FROM STATIC TO DYNAMIC: TOWARDS RUNTIME MULTI-AGENT TOPOLOGY PLANNING VIA PROGRESSIVE NEXT LAYER GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM)-driven agents are designed to handle a wide range of tasks autonomously. As tasks become increasingly complex, the multi-agent system (MAS) becomes a promising solution, where the underlying communications are typically represented as a directed acyclic graph. To autonomously construct a desired MAS topology, various design strategies are proposed based on graph-informed techniques. However, these frameworks primarily follow the "design-then-deploy" paradigm. As a result, the generated graph-structure MAS cannot satisfy runtime requirements and adapt to the dynamic environment. To break the constraints of static designs, our key insight is that the dynamic planning of interaction topology for MAS is inherently an autoregressive next layer generation problem, where nodes are agents and connections are information dependencies. Building upon this motivation, we propose a Runtime Multi-Agent topology Planning scheme named RMAP. Specifically, to capture the evolving semantics of agents during runtime topology construction, RMAP progressively integrates three agentic semantics, including role descriptions, topological context, and execution outputs, into unified task-oriented representations. To enable layer-wise execution, RMAP decomposes topology prediction into two stages: first generating the leaf agents of the current layer, followed by supplementing optional connections between these leaf agents and their ancestor agents. Experimental results show that our scheme improves task performance in 92.8% of scenarios, and can be easily implemented on personal computers with less than 1 second of additional latency. The code is available at https://anonymous.4open.science/r/RMAP-1884.

1 INTRODUCTION

Numerous large language model (LLM)-driven agents have been designed to handle various real-world tasks, such as code writing Chun et al. (2026); Zhang et al. (2026b), domain question answering Jiang et al. (2025); Wu et al. (2025), and fuzzy testing Liu et al. (2025); Yu et al. (2025). However, in complex tasks such as autonomous driving Gui et al. (2026) or smart city management Xu et al. (2025), a single agent will struggle to analyze the problem from multiple perspectives and adapt to the dynamic environment. Therefore, to satisfy composite requirements in real-world applications, the graph-structured multi-agent system (MAS) has emerged as a crucial solution, in which diverse skills could be integrated via mutual communications. Moreover, recent advanced practices Zhuge et al. (2024); Zhang et al. (2025d); Han et al. (2025) have demonstrated that, in most cases, a plurality of cross-domain knowledge outperforms individual intelligence.

The underlying topology of a MAS, i.e., how communication flows from one agent to another, is typically a directed acyclic graph (DAG) for ordered execution. It is closely related to downstream performance, task-solving efficiency, and robustness Cemri et al. (2026); Zhang et al. (2025b); Wang et al. (2025a); Li et al. (2026a). Early studies Hong et al. (2024); Chen et al. (2024) intuitively construct system topologies by adopting some well-known structures (e.g., chain, star, tree) based on prior knowledge of task requirements and role definitions. However, as the scale of the system increases and the task environment varies, such handcrafted designs require complicated testing and tuning. To architect an ideal communication topology automatically, graph-based optimiza-

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

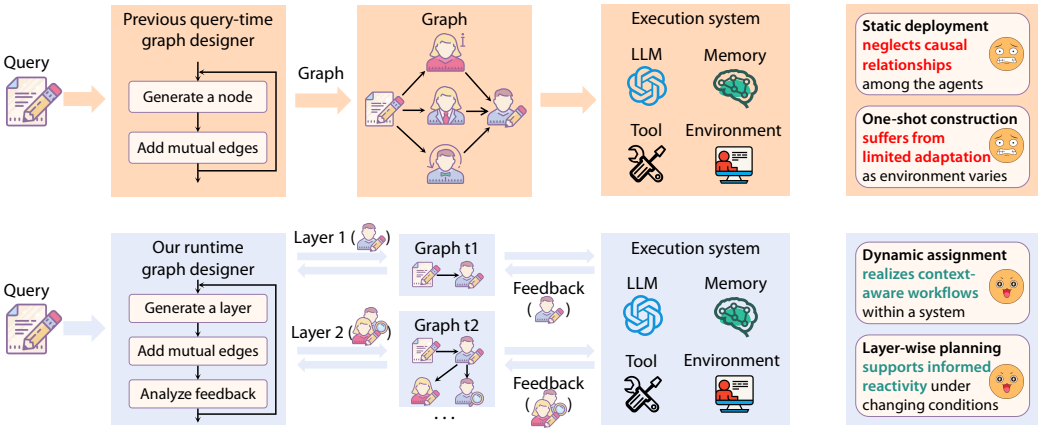


Figure 1: The comparison between previous static graph designers and our runtime graph designer (i.e., RMAP) for MAS topology construction. Unlike one-shot construction, our scheme further supports feedback-driven layer-wise generation in an asynchronous environment.

tion strategies Zhuge et al. (2024); Zhang et al. (2025d) are introduced to the agentic domain with resource savings, performance improvement, and robustness enhancement.

However, prevailing autonomous MAS topology designers Yue et al. (2025); Zhang et al. (2025a); Yu et al. (2026); Li et al. (2026b) mostly adhere to the "design-then-deploy" paradigm. Concretely, during training, they capture a query-to-graph mapping, and during inference, they directly output a graph without systematic interactions, as shown in Figure 1. In this way, such static modeling leads to two crucial shortcomings: (1) It cannot orchestrate workflows based on real-time feedback. For instance, in code generation tasks, they cannot determine whether to continue or terminate the iterative interaction between "code writer" and "code tester" based on the results of test cases; (2) Their static mapping fails in a self-evolved agentic environment. For instance, due to the application of memory Salama et al. (2025); Zhang et al. (2026a), a graph-structured MAS will produce different workflows even for the same query. Therefore, a key research question arises:

How can we dynamically construct agent dependencies to reflect evolving execution states and semantic interactions throughout runtime execution?

To address this problem, our key insight is that **the runtime planning of a graph-structured MAS can be modeled as an autoregressive next layer generation problem**, where agents are nodes and mutual relationships are connections. Building upon this motivation, we propose a **R**untime **M**ulti-**A**gent topology **P**lanning scheme named RMAP, where desired communications can be planned via a few lightweight deep modules. Specifically, to capture the evolving semantics of agents under runtime dynamics, a recurrent encoder progressively integrates three information channels, including role descriptions, topological context, and execution outputs, into unified agent representations. To preserve the topological order of the DAG, topology generation is decomposed into two decoding stages. Firstly, a decoder generates leaf agents for layer t based on the execution states of agents in layer $t-1$. Secondly, another decoder augments the generated topology by adding optional cross-layer edges between newly generated agents and their ancestor agents from preceding layers. In summary, our contributions are as follows:

- **Shortcoming Identification.** We reveal that previous MAS topology designers mostly generate a pre-defined communication template before execution. As a result, the static workflow cannot satisfy the causal interaction requirements and adapt to the evolved agentic environment.
- **Solution Proposal.** We introduce our RMAP, a dynamic MAS topology designer to support real-time layer-wise planning and asynchronous execution. Concretely, we reformulate the agentic workflow construction task as a hierarchical DAG generation process, where agents and mutual connections are progressively captured based on systematic execution states.
- **Comprehensive Evaluation.** Extensive experiments are conducted across various LLMs and tasks. Results show that our scheme improves the performance from $0.4\% \sim 9.9\%$ compared with state-of-the-art (SOTA) baselines, with only several hundred MB of GPU memory required and $1.6\% \sim 7.9\%$ additional time costs for asynchronous multiple task execution on MBPP.

