# FlowMAS: Learning Multi-Agent Workflow Topology via Information-guided Generative Flow Network

**Haitao Wang**[ab*], **Chenjing Liang**[ab*], **Haipeng Zhang**[ab], **Jiawei Hu**[ab], **Sicheng Wang**[ab], **Songzhu Mei**[c], **Chenglu Wen**[ab], **Siqi Shen**[ab†], **Cheng Wang**[ab]
[a]Fujian Key Laboratory of Urban Intelligent Sensing and Computing, Xiamen University, Xiamen, China
[b]Key Laboratory of Multimedia Trusted Perception and Efficient Computing, Ministry of Education of China, Xiamen University, Xiamen, China
[c]School of Computer, National University of Defense Technology, China
{haitaowang,liangcj,zhanghaipeng,hujiawei,sichengwang}@stu.xmu.edu.cn,
{siqishen,clwen,cwang}@xmu.edu.cn,

## Abstract

Automated multi-agent systems offer clear advantages over manually designed ones in scalability and adaptability, but existing workflow topology methods still face important limitations. Search-based methods are often computationally expensive, textual-gradient-based methods rely on coarse-grained feedback, and existing generation-based methods are not well suited to discrete workflow topologies with complex dependencies. To address these limitations, we propose FlowMAS, a multi-agent workflow topology method based on Generative Flow Networks (GFlowNets). FlowMAS models workflow generation as reward-guided flow over the topology space and introduces three components: a GFlowNet-based topology generation backbone, a curiosity-driven module for structure-aware exploration, and an information-guided optimization module for evaluating intermediate topologies. Concretely, the curiosity-driven module encourages exploration of structurally novel workflows, while the information-guided module measures both the information contribution and the communication efficiency of different operators to favor more informative and effective collaboration patterns. Experiments on six benchmark datasets with three LLM backbones show that FlowMAS consistently outperforms multiple baselines.

## 1 Introduction

As the core carrier for coordinating multi-agent collaboration and decomposing complex tasks, agentic workflows have been widely applied to complex scenarios such as multi-hop question answering [1, 2, 3], code generation [4, 5, 6], and mathematical reasoning [7, 8, 9]. Agentic workflow topology [10] refers to how multiple agents or operators are organized and connected to coordinate reasoning, communication, and decision making toward a task. Its design [3, 11] largely determines the quality of collaboration, the efficiency of execution, and the overall capability of the system.

Existing agentic workflow topology methods can be categorized into (1) manually crafted methods such as MetaGPT [6], LLM-Majority [9], and LLM-Debate [12]; (2) search-based methods that search the workflow topology in a search space, such as ADAS [13] and AFlow [14]; (3) textual-gradient-based methods that optimize workflow topology through an LLM-based critic/rectifier, such as OneFlow [15] and DebFlow [16]; and (4) generation-based methods that use a variational autoencoder (VAE) [11], graph diffusion [17], or a multi-layer perceptron (MLP) [3] to generate workflows. Generation-based methods are the best-performing genre of topology methods [3, 17].

*Equal contribution
†Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

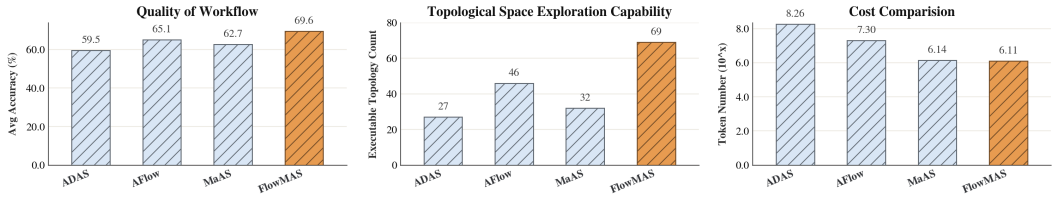


Figure 1: Experiments with Llama-3.1-8B-Instruct on HumanEval. **Left:** the quality of the generated workflow for solving specific test problems. **Middle:** the number of executable workflows discovered during topology exploration on the training set. **Right:** Total token consumption.

Manually crafted methods are labor-intensive [6] and costly to optimize for a specific query. Search-based methods need to explore a large design space (e.g., a search tree [14]), which could be token-expensive. As the number of agentic operators increases, the performance of search-based methods could drop while token usage increases significantly. Textual gradient-based methods (e.g., OneFlow [15]) are suboptimal because they usually optimize workflows through coarse-grained textual feedback, making it difficult to assign credit to specific workflow structures. Generation-based methods use generative backbones, such as VAEs [18] and diffusion models [17], to construct workflows. When trained on curated workflow datasets, these methods may inherit the structural biases of the training data, limiting the diversity of the generated topologies. Moreover, backbones originally developed for image [19] or vector generation [20] may not adequately capture the discrete structures and complex dependencies of agentic workflows. Together, these limitations can restrict the diversity of workflows produced by existing search-based and generation-based methods, hindering their adaptation to task-specific requirements, as shown in Figure 1.

To address the above workflow topology generation challenges, we propose FlowMAS, a multi-agent workflow topology method which adopts Generative Flow Networks (GFlowNets) [21, 22, 23] as the topology generation backbone. GFlowNets are a generative modeling framework whose core idea is to model the generation process as a trajectory flow on a Directed Acyclic Graph (DAG). Thanks to their efficient generation capability in discrete combinatorial spaces and reward-driven sampling characteristics, GFlowNets are a promising solution for overcoming the existing limitations.

Despite the above advantages, directly applying GFlowNets to agentic workflows faces two challenges: an information challenge and a structure challenge. Information from multiple agentic operators could be unique, redundant, or synergistic. As a method developed for molecular generation, GFlowNets fail to model the information exchange among agentic operators. However, the information exchange in workflow topology is key to task success [10]. Regarding the structure challenge, GFlowNets could be inefficient in finding diverse topology structure (shown in Figure 1 and Tab. 5) which can lead to suboptimial task performance.

FlowMAS consists of three key modules: a GFlowNet-based topology generation backbone, an information-guided optimization module for evaluating intermediate topologies and a curiosity-driven module for structure-aware exploration. Specifically, the GFlowNet backbone models workflow generation as a reward-guided flow over the topology space, the curiosity-driven module encourages exploration of structurally novel workflows during training, and the information-guided module measures the information contribution and communication efficiency of different operators to favor more informative and effective collaboration patterns.

Through extensive experiments on multiple benchmarks with multiple LLM backbones, we show that FlowMAS achieves the best performance among 15 baselines, thanks to its ability to generate diverse and query-specific workflows. Through ablation studies, we show that the partial information decomposition function could lead to better information sharing and that the curiosity-driven metric could lead to more diverse workflow topologies.

## 2 Background

### 2.1 Generative Flow Networks

Generative Flow Networks (GFlowNets) [21] are a probabilistic framework for training stochastic policies to generate discrete, combinatorial objects, originally developed to explore the diversity

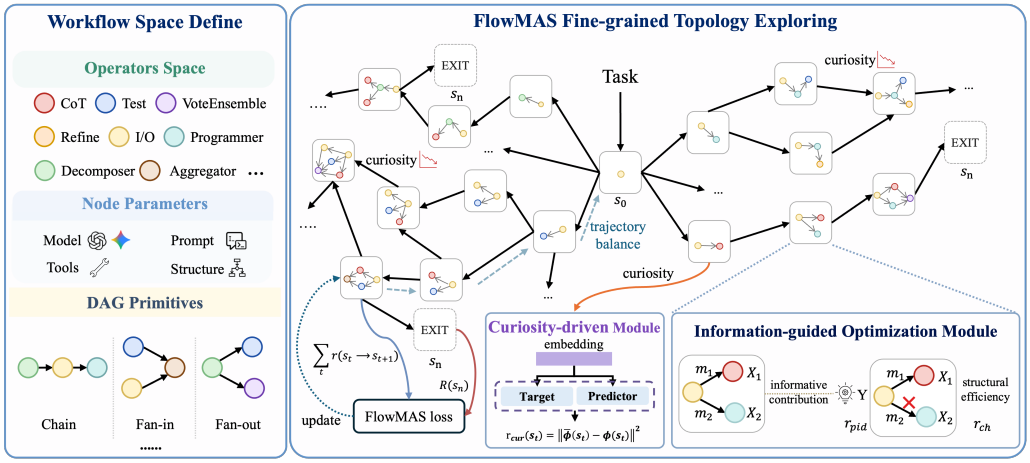


Figure 2: The overall framework of our proposed FlowMAS.

of molecular combinations in drug discovery. The generation process is modeled over a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{S}, \mathcal{A})$, where $\mathcal{S}$ denotes the state space and $\mathcal{A}$ denotes the set of valid state transitions. The goal of a GFlowNet is to learn a forward transition policy $P_F(s_{t+1}|s_t)$ such that the marginal probability of any terminal state $s_n$ is proportional to the reward function $R(x)$. A backward policy $P_B(s_t|s_{t+1})$ can also be defined to model the reverse process. Each edge is associated with a flow $F(s \to s')$, and the flow is *consistent* if, for all internal states $s$, the incoming flow equals the outgoing flow:

$$\sum_{s'' \to s} F(s'' \to s) = F(s) = \sum_{s \to s'} F(s \to s'). \tag{1}$$

GFlowNet methods offer several advantages when generating complex discrete structures. GFlowNets can generate high-quality discrete structures from scratch using only a reward function $R(\cdot)$, naturally supporting graphs and workflow substructures. Their explicit modeling of the construction process as sequences of decisions over a DAG allows consideration of dependencies and logical constraints, making them particularly well-suited for automatic exploration and generation of multi-agent workflows.

### 2.2 Partial Information Decomposition

Partial Information Decomposition (PID) [24] decomposes the information that multiple sources $\{X_k\}_{k=1}^{K}$ carry about a target $Y$ into unique, redundant, and synergistic components. For two sources, the decomposition can be written as

$$I(\{X_i, X_j\}; Y) = UI_i + UI_j + Red_{ij} + Syn_{ij}, \tag{2}$$

where $UI$, $Red$, and $Syn$ denote unique, shared, and synergistic target-relevant information, respectively. For multiple sources, PID further considers higher-order redundancy and synergy among source subsets. This framework allows quantifying the unique, shared, and synergistic contributions of multiple sources to a target, providing a principled tool to analyze redundancy and synergy in multi-agent systems or complex workflows.

## 3 Methodology

Figure 2 shows the overall framework of FlowMAS. Given a task input, FlowMAS uses GFlowNets to model the workflow distribution, a curiosity-driven module(CDM) to encourage structural exploration, and an information-guided optimization module(IOM) to favor high-quality workflows. The rest of this section presents the problem formulation in Section 3.1, the FlowMAS exploration mechanism in Section 3.2, and the information-guided optimization strategy in Section 3.3.

