

AMAT: Automated Multi-Agent Topology Design via Reinforcement Learning

Shijie Zheng^{1,2}, Zhiqiang Pu^{2,3*}, Hao Ma², Tianyi Hu², Xiaolin Ai^{2,3}, Yi Pan²

¹School of Advanced Interdisciplinary Sciences,
University of Chinese Academy of Sciences

²National Key Laboratory of Cognition and Decision Intelligence for Complex Systems,
Institute of Automation, Chinese Academy of Sciences

³School of Artificial Intelligence,
University of Chinese Academy of Sciences
{zhengshijie2025, zhiqiang.pu, mahao2021}@ia.ac.cn
{hutianyi2021, xiaolin.ai, yi.pan}@ia.ac.cn

Abstract

Large Language Model-based multi-agent systems have demonstrated remarkable capabilities in handling complex tasks through collective collaboration. However, current design approaches rely on either manual crafting or automated heuristic search, yet suffer from two critical limitations: ephemeral experience and value-agnostic optimization. To address these fundamental challenges, we propose **AMAT**, a self-bootstrapping reinforcement learning framework that shifts the paradigm from case-by-case online search to persistent offline policy learning. AMAT formulates the topology design of multi-agent systems as a constrained Markov decision process on graphs, and integrates Monte Carlo Tree Search with a policy-value network to learn reusable topology preferences over a predefined operator library. Comprehensive evaluations across five benchmarks demonstrate that AMAT (1) consistently outperforms both handcrafted and automated multi-agent baselines across all evaluated benchmarks, (2) substantially reduces both training expenditure and inference latency compared to existing search-based approaches, and (3) exhibits strong cross-task generalization within task domains and cross-LLM-backbone transferability.

1 Introduction

Multi-agent systems (MAS) based on Large Language Models (LLMs) have effectively overcome the capability bottlenecks of single agents in handling complex, long-horizon tasks through collective collaboration (Xi et al., 2023; Wang et al., 2024a; Liu et al., 2024a). However, early multi-agent systems primarily relied on expert manual design and handcrafted configurations. Representative works such as AutoGen (Wu et al., 2024), MetaGPT (Hong et al., 2024), and CAMEL (Li et al., 2023), while demonstrating stability in specific domains, typically employ fixed agent roles and interaction topologies. This paradigm lacks the flexibility to adapt to novel tasks and fails to optimize the collaboration structure itself, making it difficult to reach the optimal solution for collaborative architecture.

To address these challenges, researchers have turned to the automated design of multi-agent systems. For instance, MultiPersona (Wang et al., 2024b) and EvoAgent (Yuan et al., 2025) target agent role generation, while GPTSwarm (Zhuge et al., 2024) and DyLAN (Liu et al., 2024b) optimize collaboration relationships. More recently, heuristic search methods like ADAS (Hu et al., 2025) leverage in-context learning to have LLMs iteratively program new agents in code space, while tree search methods like AFlow (Zhang et al., 2025b) employ Monte Carlo Tree Search (MCTS) to explore configuration spaces systematically, both significantly expanding the design space.

*Corresponding author: zhiqiang.pu@ia.ac.cn

Despite these advances, search-based methods have yet to overcome two practical bottlenecks: *Ephemeral Experience* and *Value-Agnostic Optimization*. The first issue is that design knowledge accumulated during search does not transfer across tasks. Although some methods retain successful execution paths within a single task via Tree-Structured Experience (Zhang et al., 2025b), this knowledge is non-parametric and task-local—when faced with a new problem instance (even from the same distribution), the system must restart the search from scratch. The second issue concerns the absence of a global value signal. When LLMs act as black-box optimizers, they have no explicit means to estimate the long-term utility of a topology. Without a differentiable objective that jointly accounts for performance and cost, these methods tend toward over-engineered architectures that chase accuracy while ignoring redundancy, rarely reaching the Pareto frontier between *task success rate* and *inference latency*.

To address these issues, this paper proposes **AMAT**, an end-to-end automated multi-agent collaboration topology design framework based on Reinforcement Learning (RL). Our core vision is to recast automated MAS design as an *offline Meta-Policy Learning problem*: AMAT amortizes repeated online topology search into a reusable topology-construction policy, capturing structural preferences over executable multi-agent workflows and enabling efficient topology generation at test time.

Specifically, by adopting a policy-value iteration paradigm suitable for discrete decision spaces, we construct the following mechanisms for multi-agent collaboration topology generation: **(1) Topology Optimization via RL**: We establish a self-bootstrapped learning framework that utilizes MCTS as a policy improvement operator to continuously mine high-quality collaboration topologies and store them in an experience replay buffer, thereby addressing the scarcity of multi-agent topology data. Subsequently, through knowledge distillation, these discrete search trajectories are internalized into neural network weights, achieving the transformation from implicit design knowledge to explicit parameterized policies. **(2) Search-free Inference via Value-guided Policy Learning**: We introduce a value network as a differentiable predictor to assess both task performance and computational cost of substructures while training. This guides the policy network to internalize global value judgments, enabling the model to bypass expensive online searches during the inference phase and achieve search-free sequential generation of optimal collaboration topologies via efficient neural forward inference.

To comprehensively evaluate the effectiveness of this framework, we conduct extensive experiments on five authoritative benchmarks covering three major domains: code generation (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)), mathematical reasoning (MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021)), and question answering (HotpotQA (Yang et al., 2018)). Empirical results demonstrate: **(1) Strong Empirical Performance**. Through MCTS-guided exploration, AMAT outperforms all baselines including the strongest automated systems MaAS and AFlow, with gains of up to 7.28% over MaAS and 7.82% over AFlow on individual benchmarks; **(2) MCTS-Free Test-Time Topology Generation**. AMAT amortizes topology search into a learned policy and replaces test-time online topology search with direct neural generation: on MATH, inference takes 6 minutes versus 23 minutes for AFlow (3.8 $\times$) and 19 minutes for MaAS (3.2 $\times$). Training/search is also efficient under our two-stage accounting protocol: AMAT costs \$2.65 versus AFlow’s \$22.50, while being slightly faster than MaAS in wall-clock training time (49 vs. 53 minutes); **(3) Policy-Level Generalization**. The learned policy captures reusable structural preferences across related task settings—a GSM8K-trained policy maintains 98.2% relative performance on MATH without fine-tuning.