2 Related Work

LLM-Driven Multi-Agent Systems. The underlying mechanism of the MAS lies in the decomposition of a compound task into multiple sub-algorithms, which are executed by corresponding domain-specific LLM-driven agents Cemri et al. (2026); Tran et al. (2025). For example, in a bug-fixing agent system Yu et al. (2025), the retriever first identifies potential bugs, followed by the verifier who validates candidate patches, and ultimately, the bug fixer refines the patch until the issue is resolved or the assigned budget is exhausted.

Autonomous MAS Topology Designers. Generally, the existing autonomous MAS topology designers can be divided into the task-specific designer Yu et al. (2026); Zhang et al. (2025d;c;a) and the query-specific designer Li et al. (2026b); Wang et al. (2025b). For task-specific designers, they add communication edges among agents via reinforcement learning (RL) Zhuge et al. (2024) or LLM-driven Monte Carlo search Zhang et al. (2025d). The learned graphs are fixed across various queries, thereby reducing flexibility. In contrast, ARG-Designer Li et al. (2026b) re-frames MAS design as an autoregressive node generation task conditioned by queries. Moreover, Anymac Wang et al. (2025b) considers sequential runtime planning for queries, which is the most closely related to our work. However, its agent-by-agent generation hinders the DAG-informed asynchronism. In contrast, our layer-wise generation fully considers the runtime requirements of agentic systems.

Parameterized DAG Generation Strategies. Due to the topological order of the DAG, its generation can be modeled as a deep autoregressive prediction. GraphRNN You et al. (2018) and D-VAE Zhang et al. (2019) treat either a single node or a set of nodes as the token. In contrast, recent advances, such as LayerDAG Li et al. (2025), investigate layer-wise generation. Our scheme follows the layer-wise paradigm but further considers context dependencies among agents in a MAS.

3 Preliminaries

Multi-Agent Systems as DAGs. A MAS can be represented as a graph $G = (V, E, F, P)$, where V is the set of nodes (agents) with corresponding descriptions in $d_v \in D$, $E \subseteq \mathcal{E} \triangleq V \times V$ is the topology formally represented as a set of edges. $F = \{f_i | i \in V\}$ is a set of agent computational routines controlled by the prompt set $P = \{p_i | i \in V\}$, and $o \in V$ is the node (agent) for the final decision. Furthermore, all nodes are instantiated from a candidate set $\hat{V}$. The execution follows a layer-wise trajectory with T steps (layers):

$$G = \left\{G^0, G^1, ..., G^T\right\} = \left\{(V^0), (V^1, E^{<1,1}, F^1, P^1), ..., (V^T, E^{<T,T}, F^T, P^T)\right\}, \tag{1}$$

where V^t denotes the node set at the t-th step, V^0 only contains the query q and V^T only contains the final decision. $V^{<t} \triangleq \bigcup_{i=0}^{t-1} V^i$ denotes nodes in previous layers. $E^{<t,t} \subseteq \mathcal{E}^{<t,t} \triangleq V^{<t} \times V^t$. Within V^t, each node v receives messages from predecessors $\{z_u | u \in pre(v)\}$, the prompt p_v, and outputs z_v asynchronously:

$$z_v = f_v(\{z_u \mid u \in pre(v)\}, p_v). \tag{2}$$

Recurrent Agentic Layer Generation. In traditional layer-wise generation strategies, a probability distribution $P(G)$ of DAGs Zhang et al. (2019); Li et al. (2025) follows:

$$P(G) = \prod_{t=1}^{T} P(G^t \mid G^{<t}) = \prod_{t=1}^{T} \underbrace{P(V^t \mid G^{<t})}_{\text{node construction}} \underbrace{P(E^{<t,t} \mid G^{<t}, V^t)}_{\text{link prediction}}. \tag{3}$$

Intuitively, Eq.3 generates nodes first, then predicts connections without any constraints. But for a MAS topology designer, (1) Agents are pre-defined in $\hat{V}$ rather than generated; (2) An agent at layer t must have a predecessor at layer $t-1$, while connections from earlier ancestors are optional. Therefore, to satisfy the above two requirements, we reformulate the conditional probability as:

$$P(G) = \prod_{t=1}^{T} P(G^t \mid G^{<t}, \hat{V}) = \prod_{t=1}^{T} \underbrace{P(\hat{E}^t \mid G^{<t}, \hat{V})}_{\text{next-layer agent instantiation}} \underbrace{P(\tilde{E}^t \mid G^{<t}, V^t, \hat{E}^t)}_{\text{cross-layer link prediction}}, \tag{4}$$

where $\hat{E}^t \subseteq \hat{\mathcal{E}}^t \triangleq V^{t-1} \times \hat{V}$ and $\tilde{E}^t \subseteq \tilde{\mathcal{E}}^t \triangleq V^{<t-1} \times V^t$ for simplicity. Concretely, in the next-layer agent instantiation, leaf agents are induced from $\hat{E}^t$, i.e., $V^t = \{v \in \hat{V} \mid \exists u \in V^{t-1}, (u, v) \in \hat{E}^t\}$,

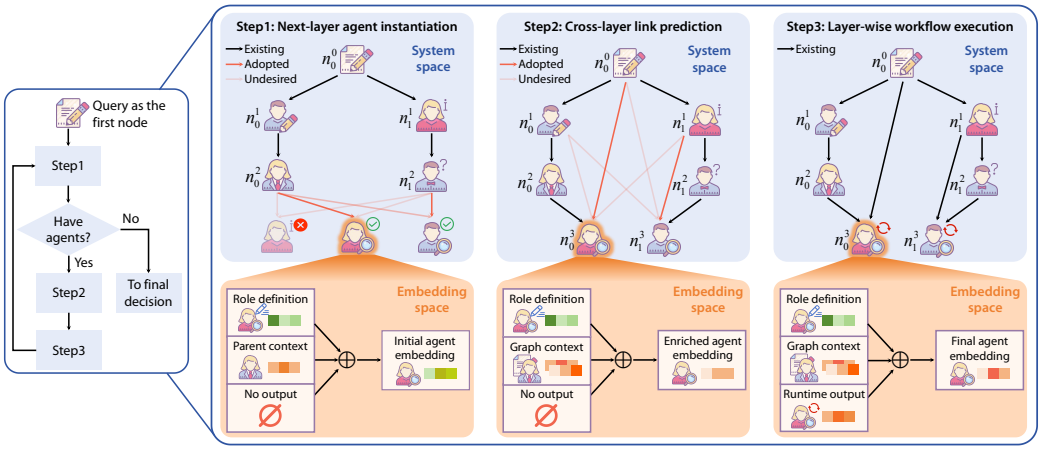


Figure 2: Our scheme has three steps: next-layer agent instantiation, cross-history link prediction and layer-wise workflow execution, during which agent semantics are progressively fused.

which ensures the topological order. Then, in the cross-layer link generation, the connections from previous layers to leaves are supplemented. When there is no connection to the candidate set (i.e., $|V^t| = 0$), the system predicts the edge to the final decision, and the generation process ends.