### 3.1 Problem Formulation

**Agentic Workflow.** We formulate the automated construction of LLM-based agentic systems as a workflow topology generation problem. Specifically, a workflow is represented as a directed acyclic graph (DAG) $\mathcal{W} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ denotes the set of nodes and $\mathcal{E}$ denotes the set of directed edges. Each node $v_i$ corresponds to an operator, i.e., a basic computational unit in the workflow. An operator is specified by its model $\mathcal{M}$, prompt $\mathcal{P}$, tool usage $\mathcal{T}$, and computational structure $\mathcal{C}$, where the computational structure defines how the operator runs internally, such as `CoT`, `Reflexion`, or `Programmer`.

A directed edge $e_{ij} = (v_i, v_j)$ represents the information flow from node $v_i$ to node $v_j$. In practice, we use three basic topology primitives to describe agentic workflows: chain, fan-in, and fan-out. A chain connects nodes sequentially and forms a linear pipeline. A fan-in merges information from multiple upstream nodes into a single downstream node, while a fan-out sends one node's output to multiple downstream nodes for parallel exploration. More complex topologies, such as tree, diamond, and hierarchical structures, can be constructed by combining these primitives.

**Automated Workflow Topology Generation.** Given a query $q$ and an evaluation function $\mathcal{R}(\cdot)$ that measures the quality of a workflow $\mathcal{W}$ generated for query $q$ from the perspectives of effectiveness and efficiency, the goal of the proposed method is to explore the combinatorial state space of workflows represented as DAG structures ($\mathcal{W} = (\mathcal{V}, \mathcal{E})$) and to identity high-quality solutions, as shown below:

$$\mathcal{P}_\theta(\mathcal{W}) \propto \mathcal{R}(\mathcal{W}), \tag{3}$$

where $\mathcal{P}_\theta(\cdot)$ is the topology generation strategy.

### 3.2 FlowMAS

FlowMAS learns a flow distribution function $Z_\theta$ such that the total flow of all generation paths from the source state $s_0$ to a terminal state $s_n$ satisfies Equation 3, i.e., the flow is proportional to the final value (e.g., task accuracy, token cost). Consequently, GFlowNets allocates limited flow across different structural paths, allowing high-value structures to receive higher flow while maintaining exploration diversity.

In FlowMAS, the workflow construction process is denoted as a trajectory $\tau = \{s_0, s_1, ..., s_n\}$, where each state $s_t$ represents a partially built workflow topology represented by DAG. Each state $s_t$ is uniformly encoded as:

$$h(s_t) = Emb(e_{task}, e_{op}, e_{topo}), \tag{4}$$

where $Emb(\cdot)$ is an embedding model, such as MiniLM [25]. The terms $e_{task}$, $e_{op}$, $e_{topo}$ denote the embeddings of the current task text, the operator pool, and the composed workflow topology, respectively. Based on the current state, the model performs actions $a_t$ to add nodes and connect edges, thereby gradually expanding the information flow paths.

However, directly applying GFlowNets to agentic workflow generation does not fully address the two challenges highlighted in the introduction. For the *structural challenge*, relying only on the terminal reward $r(s_n)$ provides little guidance about whether an intermediate topology is worth expanding, which can lead to redundant structures and insufficient exploration of diverse topologies. For the *information challenge*, the terminal reward does not indicate whether the current partial workflow already captures useful coordination patterns among operators, such as complementary or synergistic information exchange. As a result, promising intermediate structures may receive too little flow during generation.

To address these issues, the core idea of FlowMAS is to introduce *intermediate flow* into the generation process. Instead of assigning value only to the final workflow, we assign reward signals to intermediate transitions so that the flow can reflect both structural quality and information quality throughout topology construction. In this way, the model is encouraged to allocate more probability mass to trajectories whose intermediate states are already structurally efficient and information-effective, rather than waiting until the final reward to distinguish good workflows from bad ones.

Formally, we introduce an intermediate reward $r(s_t \to s_{t+1})$ on each transition and treat it as reward-induced local flow injected into the trajectory. This additional flow reshapes the original flow conservation constraint as follows:

$$\sum_{s_{t-1}} F(s_{t-1} \to s_t) = F(s_t) = \sum_{s_{t+1}} [F(s_t \to s_{t+1}) + r(s_t \to s_{t+1})], \tag{5}$$

here, $F(s_t)$ is the state flow of state $s_t$, and $F(s_t \to s_{t+1})$ is the edge flow for the transition from $s_t$ to $s_{t+1}$. Correspondingly, the trajectory balance constraint can be expressed as:

$$Z_\theta \prod_{t=0}^{n-1} P_F(s_{t+1}|s_t) = R(s_n) \prod_{t=0}^{n-1} \left[ P_B(s_t|s_{t+1}) + \frac{r(s_t \to s_{t+1})}{F(s_{t+1})} \right], \tag{6}$$

here, $P_F(\cdot)$ and $P_B(\cdot)$ denote the forward policy and the backward policy, respectively. A detailed derivation is provided in Appendix B.

### 3.3 Information-guided Topology Optimization

In this section, we describe the information-guided optimization module and curiosity-driven module in FlowMAS and specify the components of the intermediate reward $r(s_t \to s_{t+1})$ used to guide topology generation.

**Information-guided Optimization Module (IOM)** To address the information challenge, FlowMAS evaluates whether the local collaboration patterns formed during topology construction provide target-relevant and non-redundant information.

For each training instance, let $Y$ denote the representation of its ground-truth target answer. Given a local DAG primitive $D_i$ consisting of $K$ upstream operators whose messages are jointly consumed by a downstream operator, we introduce a PID-inspired synergistic information reward:

$$r_{pid}(D_i) = \max\left(0, I(D_i; Y) - \sum_{k=1}^{K} UI_k(D_i; Y) - Red(D_i; Y)\right) \in [0, 1], \tag{7}$$

where $I(D_i, Y)$ measures the target alignment of the aggregated incoming messages, $UI_k(D_i; Y)$ represents the target-specific contribution unique to the $k$-th message, and $Red(D_i; Y)$ captures the target-aligned information shared across incoming messages. Intuitively, $r_{pid}(D_i)$ favors local collaboration patterns whose aggregated messages exhibit stronger target relevance than can be explained by isolated message contributions and their shared component alone. The detailed computation of these terms is provided in Appendix C.

Besides synergistic information, we also evaluate whether a topology uses its communication paths efficiently. To capture this property, we introduce an effective information channel reward. For the $K$ upstream operator outputs in $D_i$, we construct a cosine-similarity Gram matrix $\mathcal{M}$ and compute its spectral entropy from the eigenvalues $\lambda$ to obtain

$$r_{ch}(D_i) = \frac{2^{H(\rho)} - 1}{K - 1}, \quad \text{where} \quad H(\rho) = -\sum_{j=1}^{K} \lambda_j \log_2 \lambda_j, \tag{8}$$

where $r_{ch}(\cdot)$ is the effective channel reward, $H(\cdot)$ is the spectral entropy, and $\rho$ is the trace-normalized Gram matrix (see Appendix D for details). Intuitively, this reward measures how evenly information is distributed across the communication channels of a substructure. A low reward indicates that information is concentrated in a few dominant paths, suggesting redundancy or inactive branches. A high reward indicates that multiple paths contribute more evenly, implying better coordination efficiency and less wasted computation. In multi-agent workflows, this term discourages redundant agents or edges and favors structures in which different agents make more balanced and effective contributions.

**Curiosity-driven Module (CDM)** To address the structural challenge, we introduce a curiosity-driven module that encourages exploration of under-visited topologies. It assigns higher reward to structurally novel states, improving the coverage of potentially useful workflow structures during training. Specifically, the structural reward is defined as $r_{\text{cur}}(s_t) = \|\bar{\phi}(s_t) - \phi(s_t)\|^2$, where $\phi(\cdot)$ is a fixed target network and $\bar{\phi}(\cdot)$ is a trainable prediction network.

We combine the curiosity-driven module and the information-guided optimization module into a unified transition reward for intermediate topology evaluation. Let $\mathcal{D}_t = \{\mathcal{D}_{t,1}, ..., \mathcal{D}_{t,n}\}$ denote the set of local DAG primitives evaluated at the transition $s_t \to s_{t+1}$:

$$r(s_t \to s_{t+1}) = \alpha r_{\text{cur}}(s_t) + \beta \sum_{D_i \in \mathcal{D}_t} \left(r_{pid}(D_i) + r_{ch}(D_i)\right), \tag{9}$$

where $\alpha$ and $\beta$ balance the structure-aware exploration reward and the information-guided optimization reward, respectively.

# 4 Experiment

## 4.1 Experimental Setup

**Tasks and Benchmarks.** We evaluate FlowMAS on six public benchmarks in our main experiments. Following AFlow [14], we use the same data partition protocol, splitting each benchmark into validation and test sets at a 1:4 ratio. For mathematical reasoning, we use GSM8K [7] and MATH [8]. For MATH, we select 605 level-5 problems from four representative categories: counting and probability, number theory, prealgebra, and precalculus. For code generation, we use HumanEval [4] and MBPP [5]. For reading comprehension, we use HotpotQA [1] and DROP [2], from which we randomly sample 1,000 examples each, following Hu et al. [13]. Detailed dataset statistics are provided in Appendix G.1. In addition, we evaluate the cross-task generalization ability of FlowMAS on BIG-Bench Hard (BBH) [26], with results presented in Section 4.5.

**Baselines.** We compare FlowMAS with three series of agentic baselines: (1) single-agent test-time scaling methods, including Vanilla LLM (IO), CoT [27], CoT-SC [28], ReAct [29], and Reflexion [30]; (2) hand-crafted multi-agent systems, including LLM-Majority [9], LLM-Debate [12], LLM-Blender [31], DyLAN [32], and AgentVerse [33]; (3) automated multi-agent systems, including ADAS [13], AgentNet [34], SELFORG [35], AFlow [14], and MaAS [3]. More details on baseline setups are provided in Appendix F.

**LLM backbone.** All methods are evaluated on three backbone LLMs, spanning one closed-source LLM (gpt-4o-mini [36]) and two open-source LLMs (Llama-3.1-8B-Instruct [37], Gemma-4-26B-A4B-it [38]). For all backbone LLMs, we set the sampling temperature to 0.2.

**Metrics.** For GSM8K and MATH, we report *solve rate* (%). For HumanEval and MBPP, we report *pass@1*. For HotpotQA and DROP, we report *F1*. For all methods, we additionally track total token usage as the cost metric. For each experiment type, we run five independent trials and report the mean and standard deviation.

## 4.2 Performance Analysis

Table 1 compares FlowMAS with 15 baselines on six benchmarks. Table 2 further reports the results of FlowMAS, IO, and five representative automated workflow-design baselines on GPT-4o-mini and Gemma-4-26B-A4B-it; the full comparisons with all baselines on these two backbones are provided in Appendix G.2.1. Averaging over all methods in each category and all three backbones, FlowMAS surpasses the hand-crafted multi-agent methods by 8.17 points and the existing automated workflow-design methods by 6.12 points. On Llama-3.1-8B-Instruct, FlowMAS outperforms AgentNet and SELFORG by 11.07 points on average, suggesting that its performance is less dependent on strong backbone capabilities. Moreover, compared with MaAS and AFlow, FlowMAS maintains consistent advantages across all three backbones, demonstrating stronger generality and overall effectiveness.