The main contributions of this paper are summarized as follows:

- **Reinforcement Learning Framework for Topology Generation**: We propose AMAT, a self-bootstrapping RL framework that integrates MCTS with policy-value networks. By continuously mining high-quality topologies and distilling them into a parameterized policy, the model learns reusable topology-construction preferences from MCTS-guided search trajectories.

- **MCTS-Free Test-Time Generation via Value-Guided Policy Learning:** A global value network estimates the quality of partial topologies by considering both task performance and execution efficiency during training, internalizing value judgments into the policy so that deployment constructs topologies without running MCTS— $3.2\times$ – $3.8\times$ faster than the strongest search-based baselines.
- **Strong Empirical Performance and Generalization:** AMAT outperforms all baselines across five benchmarks by up to 7.28% over MaAS and 7.82% over AFlow, and transfers without fine-tuning to unseen datasets and different LLM backbones, suggesting that the learned policy captures reusable structural preferences across related task settings and model backbones.

2 Related work

Design Space of Multi-Agent Systems. LLM-based multi-agent system design spans agent profiling, communication topology, and optimization objectives (Wang et al., 2024a). While agent profiling has received considerable attention via persona generation (Yuan et al., 2025; Wang et al., 2024b) and prompt optimization (Fernando et al., 2023), the *topology design* problem—determining which agents communicate and in what order—remains underexplored. Our work addresses this gap by formulating topology construction as a sequential DAG-building process.

Technical Approaches to Topology Optimization. Existing methods fall into three categories: *differentiable approaches* (Zhuge et al., 2024) that optimize edge weights via gradient descent but assume fixed node sets; *LLM-as-optimizer* methods (Hu et al., 2025) that propose modifications via in-context learning but lack explicit long-term value signals; and *tree search methods* (Zhang et al., 2025b) that systematically explore workflow configurations. Recent automated workflow and topology optimization methods, such as AFlow and MaAS, further demonstrate the potential of search-based or supernet-based workflow construction. AFlow searches for optimized workflows, while MaAS explores agentic supernets and operator compositions. AMAT complements these approaches by learning a state-conditioned graph-edit policy over partial DAG states, using MCTS-generated construction trajectories and execution feedback to train a reusable policy-value network. At test time, the learned policy synthesizes executable workflows without running MCTS, removing the per-instance N_{sim} search overhead.

Policy Learning for Combinatorial Structures. Policies over combinatorial structures have been studied in neural combinatorial optimization (Vinyals et al., 2017; Kool et al., 2019) and graph-based program synthesis (Allamanis et al., 2018). The AlphaGo framework (Silver et al., 2016) provides a principled bootstrap via MCTS-supervised policy training, with the value network predicting long-term returns. We adapt this framework to executable DAG construction using a Graph Transformer encoder and MCTS-generated trajectories from our dual-buffer replay mechanism. MDPs, MCTS, and neural combinatorial optimization are established tools; AMAT instantiates them for LLM-based multi-agent workflow topology learning with performance and cost feedback.

3 Methodology

We propose AMAT, a framework for the automated design of multi-agent collaborative topological structures. In this framework, we formally characterize the workflow of a MAS as a finite executable DAG, where nodes instantiate functional agents/operators and edges denote data-flow dependencies. Crucially, our work centers on improving search efficiency within the vast combinatorial topological space. By modeling the MAS generation process as a constrained Markov decision process (CMDP) (Altman, 1999) on graphs, we employ an MCTS-augmented RL approach to efficiently navigate the search space and identify high-performing executable workflows under validity constraints. Meanwhile, this paradigm enables the system to distill MCTS-generated topology-construction trajectories

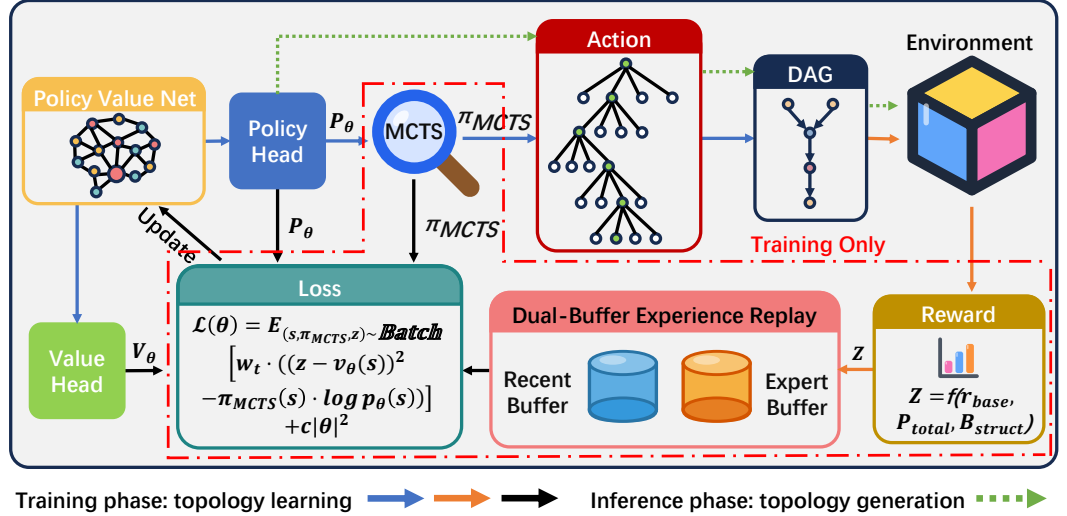


Figure 1: The overall framework of AMAT.

and execution feedback into a policy-value network, thereby achieving MCTS-free topology generation during inference.

3.1 Modeling

We model the design of multi-agent workflows as a generative decision process. Unlike fixed hand-designed workflows, we formalize this problem as a CMDP, represented by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. This formulation uses reinforcement learning to search for high-performing executable structures within a large topological space, while lightweight validity constraints restrict exploration to feasible DAGs and accelerate convergence.

State Space (S). The state s_t fully describes the partial DAG currently under construction. Formally:

$$s_t = (\mathcal{G}_t, \mathbf{X}_t, \text{meta}_t), \quad (1)$$

where $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ represents the current topological structure (including the node set and data flow edge set). $\mathbf{X}_t$ encodes the semantic roles and parameter configurations of each node via one-hot encodings. Specifically, the semantic roles correspond to various reasoning paradigms, such as *Generator* (implementing Chain-of-Thought (Wei et al., 2023)), *Voter* (employing Self-Consistency (Wang et al., 2023)), *Reflector* (utilizing Reflexion (Shinn et al., 2023)), and *Programmer* (adopting Program-Aided Language Models (Gao et al., 2023)). Note that these roles serve as abstract classifications; in practice, diverse operators are designated to execute specific functions tailored to different tasks. meta_t contains global resource states such as node counts, assisting the value network in global performance assessment. The policy-value network takes this partial DAG state as its input. The concrete task instance is supplied during workflow execution, where each selected agent/operator receives the problem, available context, and upstream outputs through predefined input slots.