Multi-Channel Representations of Agents. For the representation of agent v in step t, the current input is composed of two channels, including textual role definition d_v and output z_v. The context of topological relationships, the third channel, is the set of all in-neighbors' hidden states. In this way, the embedding of an agent is:

$$h_v = f_\theta^{enc}\big((d_v, z_v), \{h_u | u \in pre(v)\}\big), \tag{5}$$

where f_θ^{enc} is an encoder. During layer-to-layer generation, Eq.5 could be viewed as a variant of recurrent modeling Yu et al. (2019). In the next section, we will introduce how to progressively update agent semantics and represent workflow relationships between agents based on task steps.

4 Our Proposed RMAP

Overview. In section 4.1, we introduce how to parameterize the MAS topology generation and progressively represent agent semantics according to the running stages. In section 4.2, we discuss how to conduct supervised pre-training for utility and then employ RL-based post-training for efficiency.

4.1 Parameterization of Layer-Wise Topology Generation.

As shown in Figure 2, the generation is a loop with the following three steps to realize the layer-wise agent generation and execution progressively.

Step 1: Next-Layer Agent Instantiation. To approximate $P(\hat{E}^t \mid G^{<t}, \hat{V})$, there are two crucial inductive biases. Firstly, the recurrent agent encoder captures systemic context until step $t-1$. In this way, the embeddings of V^{t-1} represent previous graph information, i.e., $P(\hat{E}^t \mid G^{<t}, \hat{V}) \approx P_\theta(\hat{E}^t \mid V^{t-1}, \hat{V})$. Secondly, the probability of each potential link is modeled independently because each largely represents a distinct dependency between two agents. Therefore, the distribution is further approximated as the joint probability of individual links. Moreover, the embeddings of V^{t-1} are derived from the previous execution, whereas those of $\hat{V}$ depend only on descriptions:

$$\forall u \in V^{t-1}, h_u = f_\theta^{enc}\big((d_u, z_u), \{h_w | w \in pre(u)\}\big), \forall v \in \hat{V}, h_v = f_\theta^{enc}((d_v, \emptyset), \emptyset). \tag{6}$$

Following the above analysis, the parameterization is denoted as:

$$\begin{aligned} P(\hat{E}^t \mid G^{<t}, \hat{V}) \approx P_\theta(\hat{E}^t \mid V^{t-1}, \hat{V}) &\approx \prod_{(u,v)\in\hat{\mathcal{E}}^t} \big[p_\theta(e|u,v)\big]^{y_e} \big[1 - p_\theta(e|u,v)\big]^{1-y_e} \\ &\approx \prod_{(u,v)\in\hat{\mathcal{E}}^t} \big[p_\theta^{agt}(h_u, h_v)\big]^{y_e} \big[1 - p_\theta^{agt}(h_u, h_v)\big]^{1-y_e}, \end{aligned} \tag{7}$$

where p_θ^{agt} is a deep link predictor (decoder) and $y_e \in \{0, 1\}$ denotes whether e exists in $\hat{E}^t$. After prediction, edges with predicted probabilities $p_\theta^{agt} > \delta$ are selected, where δ defaults to 0.5. The embeddings of leaves are updated with structural information originating from their parents:

$$h_v \leftarrow f_\theta^{enc}\big((d_v, \emptyset), \{h_u | u \in parent(v)\}\big). \tag{8}$$

Step 2: Cross-Layer Link Prediction. To approximate $P(\tilde{E}^t \mid G^{<t}, V^t, \hat{E}^t)$, we also follow two inductive biases in step 1. That is, the agent representation captures all execution semantics, and links are independent. Moreover, the information of $(V^t, \hat{E}^t)$ has been captured during the update in Eq.8. In this way, the parameterization of cross-layer link prediction is defined as:

$$P(\tilde{E}^t \mid G^{<t}, V^t, \hat{E}^t) \approx \prod_{(u,v)\in\tilde{\mathcal{E}}^t} \big[p_\theta^{link}(h_u, h_v)\big]^{y_e} \big[1 - p_\theta^{link}(h_u, h_v)\big]^{1-y_e}, \tag{9}$$

where p_θ^{link} is another deep link predictor. Similarly, edges with $p_\theta^{link} > \delta$ are selected. After prediction, the embeddings of each leaf are updated as:

$$h_v \leftarrow f_\theta^{enc}\big((d_v, \emptyset), \{h_u | u \in pre(v)\}\big). \tag{10}$$

That is, the topological channel contains both parent information and ancestor information.

Step 3: Layer-Wise Workflow Execution. After generating $(V^t, E^{<t,t})$, the agentic system executes the t-th layer asynchronously. As all nodes in V^t finish their pipelines, the textual execution outputs are incorporated into the representations:

$$h_v \leftarrow f_\theta^{enc}\big((d_v, z_v), \{h_u | u \in pre(v)\}\big). \tag{11}$$

Eventually, h_v represents all accumulated information related to node v until step t.

Generation to Final Decision. If there is no connection to candidates (i.e., $|V^t| = 0$) in Eq. 7, layer-wise iteration terminates. We replace the candidate set $\hat{V}$ with the final decision agent o and repeat Step 1$\sim$Step 3 to generate edges pointing to o. Notably, to ensure that the workflow reaches the final decision, if no edge pointed to o satisfies $p_\theta^{agt} > \delta$ in Step 1, we forcibly select the edge with the highest predicted probability.

Technically, any autoregressive encoder could be extended for the tasks, and the decoders could simply follow traditional link predictors Zhang & Chen (2018). In our study, f_θ^{enc} is a variant of the recurrent neural network (RNN) Yu et al. (2019) for parameter-efficiency. p_θ^{agt} and p_θ^{link} are linearized transformations of node pairs. Detailed reports are presented in Appendix C.

4.2 Training Strategy of Layer-Wise Topology Generation

Supervised Pre-Training for Utility. For each query in the training set, we utilize the LLM-driven Monte Carlo search to investigate a feasible query-specific graph. The detailed designs of our LLM-driven search are presented in Appendix E. After query-wise system generation, to avoid bias toward specific structures, we retain only one among a set of isomorphic graphs.

Table 1: Rate of "generating only the final decision" across topology generations after training with BCE and WBCE.

Dataset	BCE	WBCE
MMLU	99.4	0.2
SVAMP	97.4	0.0
AQuA	98.2	0.2
HumanEval	96.0	0.0
MBPP	95.6	0.0
AsyncHow	74.6	0.0

Subsequently, the supervised task becomes a multi-step link prediction problem. Similar to traditional link prediction tasks, the number of negative samples (denoted as n_{neg}) is much larger than that of positive samples (denoted as n_{pos}). Under class imbalance, the direct utilization of binary cross entropy (BCE) leads to a preference over negative samples. As a result, models will only output the final decision without generation during inference as shown in Table 1. Therefore, we utilize weighted binary cross entropy (WBCE) as a practical solution, where the gradients of positive samples are scaled by a weight n_{neg}/n_{pos}. Finally, our training objective becomes:

$$\mathcal{L}_u = \frac{1}{N_{agt}} \sum_{t=1}^{T} \sum_{e\in\hat{\mathcal{E}}^t} \mathcal{L}_{WBCE}\big(p_\theta^{agt}(e), y_e\big) + \frac{1}{N_{link}} \sum_{t=1}^{T} \sum_{e\in\tilde{\mathcal{E}}^t} \mathcal{L}_{WBCE}\big(p_\theta^{link}(e), y_e\big), \tag{12}$$

where $N_{agt} = \text{sum}(|\hat{\mathcal{E}}^1|, .., |\hat{\mathcal{E}}^T|)$ and $N_{link} = \text{sum}(|\tilde{\mathcal{E}}^1|, .., |\tilde{\mathcal{E}}^T|)$.