Table 1: Main results on Llama-3.1-8B-Instruct. We **bold** the best results and underline the second-best results. Each experiment is run five times to record the mean and standard deviation.

| Method | HumanEval | GSM8K | MATH | MBPP | HotpotQA | DROP | Avg. |
|---|---|---|---|---|---|---|---|
| IO | 50.1±1.4 | 73.7±1.4 | 18.9±1.5 | 67.1±1.3 | 60.5±1.6 | 68.8±2.1 | 56.52 |
| CoT [27] | 51.2±1.5 | 75.6±1.1 | 21.0±1.7 | 69.4±1.4 | 64.0±1.9 | 73.1±1.9 | 59.05 |
| CoT-SC [28] | 52.4±0.8 | 77.5±0.9 | 22.1±0.0 | 66.8±0.8 | 66.0±0.9 | 75.4±1.3 | 60.03 |
| ReAct [29] | 51.9±1.9 | 75.2±1.7 | 28.1±2.0 | 65.4±1.5 | 63.7±1.7 | 71.9±1.7 | 59.37 |
| Reflexion [30] | 50.5±1.7 | 79.1±1.1 | 22.0±2.1 | 65.7±1.4 | 57.6±1.5 | 64.6±2.0 | 56.58 |
| LLM-Majority [9] | 56.6±0.0 | 76.7±1.2 | 22.4±1.6 | 67.3±0.9 | 62.0±1.1 | 71.4±1.2 | 59.40 |
| LLM-Debate [12] | 53.4±1.3 | 82.7±1.4 | 19.9±1.7 | 65.4±1.2 | 66.9±1.6 | 64.4±1.4 | 58.78 |
| LLM-Blender [31] | 52.5±1.5 | 82.1±1.9 | 22.3±1.8 | 64.5±1.4 | 68.0±1.8 | 75.6±1.6 | 60.83 |
| DyLAN [32] | 51.9±1.4 | 83.0±1.7 | 27.6±2.0 | 68.3±1.5 | 63.2±1.9 | 70.2±1.8 | 60.70 |
| AgentVerse [33] | 52.4±1.7 | 76.9±2.1 | 26.7±2.1 | 66.3±1.6 | 68.2±1.5 | 70.8±1.6 | 60.22 |
| ADAS [13] | 59.5±2.4 | 79.5±1.8 | 22.0±3.0 | 64.0±1.4 | 60.9±2.9 | 69.0±2.6 | 59.15 |
| AgentNet [34] | 50.8±2.2 | 75.0±0.4 | 22.5±0.1 | 60.4±0.4 | 61.7±0.1 | 67.9±0.2 | 56.38 |
| SELFORG [35] | 61.9±0.6 | 83.1±0.1 | 26.0±0.2 | 67.0±0.5 | 65.7±0.1 | 71.2±0.1 | 62.48 |
| AFlow [14] | 65.1±1.4 | 85.1±1.6 | 35.3±2.2 | 72.9±1.4 | 69.2±1.9 | 77.7±2.1 | 67.55 |
| MaAS [3] | 62.7±0.9 | 85.0±1.1 | 37.5±1.9 | 73.4±1.0 | 70.3±1.6 | 77.5±1.2 | 67.73 |
| FlowMAS | **69.6**±1.5 | **86.2**±1.0 | **38.8**±1.8 | **75.6**±1.1 | **73.6**±1.2 | **79.2**±1.4 | **70.50** |

Table 2: Results of IO and selected automated MAS-design methods on Gemma-4-26B-A4B-it and GPT-4o-mini backbones. We **bold** the best results and underline the second-best results.

| Method | HumanEval | GSM8K | MATH | MBPP | HotpotQA | DROP | Avg. |
|---|---|---|---|---|---|---|---|
| **GPT-4o-mini** | | | | | | | |
| IO | 86.3±0.6 | 86.3±0.5 | 45.0±0.7 | 71.2±0.4 | 66.7±0.6 | 73.2±0.9 | 71.45 |
| ADAS [13] | 83.4±1.2 | 84.9±0.9 | 42.7±1.2 | 67.3±0.8 | 63.9±0.9 | 75.5±1.2 | 69.62 |
| AgentNet [34] | 86.8±0.9 | 91.8±0.0 | 47.5±0.1 | 71.1±0.2 | 64.5±0.2 | 80.8±0.0 | 73.75 |
| SELFORG [35] | 86.0±2.0 | 92.7±0.1 | 50.7±0.7 | 71.7±0.3 | 68.9±0.0 | 84.3±0.1 | 75.72 |
| AFlow [14] | 89.6±0.7 | 90.5±0.7 | 50.5±0.9 | 81.0±0.6 | 72.6±0.9 | 79.7±0.5 | 77.32 |
| MaAS [3] | 92.2±0.5 | 91.7±0.6 | 51.1±0.8 | 81.4±0.5 | 77.3±0.7 | 83.1±0.6 | 79.47 |
| FlowMAS | **94.4**±0.8 | **93.1**±0.4 | **52.6**±0.7 | **83.4**±0.5 | **80.0**±0.4 | **84.9**±0.2 | **81.40** |
| **Gemma-4-26B-A4B-it** | | | | | | | |
| IO | 90.8±0.8 | 90.5±0.7 | 61.3±0.9 | 73.3±0.6 | 74.9±0.8 | 83.9±1.0 | 79.12 |
| ADAS [13] | 88.5±1.2 | 90.0±1.0 | 59.2±1.2 | 71.2±0.8 | 76.7±1.1 | 86.5±1.2 | 78.68 |
| AgentNet [34] | 90.9±1.2 | 91.9±1.0 | 62.7±0.3 | 81.6±0.3 | 79.5±0.4 | 88.8±1.0 | 82.57 |
| SELFORG [35] | 91.3±0.3 | 92.1±0.3 | 63.1±0.4 | 77.8±0.4 | 79.5±0.5 | 90.4±0.5 | 82.37 |
| AFlow [14] | 92.6±0.8 | 93.4±0.9 | 65.0±1.0 | 83.5±0.7 | 82.7±1.0 | 89.9±1.1 | 84.52 |
| MaAS [3] | 92.4±0.6 | 95.1±0.7 | 65.1±1.0 | 83.1±0.6 | 83.8±0.8 | 90.7±0.7 | 85.03 |
| FlowMAS | **94.8**±0.8 | **95.6**±0.8 | **66.7**±0.9 | **87.1**±0.6 | **85.3**±0.8 | **92.0**±0.7 | **86.92** |

### 4.3 Cost Analysis

To assess whether FlowMAS is cost-effective in both optimization and deployment, we report the training and inference token consumption of different baselines on HumanEval in Table 3. We focus on token usage and accuracy because they are the most direct and comparable measures of efficiency across methods.

**Obs. 1: FlowMAS's optimization is resource-friendly.** As shown in Table 3, among optimization-oriented agentic workflow methods, FlowMAS achieves the highest accuracy with relatively low training token consumption. MaAS uses fewer training tokens than FlowMAS but is 6.9 accuracy points lower. This accuracy–cost trade-off is consistent with the BBH analysis in Figure 1, where MaAS explores substantially fewer executable topologies than FlowMAS.

Table 3: Efficiency comparison between FlowMAS and state-of-the-art baselines on the HumanEval. Train× and Infer× are relative to FlowMAS

| Method | Acc. (%) | Training tokens | Train× | Inference tokens | Infer× |
|---|---|---|---|---|---|
| DyLAN [32] | 51.9 | 5,842,507 | 15.2× | 25,064,472 | 27.8× |
| ADAS [13] | 59.5 | 84,526,998 | 220.0× | 97,526,460 | 108.3× |
| AgentNet [34] | 50.8 | 792,775 | 2.1× | 451,047 | 0.5× |
| SELFORG [35] | 61.9 | 853,528 | 2.2× | 2,755,046 | 3.1× |
| AFlow [14] | 65.1 | 6,253,861 | 16.3× | 13,690,205 | 15.2× |
| MaAS [3] | 62.7 | 223,042 | 0.6× | 1,151,392 | 1.3× |
| FlowMAS (Ours) | 69.6 | 384,278 | 1.0× | 900,287 | 1.0× |

**Obs. 2: FlowMAS enjoys superior token economy during inference.** As shown in Table 3, AFlow uses more than 15× the inference tokens of FlowMAS and still fails to match its accuracy. AgentNet substantially reduces inference cost by reusing experience summarized during training, but the reused experience is less reliable because it is entirely induced by LLM summarization, leading to noticeably lower accuracy. In contrast, FlowMAS achieves the best overall performance with a relatively low token budget.

### 4.4 Ablation Study

We conduct two ablations to answer two questions. **Q1:** Is GFlowNet a better generation-based backbone for agentic flow than other generation models? **Q2:** Are the two key method components, the curiosity-driven module (CDM) and the information-guided optimization mechanism (IOM), introduced in Section 3.3 necessary, and can they not be replaced arbitrarily by other alternatives? Tables 4 and 5 summarize the results.

**Q1: Generator backbone ablation.** Table 4 examines whether GFlowNet is a stronger core architecture than other advanced topology generators. To keep the comparison as clean as possible, we remove the curiosity-driven module (CDM) and information-guided optimization mechanism (IOM), and compare the GFlowNet backbone with a Graph Diffusion Model [20, 39], a VAE [18], a Transformer [40] and a MLP [41] controller under otherwise aligned settings, using GTD [17], G-Designer [11], and MaAS [3] as references. Among the five generator-backbone variants, GFlowNet achieves the best result on all three benchmarks. This suggests that, relative to the alternative backbones, it provides a more suitable inductive bias for modeling complex discrete workflow structures with logical constraints. More specifically, the Graph Diffusion Model [20] and the Transformer [40] obtain HumanEval accuracies below 60%. The VAE attains 46.1 on HotpotQA, substantially below the 67.3 achieved by the GFlowNet variant.