Atomic Action Space ($\mathcal{A}$). We define a fine-grained action space over a predefined agent/operator library. The agent decides on node instantiation, edge construction, structured operator refinement, and termination. Specific actions include: `AddNode( $r, \theta_{\text{init}}$ )`, which instantiates an agent/operator type r with its default structured configuration θ_{init} ; `AddEdge( $u, v$ )`, which establishes a directed data-flow dependency between nodes u and v ; `RefineOps`, which updates structured configurations exposed by the selected operators; and `Stop`, which submits the current DAG for execution. Each agent/operator is associated with

a predefined prompt template or prompt-construction rule, while graph actions determine the topology and data flow.

Transition & Constraints ($\mathcal{P}$). State transitions follow deterministic logic: $s_{t+1} = \mathcal{T}(s_t, a_t)$. Given the super-exponential growth of the DAG search space ($O(N! \cdot 2^N)$), we introduce a dynamic action masking mechanism as an inductive bias. The set of valid actions is defined as:

$$\mathcal{A}_{\text{valid}}(s_t) = \{a \in \mathcal{A} \mid \text{Valid}_{\text{DAG}}(s_t \oplus a) \wedge \text{Valid}_{\text{Flow}}(s_t, a)\}, \quad (2)$$

where $\text{Valid}_{\text{DAG}}$ enforces the acyclicity constraint, and $\text{Valid}_{\text{Flow}}$ ensures the weak connectivity of the topology and the semantic compatibility of node interfaces. This guarantees that the policy π_θ explores only within valid and executable topological spaces.

Reward & Discount ($\mathcal{R}, \gamma$). The reward function $\mathcal{R}(s_t, a_t)$ is designed as a sparse, delayed signal. It returns zero for all intermediate construction steps and yields a scalar performance metric r_{final} (detailed in Sec. 3.3) only at the terminal state. Consistent with the objective of global structural optimization, we set the discount factor $\gamma = 1$, ensuring that early topological decisions are weighted equally with subsequent refinements during value estimation.

3.2 Neural-guided topology search on graph structures

We model the DAG construction process as MCTS on graph structures. To navigate efficiently within the immense combinatorial space, we combine the lookahead planning capability of MCTS with the generalization ability of policy-value networks.

Search Loop on Graph Structures. We parameterize f_θ as a Graph Transformer (Dwivedi & Bresson, 2021) with adjacency-based attention masking to respect the DAG topology. In our implementation, f_θ uses one Graph Transformer encoder layer with hidden dimension 64 and four self-attention heads. It operates over fixed node slots and one global graph token, with input features including agent/operator types, structured configurations, directed edges, graph size, and construction progress.

Using this network as the prior and value estimator, we execute N_{sim} simulations on the partial DAG state s_t (Silver et al., 2016). Each simulation follows the standard *Select-Expand-Evaluate-Backup* paradigm. Specifically, the selection phase employs the Predictor Upper Confidence Bound (PUCT) (Rosin, 2010) rule to balance exploration and exploitation, introducing Dirichlet noise $\eta \sim \text{Dir}(\alpha)$ at the root node to enhance topological diversity; the expansion phase is restricted to $\mathcal{A}_{\text{valid}}(s_\ell)$ at the expanded leaf state s_ℓ to ensure structural validity; and backpropagation updates the visit count $N(s, a)$ and mean value $Q(s, a)$ of nodes along the search path.

After simulations conclude, we sample the actual action based on the distribution $\pi(a|s) \propto N(s, a)^{1/\tau}$. Additionally, to penalize non-functional DAGs (e.g., lacking a generator node) encountered during the exploration phase, we introduce a validity-constrained penalty mechanism: if a terminal graph fails the integrity check, the environment assigns a penalty reward $z_{\text{penalty}} = -1$. This negative feedback signal compels the policy network to implicitly internalize global structural constraints, ultimately enabling the direct generation of executable topologies during inference without relying on post-hoc rejection sampling or retries.

Policy Deployment: MCTS-Free Inference. During training, MCTS acts as a *Policy Improvement Operator*, supplying an improved action distribution π_{MCTS} that supervises the policy network. During testing, MCTS is dropped: starting from an empty DAG, the learned policy repeatedly selects legal graph-edit actions under the same validity mask until Stop. This removes the per-instance MCTS overhead and enables efficient topology generation for new tasks.

3.3 Reward design and policy optimization

We optimize the policy network θ via RL based on execution feedback. Specifically, the process comprises three key steps: first, quantifying the quality of the generated topology through a physical execution engine (Reward Shaping (Ng et al., 1999)); second, managing training data using a dual-buffer mechanism (Experience Replay); and finally, updating network parameters based on a time-weighted objective function (Optimization Objective).

Topology Execution and Performance-First Reward. Since the generated DAG is an abstract workflow specification, we map the terminal state s_L to the execution environment via the asynchronous engine (Wang et al., 2025), and execute the agent workflow after verifying acyclicity. We first compute a task-performance base reward r_{base} from the benchmark metric A , using Pass@1 for code generation, Accuracy for mathematical reasoning, and F1 for question answering. Token consumption, inference latency, structural redundancy, and structural completeness are then used as normalized adjustments to encourage compact and efficient workflows among high-quality candidates:

$$r_{\text{base}} = \begin{cases} \lambda(A - \tau_{\text{high}})^\kappa & \text{if } A \geq \tau_{\text{high}} \\ -1 & \text{if } A < \tau_{\text{elim}} \\ -1 + \frac{A - \tau_{\text{elim}}}{\tau_{\text{high}} - \tau_{\text{elim}}} & \text{otherwise} \end{cases}, \quad (3)$$

$$r_{\text{final}} = \begin{cases} [r_{\text{base}} \cdot (1 - P_{\text{total}}) + B_{\text{struct}}]_{-1}^{+1} & \text{if } r_{\text{base}} > 0 \\ [r_{\text{base}} - P_{\text{total}} + B_{\text{struct}}]_{-1}^{+1} & \text{otherwise} \end{cases},$$