RL-Based Post-Training for Efficiency. A desired graph design should not only ensure task solving but also minimize redundant operations. Consequently, we utilize RL-based post-training for more efficient strategies. Firstly, for each query, we treat $p_\theta^{agt}(\cdot)$ and $p_\theta^{link}(\cdot)$ as two Bernoulli distributions to sample a group of trajectories $\{\tau_1, \tau_2, ..., \tau_n\}$. Each trajectory τ_i is:

$$\begin{aligned} \tau_i = \{G_i^0, G_i^1, ..., G_i^T\},\ G_i^t = \big(V_i^t,\ \{e \mid y_e = 1,\ e \in \hat{\mathcal{E}}_i^t \cup \tilde{\mathcal{E}}_i^t\}\big), \\ \forall e \in \hat{\mathcal{E}}_i^t,\ y_e \sim \text{Bernoulli}\big(p_\theta^{agt}(e)\big),\ \forall e \in \tilde{\mathcal{E}}_i^t,\ y_e \sim \text{Bernoulli}\big(p_\theta^{link}(e)\big). \end{aligned} \tag{13}$$

The reward R_i for a sampled trajectory within a group is defined as:

$$\begin{aligned} R_i = R_i^u + R_i^s, \quad R_i^u = \mathbb{I}\{\tau_i \text{ solves the query}\}, \\ R_i^s = -\frac{1}{2}\left(\frac{|V_i| - \min_j |V_j|}{\max_j |V_j| - \min_j |V_j| + \eta} + \frac{|E_i| - \min_j |E_j|}{\max_j |E_j| - \min_j |E_j| + \eta}\right), \end{aligned} \tag{14}$$

where η is a small constant for numerical stability. Intuitively, R_i^s encourages fewer agents and communication channels. Moreover, the rewards of successful trajectories are strictly larger than those of failed ones to ensure utility. Subsequently, advanced RL strategies (e.g., GRPO Shao et al. (2024)) can be applied:

$$\theta' = \mathcal{J}_{RL}(\theta, \{\tau_1, \tau_2, ..., \tau_n\}, \{R_1, R_2, ..., R_n\}), \tag{15}$$

where $\mathcal{J}_{RL}$ denotes the update process of RL and θ is the parameters of deep modules in RMAP.

5 Experimental Results

We conduct experiments to answer the questions: **(RQ1)** Could our scheme outperform SOTA MAS topology designers across various tasks? **(RQ2)** Are three context channels necessary for capturing agent semantics and dependencies? **(RQ3)** Could our post-training strategy improve the efficiency of the planned MAS? **(RQ4)** How much overhead does RMAP introduce during task solving?

5.1 Experimental Settings

Datasets. Four real-world tasks are used: (1) General question answering, with MMLU Hendrycks et al. (2021); (2) Mathematical reasoning, including SVAMP Patel et al. (2021) and AQuA Ling et al. (2017); (3) Code generation, including HumanEval Chen et al. (2021), MBPP Austin et al. (2021), and DS1000 Lai et al. (2023); (4) Asynchronous planning, with AsyncHow Lin et al. (2024).

Baselines. RMAP is compared with: (1) A single node baseline with only final decision; (2) Task-specific MAS designers, including G-Designer Zhang et al. (2025c), MaAS Zhang et al. (2025a), AgentPrune Zhang et al. (2025b), and AFlow Zhang et al. (2025d); (3) A query-specific MAS designer, ARG-Designer Li et al. (2026b); (4) A runtime designer, Anymac Wang et al. (2025b) with a one-by-one generation paradigm. More implementation details are provided in Appendix B.

LLM Backbones. For agent executors, we utilize two closed-source LLMs, GPT-4o-mini Achiam et al. (2023) and Gemini-3.1-flash Team et al. (2023), and two open-source LLMs, Llama-4-scout Adcock et al. (2026) and DeepSeek-v3 Liu et al. (2024). For the data generator, we only utilize Gemini-3.1-flash due to its efficiency and capacity. For efficient text encoding of agent semantics, we use MiniLM-L6-v2 Wang et al. (2020), a small-scale language model.

5.2 Performance Evaluation (RQ1)

To demonstrate the performance improvement, we compare our scheme with SOTA baselines. For a fair comparison, each agent role can be instantiated no more than twice, as in static designers operating on a predefined graph. Moreover, RMAP in this experiment is only pre-trained for fair comparison. The experimental results across five runs are shown in Table 2. From the experimental results, we have the following observations:

Our scheme effectively improves task performance across scenarios. Specifically, RMAP outperforms all baselines on 26 of the 28 evaluated tasks. Notably, when driven by Gemini-3.1-flash, RMAP improves the MAS pass rate on DS1000 from over 60% to over 70%.

Table 2: Performance (in $\mu \pm \sigma$ %) comparison among MAS topology designers.