Table 4: Q1. Generator backbone ablation.

| Method | HumanEval | GSM8K | HotpotQA |
|---|---|---|---|
| ADAS [13] | 59.5 | 79.5 | 60.9 |
| FlowMAS | **69.6** | **86.2** | **73.6** |
| FlowMAS (VAE) | 62.6 | 81.6 | 46.1 |
| FlowMAS (Graph Diffusion Model) | 55.7 | 80.8 | 50.6 |
| FlowMAS (Transformer) | 58.8 | 82.5 | 53.2 |
| FlowMAS (MLP) | 60.3 | 81.1 | 62.9 |
| FlowMAS (GFlowNet, w/o CDM, w/o IOM) | 62.8 | 82.9 | 67.3 |

**Q2: FlowMAS-component ablation.** Table 5 studies the two core components of FlowMAS: the curiosity-driven module (CDM) and the information-guided optimization mechanism (IOM). The results show that removing either component degrades performance across all three benchmarks. Among the direct component-removal variants, removing both causes the largest drop, reducing HumanEval from 69.6 to 62.8. Replacing CDM with graph edit distance (GED) [42] or intrinsic curiosity module (ICM) [43] also remains worse than the full model: the ICM variant reaches 67.9 on HumanEval, only slightly above 67.3 without CDM. Replacing IOM with alternative reward designs is similarly ineffective: both Embedding Similarity and LLM-as-Judge consistently underperform the

Table 5: Q2. Method-component ablation. HE, G8K, and HQ denote accuracy on HumanEval, GSM8K, and HotpotQA, respectively. "Avg. Topo. Count" denotes the average number of executable topologies discovered during training across the three benchmarks.

| Variant | HE Acc. | G8K Acc. | HQ F1. | Avg. Topo. Count |
|---|---|---|---|---|
| ADAS [13] | 59.5 | 79.5 | 60.9 | 31 |
| w/o CDM | 67.3 | 84.9 | 72.0 | 68 |
| w/o CDM, w/ Graph Edit Distance [42] | 66.8 | 85.2 | 72.1 | 73 |
| w/o CDM, w/ Intrinsic Curiosity Module [43] | 67.9 | 85.4 | 72.7 | 73 |
| w/o IOM | 65.0 | 84.2 | 69.3 | 60 |
| w/o IOM, w/ Embedding Similarity | 62.0 | 84.2 | 68.5 | 56 |
| w/o IOM, w/ LLM-as-Judge | 65.6 | 84.9 | 69.3 | 60 |
| w/o IOM, w/ O-Information [44] | 65.0 | 84.9 | 70.6 | 66 |
| w/o CDM & IOM | 62.8 | 82.9 | 67.3 | 51 |
| FlowMAS | **69.6** | **86.2** | **73.6** | 81 |

full model across all three benchmarks, with Embedding Similarity reducing HumanEval accuracy to 62.0. The topology count shows a similar trend, where the full FlowMAS explores more executable topologies than its component-ablated and alternative-reward variants. Overall, these results suggest that performance gains do not come from simply combining related modules; instead, the two proposed components are jointly necessary for improving both exploration diversity and workflow quality.

More ablation and analysis experiments are provided in Appendix G.

### 4.5 Generalization Analysis

To further evaluate the generalization ability of FlowMAS, we conduct experiments on BIG-Bench Hard (BBH) [26], which covers 23 distinct task types. We sample approximately 20% of the examples from 10 BBH task types, including Object Counting, Causal Judgement, and Formal Fallacies, to train FlowMAS, and evaluate on test sets totaling 5,000 examples from both seen and unseen task types. The unseen task types include Multistep Arithmetic, Temporal Sequences, and Logical Deduction.

As shown in Table 6, FlowMAS consistently outperforms AFlow and MaAS under both Llama-3.1-8B-Instruct and GPT-4o-mini, demonstrating its ability to generalize across diverse reasoning tasks.

Table 6: Generalization results on BIG-Bench Hard across different LLM backbones.

| Backbone | Method | Accuracy |
|---|---|---|
| Llama-3.1-8B-Instruct | AFlow | 43.7 |
| | MaAS | 60.0 |
| | FlowMAS | **69.0** |
| GPT-4o-mini | AFlow | 75.7 |
| | MaAS | 81.6 |
| | FlowMAS | **85.0** |

### 4.6 Scalability with Larger Operator Spaces

We also examine how FlowMAS scales to larger operator spaces by expanding the candidate operator set on HumanEval from 10 to 15 and 20. The added operators are deliberately chosen to be less relevant to the benchmark, including `RhetoricalCritic`, which performs rhetorical and stylistic analysis, and `CreativeStoryteller`, which generates fictional stories based on the problem statement. This setting enlarges the workflow search space and introduces distracting alternatives, testing whether FlowMAS can identify effective workflows from a broader, less task-focused operator pool.

Table 7: Scalability comparison on HumanEval with different numbers of candidate operators.

| # Operators | Method | Training Tokens | GPU Hours | Wall-clock Time (min) | Accuracy |
|---|---|---|---|---|---|
| 10 | AFlow | 6,253,861 | 1.61 | 97 | 65.1 |
| | MaAS | 223,042 | 0.52 | 31 | 62.7 |
| | FlowMAS | 384,278 | 0.65 | 39 | **69.6** |
| 15 | AFlow | 9,746,315 | 2.90 | 174 | 61.8 |
| | MaAS | 493,110 | 0.78 | 47 | 61.8 |
| | FlowMAS | 427,057 | 0.70 | 42 | **68.7** |
| 20 | AFlow | 14,284,672 | 3.92 | 235 | 58.8 |
| | MaAS | 627,634 | 0.83 | 50 | 59.5 |
| | FlowMAS | 652,095 | 0.85 | 51 | **65.6** |

As shown in Table 7, the performance of all methods decreases as the operator space grows, while the optimization cost generally increases. Nevertheless, FlowMAS consistently achieves the best accuracy across different operator-space sizes and maintains a relatively low optimization cost. Even with 20 candidate operators, FlowMAS achieves 65.6 accuracy, compared with 58.8 for AFlow and 59.5 for MaAS.

## 5 Related Work

Existing methods for agentic workflow topology can be roughly grouped into four categories. Static methods, such as AgentVerse [33], LLM-Debate [12], and MetaGPT [6], rely on manually designed topologies and are therefore labor-intensive. Search-based methods treat the workflow as a design object and optimize it through repeated evaluation. Representative examples include AFlow [14], which searches over topologies with tree search, ADAS [13], which searches in a code space, and EvoAgent [45], which uses population-based search. While effective, these methods usually incur high evaluation cost, and their search quality is often constrained by the workflow representation and the search strategy.

Related to search are topology pruning methods, such as DyLAN [32] and AgentDropout [46], which start from a fixed super-graph and perform run-specific activation or pruning. Although effective, their performance is bounded by the quality of the initial super-graph. Textual gradient-based methods, such as OneFlow [15], optimize workflows through text feedback from LLM-based critics or rectifiers, but such feedback is typically coarse-grained and makes structure-level credit assignment difficult.

Generation-based methods, including VAE-based G-Designer [11], MLP-based MaAS [3], and diffusion-based GTD [17], directly generate workflow topologies at inference time. However, these architectures are not specifically designed for discrete workflow generation with complex dependencies, and they often lack autonomous exploration. As a result, they may produce limited topology diversity and struggle to generate task-specific workflows that adapt well to different problems. In contrast, our method performs fine-grained step-level generation during training and learns a topology distribution through active exploration over the combinatorial workflow space.

Our work focuses on DAG-structured workflow topologies. Other workflow representations also exist, such as Python code in MAS-GPT [47], domain-specific languages in AutoFlow [48], and symbolic primitive representations for executable workflow construction in AutoRAS [49].

## 6 Conclusion

In this work, we propose FlowMAS, which combines a generative flow network topology generation backbone, an information-guided optimization module for evaluating intermediate topologies, and a curiosity-driven module for structure-aware exploration. Together, these components address both the structural challenge of exploring diverse workflow topologies and the information challenge of identifying effective collaboration patterns among operators. Extensive experiments show that FlowMAS can generate more diverse and task-adaptive workflows and consistently outperforms strong baselines across different LLM backbones.

## 7 Acknowledgments

This work was partially supported by a grant from the National Key Laboratory of Unmanned Aerial Vehicle Technology Intelligence Development Fund. We thank the anonymous reviewer for their valuable suggestions.

## References

[1] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2369–2380, 2018.

[2] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), Volume 1 (Long and Short Papers)*, pages 2368–2378, 2019.

[3] Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pages 75834–75852, 2025.

[4] 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.

[5] 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.

[6] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu 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, ICLR*, 2024.

[7] 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.

[8] 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. *arXiv preprint arXiv:2103.03874*, 2021.

[9] Vighnesh Subramaniam, Yilun Du, Joshua B Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[10] Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for llm agents, 2026.

[11] 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 *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pages 76678–76692, 2025.

[12] 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 (ICML)*, 2024.

[13] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

[14] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025.

[15] Jiawei Xu, Arief Koesdwiady, Sisong Bei, Yan Han, Baixiang Huang, Dakuo Wang, Yutong Chen, Zheshen Wang, Peihao Wang, Pan Li, et al. Rethinking the value of multi-agent workflow: A strong single agent baseline. *arXiv preprint arXiv:2601.12307*, 2026.

[16] Jinwei Su, Yinghui Xia, Yiqun Duan, Jun Du, Jianuo Huang, Tianyu Shi, and Lewei He. Debflow: Automating agent creation via agent debate, 2025.

[17] Eric Hanchen Jiang, Guancheng Wan, Sophia Yin, Mengting Li, Yuchen Wu, Xiao Liang, Xinfeng Li, Yizhou Sun, Wei Wang, Kai-Wei Chang, et al. Dynamic generation of multi-llm agents communication topologies with graph diffusion models. *arXiv preprint arXiv:2510.07799*, 2025.

[18] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR*, 2014.

[19] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695. IEEE, 2022.

[20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

[21] Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.

[22] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:27381–27394, 2021.

[23] Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in gflownets. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:5955–5967, 2022.

[24] Paul L Williams and Randall D Beer. Nonnegative decomposition of multivariate information. *arXiv preprint arXiv:1004.2515*, 2010.

[25] 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. *Advances in neural information processing systems (NeurIPS)*, 33:5776–5788, 2020.

[26] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023.

[27] 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 *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[28] 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 (ICLR)*, 2023.

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

[30] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:8634–8652, 2023.

[31] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pages 14165–14178, 2023.

[32] 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 (COLM)*, 2024.

[33] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[34] Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025), Poster Session*, 2025.

[35] Nurbek Tastan, Samuel Horváth, and Karthik Nandakumar. Stochastic self-organization in multi-agent systems. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

[36] OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence. `https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence`, 2024.

[37] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[38] Google AI. Gemma 4 model card. `https://ai.google.dev/gemma/docs/core/model_card_4`, 2026.

[39] Zhe Xu, Ruizhong Qiu, Yuzhong Chen, Huiyuan Chen, Xiran Fan, Menghai Pan, Zhichen Zeng, Mahashweta Das, and Hanghang Tong. Discrete-state continuous-time diffusion for graph generation. *Advances in Neural Information Processing Systems*, 37:79704–79740, 2024.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[41] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[42] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

[43] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[44] Fernando E Rosas, Pedro AM Mediano, Michael Gastpar, and Henrik J Jensen. Quantifying high-order interdependencies via multivariate extensions of the mutual information. *Physical Review E*, 100(3):032305, 2019.

[45] Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In *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)*, pages 6192–6217, 2025.

[46] Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *ACL*, pages 24013–24035, 2025.

[47] 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 *ICML*, 2025.

[48] Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. Autoflow: Automated workflow generation for large language model agents. *CoRR*, abs/2407.12821, 2024.