where r_{base} incorporates an exponent κ (e.g., 1.5) to provide super-linear incentives for architectures exceeding the high-score threshold τ_{high} , amplifying differences among top-performing candidates. For the middle tier, a linear interpolation bridges the gap between the elimination threshold τ_{elim} and τ_{high} . $P_{\text{total}} = \sum_k w_k C_k$ represents the aggregated normalized efficiency penalty, including token consumption (C_{token}), inference latency (C_{latency}), and structural redundancy ($C_{\text{redundancy}}$). B_{struct} denotes the structural completeness bonus, such as forming a complete verification or refinement path. For positive base rewards, efficiency penalties scale down the achieved gain; for non-positive base rewards, they are applied additively to penalize inefficient low-quality workflows. The final reward is clipped to $[-1, 1]$ to match the Tanh activation range of the value network, serving as the value target z for subsequent optimization. This design keeps task performance as the primary signal, while cost and structural terms regularize workflow compactness and execution efficiency.

Dual-Buffer Experience Replay. To address the non-stationarity of online generated data and the catastrophic forgetting of high-performance architectures, we design a hybrid experience replay mechanism (Schaul et al., 2016) maintaining two independent buffers:

- **Recent Buffer ($\mathcal{B}_{\text{recent}}$):** Uses a first-in-first-out (FIFO) queue to store the latest N_{rec} online trajectories. This approximates the current marginal policy distribution, ensuring gradient updates possess near on-policy characteristics, facilitating rapid adaptation to the current search space.
- **Expert Buffer ($\mathcal{B}_{\text{expert}}$):** Uses a priority queue to permanently store high-quality topologies meeting specific thresholds (e.g., $\text{Accuracy} \geq 0.7$).

This mechanism ensures the model continuously reviews and internalizes historical optimal structural patterns while exploring new topologies.

Time-Weighted Optimization Objective. Our final optimization objective is to minimize the weighted sum of policy distillation error and value regression error. Addressing the credit assignment problem caused by long horizons in DAG generation tasks, we introduce time importance weights w_t . The loss function $\mathcal{L}(\theta)$ is defined as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s, \pi_{MCTS}, z) \sim \text{Batch}} \left[w_t \cdot \left((z - v_\theta(s))^2 - \pi_{MCTS}(s) \cdot \log p_\theta(s) \right) \right] + c \|\theta\|^2, \quad (4)$$

where: z is the actual return r_{final} computed via Eq. (3). π_{MCTS} is the policy distribution augmented by search in Section 3.2, serving as the distillation target (Rusu et al., 2016) for the policy network p_θ . w_t is a binary time weight: set to 1.0 for terminal states yielding explicit rewards, and 0.5 for intermediate states. This significantly accelerates the convergence of the value head in long-sequence decisions.

4 Experiments

In this section, we organize our experiments around four research questions (RQ): **RQ1 (Performance Boundary)**: Can AMAT discover superior multi-agent topologies compared to SOTA manual designs and online search-based methods? **RQ2 (Efficiency Analysis)**: Does the “offline learning, online generation” paradigm achieve better training and inference efficiency compared to existing methods? **RQ3 (Generalization)**: Can the learned policy network capture generalizable collaboration primitives and transfer to unseen tasks or different LLM backbones? **RQ4 (Ablation)**: How do key components contribute to the final performance?

4.1 Experimental setup

Tasks and Benchmarks. We evaluate AMAT on five benchmarks across three domains: **Code Generation** (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)), **Mathematical Reasoning** (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021)), and **Question Answering** (HotpotQA (Yang et al., 2018)). For MATH, we select 605 Level-5 problems following established protocols.

Evaluation Metrics. We adopt standard task-specific metrics: Pass@1 for code generation, Accuracy for mathematical reasoning, and F1 Score for question answering. We also report inference cost (dollar cost per 1k tokens) to assess computational efficiency.

Baselines. We compare against three categories of methods: **(1) Single-Agent**: Direct I/O, CoT (Wei et al., 2023), Complex CoT (Fu et al., 2023), and Self-Consistency (Wang et al., 2023). **(2) Handcrafted MAS**: MultiPersona (Wang et al., 2024b), LLM-Debate (Du et al., 2024), LLM-Blender (Jiang et al., 2023), DyLAN (Liu et al., 2024b), AgentVerse (Chen et al., 2023), and MacNet (Qian et al., 2025). **(3) Automated MAS**: GPTSwarm (Zhuge et al., 2024), AutoAgents (Chen et al., 2024), ADAS (Hu et al., 2025), AgentSquare (Shang et al., 2025), AFlow (Zhang et al., 2025b), and MaAS (Zhang et al., 2025a).

Implementation Details. We employ Qwen-Turbo and GPT-4o-mini-0718 as backbone LLMs. The topological search space is constrained to DAGs with at most 6 nodes. The policy-value network is a lightweight structure-masked Transformer with hidden dimension 64. During inference, executable topologies are generated via neural forward passes without online search. All results are averaged over 4 runs. For fair comparison, we adopt the same experimental protocol as (Zhang et al., 2025a), including identical model versions, temperature settings, and rollout counts. Baseline results for all compared methods are taken directly from Zhang et al. (2025a).

4.2 Main results: performance comparison (RQ1)

As shown in Table 1, AMAT achieves 80.50% on average, outperforming all baselines. Compared to MaAS (77.73%), the improvement is most pronounced on MATH (+7.28%, 59.10% vs. 51.82%), suggesting that RL-driven topology search yields greater returns on harder reasoning tasks where architectural diversity matters more. Gains on code generation and QA tasks are smaller but consistent. Compared to AFlow (76.86%), AMAT improves by 3.64% on average.

