.
**Problem:** {`query`} **Output:** `keywords` (1–3, comma-separated).

**[Step 2 — Retrieval]** (executed in code, no LLM call)
For each keyword, search Wikipedia and fetch the best-matching article's introduction; concatenate the extracts as {`wiki`}.

**[Step 3 — Synthesize]**
You are compiling a BACKGROUND note of reference knowledge that will help someone solve the problem. You are given the problem, the extracted keywords, and encyclopedia extracts for those keywords ({`query`}, {`keywords`}, {`wiki`}). Write a concise background of the knowledge most useful for solving THIS problem (relevant definitions, key formulas, identities, theorems). **Strict rules:** do NOT solve the problem or state/compute its final answer; provide only general reference facts, distilled from the extracts and your own knowledge, ignoring irrelevant parts.

**[Output]** `background`.

**Operator 6: ScEnsemble**

**[Background]** {`background`}

You are combining several candidate solutions to the SAME problem; each candidate contains a reasoning process and a final answer. Follow this decision procedure:

1. Extract the final answer from every candidate; treat equivalent answers as the SAME answer.
2. Count how many candidates support each answer. If one answer has a strict plurality/majority, select a candidate supporting it (independent agreement is strong evidence).
3. If the highest vote is tied, do NOT break the tie by candidate order — inspect the competing reasoning, verify it yourself, and choose the most sound one (reasoning quality takes priority over vote count).
4. Write a self-contained synthesis of the selected answer's reasoning, correcting any clear mistake; do not invent an answer unsupported by the candidates.

**Problem:** {query} **Candidate solutions:** {candidates}

**[Output]** reasoning_process: a complete comparison and synthesis (state the vote pattern, verify the selected derivation, explain any tie-break); solution_letter: the single letter (A, B, C, ...) of the selected candidate.

**Operator 7: SelfRefine**

**[Background]** {background}

Your task is to REFINE a proposed solution: find and fix any errors or gaps, improve the rigor, and keep the result correct. You are given the problem plus the parent solution split into its reasoning and its final answer.
**Problem:** {query}
**Parent — reasoning:** {parent_reasoning}
**Parent — final answer:** {parent_answer}

**[Output]** reasoning_process: the corrected, self-contained reasoning; final_answer: the final answer only.

**Operator 8: Variation**

**[Background]** {background}

Produce a NEW solution that deliberately follows a DIFFERENT line of reasoning from the parent solution. The goal is to EXPLORE an alternative approach (to diversify the candidate pool), NOT to critique or merely rephrase the parent.
**Problem:** {query}
**Parent — reasoning:** {parent_reasoning} **Parent — final answer:** {parent_answer}

1. Solve correctly; the final answer must be fully derivable from your reasoning. If the parent's answer is already correct, reach that SAME answer by a different route.
2. Take a FUNDAMENTALLY different approach (e.g., a different decomposition, working backward vs. forward, exploiting structure vs. brute force). Do not paraphrase or reorder the parent's steps.
3. Do NOT critique the parent — this is exploration, not evaluation.
4. Before finalizing, VERIFY your own answer against the problem's conditions.

Your reasoning must be COMPLETE and self-contained (readable without seeing the parent).

**[Output]** `reasoning_process`; `final_answer`.

**Operator 9: Debate (sequential two agents, two rounds, four calls)**

**[Round 0 — Debater A: careful & rigorous]**
A solution has been proposed (reasoning and final answer supplied separately). Examine it: if correct, defend it with stronger justification and clearer steps; if flawed, produce a corrected solution. Re-derive from scratch rather than trusting the proposal; first look for a structural shortcut, then VERIFY by checking the result against the problem's conditions. If the proposal is correct, keep the answer UNCHANGED. Produce your OWN complete, self-contained derivation.
**Inputs:** {`query`}, {`parent_reasoning`}, {`parent_answer`}, {`background`}.

**[Round 0 — Debater B: skeptical & creative]**
A solution has been proposed and Debater A has responded. Critically examine BOTH for logical flaws, unstated assumptions, edge cases, or better approaches. Target common failure points: a missed simplifying insight; an arithmetic/sign slip; an unchecked edge case or unmet constraint; incomplete or double-counted cases. Even when the proposal and A agree, independently verify — agreement can hide a shared mistake. Produce your OWN complete, self-contained derivation.
**Inputs:** {`query`}, {`parent_reasoning`}, {`parent_answer`}, {`a_reasoning`}, {`a_answer`}, {`background`}.

**[Round 1 — Debater {role} (A then B): final round]**
Read your opponent's latest response: (1) are there valid points that should change your answer? (2) flaws to rebut? (3) can you strengthen your own solution? Then produce your final, definitive solution and verify it against the problem's conditions. If your previous answer is already correct, keep it UNCHANGED.
**Inputs:** {`query`}, {`own_reasoning`}, {`own_answer`}, {`opp_reasoning`}, {`opp_answer`}, {`background`}.