[49] Yang Yue, Xuancheng Zhu, Yuyang Ma, Guoshun Nan, Zihan Dou, Jingru Shan, Congyu Guo, Ji Zhang, Hua Wang, and Jingfeng Zhang. Autoras: Learning robust agentic systems with primitive representations. *arXiv preprint arXiv:2606.21445*, 2026.

[50] Ling Pan, Dinghuai Zhang, Aaron Courville, Longbo Huang, and Yoshua Bengio. Generative augmented flow networks. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[51] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

# Appendix

## A FlowMAS Algorithm

**Algorithm 1** FlowMAS

**Require:** Training set $\mathcal{D}$, operator set $\mathcal{O}$, forward-policy parameters $\theta_F$, backward-policy parameters $\theta_B$, state-flow parameters $\theta_Z$, maximum steps $T$, repetitions $N$, mini-batch $\mathcal{B}$
**Require:** Reward coefficients $\alpha, \beta, \lambda_{\text{cost}}$
Initialize graph-edit GFlowNet with forward policy $P_F(\cdot \mid s; \theta_F)$, backward policy $P_B(\cdot \mid s'; \theta_B)$, and flow normalizer parameters $\theta_Z$
Initialize optimizer and CDM predictor / target networks
**for** $i \leftarrow 1$ to $N$ **do**
  **for** each batch $\mathcal{B} \subset \mathcal{D}$ of size $|\mathcal{B}|$ **do**
    Initialize an empty trajectory set $\mathcal{T} \leftarrow \emptyset$
    **for** each problem $x \in \mathcal{B}$ **do**
      Create a workflow trajectory $\tau$ with `ROOT` node
      **for** $t \leftarrow 1$ to $T$ **do**
        Encode the current state $s_t = (e_{task}, e_{op}, e_{topo})$
        Build valid graph-edit action set $\mathcal{A}(s_t)$
        Sample a forward action $a_t \sim P_F(\cdot \mid s_t; \theta_F)$
        **if** $\mathcal{O}_{\texttt{EarlyStop}} \in a_t$ **then**
          **break**
        **end if**
        Execute graph edit $a_t$, transition to state $s_{t+1}$
        Infer the reverse edit probability $P_B(a_t^{-1} \mid s_{t+1}; \theta_B)$
        Compute CDM exploration reward $r_{\text{cur}}^{(t)}$
        $r_{\text{step}}^{(t)} \leftarrow \alpha \cdot r_{\text{cur}}^{(t)}$
        **if** the updated workflow induces a fan-out or fan-in structure **then**
          Compute PID-inspired synergistic information reward $r_{\text{pid}}^{(t)}$
          Compute effective channel reward $r_{\text{ch}}^{(t)}$

$$r_{\text{step}}^{(t)} \leftarrow \alpha \cdot r_{\text{cur}}^{(t)} + \beta \cdot \sum_{D_i} (r_{\text{pid}}^{(t)} + r_{\text{ch}}^{(t)})$$

        **end if**
      **end for**
      Run final verification / testing to obtain prediction $\hat{y}$ and score $s(\hat{y})$
      Compute terminal reward only after the full workflow topology is completed:

$$R_{\text{term}} = max(0, s(\hat{y}) - \lambda_{\text{cost}} \cdot c(\hat{y})) \in [0, 1]$$

      Store the trajectory $\tau$ together with $R_{\text{term}}$ and $\{r_{\text{step}}^{(t)}\}_{t=1}^{|\tau|}$ in $\mathcal{T}$
    **end for**
    Compute the GFlowNet trajectory objective $\mathcal{L}_{\text{TB}}$ over $\mathcal{T}$
    Update $\theta_F$, $\theta_B$, and $\theta_Z$ by backpropagating $\mathcal{L}_{\text{TB}}$
    Update the CDM predictor network with the visited states in $\mathcal{T}$
  **end for**
**end for**
**Return:** trained graph-edit GFlowNet $\{P_F, P_B, Z\}$

## B Derivation of Trajectory Balance Constraint

In standard GFlowNets, the flow consistency constraint is defined as:

$$F(s_{t-1}) P_F(s_t \mid s_{t-1}) = F(s_t) P_B(s_{t-1} \mid s_t), \tag{10}$$

This constraint ensures that the trajectory probability from the initial state to the terminal state is consistent with the terminal reward. By introducing an intermediate reward $r(s_{t-1} \to s_t)$ at the state transition level, we obtain a new flow consistency relation:

$$F(s_{t-1})P_F(s_t \mid s_{t-1}) = F(s_t)P_B(s_{t-1} \mid s_t) + r(s_{t-1} \to s_t), \tag{11}$$

Based on Equation 5, we define the forward and backward policies as:

$$P_F(s_t|s_{t-1}) = \frac{F(s_{t-1} \to s_t) + r(s_{t-1} \to s_t)}{F(s_{t-1})}, \qquad P_B(s_{t-1}|s_t) = \frac{F(s_{t-1} \to s_t)}{F(s_t)}, \tag{12}$$

For a trajectory $\tau = \{s_0, s_1, ..., s_n\}$ of length $n$, by recursively expanding Equation 11, we have:

$$\begin{aligned} F(s_0)P_F(s_1 \mid s_0) &= F(s_1)P_B(s_0 \mid s_1) + r(s_0 \to s_1), \\ F(s_1)P_F(s_2 \mid s_1) &= F(s_2)P_B(s_1 \mid s_2) + r(s_1 \to s_2), \\ &\vdots \\ F(s_{n-1})P_F(s_n \mid s_{n-1}) &= F(s_n)P_B(s_{n-1} \mid s_n) + r(s_{n-1} \to s_n). \end{aligned} \tag{13}$$

By multiplying both sides of these equations over all transitions, we obtain:

$$F(s_0) \cdots F(s_{n-1}) \prod_{t=0}^{n-1} P_F(s_{t+1}|s_t) = F(s_1) \cdots F(s_n) \prod_{t=0}^{n-1} [P_B(s_t|s_{t+1}) + \frac{r(s_t \to s_{t+1})}{F(s_{t+1})}], \tag{14}$$

By rearranging the above equation, we obtain the corresponding edge-based reward-augmented formulation for trajectory balance as:

$$F(s_0) \prod_{t=0}^{n-1} P_F(s_{t+1} \mid s_t) = F(s_n) \prod_{t=0}^{n-1} \left[ P_B(s_t \mid s_{t+1}) + \frac{r(s_t \to s_{t+1})}{F(s_{t+1})} \right], \tag{15}$$

where $F(s_0) = Z_\theta = \sum_x R(x) + \sum_{s_{t-1} \to s_t} r(s_{t-1} \to s_t)$, and $F(s_n) = R(x)$. Our method relies on edge-based intermediate reward augmentation. The design details of state-based intermediate reward augmentation can be found in [50].

## C Detailed Calculation of the PID-inspired Reward

This section provides the detailed computation of the PID-inspired reward introduced in Eq. 7. The reward is computed independently for each training instance and each local DAG primitive, without estimating mutual information from batch-level or dataset-level empirical distributions.

For a training instance, let $T$ denote the ground-truth target answer. Consider a local DAG primitive $D_i$ in which $K$ upstream operators $\{o_k\}_{k=1}^K$ produce messages $\{m_k\}_{k=1}^K$ that are jointly consumed by a downstream operator. We first encode the target answer and the incoming messages into normalized representations:

$$Y = \frac{\mathrm{Emb}(T)}{\| \mathrm{Emb}(T) \|_2}, \qquad X_k = \frac{\mathrm{Emb}(m_k)}{\| \mathrm{Emb}(m_k) \|_2}, \tag{16}$$

where $\mathrm{Emb}(\cdot)$ denotes the embedding encoder.

We measure the target alignment of each incoming message as

$$I(X_k; Y) = \max\left(0, \mathrm{sim}(X_k, Y)\right), \tag{17}$$

where $\mathrm{sim}(\cdot, \cdot)$ denotes cosine similarity.

To characterize the target alignment of the aggregated incoming messages, we compute

$$I(D_i;Y) = \max\left(0, \operatorname{sim}\left(\frac{\sum_{k=1}^{K} X_k}{\left\|\sum_{k=1}^{K} X_k\right\|_2}, Y\right)\right). \tag{18}$$

Following the redundancy–uniqueness–synergy intuition of Partial Information Decomposition (PID), we approximate the target-aligned information shared across all incoming messages as

$$Red(D_i;Y) = \min_k I(X_k;Y). \tag{19}$$

The contribution unique to the $k$-th message is then defined as

$$UI_k(D_i;Y) = I(X_k;Y) - Red(D_i;Y). \tag{20}$$

Finally, the PID-inspired synergistic information reward is computed as

$$r_{pid}(D_i) = \max\left(0, I(D_i;Y) - \sum_{k=1}^{K} UI_k(D_i;Y) - Red(D_i;Y)\right) \in [0,1]. \tag{21}$$

Notably, the target representation $Y$ and the resulting reward are used only during training; neither the ground-truth answer nor the IOM reward is required during inference.

## D Derivation of Spectral Entropy

For a local DAG primitive $D_i$, the outputs of its $K$ upstream operators are denoted as $\{Z_1, Z_2, \ldots, Z_K\}$. For each output, the normalized embedding is computed as

$$\tilde{z}_j := \frac{\mathrm{Emb}(Z_j)}{\|\mathrm{Emb}(Z_j)\|_2} \in \mathbb{R}^d, \tag{22}$$

where $\mathrm{Emb}(\cdot)$ denotes the embedding model. Further, the cosine-similarity Gram matrix $\mathcal{M}$ is computed as

$$\mathcal{M} \in \mathbb{R}^{K\times K}, \qquad \mathcal{M}_{pq} = \tilde{z}_p^\top \tilde{z}_q. \tag{23}$$

Applying trace normalization to the Gram matrix $\mathcal{M}$ yields

$$\rho = \frac{\mathcal{M}}{\mathrm{Tr}(\mathcal{M})}. \tag{24}$$

Let $\{\lambda_1, \lambda_2, \ldots, \lambda_K\}$ denote the eigenvalues of $\rho$. Based on the eigenvalue distribution, the spectral entropy $H(\rho)$ is defined as

$$H(\rho) = -\sum_{j=1}^{K} \lambda_j \log_2 \lambda_j. \tag{25}$$

Accordingly, the effective channel reward is computed as

$$r_{ch}(D_i) = \frac{2^{H(\rho)} - 1}{K - 1}. \tag{26}$$

## E Case Study

Figure 3 visualizes the different workflow topologies designed by AFlow, MaAS, and FlowMAS on the DROP benchmark. Compared with AFlow and MaAS, FlowMAS produces a more task-adaptive workflow structure that better separates evidence extraction, reasoning, and answer aggregation.