LLM	Designer	MMLU	SVAMP	AQuA	HumanEval	MBPP	DS1000	AsyncHow
GPT-4o-mini	Single node	72.4±1.2	87.6±2.0	75.8±1.4	82.9±1.3	72.3±0.8	29.6±2.4	58.7±1.8
	G-Designer	76.2±2.0	90.8±1.2	79.2±1.1	86.0±0.9	75.0±1.4	35.8±1.9	61.8±1.5
	MaAS	75.4±2.2	91.2±2.0	77.4±1.5	86.4±1.3	75.9±0.9	38.2±1.3	60.7±1.7
	AgentPrune	76.0±1.3	91.6±1.6	78.3±1.0	86.5±1.4	74.8±1.2	36.9±2.7	61.1±1.3
	AFlow	77.0±1.7	92.4±0.8	79.1±0.8	86.9±1.7	76.6±2.0	39.8±2.2	62.6±1.4
	ARG-Designer	77.8±2.2	92.2±0.7	78.8±0.8	87.5±1.6	75.9±2.3	41.0±2.5	62.0±1.8
	Anymac	77.5±2.1	92.6±1.3	80.5±1.2	86.3±1.2	76.0±3.4	44.2±1.8	61.7±2.6
	RMAP (ours)	**80.1±1.1**	**93.0±1.0**	**81.3±0.7**	**89.4±1.4**	**80.3±0.8**	**47.7±1.6**	**63.8±1.5**
Gemini-3.1-flash	Single node	84.5±1.6	88.9±1.2	83.2±1.4	90.4±0.7	85.4±1.9	50.9±1.0	80.7±1.3
	G-Designer	87.3±1.8	93.6±1.2	89.4±1.2	94.1±0.8	89.5±1.2	59.4±1.7	85.6±1.5
	MaAS	87.1±2.0	92.8±0.8	88.2±0.8	90.7±0.8	88.6±1.4	60.9±2.7	86.1±0.8
	AgentPrune	88.8±1.4	93.0±1.2	89.2±1.1	93.6±1.4	87.3±0.8	58.6±3.6	85.0±0.9
	AFlow	90.3±1.7	93.5±1.5	89.9±1.0	95.5±1.7	88.0±1.0	64.2±2.9	86.2±0.9
	ARG-Designer	89.0±2.0	93.4±1.5	**90.4±1.0**	94.4±1.2	90.0±1.5	66.5±2.0	85.5±1.4
	Anymac	89.2±2.1	92.6±1.0	90.0±0.7	94.6±0.8	87.9±1.8	66.0±1.4	84.9±2.0
	RMAP (ours)	**92.1±0.5**	**95.4±1.1**	90.2±1.2	**96.8±0.5**	**92.5±1.2**	**76.4±1.9**	**92.8±1.1**
Llama-4-scout	Single node	76.5±1.5	90.2±1.7	71.5±1.9	74.4±1.6	65.0±1.0	30.2±2.2	57.8±1.7
	G-Designer	78.3±2.0	92.4±1.7	74.1±1.5	77.1±0.9	67.8±0.8	39.1±1.6	61.4±1.4
	MaAS	79.8±1.8	93.0±0.8	75.1±1.4	78.0±1.1	69.2±0.8	39.5±1.8	60.8±0.9
	AgentPrune	79.0±1.1	92.5±1.8	73.0±2.2	76.8±1.4	68.0±1.4	38.0±1.6	59.9±2.0
	AFlow	80.8±1.9	**94.4±0.5**	74.8±2.4	78.6±2.0	69.2±1.5	42.6±2.8	64.7±2.9
	ARG-Designer	81.0±2.3	93.4±1.0	75.6±2.3	77.5±1.7	68.5±1.7	44.3±1.7	62.3±1.5
	Anymac	80.5±1.7	93.0±1.8	76.9±1.4	78.0±1.2	68.7±1.3	43.5±3.4	61.1±2.0
	RMAP (ours)	**82.0±1.8**	93.7±0.6	**78.7±1.4**	**82.9±1.0**	**73.8±1.9**	**51.9±2.1**	**67.9±1.3**
DeepSeek-v3	Single node	77.5±0.6	89.0±2.0	84.2±1.2	70.5±2.1	63.4±1.2	46.7±3.4	86.3±2.3
	G-Designer	81.7±1.3	91.5±1.7	87.4±0.8	74.5±1.7	65.7±1.6	49.8±1.8	88.2±1.1
	MaAS	83.4±1.4	92.4±1.1	86.8±0.9	73.3±0.5	68.5±1.5	52.8±1.3	87.6±1.5
	AgentPrune	80.5±1.9	91.7±1.2	86.5±1.5	74.0±1.0	66.5±2.1	50.9±1.0	86.4±2.2
	AFlow	83.2±1.2	92.6±1.1	87.9±1.5	74.6±1.4	67.8±2.4	53.2±2.0	88.5±1.5
	ARG-Designer	83.9±2.0	92.4±0.9	89.0±1.8	75.7±0.7	68.2±1.3	55.4±2.5	87.9±2.2
	Anymac	85.6±2.3	92.0±1.5	89.5±1.2	74.5±0.8	68.8±1.6	54.2±2.8	87.4±1.6
	RMAP (ours)	**88.8±1.1**	**93.2±0.7**	**90.5±1.6**	**76.8±1.1**	**71.6±1.5**	**60.7±1.3**	**90.9±1.0**

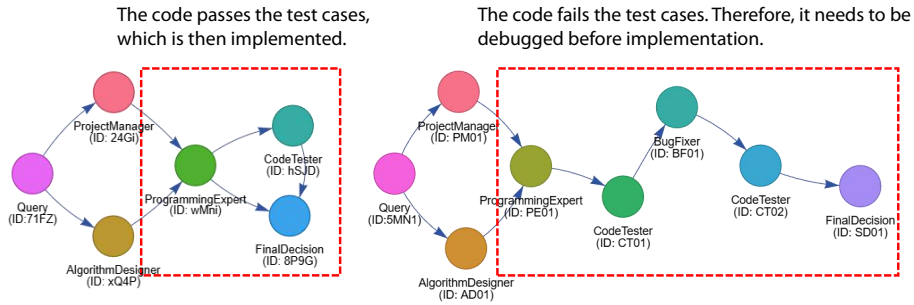


Figure 3: A case study in HumanEval: For query 71FZ, the generated code passes CodeTester's evaluation, which results in a concise topology without BugFixer. For query 5MN1, failed testing results trigger the introduction of BugFixer and an additional CodeTester before final decision.

Our scheme greatly facilitates multi-step reasoning tasks. Concretely, compared with QA tasks, multi-step reasoning tasks, such as code generation and graph planning, require precise planning to explore potential solutions, evaluate their correctness, and make improvements. Therefore, in these component tasks, our scheme assigns agents based on the runtime systematic requirements, thereby realizing more significant improvements. In addition, we provide a case study in Figure 3 to visualize how RMAP assigns agents based on information dependencies.

5.3 Ablation Study (RQ2)

To reveal the necessity of specific modules, we compare the full scheme with three variants after pre-training: (1) Prediction without the topological semantics (denoted as "w/o topo"); (2) Prediction but without the output semantics (denoted as "w/o output"); (3) Systematic generation but without step 2 (denoted as "w/o cross"). In this way, the DAG is limited to a hierarchical structure. Other

Table 3: Performance (in %) comparison between RMAP and its variants.

LLM	Variant	MMLU	SVAMP	AQuA	HumanEval	MBPP	DS1000	AsyncHow
GPT-4o-mini	RMAP	**80.1**	**93.0**	**81.3**	**89.4**	**80.3**	47.7	63.8
	w/o topo	79.2↓0.9	92.6↓0.4	80.3↓1.0	87.5↓1.9	78.5↓1.8	46.0↓1.7	61.3↓2.5
	w/o output	74.5↓5.6	91.8↓1.2	77.1↓4.2	85.7↓3.7	74.3↓6.0	45.7↓2.0	59.8↓4.0
	w/o cross	77.7↓2.4	92.4↓0.6	77.9↓3.4	86.4↓3.0	76.8↓3.5	**53.8↑6.1**	**65.1↑1.3**
Gemini-3.1-flash	RMAP	**92.1**	**95.4**	**90.2**	**96.8**	**92.5**	**76.4**	92.8
	w/o topo	91.1↓1.0	94.2↓1.2	89.6↓0.6	94.4↓2.4	91.4↓1.1	75.0↓1.4	90.2↓2.6
	w/o output	91.5↓0.6	94.5↓0.9	88.1↓2.1	95.2↓1.6	89.5↓3.0	74.4↓2.0	91.6↓1.2
	w/o cross	90.2↓1.9	93.5↓1.9	89.2↓1.0	96.4↓0.4	88.2↓4.3	75.7↓0.7	**93.4↑0.6**
Llama-4-scout	RMAP	**82.0**	**93.7**	**78.7**	**82.9**	73.8	**51.9**	**67.9**
	w/o topo	81.3↓0.7	92.0↓1.7	77.6↓1.1	81.0↓1.9	72.2↓1.6	49.3↓2.6	66.6↓1.3
	w/o output	79.3↓2.7	92.8↓0.9	75.4↓3.3	78.6↓4.3	70.0↓3.8	47.8↓4.1	65.8↓2.1
	w/o cross	81.6↓0.4	93.0↓0.7	78.2↓0.5	80.7↓2.2	**75.1↑1.3**	50.5↓1.4	66.9↓1.0
DeepSeek-v3	RMAP	**88.8**	**93.2**	**90.5**	**76.8**	**71.6**	**60.7**	**90.9**
	w/o topo	87.6↓1.2	92.4↓0.8	89.8↓0.7	75.4↓1.4	70.3↓1.3	58.9↓1.8	89.2↓1.7
	w/o output	86.2↓2.6	91.8↓1.4	88.1↓2.4	73.2↓3.6	67.8↓3.8	57.5↓3.2	87.6↓3.3
	w/o cross	85.9↓2.9	91.7↓1.5	90.0↓0.5	74.8↓2.0	70.8↓0.8	58.3↓2.4	88.1↓2.8