**[Output]** `reasoning_process`; `final_answer` (the operator returns Debater B's final-round output).

**Operator 10: Programmer (two calls: code then synthesize)**

**[Background]** {`background`}

**[Call 1 — Code generation]**
Solve the problem by writing EXACT Python. Computing with code avoids the slips that ruin long hand derivations, so lean on it for heavy or error-prone calculation. Classify the problem and pick the right tools (prefer exact/symbolic computation); use the background to set up the computation correctly; the parent reasoning/answer are REFERENCE ONLY (may be wrong; re-derive the setup). **Rules:** define `def solve()` taking NO arguments and RETURNING the result; return EXACT values; no `print`/plots/file/network I/O; give EVERY loop a finite termination bound so it finishes well under the sandbox time limit.
**Problem:** {`query`}
**Parent reasoning (reference only):** {`parent_reasoning`} **Parent final answer (reference only):** {`parent_answer`}
**Feedback (on a failed run):** {`feedback`} — the previous code and its error are fed back so the model can fix it (up to three tries).
**Output:** `code` (complete, self-contained, ready to run), then executed in a sandbox.

**[Call 2 — Synthesis]**
You solved the problem by writing and running Python. Given the problem, the code, and the execution result ({`query`}, {`code`}, {`status`}, {`run_output`}), turn this into the final answer: `reasoning_process` explains the approach the code took and interprets the result; `final_answer` is DERIVED from the execution result in the exact form asked. If the status is an error, or the result is empty or clearly inconsistent, do NOT trust it — re-reason using the background.

**[Output]** `reasoning_process`; `final_answer`.

## APPENDIX REFERENCES


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

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

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/578e65cdee35d00c708d4c64bce32971-Paper-Conference.pdf.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Xin Cheng, Shuo He, Lang Feng, Haiyang Xu, Ming Yan, Lei Feng, and Bo An. Beyond trajectory-level attribution: Graph-based credit assignment for agentic reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=zAFrIOeTCp.

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

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. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 165025–165059. Curran Associates, Inc., 2025. doi: 10.52202/085713-5502. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f1320d2e2842169c6fc89dcbd80e94d0-Paper-Conference.pdf.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1515–1528. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/florensa18a.html.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Olga Golovneva, Moya Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. Roscoe: A suite of metrics for scoring step-by-step reasoning, 2022. URL http://arxiv.org/abs/2212.07919v2.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29557–29590. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/kazemnejad25a.html.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In Bruce W. Croft and C. J. van Rijsbergen (eds.), *SIGIR '94*, pp. 3–12, London, 1994. Springer London. ISBN 978-1-4471-2099-5.

Yanming Li, Xuelin Zhang, WenJie Lu, Ziye Tang, Maodong Wu, Haotian Luo, Tongtong Wu, Zijie Peng, Hongze Mi, Yibo Feng, et al. Who deserves the reward? sharp: Shapley credit-based optimization for multi-agent system. *arXiv preprint arXiv:2602.08335*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Yun Qu, Qi Wang, Yixiu Mao, Vincent Tao Hu, Björn Ommer, and Xiangyang Ji. Can prompt difficulty be online predicted for accelerating rl finetuning of reasoning models? In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, KDD '26, pp. 1240–1250, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722585. doi: 10.1145/3770854.3780263. URL https://doi.org/10.1145/3770854.3780263.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Burr Settles. Active learning literature survey. 2009.

Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis (eds.), *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper_files/paper/2007/file/a1519de5b5d44b31a01de013b9b51a80-Paper.pdf.

Myan Vu, Harrish Ayyanar, PANG JIANG, Anwiketh Reddy, and Mayank Goel. Automated stateful specialization for adaptive agent systems. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 90290–90313, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/9230cf96fb02545f83e34dfe2810e204-Paper-Conference.pdf.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634, Toronto, Canada, July 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.147. URL https://aclanthology.org/2023.acl-long.147/.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5776–5788. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ad236edc564f3e3156e1b2feafb99a24-Paper-Datasets_and_Benchmarks_Track.pdf.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Yangbo Wei, Zhen Huang, Ronghao Xu, Hong Wang, and WEI W. XING. EvoMAS: Heuristics in the loop—evolving smarter agentic workflows. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=cdWnpXkcQo.

Jiayi Yang, Guancheng Wan, Man Zhang, and Mang Ye. Raas: Llm agentic system architecture search with grpo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 34470–34479, June 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, juncai liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 113222–113244. Curran Associates, Inc., 2025. doi: 10.52202/085713-3775. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a4277440d50f1f15d2cb4c14f7e0c0d2-Paper-Conference.pdf.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267

of *Proceedings of Machine Learning Research*, pp. 73511–73525. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/yuan25c.html.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75834–75852. PMLR, 13–19 Jul 2025a. URL https://proceedings.mlr.press/v267/zhang25bi.html.

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. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5492ecbce4439401798dcd2c90be94cd-Paper-Conference.pdf.

Zhen Zhang, Wanjing Zhou, Juncheng Li, Hao Fei, Jun Wen, and Wei Ji. RADAR: Redundancy-aware diffusion for multi-agent communication structure generation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=GtpiqFaJtZ.

Haizhong Zheng, Yang Zhou, Brian Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 124321–124346. Curran Associates, Inc., 2025. doi: 10.52202/085713-4143. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/b42fb82fc3ffb88b872c4714f093875d-Paper-Conference.pdf.