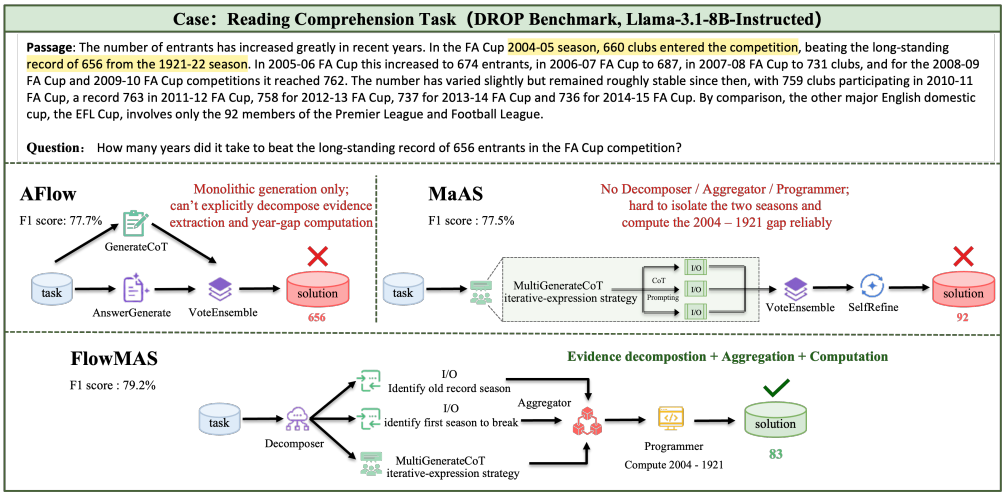


Figure 3: Case study of the workflow topologies designed by AFlow, MaAS, and FlowMAS on the DROP benchmark.

## F Baseline Methods and FlowMAS Variants

We consider three categories of methods for comparison: (1) single-agent test-time scaling methods: Vanilla LLM (IO), CoT, CoT-SC, ReAct, and Reflexion; (2) hand-crafted multi-agent systems: LLM-Majority, LLM-Debate, LLM-Blender, DyLAN, and AgentVerse; (3) automated multi-agent systems: ADAS, AgentNet, SELFORG, AFlow, and MaAS.

1. **Vanilla LLM (IO)**: a direct input-output baseline that queries the LLM once and uses the returned answer without explicit deliberation, search, or multi-agent coordination.
2. **CoT [27]**: a prompt-based method that guides the LLM to generate intermediate reasoning steps before producing the final answer.
3. **CoT-SC [28]**: a test-time scaling method that samples multiple chain-of-thought solutions and selects the final answer by majority voting across sampled outputs.
4. **ReAct [29]**: a prompting framework that interleaves reasoning and action generation, allowing the model to iteratively reason over observations and external feedback[1].
5. **Reflexion [30]**: a prompt-based method that converts verbalized environmental feedback into self-reflection to improve subsequent attempts[2].
6. **LLM-Majority [9]**: a simple multi-agent baseline that queries multiple LLM instances independently and uses majority voting to obtain the final prediction.
7. **LLM-Debate [12]**: a multi-agent deliberation method in which several LLM agents iteratively debate and refine their positions before reaching a final answer[3].
8. **LLM-Blender [31]**: an ensemble framework that combines and reranks candidate outputs from multiple LLMs or decoding paths to produce a stronger final response[4].
9. **DyLAN [32]**: a dynamic multi-agent system that adaptively invokes different agents and interaction patterns according to the evolving problem-solving state[5].
10. **AgentVerse [33]**: a configurable multi-agent platform that organizes role-based agents into predefined collaboration workflows for task solving[6].

11. **ADAS [13]**: an automated design framework that utilizes a search algorithm and a meta-agent to iteratively invent, program, and optimize novel agentic building blocks and workflows directly in code[7].
12. **AgentNet [34]**: a decentralized, RAG-based framework that enables LLM-based agents to specialize, evolve, and collaborate autonomously in a dynamically structured Directed Acyclic Graph[8]
13. **SELFORG [35]**: a decentralized multi-agent framework that dynamically constructs an adaptive communication graph on-the-fly by evaluating peer responses using Shapley-based contribution scores, routing information from high-contributing agents to others without relying on external judges or fixed topologies[9].
14. **AFlow [14]**: an automated framework that leverages Monte Carlo Tree Search and LLMs as optimizers to systematically explore, iteratively refine, and discover optimal code-represented agentic workflows using execution feedback[10].
15. **MaAS [3]**: an automated framework that optimizes a probabilistic agentic supernet to dynamically sample query-dependent multi-agent architectures, delivering tailored solutions and efficient resource allocation[11].

In this work, we further consider several FlowMAS variants to evaluate the contribution of different design choices. In particular, we include these variants to compare different exploration-encouraging strategies [51, 43, 42] and information decomposition mechanisms [24, 44]. These variants are listed as follows.

1. **FlowMAS w/o IOM**: a variant of FlowMAS, where the information-guided optimization mechanism (IOM) is removed.
2. **FlowMAS w/o CDM**: a variant of FlowMAS, where the curiosity-driven module (CDM) is removed.
3. **FlowMAS w/o IOM, w/o CDM**: a variant of FlowMAS, where the information-guided optimization mechanism (IOM) and the curiosity-driven module (CDM) are removed.
4. **FlowMAS w/o IOM, w/ O-Information [44]**: a variant of FlowMAS, where the information-guided optimization mechanism (IOM) is removed and replaced with an O-Information-based reward.
5. **FlowMAS w/o CDM, w/ Graph Edit Distance [42]**: a variant of FlowMAS, where the curiosity-driven module (CDM) is removed and replaced with a Graph Edit Distance-based diversity reward.
6. **FlowMAS w/o CDM, w/ Intrinsic Curiosity Module [43]**: a variant of FlowMAS, where the curiosity-driven module (CDM) is removed and replaced with an Intrinsic Curiosity Module-based exploration reward.
7. **FlowMAS w/o IOM, w/ Embedding Similarity**: a variant of FlowMAS, where the information-guided optimization mechanism (IOM) is removed and replaced with a direct embedding similarity-based reward. Specifically, given the normalized representations $\{X_k\}_{k=1}^{K}$ of the incoming messages and the target representation $Y$, the reward is defined as

$$r_{\text{emb-sim}} = \frac{1 + \operatorname{sim}\left(\frac{\sum_{k=1}^{K} X_k}{\left\|\sum_{k=1}^{K} X_k\right\|_2}, Y\right)}{2} \in [0, 1], \tag{27}$$

[1]ReAct: `https://github.com/ysymyth/ReAct`
[2]Reflexion: `https://github.com/noahshinn/reflexion`
[3]LLM-Debate: `https://github.com/ucl-dark/llm_debate`
[4]LLM-Blender: `https://github.com/yuchenlin/LLM-Blender`
[5]DyLAN: `https://github.com/SALT-NLP/DyLAN`
[6]AgentVerse: `https://github.com/openbmb/agentverse`
[7]ADAS: `https://github.com/ShengranHu/ADAS`
[8]AgentNet: `https://github.com/zoe-yyx/AgentNet`
[9]SELFORG: `https://github.com/tnurbek/selforg`
[10]AFlow: `https://github.com/FoundationAgents/AFlow`
[11]MaAS: `https://github.com/bingreeky/MaAS`

Table 8: Dataset statistics used in our experiments.

| Domain | Dataset | #Train/Val | #Test | Metric |
|---|---|---|---|---|
| Code Generation | HumanEval | 33 | 131 | pass@1 |
| | MBPP | 86 | 341 | pass@1 |
| Math Reasoning | GSM8K | 264 | 1,055 | Solve rate |
| | MATH | 119 | 486 | Solve rate |
| Reading Comprehension | HotpotQA | 200 | 800 | F1 |
| | DROP | 200 | 800 | F1 |

where $\mathrm{sim}(\cdot,\cdot)$ denotes cosine similarity. This reward directly measures the similarity between the aggregated incoming messages and the target representation.

8. **FlowMAS w/o IOM, w/ LLM-as-Judge**: a variant of FlowMAS, where the information-guided optimization mechanism (IOM) is removed and replaced with an LLM-as-Judge reward. Specifically, an LLM judge is prompted to estimate the information reward captured by IOM for each intermediate collaboration pattern.

In addition, we include the designations used for various ablation experiments.

1. **Graph Diffusion Model [20, 39] (ref. GTD [17])**: following GTD, we use a Graph Diffusion Model as the basic network backbone for multi-agent system generation, with other method-specific optimizations removed[12].
2. **VAE [18] (ref. G-Designer [11])**: following G-Designer, we use a VAE as the basic network backbone for multi-agent system generation, with other method-specific optimizations removed[13].
3. **MLP [41] (ref. MaAS [3])**: following MaAS, we use a multi-layer MLP as the basic network backbone for multi-agent system generation, with other method-specific optimizations removed[11].
4. **Transformer [40] (ref. GTD [17])**: we use a Transformer as the basic network backbone for multi-agent system generation. The supervision data are constructed following GTD[12].

# G Benchmark Detail and More Experimental

## G.1 Benchmark Statistics

Following established data partition protocols used in prior workflow-automation studies, we split each benchmark into a training/validation portion and a test portion with a 1:4 ratio. For MATH, we select 605 difficulty-level-5 problems from four representative categories: combinatorics and probability, number theory, pre-algebra, and pre-calculus. For HotpotQA and DROP, we randomly sample 1,000 examples from each benchmark following ADAS [13]. The resulting dataset statistics are summarized in Table 8.

## G.2 Supplementary Experiments

### G.2.1 Additional Results on More LLM Backbones

Detailed results of all methods on Gemma-4-26B-A4B-it and GPT-4o-mini are reported in Table 10 and Table 9, respectively.

### G.2.2 Hyperparameter Sensitivity

We further analyze the sensitivity of the reward coefficients $\alpha$ and $\beta$ in Eq. 9, which control the contributions of CDM and IOM, respectively. The experiments are conducted on HumanEval with Llama-3.1-8B-Instruct. When varying $\alpha$, we fix $\beta = 0.8$; when varying $\beta$, we fix $\alpha = 0.2$.