Table 4: Performance (denoted as Perf, in %), time cost per task (denoted as Time, in seconds), and token cost per task (denoted as Token, in tokens) as the number of post-training steps increases.

N-step	MMLU			SVAMP			HumanEval			AsyncHow		
	Perf	Time	Token	Perf	Time	Token	Perf	Time	Token	Perf	Time	Token
0	80.1	9.7	2643.6	93.0	17.0	4873.2	89.4	20.4	4643.5	63.8	22.3	6083.2
5	79.5	5.9	1032.6	93.2	10.0	2151.4	88.6	10.0	2143.7	64.1	20.1	5642.6
10	79.7	3.9	723.5	92.4	5.9	1459.2	88.3	6.8	1580.3	64.4	16.2	5268.4
15	79.3	2.1	423.8	93.0	6.5	1886.9	87.8	5.6	1185.4	64.1	17.2	5519.3
20	79.9	2.8	568.2	**93.7**	**4.1**	**1066.0**	**89.4**	**8.0**	**1854.2**	64.7	14.5	4844.1
25	**80.7**	**3.1**	**643.5**	93.2	2.9	763.6	88.8	5.6	1287.6	**65.4**	**13.0**	**4477.1**

modules cannot be forbidden. For instance, if the role definition channel is removed, the candidate agents become invalid for the next-layer agent generation. From Table 3, we observe that:

Both topological context and execution information are critical for RMAP to discover effective multi-agent collaboration strategies. For instance, under GPT-4o-mini, removing topological context (w/o topo) decreases performance from 89.4% to 87.5% on HumanEval and from 47.7% to 46.0% on DS1000. Similarly, removing execution output (w/o output) reduces performance on AsyncHow from 63.8% to 59.8%.

Generally, allowing DAG structures provides greater flexibility than restricting the topology to a hierarchical structure. This is because a DAG provides more flexible communication patterns and allows agents to establish long-range information dependencies beyond adjacent stages. However, hierarchical structures remain effective when prior knowledge suggests that agents only require stage-wise dependencies. In such cases, next-layer leaf prediction can construct efficient communication patterns by avoiding unnecessary long-range interactions. **Therefore, we recommend that developers select the generated topology form based on the task requirements.**

5.4 Analysis of Post-Training (RQ3)

To evaluate the efficiency gains from post-training, we compare RMAP before and after post-training in terms of performance, token usage, and execution time. The strategy mainly follows GRPO, and the LLM backbone of agents is GPT-4o-mini. From Table 4, we find that:

The RL-based post-training could find solutions with substantially improved systemic efficiency. In terms of time costs, post-training yields a 41.7% to 82.9% reduction across all evaluated tasks. With regard to token consumption, post-training cuts the usage from around 4600 to 1300 on HumanEval and from around 6100 to 4500 on AsyncHow.

After post-training, RMAP achieves competitive or even better performance than its pre-trained version. For instance, performance is enhanced from 80.1% to 80.7% on MMLU and from 63.8% to 65.4% on AsyncHow. The case study is presented in Appendix D.

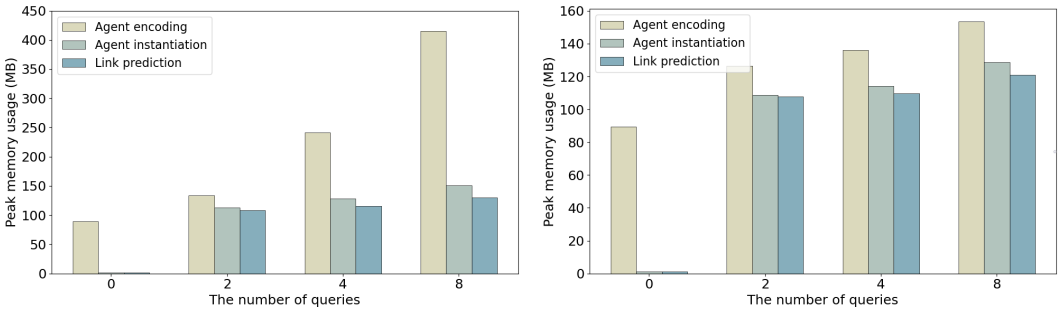


Figure 4: The peak memory usages across steps on MBPP (left) and AsyncHow (right). When the number of queries is 0, the memory for module loading is reported.

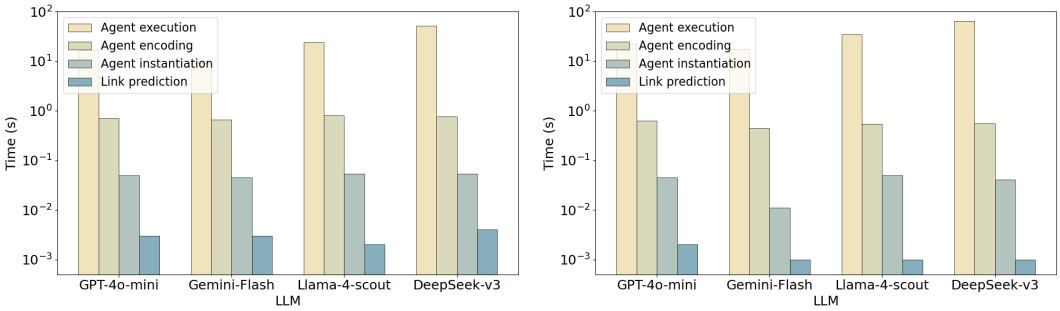


Figure 5: The average time costs per query across steps on MBPP (left) and AsyncHow (right).

5.5 Inference Overhead Report (RQ4)

To quantify the introduced GPU overhead and time latency, we report memory peaks as the number of queries increases and the average time cost per query across the underlying LLMs. One RTX 4090 is used, and the APIs of the LLMs are sufficient for each query. RMAP is only pre-trained to test the worst case. As shown in Figure 4 and Figure 5, results reveal the following conclusions:

Within RMAP, the agent encoding is the most resource-intensive step for generation. This is mainly because MiniLM-L6-v2 is used for text encoding and is invoked multiple times in the asynchronous execution. In contrast, agent generation and link generation only involve lightweight linear transformations of embeddings, resulting in substantially lower memory consumption.