Method	HumanEval	MBPP	HotpotQA	GSM8K	MATH	Avg.
Vanilla	87.08	71.83	68.81	87.45	46.29	72.29
CoT	88.13	71.83	68.93	87.10	46.40	72.48
Complex CoT	87.49	72.36	69.43	86.89	46.53	72.54
Self-Consistency	88.60	73.60	69.79	87.57	47.91	73.49
MultiPersona	88.32	73.19	69.31	87.50	45.43	72.75
LLM-Debate	88.68	70.29	69.59	89.47	48.54	73.31
LLM-Blender	88.80	77.05	67.28	88.35	46.92	73.68
DyLAN	90.42	77.30	68.49	89.98	48.63	74.96
AgentVerse	89.29	74.28	67.47	89.91	47.35	73.66
MacNet	84.57	65.28	66.39	87.95	45.18	69.87
AutoAgents	87.64	71.95	68.24	87.69	45.32	72.17
GPTSwarm	89.32	77.43	69.65	89.14	47.88	74.68
ADAS	84.19	68.13	65.51	86.12	43.18	69.43
AgentSquare	89.08	78.46	69.53	87.62	48.51	74.64
AFlow	90.93	81.67	69.28	91.16	51.28	76.86
MaAS	<u>92.85</u>	<u>82.17</u>	69.49	<u>92.30</u>	<u>51.82</u>	<u>77.73</u>
AMAT (Ours)	93.32	83.87	72.41	93.81	59.10	80.50

Table 1: Performance comparison across five benchmarks using GPT-4o-mini-0718. We report Pass@1 (%) for code generation tasks (HumanEval, MBPP), Accuracy (%) for mathematical reasoning tasks (GSM8K, MATH), and F1 score (%) for question answering (HotpotQA). The best results are in **bold** and the second-best results are underlined. For AMAT, additional run-level statistics, including standard deviation, standard error, and 95% confidence intervals, are reported in Appendix E.

Method	Training				Inference				Overall
	Prompt Token	Completion Token	Cost	Time (min)	Prompt Token	Completion Token	Cost	Time (min)	Acc. (%)
LLM-Debate	-	-	-	-	3,275,764	10,459,097	6.76	92	48.54
DyLAN	22,152,407	16,147,052	13.01	508	6,081,483	3,303,522	2.89	39	48.63
MacNet	-	-	-	-	7,522,057	2,043,600	2.35	47	45.18
GPTSwarm	21,325,266	6,369,884	7.02	129	3,105,571	788,273	0.93	30	47.88
AFlow	33,831,239	29,051,840	22.50	184	2,505,944	2,151,931	1.66	23	51.28
MaAS	3,052,159	2,380,505	1.88	53	1,311,669	853,116	0.42	19	51.82
AMAT (Ours)	3,884,265	3,448,217	2.65	49	1,026,374	855,368	0.67	6	59.10

Table 2: Efficiency comparison on the MATH benchmark using GPT-4o-mini-0718. We report token consumption, total cost, and wall-clock time for both training and inference phases. The best results are highlighted with gray background .

4.3 Training and inference efficiency (RQ2)

We evaluate efficiency under a two-stage accounting protocol. Each method first performs its training, search, or optimization phase on the training split, and the resulting workflow, topology policy, or operator-selection distribution is then evaluated on the test split. For AMAT, MCTS expands graph states and queries the policy-value network during training; LLM token cost is incurred when a complete candidate DAG is instantiated and executed as a workflow. At test time, the learned policy generates the topology without running MCTS.

Table 2 reports training/search and test-time execution costs on MATH under this protocol. **Training:** AMAT finishes in 49 minutes—slightly faster than MaAS (53 min) and $3.8\times$ faster than AFlow (184 min). MaAS has lower raw API cost (\$1.88 vs. \$2.65), but both are far cheaper than AFlow (\$22.50), reflecting the benefit of amortizing topology search into reusable artifacts rather than relying on repeated workflow search. **Inference:** AMAT

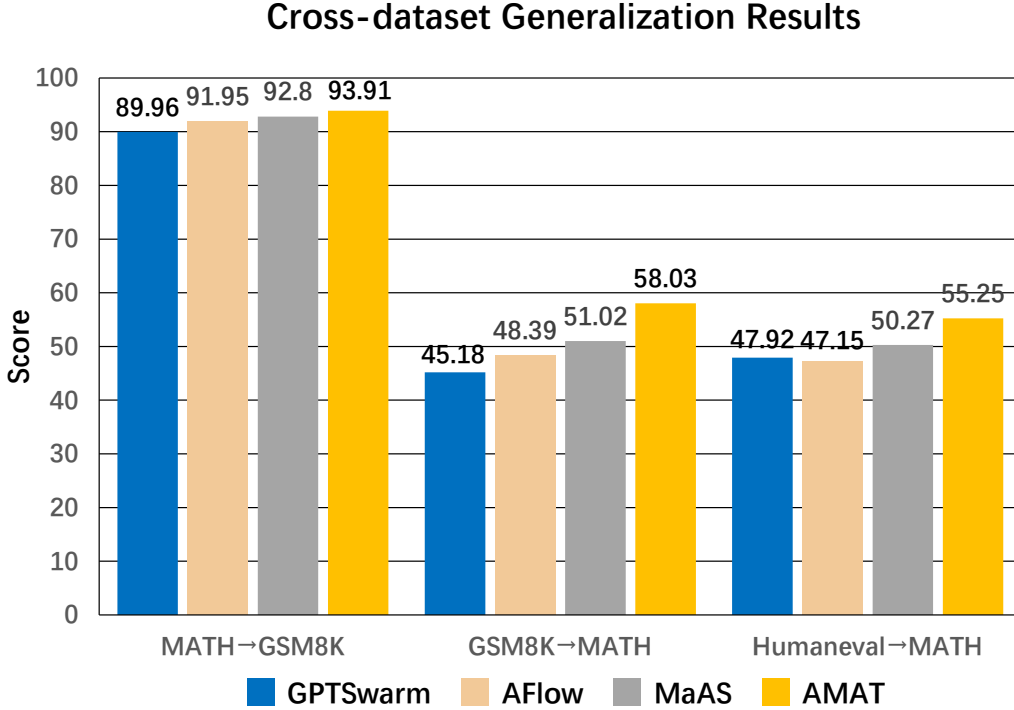


Figure 2: Cross-dataset generalization results across mathematical reasoning and code generation benchmarks using GPT-4o-mini-0718. Each method is trained on the source dataset and directly applied to the target dataset without fine-tuning. MATH→GSM8K and GSM8K→MATH represent within-domain transfer; HumanEval→MATH represents cross-domain transfer from code generation to mathematical reasoning.

completes in 6 minutes, $3.2\times$ faster than MaAS (19 min) and $3.8\times$ faster than AFlow (23 min). MaAS’s per-run cost (\$0.42) is lower than AMAT’s (\$0.67), but its accuracy is 7.28 points behind (51.82% vs. 59.10%). Put differently, AMAT reaches a higher performance level in less time, even if its test-time API cost is slightly higher.