[12]GTD: `https://github.com/ericjiang18/diffusion_agent`
[13]G-Designer: `https://github.com/yanweiyue/GDesigner`

Table 9: Detailed results on GPT-4o-mini. The best result in each column is boldfaced, and the second-best is underlined. Each experiment is run five times to record the mean, standard deviation.

| Method | HumanEval | GSM8K | MATH | MBPP | HotpotQA | DROP | Avg. |
|---|---|---|---|---|---|---|---|
| IO | 86.3±0.6 | 86.3±0.5 | 45.0±0.7 | 71.2±0.4 | 66.7±0.6 | 73.2±0.9 | 71.45 |
| CoT [27] | 86.6±0.7 | 86.4±0.5 | 45.8±0.6 | 72.6±0.6 | 66.4±0.9 | 77.0±0.8 | 72.47 |
| CoT-SC [28] | 87.6±0.0 | 86.5±0.3 | 46.5±0.0 | 74.2±0.5 | 67.4±0.6 | 77.5±0.7 | 73.28 |
| ReAct [29] | 87.0±0.8 | 86.5±0.7 | 43.3±0.9 | 77.7±0.6 | 71.0±0.8 | 73.8±0.9 | 73.22 |
| Reflexion [30] | 88.7±0.7 | 87.5±0.4 | 45.9±0.9 | 76.8±0.6 | 66.2±0.7 | 77.1±0.8 | 73.70 |
| LLM-Majority [9] | 87.9±0.3 | 86.5±0.6 | 46.8±0.0 | 74.4±0.0 | 68.3±0.6 | 74.6±0.5 | 73.08 |
| LLM-Debate [12] | 87.9±0.6 | 88.2±0.9 | 47.8±0.8 | 72.1±0.5 | 65.5±0.9 | 71.9±0.6 | 72.23 |
| LLM-Blender [31] | 87.9±0.8 | 87.5±0.9 | 46.2±0.8 | 75.6±0.6 | 68.1±0.9 | 73.9±0.7 | 73.20 |
| DyLAN [32] | 89.6±0.7 | 88.9±0.9 | 47.1±1.2 | 78.1±0.7 | 70.4±0.9 | 76.2±0.8 | 75.05 |
| AgentVerse [33] | 88.3±0.9 | 89.4±1.2 | 46.3±0.9 | 78.0±0.7 | 68.2±0.6 | 75.4±0.8 | 74.27 |
| ADAS [13] | 83.4±1.2 | 84.9±0.9 | 42.7±1.2 | 67.3±0.8 | 63.9±0.9 | 75.5±1.2 | 69.62 |
| AgentNet [34] | 86.8±0.9 | 91.8±0.0 | 47.5±0.1 | 71.1±0.2 | 64.5±0.2 | 80.8±0.0 | 73.74 |
| SELFORG [35] | 86.0±2.0 | 92.7±0.1 | 50.7±0.7 | 71.7±0.3 | 68.9±0.0 | 84.3±0.1 | 75.71 |
| AFlow [14] | 89.6±0.7 | 90.5±0.7 | 50.5±0.9 | 81.0±0.6 | 72.6±0.9 | 79.7±0.5 | 77.32 |
| MaAS [3] | 92.2±0.5 | 91.7±0.6 | 51.1±0.8 | 81.4±0.5 | 77.3±0.7 | 83.1±0.6 | 79.47 |
| FlowMAS | **94.4**±0.8 | **93.1**±0.4 | **52.6**±0.7 | **83.4**±0.5 | **80.0**±0.4 | **84.9**±0.2 | **81.40** |

Table 10: Detailed results on Gemma-4-26B-A4B-it. The best result in each column is boldfaced, and the second-best is underlined. Each experiment is run five times to record the mean, standard deviation.

| Method | HumanEval | GSM8K | MATH | MBPP | HotpotQA | DROP | Avg. |
|---|---|---|---|---|---|---|---|
| IO | 90.8±0.8 | 90.5±0.7 | 61.3±0.9 | 73.3±0.6 | 74.9±0.8 | 83.9±1.0 | 79.12 |
| CoT [27] | 91.6±0.9 | 91.6±0.6 | 61.7±1.0 | 76.7±0.7 | 79.3±1.1 | 87.4±0.9 | 81.38 |
| CoT-SC [28] | 87.6±0.0 | 91.2±0.5 | 61.7±0.3 | 78.0±0.4 | 78.3±0.6 | 87.8±0.7 | 80.77 |
| ReAct [29] | 90.0±1.0 | 90.3±0.9 | 60.0±1.1 | 77.4±0.8 | 78.6±1.0 | 88.2±0.9 | 80.75 |
| Reflexion [30] | 91.8±0.9 | 92.1±0.6 | 62.2±1.2 | 76.2±0.7 | 73.5±0.8 | 85.1±1.0 | 80.15 |
| LLM-Majority [9] | 91.7±0.8 | 91.2±0.4 | 61.3±0.4 | 78.5±0.0 | 79.3±0.7 | 88.2±0.6 | 81.70 |
| LLM-Debate [12] | 92.3±0.8 | 89.3±0.9 | 56.9±1.0 | 75.3±0.6 | 76.6±0.9 | 84.3±0.8 | 79.12 |
| LLM-Blender [31] | 89.9±0.9 | 90.1±1.1 | 61.4±1.0 | 77.1±0.7 | 77.7±1.0 | 87.6±0.8 | 80.63 |
| DyLAN [32] | 92.3±0.8 | 91.8±1.0 | 62.7±1.1 | 79.5±0.8 | 70.3±1.0 | 88.2±0.9 | 80.80 |
| AgentVerse [33] | 89.8±1.0 | 92.6±1.2 | 62.0±1.1 | 79.2±0.9 | 78.8±0.8 | 86.6±0.9 | 81.50 |
| ADAS [13] | 88.5±1.2 | 90.0±1.0 | 59.2±1.2 | 71.2±0.8 | 76.7±1.1 | 86.5±1.2 | 78.68 |
| AgentNet [34] | 90.9±1.2 | 91.9±1.0 | 62.7±0.3 | 81.6±0.3 | 79.5±0.4 | 88.8±1.0 | 82.57 |
| SELFORG [35] | 91.3±0.3 | 92.1±0.3 | 63.1±0.4 | 77.8±0.4 | 79.5±0.5 | 90.4±0.5 | 82.37 |
| AFlow [14] | 92.6±0.8 | 93.4±0.9 | 65.0±1.0 | 83.5±0.7 | 82.7±1.0 | 89.9±1.1 | 84.52 |
| MaAS [3] | 92.4±0.6 | 95.1±0.7 | 65.1±1.0 | 83.1±0.6 | 83.8±0.8 | 90.7±0.7 | 85.03 |
| FlowMAS | **94.8**±0.8 | **95.6**±0.8 | **66.7**±0.9 | **87.1**±0.6 | **85.3**±0.8 | **92.0**±0.7 | **86.92** |

As shown in Table 11, FlowMAS remains relatively stable across the evaluated hyperparameter ranges. We therefore use $\alpha = 0.2$ and $\beta = 0.8$ as the default configuration in our experiments.

### G.2.3 Training and Inference Wall-clock Time

In addition to token consumption, we report the end-to-end wall-clock time of representative automated workflow optimization methods on HumanEval. All experiments are measured on a single NVIDIA GeForce RTX 5090 with Llama-3.1-8B-Instruct. The reported time includes the complete optimization or inference procedure of each method.

Table 11: Hyperparameter sensitivity analysis on HumanEval with Llama-3.1-8B-Instruct.

| $\alpha$ | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
|---|---|---|---|---|---|
| Accuracy | 67.9 | **69.6** | **69.6** | 68.7 | 67.9 |
| $\beta$ | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
| Accuracy | 66.4 | 68.7 | **69.6** | 67.9 | 67.9 |

Table 12: End-to-end training and inference wall-clock time on HumanEval with Llama-3.1-8B-Instruct. All experiments are measured on a single NVIDIA GeForce RTX 5090.

| Method | Training Time (min) | Inference Time (min) | Accuracy |
|---|---|---|---|
| ADAS | 200 | 91 | 59.5 |
| AFlow | 97 | 66 | 65.1 |
| MaAS | **31** | 22 | 62.7 |
| FlowMAS | 39 | **13** | 69.6 |

Table 13: Detailed training-time overhead of FlowMAS on HumanEval. IOM Time includes the embedding computation required for the IOM reward, while Embedding Time denotes the total embedding-model computation throughout training.

| Method | Total Training Time (min) | IOM Time (min) | CDM Time (min) | Embedding Time (min) |
|---|---|---|---|---|
| FlowMAS | 39 | 1.81 | 0.74 | 2.89 |

As shown in Table 12, FlowMAS requires substantially less training time than AFlow and ADAS while achieving the shortest inference time among the compared methods. Although MaAS requires slightly less training time, FlowMAS provides a better trade-off between optimization cost and task performance.

### G.2.4 Detailed Computational Overhead

We further profile the additional computation introduced by the IOM and CDM modules during FlowMAS training. Table 13 reports the time spent on IOM, CDM, and embedding computation on HumanEval. The IOM time includes both encoding the operator outputs and computing the information-guided reward, while the embedding time denotes the total time spent invoking the embedding model throughout training.

The IOM and CDM computations require only 1.81 and 0.74 minutes, respectively, compared with 39 minutes of total training time. Together, these two modules account for approximately 6.5% of the overall training wall-clock time, indicating that the intermediate reward computation introduces limited additional overhead.

## H OPERATORS

### H.1 OPERATORS Register

**OPERATOR REGISTRY**

```
{
  "Generate": {
    "description": "Generates anything based on customized input and instruction.",
    "interface": "generate(input: str, instruction: str) -> dict with key 'response' of type
      str"
  },
  "GenerateCoT": {
```

```
    "description": "Generates anything using a chain-of-thought approach, providing step-by-
     step reasoning before producing the final solution. If you need to solve mathematical or
      difficult reasoning problems, you can use this operator.",
    "interface": "generate_cot(input: str, instruction: str) -> dict with key 'response' of
     type str"
  },
  "MultiGenerateCoT": {
    "description": "Generates multiple solutions using diverse chain-of-thought reasoning
     processes to increase solution variety and robustness.",
    "interface": "multi_generate_cot(input: str, instruction: str) -> dict with key 'response
     ' of type List[str]"
  },
  "VoteEnsemble": {
    "description": "Selects the best solution from multiple candidates using voting, semantic
      equivalence, and reasoning-based judgment to improve robustness and quality.",
    "interface": "vote_ensemble(solutions: List[str], problem: str) -> dict with key '
     response' of type str"
  },
  "Decomposer": {
    "description": "Decomposes a complex problem into structured subtasks with potential
     dependencies and execution strategies.",
    "interface": "decompose(problem: str) -> dict with key 'subtasks' of type List[dict]"
  },
  "Aggregator": {
    "description": "Aggregates and fuses multiple inputs, intermediate results, or task
     outputs into a unified and coherent representation.",
    "interface": "aggregate(inputs: List[str], instruction: str) -> dict with key 'response'
     of type str"
  },
  "Programmer": {
    "description": "Automatically writes, executes Python code, and returns the solution
     based on the provided problem description and analysis. The `output` only contains the
     final answer. If you want to see the detailed solution process, it's recommended to
     retrieve the `code`.",
    "interface": "programmer(problem: str, analysis: str = 'None') -> dict with keys 'code'
     and 'output' of type str"
  },
  "Test": {
    "description": "Tests the solution using public test cases. If the solution fails, it
     reflects on the errors and attempts to modify the solution. Returns True and the
     solution if all tests pass after modifications. Returns False and the current solution
     if it still fails after modifications.",
    "interface": "test(problem: str, solution: str, entry_point: str) -> dict with key '
     result' of type bool and key 'solution' of type str"
  },
  "SelfRefine": {
    "description": "Refines the generated solution by analyzing errors or suboptimal aspects
     and making iterative improvements to enhance correctness and efficiency.",
    "interface": "self_refine(problem: str, solution: str) -> dict with key 'response' of
     type str"
  },
  "EarlyStop": {
    "description": "This issue has been correctly and effectively resolved. In order to
     prevent further activation of operators from wasting resources, we will immediately stop
      answering this question. Immediately terminates the workflow when selected, preventing
     any further operators from being invoked.",
    "interface": "early_stop() -> dict with key 'terminate' of type bool"
  }
}
```