Within a MAS task, RMAP introduces limited overhead. Concurrent planning with RMAP consumes only several hundred MB of GPU memory, which is readily manageable on modern personal computers. In terms of latency, most of the inference time is spent on agent execution. Specifically, RMAP adds less than one second of planning time per query, corresponding to 1.6% and 7.9% additional execution latency with DeepSeek-v3 and Gemini-3.1-flash on MBPP, respectively.

6 Conclusion

In this study, we reveal that prevalent MAS topology planning strategies mostly follow a "design-then-deploy" paradigm, which limits their adaptability to dynamic environments. To enable runtime topology construction, we propose RMAP, which formulates topology planning as an autoregressive next-layer generation. Extensive experiments across tasks lead to the following findings: (1) Our dynamic designer generally outperforms SOTA static designers across various LLMs and scenarios; (2) Agent semantics can be represented by three complementary information channels: role descriptions, topological relationships, and execution feedback. They should be progressively incorporated according to the execution stage; (3) Supervised pre-training provides preliminary usability, while RL-based post-training further improves systemic efficiency; and (4) RMAP can be efficiently deployed on a single GPU with limited additional latency relative to the cost of agentic reasoning while supporting multi-query asynchronous inference. In future work, we will investigate how agent capabilities and MAS topologies can jointly evolve during execution to support long-horizon adaptation.

AI USE STATEMENT

We used generative AI tools for:

- Generate synthetic data sets.

We have not used generative AI tools for:

- Help develop theoretical models or conceptual frameworks.
- Propose or refine hypotheses.
- Design or provide feedback on research methodology or experiments.
- Implement methods.
- Support qualitative and thematic data analysis.
- Interpret results.

The following terms are not applicable to this work:

- Formulate mathematical claims.
- Provide critical ingredients for proving mathematical claims.
- Assist in the writing of proofs.
- Assist with translation.
- Clean and reformat dataset.

Additionally, we used generative AI tools for:

- Create or edit software code.
- Discover research topics or identify gaps.
- Edit a research paper to improve readability.

We have reviewed all AI-assisted work. The validity of the generated data is ensured through strict format checks. All generated code has been tested by the authors. Articles listed by the LLM are examined for potential hallucinations. We only employ the LLM to provide revision suggestions, while the actual writing is completed by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personal or sensitive data, or human-subject experiments. All datasets used in our experiments are publicly available and were used in accordance with their respective licenses and intended research purposes. We do not identify any additional ethical concerns associated with this work.

REPRODUCIBILITY STATEMENT

We have released our code achievement and evaluation environment in an anonymous repository (i.e., https://anonymous.4open.science/r/RMAP-1884), with extensive comments to facilitate understanding and usage. We also provide detailed descriptions of the hyper-parameters, model architectures, and data generation procedures in the Appendix to facilitate reproducibility.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

Aaron Adcock, Aayushi Srivastava, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pande, Abhinav Pandey, Abhinav Sharma, Abhishek Kadian, Abhishek Kumawat, Adam Kelsey, et al. The llama 4 herd: Architecture, training, evaluation, and deployment notes. arXiv preprint arXiv:2601.11659, 2026.

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.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? Advances in Neural Information Processing Systems, 38, 2026.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje Karlsson, Jie Fu, and Yemin Shi. Autoagents: a framework for automatic agent generation. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, pp. 22–30, 2024.

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.

Saehun Chun, Wonje Choi, Sera Choi, Sanghyun Ahn, and Honguk Woo. Functional cache grafting: Robust and rapid code-policy synthesis for embodied agents. In Proceedings of the 43st International Conference on Machine Learning, 2026.

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.

Yuan Gui, Hongchen Luo, Jiao Wang, and Liqi Qu. Beyond rule-based agents: Active markov games for realistic multi-agent interaction in autonomous driving. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10689–10698, 2026.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. Advances in neural information processing systems, 30, 2017.

Xiao Han, Chen Zhu, Hengshu Zhu, and Xiangyu Zhao. Swarm intelligence in geo-localization: A multi-agent large vision-language model collaborative framework. In Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2, pp. 814–825, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In International Conference on Learning Representations, 2021.

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

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 9505–9523, 2025.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In International Conference on Machine Learning, pp. 18319–18345. PMLR, 2023.

Hao Li, Xiaogeng Liu, CHIU Chun, Dianqi Li, Ning Zhang, and Chaowei Xiao. Drift: Dynamic rule-based defense with injection isolation for securing llm agents. Advances in Neural Information Processing Systems, 38:83262–83290, 2026a.

Mufei Li, Viraj Shitole, Eli Chien, Changhai Man, Zhaodong Wang, Ying Zhang, Tushar Krishna, Pan Li, et al. Layerdag: A layerwise autoregressive diffusion model for directed acyclic graph generation. In The Thirteenth International Conference on Learning Representations, 2025.

Shiyuan Li, Yixin Liu, Qingsong Wen, Chengqi Zhang, and Shirui Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 40, pp. 23142–23150, 2026b.

Fangru Lin, Emanuele La Malfa, Valentin Hofmann, Elle Michelle Yang, Anthony Cohn, and Janet B Pierrehumbert. Graph-enhanced large language models in asynchronous plan reasoning. arXiv preprint arXiv:2402.02805, 2024.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 158–167, 2017.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437, 2024.

Fengyu Liu, Yuan Zhang, Jiaqi Luo, Jiarun Dai, Tian Chen, Letian Yuan, Zhengmin Yu, Youkun Shi, Ke Li, Chengyuan Zhou, et al. Make agent defeat agent: Automatic detection of taint-style vulnerabilities in llm-based agents. In 34th USENIX Security Symposium (USENIX Security 25), pp. 3767–3786, 2025.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 2080–2094, 2021.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, pp. 1743–1752. Association for Computational Linguistics (ACL), 2015.

Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. Meminsight: Autonomous memory augmentation for llm agents. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pp. 33124–33140, 2025.

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

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805, 2023.

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

Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. G-safeguard: A topology-guided security lens and treatment on llm-based multi-agent systems. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 7261–7276, 2025a.

Song Wang, Zhen Tan, Zihan Chen, Shuang Zhou, Tianlong Chen, and Jundong Li. Anymac: Cascading flexible multi-agent collaboration via next-agent prediction. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pp. 11566–11578, 2025b.

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, 33:5776–5788, 2020.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, et al. Tablebench: A comprehensive and complex benchmark for table question answering. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, pp. 25497–25506, 2025.

Yunzhe Xu, Yiyuan Pan, Zhe Liu, and Hesheng Wang. Flame: Learning to navigate with multimodal llm in urban environments. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, pp. 9005–9013, 2025.

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, 2022.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In International conference on machine learning, pp. 5708–5717. PMLR, 2018.

Jingxuan Yu, Ju Jia, Simeng Qin, Xiaojun Jia, Siqi Ma, Yihao Huang, Yali Yuan, and Guang Cheng. Mpas: Breaking sequential constraints of multi-agent communication topologies via individual-epistemic message propagation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 40, pp. 29847–29855, 2026.

Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. Neural computation, 31(7):1235–1270, 2019.