4.4 Cross-task and cross-backbone transferability (RQ3)

To evaluate whether AMAT learns reusable structural preferences beyond a single training split, we conduct cross-dataset transfer experiments. Specifically, we train the policy-value network on one dataset and directly apply it to another dataset, either within the same broad domain or across domains, without any fine-tuning.

Figure 2 summarizes transfer performance across three scenarios. Within the mathematical domain, AMAT transfers well in both directions: the MATH-trained policy scores 93.91% on GSM8K (above its in-domain result), and the GSM8K-trained policy reaches 58.03% on MATH—only 1.07 points below in-domain. AFlow loses 2.89 points in the same GSM8K→MATH direction, and even at its in-domain score (51.82%), MaAS remains below AMAT’s transferred result. The cross-domain setting is more telling. A policy trained solely on HumanEval (code generation) achieves 55.25% on MATH, exceeding the in-domain scores of both AFlow (51.28%) and MaAS (51.82%). AFlow and GPTSwarm drop to 47.15% and 47.92% respectively, suggesting their performance depends more heavily on task-specific search. These results suggest that AMAT can transfer useful topology preferences across related task settings, with the strongest transfer observed within the same broad domain.

Method	MATH	
	Score (%)	Cost
Vanilla AMAT	59.10	0.67
w/o MCTS	56.48	0.84
w/o Action Mask	51.29	0.62
w/o Expert Buffer	53.14	0.90
w/o Multi-Obj Reward	59.00	1.00

Table 3: Ablation study on key components of AMAT using GPT-4o-mini-0718 on the MATH benchmark.

We further evaluate cross-LLM-backbone transferability. As shown in Table 8 (Appendix F), AMAT consistently improves over Vanilla on all benchmarks under both GPT-4o-mini-0718 and Qwen-Turbo, indicating that the learned topologies capture backbone-agnostic structural patterns.

4.5 Ablation study: component contributions (RQ4)

To understand the contribution of each component in AMAT, we conduct ablation studies on the MATH benchmark. As shown in Table 3, the **Action Mask** is the most critical component, with its removal causing the largest accuracy drop (59.10%→51.29%, -7.81%), confirming that constraining the search space is essential for discovering valid topologies. The **Expert Buffer** ranks second in importance (59.10%→53.14%, -5.96%) while also increasing cost (0.67→0.90), demonstrating its role in preserving high-quality architectural patterns and preventing catastrophic forgetting. Removing **MCTS** leads to moderate degradation (59.10%→56.48%, -2.62%) with increased cost (0.67→0.84), validating tree search as an effective policy improvement operator. Notably, the **Multi-Objective Reward** shows minimal accuracy impact (-0.10%) but substantially increases cost (0.67→1.00, +49.3%), indicating its primary contribution lies in efficiency optimization rather than performance gains.

A separate head-level ablation that isolates the policy-only, value-only, and full policy-value variants is provided in Appendix H. To provide qualitative intuition for the structures AMAT discovers, we visualize representative topologies for code generation and question answering tasks in Appendix I.

5 Conclusion

We presented AMAT, a framework that shifts automated multi-agent system design from transient online search to persistent offline policy learning. By formulating topology generation as a constrained Markov decision process and integrating MCTS with policy-value networks, AMAT addresses the critical limitations of ephemeral experience and value-agnostic optimization in existing approaches. Extensive experiments across five benchmarks demonstrate that AMAT achieves a 2.77–3.64 percentage-point improvement in average score over state-of-the-art baselines (MaAS and AFlow) while, on the MATH efficiency benchmark, requiring only 12%–38% of the training API cost of representative workflow-search baselines (AFlow and GPTSwarm) and completing inference $3.2\times$ – $3.8\times$ faster than the strongest baselines (MaAS and AFlow), with strong cross-task and cross-backbone generalization. Additional discussion on DAG workflow expressiveness and the relation to parameter- or prompt-level optimization is provided in Appendix J.

Acknowledgments

This work was supported by the Key Research Project of Chinese Academy of Sciences under Grant No. KGFZD-145-25-21, the National Natural Science Foundation of China under Grant No. 62322316 and 62503472, the CAS Project for Young Scientists in Basic

Research (Grant No. YSBR-123), and the Young Scientists Foundation of CSAA (Guidance Navigation and Control, GNC) under Grant CSAA-YSF2025-GNC-08.

Ethics Statement

All benchmarks used in our experiments (HumanEval, MBPP, GSM8K, MATH, and HotpotQA) are publicly available; no datasets involving personal information were collected or used. The research involves no human subjects. On the societal side, our primary contribution is reducing the computational cost of multi-agent system design, which we view as a positive development given the high API costs associated with existing methods. During the writing of this paper, large language models (LLMs) were used in a limited capacity to assist with writing tasks such as logical consistency review and language refinement. All scientific content, experimental design, analysis, and conclusions are entirely the work of the authors.

References

- Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs, 2018. URL <https://arxiv.org/abs/1711.00740>.
- Eitan Altman. Constrained markov decision processes. 1999. URL <https://api.semanticscholar.org/CorpusID:14906227>.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL <https://arxiv.org/abs/2108.07732>.
- Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation, 2024. URL <https://arxiv.org/abs/2309.17288>.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgén Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL <https://arxiv.org/abs/2107.03374>.
- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *CoRR*, abs/2308.10848, 2023. URL <https://doi.org/10.48550/arXiv.2308.10848>.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL <https://arxiv.org/abs/2110.14168>.
- Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2024. URL <https://openreview.net/forum?id=zj7YuTE4t8>.
- Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2021. URL <https://arxiv.org/abs/2012.09699>.

- Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution, 2023. URL <https://arxiv.org/abs/2309.16797>.
- Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=yflicZHC-19>.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023. URL <https://arxiv.org/abs/2211.10435>.
- 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, Xiwu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VtmBAGCN7o>.
- Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=t9U3LW7JVX>.
- Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *CoRR*, abs/2306.02561, 2023. URL <https://doi.org/10.48550/arXiv.2306.02561>.
- Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems!, 2019. URL <https://arxiv.org/abs/1803.08475>.
- Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for “mind” exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=3IyL2XWDkG>.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024a. URL <https://openreview.net/forum?id=zAdUB0aCTQ>.
- 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*, 2024b. URL <https://openreview.net/forum?id=XII0Wp1XA9>.
- A. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999. URL <https://api.semanticscholar.org/CorpusID:5730166>.
- Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large language model-based multi-agent collaboration. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=K3n5jPkrU6>.
- Christopher D. Rosin. Multi-armed bandits with episode context. In *ISAIM*, 2010. URL <http://dblp.uni-trier.de/db/conf/isaim/isaim2010.html#Rosin10>.

- Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation, 2016. URL <https://arxiv.org/abs/1511.06295>.
- Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016. URL <https://arxiv.org/abs/1511.05952>.
- 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 *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=mPdmDYIQ7f>.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=vAE1hFckW6>.
- David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529 (7587):484–489, 2016. doi: 10.1038/nature16961.
- Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2017. URL <https://arxiv.org/abs/1506.03134>.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024a. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL <http://dx.doi.org/10.1007/s11704-024-40231-1>.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL <https://arxiv.org/abs/2203.11171>.
- Yingxu Wang, Siwei Liu, Jinyuan Fang, and Zaiqiao Meng. Evoagentx: An automated framework for evolving agentic workflows, 2025. URL <https://arxiv.org/abs/2507.03616>.
- Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 257–279, Mexico City, Mexico, June 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.15. URL <https://aclanthology.org/2024.naacl-long.15/>.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL <https://arxiv.org/abs/2201.11903>.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=BAakY1hNKS>.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey, 2023. URL <https://arxiv.org/abs/2309.07864>.

- 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/>.
- Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6192–6217, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.315. URL <https://aclanthology.org/2025.naacl-long.315/>.
- Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet, 2025a. URL <https://arxiv.org/abs/2502.04180>.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui 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 *The Thirteenth International Conference on Learning Representations*, 2025b. URL <https://openreview.net/forum?id=z5uVAKwmjf>.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024. URL <https://openreview.net/forum?id=uTC9AFXIhg>.

A Dataset statistics

Building upon established methodologies in workflow automation, we evaluate our approach across three domains: code generation (HumanEval, MBPP), question answering (HotpotQA), and mathematical reasoning (GSM8K, MATH). Following prior work, we divide each dataset into training and test sets using a TRAIN:TEST ratio of approximately 1:4. We adopt standard task-specific metrics: Pass@1 for code generation, F1 score for question answering, and Accuracy for mathematical reasoning tasks. The detailed dataset statistics are provided in Table 4.

Domain	Dataset	Train	Test	Metric
Code Generation	HumanEval	33	131	pass@1
	MBPP	86	341	pass@1
Question Answering	HotpotQA	200	800	F1
Math Reasoning	GSM8K	264	1055	Accuracy
	MATH	119	486	Accuracy

Table 4: Dataset statistics.

B Training algorithm

Algorithm 1 AMAT: MCTS-bootstrapped training

Input: Task set $\mathcal{T}$, policy-value network f_θ , simulations N_{sim} , epochs E

Output: Trained policy-value network f_θ

Initialize dual buffers: $\mathcal{B}_{recent} \leftarrow \emptyset, \mathcal{B}_{expert} \leftarrow \emptyset$

for epoch = 1 **to** E **do**

 Sample task $t \sim \mathcal{T}$

 Initialize empty DAG state $s_0 \leftarrow (\mathcal{G}_0, \mathbf{X}_0, \text{meta}_0)$

$\mathcal{H} \leftarrow []$ {Trajectory buffer}

while $a_t \neq \text{Stop}$ **do**

 Compute valid actions $\mathcal{A}_{\text{valid}}(s_t)$ via DAG constraints

 Run N_{sim} MCTS simulations with PUCT and Dirichlet noise

 Compute $\pi_{MCTS}(a|s_t) \propto N(s_t, a)^{1/\tau}$

 Sample $a_t \sim \pi_{MCTS}$, append (s_t, π_{MCTS}) to $\mathcal{H}$

$s_{t+1} \leftarrow \mathcal{T}(s_t, a_t)$

end while

 Execute terminal DAG and compute r_{final} via Eq. (3)

 Store $(\mathcal{H}, r_{\text{final}})$ in $\mathcal{B}_{recent}$

if $r_{\text{final}} \geq \tau_{\text{expert}}$ **then**

 Store $(\mathcal{H}, r_{\text{final}})$ in $\mathcal{B}_{expert}$

end if

 Sample batch $\sim \alpha \cdot \mathcal{B}_{expert} + (1 - \alpha) \cdot \mathcal{B}_{recent}$

 Update θ by minimizing $\mathcal{L}(\theta)$ with time weights w_t

end for

return f_θ

C PVN configuration

D Operator and prompt interface

Table 6 summarizes the operator interface used in AMAT. Each operator is associated with a predefined prompt template or prompt-construction rule. Graph actions determine which operators are instantiated and how their outputs are connected.

Item	Configuration
Encoder	Structure-aware Transformer
Transformer layers	1
Hidden dimension	64
Attention heads	4
Graph representation	Fixed node slots + one global graph token
Input features	Operator/configuration, edges, graph size/progress
Topology budget	At most 6 nodes and 15 construction steps
MCTS budget	50 simulations per search step

Table 5: Compact configuration of the policy-value network used in AMAT.

Operator	Main input slots	Output
CustomCodeGenerate	problem statement, entry point, upstream feedback	candidate solution
Test	candidate solution, tests, error messages	verified/refined solution
ScEnsemble	candidate solutions, test metadata	selected solution
AnswerGenerate	question, context, upstream outputs	answer candidate
QAScEnsemble	answer candidates, context	final answer
Programmer	problem/context, code prompt slots	executable reasoning output

Table 6: Operator input-output interface used by AMAT. The concrete problem instance is supplied through the input slots during workflow execution.

E Run-level statistics

To supplement the main comparison table, we report AMAT’s run-level statistics over four independent runs in Table 7. We include the standard deviation, standard error, and 95% confidence interval for each benchmark.

F Cross-backbone generalization

We evaluate whether AMAT’s learned policies transfer across different LLM backbones. Table 8 compares Vanilla (direct prompting) and AMAT using GPT-4o-mini-0718 and Qwen-Turbo on all five benchmarks.

G Additional search comparison

We compare an intermediate AMAT checkpoint with a direct online-search baseline on HumanEval under the same operator set, valid-DAG constraints, GPT-4o-mini backbone, and decoding temperature. The direct online-search baseline evaluates 70 unique valid DAGs on the training split, selects the best training candidate, and evaluates only that candidate on the test split.

This checkpoint-level comparison (table 9) illustrates the amortized-search behavior of AMAT: offline MCTS trajectories provide training signals for the policy-value network, while test-time topology generation uses the learned policy.