## H.2 OPERATORS Implementation

**OPERATORS**

```python
class Operator:
    def __init__(self, llm: LLM, name: str):
        self.name = name
        self.llm = llm
    def __call__(self, *args, **kwargs):
        raise NotImplementedError
    async def _fill_node(self, op_class, prompt, mode=None, **extra_kwargs):
        fill_kwargs = {"context": prompt, "llm": self.llm}
        if mode:
```

```
            fill_kwargs["mode"] = mode
        fill_kwargs.update(extra_kwargs)
        node = await ActionNode.from_pydantic(op_class).fill(**fill_kwargs)
        result = node.instruct_content.model_dump()
        if any(str(v or "").strip() for v in result.values()):
            return result
        raw_text = await self.llm.aask(prompt)
        raw_text = str(raw_text or "").strip()
        if not raw_text:
            return result
        if "reflection_and_solution" in result:
            result["reflection_and_solution"] = raw_text
        elif "response" in result:
            result["response"] = raw_text
        return result
    async def _ask_text(self, prompt: str) -> str:
        response = await self.llm.aask(prompt)
        return response

class Generate(Operator):
    def __init__(self, llm: LLM, name: str = "Generate"):
        super().__init__(llm, name)
    async def __call__(self, problem, entry_point, instruction):
        prompt = instruction + problem
        return await self._fill_node(GenerateOp, prompt, mode="code_fill", function_name=
     entry_point)

class GenerateCoT(Operator):
    def __init__(self, llm: LLM, name: str = "GenerateCoT"):
        super().__init__(llm, name)
    async def __call__(self, problem, entry_point, subtask="", current_solution=""):
        prompt = GENERATE_COT_PROMPT.format(
            problem=problem,
            entry_point=entry_point,
            subtask_block=f"Subtask focus: {subtask}\n" if subtask else "",
            current_solution_block=(
                f"Current draft solution:\n{current_solution}\n\nRevise or improve it.\n"
                if current_solution else ""
            ),
        )
        return await self._fill_node(GenerateOp, prompt, mode="code_fill", function_name=
     entry_point)

class MultiGenerateCoT(Operator):
    def __init__(self, llm: LLM, name: str = "MultiGenerateCoT"):
        super().__init__(llm, name)
    async def __call__(self, problem, entry_point, current_solution=""):
        candidates = []
        seen = set()
        for variant in MULTI_GENERATE_COT_VARIANTS:
            prompt = GENERATE_COT_PROMPT.format(
                problem=problem,
                entry_point=entry_point,
                subtask_block=f"Additional guidance: {variant}\n",
                current_solution_block=(
                    f"Current draft solution:\n{current_solution}\n\nProduce an alternative
     improved implementation.\n"
                    if current_solution else ""
                ),
            )
            response = await self._fill_node(GenerateOp, prompt, mode="code_fill",
     function_name=entry_point)
            candidate = str(response.get("response", "")).strip()
            if candidate and candidate not in seen:
                seen.add(candidate)
                candidates.append(candidate)
        if not candidates:
            fallback = await GenerateCoT(self.llm).__call__(
                problem=problem,
                entry_point=entry_point,
                current_solution=current_solution,
            )
            if str(fallback.get("response", "")).strip():
                candidates.append(fallback["response"])
        return {"response": candidates}

class VoteEnsemble(Operator):
    def __init__(self, llm: LLM, name: str = "VoteEnsemble"):
```

```
        super().__init__(llm, name)
    async def __call__(self, problem, entry_point, parent_solutions: List[str]):
        formatted_solutions = []
        for index, solution in enumerate(parent_solutions, start=1):
            formatted_solutions.append(f"Candidate {index}:\n{solution}")
        prompt = VOTE_ENSEMBLE_PROMPT.format(
            problem=problem,
            entry_point=entry_point,
            parent_solutions="\n\n".join(formatted_solutions) if formatted_solutions else "<
     empty>",
        )
        return await self._fill_node(GenerateOp, prompt, mode="code_fill", function_name=
     entry_point)

class Decomposer(Operator):
    def __init__(self, llm: LLM, name: str = "Decomposer"):
        super().__init__(llm, name)
    @staticmethod
    def _parse_subtasks(raw_text: str) -> List[str]:
        lines = [line.strip() for line in raw_text.splitlines() if line.strip()]
        subtasks = []
        for line in lines:
            cleaned = re.sub(r"^(?:[-*]|\d+[.)])\\s*", "", line).strip()
            if cleaned:
                subtasks.append(cleaned)
        deduped = []
        seen = set()
        for task in subtasks:
            key = task.lower()
            if key not in seen:
                seen.add(key)
                deduped.append(task)
        return deduped[:4]
    async def __call__(self, problem, entry_point, current_solution=""):
        prompt = DECOMPOSER_PROMPT.format(
            problem=problem,
            entry_point=entry_point,
            current_solution=current_solution or "<empty>",
        )
        raw_response = await self._ask_text(prompt)
        subtasks = self._parse_subtasks(raw_response)
        if not subtasks:
            subtasks = [
                "Derive a correct algorithm and invariants for the target function.",
                "Stress test edge cases and boundary conditions.",
            ]
        return {"response": raw_response, "subtasks": subtasks}

class Aggregator(Operator):
    def __init__(self, llm: LLM, name: str = "Aggregator"):
        super().__init__(llm, name)
    async def __call__(self, problem, entry_point, parent_solutions: List[str]):
        formatted_solutions = []
        for index, solution in enumerate(parent_solutions, start=1):
            formatted_solutions.append(f"Candidate {index}:\n{solution}")
        prompt = AGGREGATOR_PROMPT.format(
            problem=problem,
            entry_point=entry_point,
            parent_solutions="\n\n".join(formatted_solutions) if formatted_solutions else "<
     empty>",
        )
        return await self._fill_node(GenerateOp, prompt, mode="code_fill", function_name=
     entry_point)

class Programmer(Operator):
    def __init__(self, llm: LLM, name: str = "Programmer"):
        super().__init__(llm, name)
    async def __call__(self, problem, entry_point):
        prompt = PROGRAMMER_PROMPT.format(problem=problem, entry_point=entry_point)
        return await self._fill_node(GenerateOp, prompt, mode="code_fill", function_name=
     entry_point)

class Test(Operator):
    INTERNAL_TEST_TIMEOUT_SECONDS = int(os.getenv("
     MAAS_HUMANEVAL_INTERNAL_TEST_TIMEOUT_SECONDS", "10"))
    def __init__(self, llm: LLM, name: str = "Test"):
        super().__init__(llm, name)
    @staticmethod
```

```
    def _exec_code_impl(solution, entry_point):
        test_cases = extract_test_cases_from_jsonl(entry_point, dataset="HumanEval")
        fail_cases = []
        for test_case in test_cases:
            test_code = test_case_2_test_function(solution, test_case, entry_point)
            try:
                exec(test_code, globals())
            except AssertionError as e:
                exc_type, exc_value, exc_traceback = sys.exc_info()
                tb_str = traceback.format_exception(exc_type, exc_value, exc_traceback)
                error_information = {
                    "test_fail_case": {
                        "test_case": test_case,
                        "error_type": "AssertionError",
                        "error_message": str(e),
                        "traceback": tb_str,
                    }
                }
                fail_cases.append(error_information)
            except Exception as e:
                return {"exec_fail_case": str(e)}
        return fail_cases if fail_cases else "no error"
    def exec_code(self, solution, entry_point):
        result_queue = mp.Queue()
        def target(queue, code_solution, code_entry_point):
            try:
                queue.put(("result", self._exec_code_impl(code_solution, code_entry_point)))
            except Exception as e:
                queue.put(("error", str(e)))
        process = mp.Process(target=target, args=(result_queue, solution, entry_point),
     daemon=True)
        process.start()
        process.join(self.INTERNAL_TEST_TIMEOUT_SECONDS)
        try:
            if process.is_alive():
                process.terminate()
                process.join(timeout=1)
                return {
                    "exec_fail_case": (
                        f"Internal HumanEval public-test execution timed out after "
                        f"{self.INTERNAL_TEST_TIMEOUT_SECONDS} seconds"
                    )
                }
            if result_queue.empty():
                return {"exec_fail_case": "Internal HumanEval public-test execution returned
     no result"}
            status, payload = result_queue.get_nowait()
            if status == "error":
                return {"exec_fail_case": payload}
            return payload
        finally:
            result_queue.close()
            result_queue.join_thread()
    async def __call__(self, problem, solution, entry_point, test_loop: int = 1):
        if not str(solution or "").strip():
            return {"result": False, "solution": ""}
        for _ in range(test_loop):
            result = self.exec_code(solution, entry_point)
            if result == "no error":
                return {"result": True, "solution": solution}
            if "exec_fail_case" in result:
                exec_result = result["exec_fail_case"]
                prompt = REFLECTION_ON_PUBLIC_TEST_PROMPT.format(
                    problem=problem,
                    solution=solution,
                    exec_pass=f"executed unsuccessfully, error:\n{exec_result}",
                    test_fail="executed unsuccessfully",
                )
            else:
                prompt = REFLECTION_ON_PUBLIC_TEST_PROMPT.format(
                    problem=problem,
                    solution=solution,
                    exec_pass="executed successfully",
                    test_fail=result,
                )
            response = await self._fill_node(ReflectionTestOp, prompt, mode="code_fill")
            solution = response["reflection_and_solution"]
        result = self.exec_code(solution, entry_point)
```

```
        return {"result": result == "no error", "solution": solution}

class SelfRefine(Operator):
    def __init__(self, llm: LLM, name: str = "SelfRefine"):
        super().__init__(llm, name)
    async def __call__(self, problem, solution):
        prompt = SELFREFINE_PROMPT.format(problem=problem, solution=solution)
        return await self._fill_node(SelfRefineOp, prompt, mode="code_fill")
```

## NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

**The checklist answers are an integral part of your paper submission.** They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have precisely described them in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the main limitations and computational efficiency in the experimental section and supplementary appendix.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: This is not a theory paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The source code are provided in the appendix. The experimental settings are described in the experimental section, and the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The source code are included in the supplementary materials. All the data and the source code will be made open-access upon publication.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The details are provided in the experimental section and the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The variance are reported along with the mean value.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experiments are conducted in a computer clusters with multiple Nvidia H100 (80G) GPUs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We follow the NeurIPS Code of Ethics. No human subject, and private datasets are involed. We use public available datasets that are shared to the research community.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Our work is foundational research and not tied to particular applications.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Such paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have cited the original paper that produced the code and the dataset.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We will release the code of this work.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: No human subjects were involved.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: Not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: We use an LLM to polish the paper. In this work, we study LLM-based agentic workflows. LLMs are used as the base models invoked by our proposed workflow method. An LLM is a necessary component of our work.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.