H Head-level ablation

We further isolate the contribution of the policy and value heads on the MATH benchmark using GPT-4o-mini-0718. Table 10 shows that the policy head provides local action priors over legal graph-edit actions, while the value head estimates the utility of partial DAG states.

Benchmark	Run Scores	Mean	Std.	SE	95% CI
HumanEval	92.37 / 93.13 / 93.89 / 93.89	93.32	0.73	0.36	± 1.16
MBPP	82.40 / 84.75 / 84.75 / 83.58	83.87	1.12	0.56	± 1.79
HotpotQA	71.89 / 73.02 / 72.24 / 72.50	72.41	0.48	0.24	± 0.76
GSM8K	93.74 / 93.36 / 94.12 / 94.03	93.81	0.34	0.17	± 0.54
MATH	59.47 / 59.47 / 58.64 / 58.85	59.11	0.43	0.21	± 0.68

Table 7: Run-level statistics of AMAT over four independent runs. Std. denotes the sample standard deviation, SE denotes the standard error, and 95% CI is computed as the two-sided confidence interval over four runs.

Dataset	GPT-4o-mini-0718		Qwen-Turbo	
	Vanilla	AMAT	Vanilla	AMAT
HumanEval	87.08	93.32	88.93	95.04
MBPP	71.83	83.87	71.92	84.46
HotpotQA	68.81	72.41	64.45	72.88
GSM8K	87.45	93.81	88.60	94.55
MATH	46.29	59.10	65.69	76.49

Table 8: Cross-backbone generalization results. We compare Vanilla (direct prompting) and AMAT across different LLM backbones on all five benchmarks.

I Case study

To provide insights into the topologies discovered by AMAT, we visualize representative structures generated for different task domains in Figure 3.

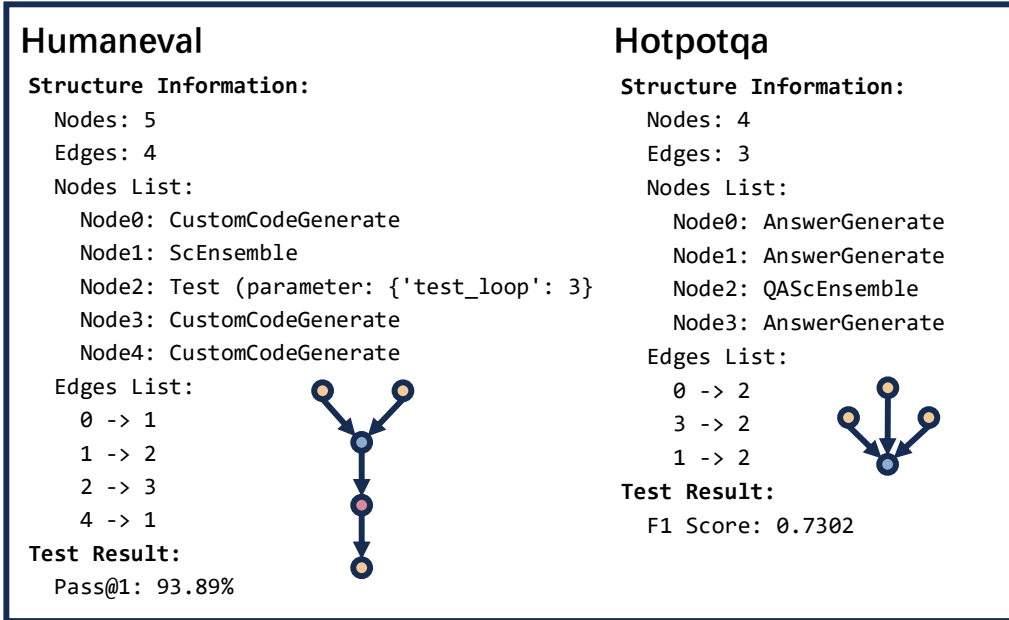


Figure 3: Visualization of discovered topologies for code generation (HumanEval) and question answering (HotpotQA). Node colors indicate agent roles: Generator (orange), Voter (blue), and Reflector (red).

Method	Search / training budget	HumanEval Pass@1
Direct online search	70 valid DAGs per run	90.08 $\pm$ 5.68
AMAT checkpoint	40 offline MCTS episodes	92.56 $\pm$ 1.13

Table 9: Checkpoint-level comparison between AMAT and direct online search on HumanEval.

Variant	Generated topology	MATH Acc.	Test cost
Policy-only	Custom	53.81	\$0.232
Value-only	Custom + Custom $\rightarrow$ ScEnsemble $\rightarrow$ Custom	54.84	\$0.880
Policy-value	Full AMAT	59.10	\$0.67

Table 10: Head-level ablation on MATH. The full policy-value model achieves the best accuracy, while the policy-only and value-only variants show different structural and cost preferences.

AMAT discovers distinct domain-specific topological patterns: for code generation, the optimal topology features two parallel generators feeding into a voter, followed by a reflector for error correction, which then connects to an additional generator for refinement; for question answering, a simpler parallel generation-aggregation pattern emerges with three generators converging to a single voter for consensus. These domain-specific structures suggest that code generation benefits from iterative self-reflection loops, while question answering tasks favor aggregating diverse perspectives through majority voting. Throughout training, topologies evolve from simple linear chains (1-2 nodes) in early stages to these sophisticated task-adapted patterns—structures that would be difficult to design manually.

J Additional scope discussion

We provide two brief scope clarifications. First, Table 11 summarizes the expressiveness of the finite-DAG workflow representation used in AMAT. Second, Table 12 clarifies how topology learning relates to parameter- and prompt-level optimization.

Pattern	Current support	Representation / extension
Parallel generation	Supported	Multiple generators feed aggregation
Voting / ensemble	Supported	Aggregator over upstream outputs
Generate-test-refine	Supported	Sequential test/refine chain
Fixed-round reflection	Supported	Unrolled as finite DAG nodes
Open-ended debate	Extension	Requires cyclic/dynamic execution
Persistent memory	Extension	Requires memory state
Conditional control flow	Extension	Requires branch actions and routing
Hyper-edge communication	Extension	Requires generalized edge actions

Table 11: Scope of the finite-DAG workflow representation used in AMAT.

Direction	Optimized object	Relation to AMAT
LLM fine-tuning	Model parameters	Can improve operators
Prompt optimization	Prompt text	Can improve templates
Workflow learning	Operator composition/data flow	AMAT setting

Table 12: Relation between AMAT and parameter- or prompt-level optimization.