Zheng Yu, Ziyi Guo, Yuhang Wu, Jiahao Yu, Meng Xu, Dongliang Mu, Yan Chen, and Xinyu Xing. Patchagent: A practical program repair agent mimicking human expertise. In 34th USENIX Security Symposium (USENIX Security 25), pp. 4381–4400, 2025.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 15549–15572, 2025.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In International Conference on Machine Learning, pp. 75834–75852. PMLR, 2025a.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In The Thirteenth International Conference on Learning Representations, 2025b.

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 International Conference on Machine Learning, pp. 76678–76692. PMLR, 2025c.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems. Advances in Neural Information Processing Systems, 38:12988–13018, 2026a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In The Thirteenth International Conference on Learning Representations, 2025d.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. Advances in neural information processing systems, 31, 2018.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. Advances in neural information processing systems, 32, 2019.

Zhe Zhang, Runlin Liu, Aishan Liu, Xingyu Liu, Xiang Gao, and Hailong Sun. Code2bench: Scaling source and rigor for dynamic benchmark construction. In The Fourteenth International Conference on Learning Representations, 2026b.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: language agents as optimizable graphs. In Proceedings of the 41st International Conference on Machine Learning, pp. 62743–62767, 2024.

A Dataset Statistic

The descriptions and statistics of the benchmark datasets are presented in Table 5.

Table 5: Dataset descriptions and statistics.

Dataset	Task Type	Metric	Questions
MMLU Hendrycks et al. (2021)	4-choice question answering	Accuracy	1816
SVAMP Patel et al. (2021)	Mathematical problem solving	Accuracy	1000
AQuA Ling et al. (2017)	5-choice mathematical answering	Accuracy	254
HumanEval Chen et al. (2021)	Python code generation	Passing rate	161
MBPP Austin et al. (2021)	Python code generation	Passing rate	974
DS1000 Lai et al. (2023)	Science code generation	Passing rate	1000
AsyncHow Lin et al. (2024)	Shortest path planning	Accuracy	1598

B Implementation Details

The experimental environment is as follows:

GPU: One RTX 4090

Python version: 3.10

Torch version: 2.1.2+cu121

Transformers version: 4.36.2

OpenAI version: 1.65.2

For training data generation, the hyper-parameters are as follows:

LLM: Gemini-3.1-flash

LLM temperature: 0.2

LLM Max tokens: 8192

Maximum number of attempts per query: 4

For supervised pre-training, the hyper-parameters are as follows:

Optimizer: Adam with default setting

learning rate: 0.0001

The number of epochs: 1000

Batch size: 8

Early stopping: True with patience of 20 epochs based on the best training loss

For RL-based post-training, the hyper-parameters are as follows:

Optimizer: Adam with default setting

Learning rate: 0.0001

The number of steps: 25

Group size: 8

The number of trajectories: 8

Group normalization: True

Policy clipping: True with $\epsilon = 0.2$

Update epochs within a step: 4

KL regularization: False

For inference, our solution is subject to the following conditions:

Maximum number of layers: 8 to avoid infinite generation

Sampling threshold: $\delta = 0.5$

For MAS execution, the following hyper-parameters remain the same across various baselines:

The number of queries per execution: 4 by default with asynchronous execution

LLM temperature: 0.2

LLM Max tokens: 8192

Agents: Due to the large number of agents, we do not introduce them one-by-one in the appendix. If interested, please refer to /rmap/environment/agents in our anonymous repository

Maximum number of initializations for each agent: 2. We impose this constraint for fair comparison, as some task-specific static designers must predefine the involved agents before optimizing their mutual edges. Without this constraint, generation-based designers could use larger graph scales, giving them an unfair advantage.

Train Test Split. If the dataset follows the default standard (e.g., MMLU), we will adhere to it. In contrast, the data will be split into 60% for training and 40% for testing. All baselines are trained on a specific dataset and a specific LLM backbone.

C Encoder and Decoder Designs

C.1 Agent Encoder

For agent v, the textual description d_v and textual execution output z_v should be encoded by language models (LMs), and we use MiniLM Wang et al. (2020):

$$h_v^d = f_{lm}(d_v), h_v^z = f_{lm}(z_v). \tag{16}$$

Following traditional graph neural networks Hamilton et al. (2017), the structural information is combined via an inductive pooling function, and the widely-used mean pooling is selected.

$$h_v^{s1} = \frac{1}{|parent(v)|} \sum_{u \in parent(v)} h_u \text{ and } h_v^{s2} = \frac{1}{|pre(v)|} \sum_{u \in pre(v)} h_u. \tag{17}$$

Table 6: Performance (in %) comparison between RMAP and its variants after supervised pre-training. Specifically, when softmax gate is removed, the variant is denoted as "w/o softmax gate". When layer normalization is removed, the variant is denoted as "w/o layer norm". When GELU is added to each channel, the variant is denoted as "w/ gelu".

LLM	Variant	MMLU	SVAMP	AQuA	HumanEval	MBPP	DS1000	AsyncHow
GPT-4o-mini	RMAP	80.1	**93.0**	**81.3**	89.4	**80.3**	47.7	63.8
	w/o softmax gate	78.4↓1.7	91.5↓1.5	80.2↓1.1	80.0↓9.4	79.8↓0.5	45.7↓2.0	62.5↓1.3
	w/o layer norm	79.2↓0.9	92.5↓0.5	81.6↑0.3	88.8↓0.6	79.5↓0.8	46.6↓1.1	63.1↓0.7
	w/ gelu	**80.4**↑0.3	92.7↓0.3	80.7↓0.6	**89.9**↑0.5	79.4↓0.9	**48.0**↑0.3	**64.2**↑0.4
Gemini-3.1-flash	RMAP	**92.1**	**95.4**	90.2	**96.8**	**92.5**	76.4	**92.8**
	w/o softmax gate	90.8↓1.3	93.6↓1.8	88.0↓2.2	95.0↓1.8	90.6↓1.9	73.4↓3.0	89.9↓2.9
	w/o layer norm	91.4↓0.7	94.2↓1.2	90.5↑0.3	96.0↓0.8	91.3↓1.2	75.2↓1.2	91.5↓1.3
	w/ gelu	91.4↓0.7	95.1↓0.3	**90.6**↑0.4	96.4↓0.4	91.8↓0.7	**76.9**↑0.5	92.3↓0.5
Llama-4-scout	RMAP	82.0	**93.7**	**78.7**	**82.9**	**73.8**	**51.9**	67.9
	w/o softmax gate	81.4↓0.6	92.1↓1.6	76.5↓2.2	79.7↓3.2	71.5↓2.3	48.8↓3.1	66.4↓1.5
	w/o layer norm	81.0↓1.0	92.5↓1.2	77.9↓0.8	81.7↓1.2	72.5↓1.3	50.4↓1.5	66.8↓1.1
	w/ gelu	**82.3**↑0.3	93.3↓0.4	78.0↓0.7	82.3↓0.6	73.0↓0.8	51.4↓0.5	**68.5**↑0.6
DeepSeek-v3	RMAP	88.8	**93.2**	90.5	76.8	**71.6**	**60.7**	**90.9**
	w/o softmax gate	88.0↓0.8	91.2↓2.0	87.6↓2.9	74.7↓2.1	70.0↓1.6	58.5↓2.2	87.4↓3.5
	w/o layer norm	87.5↓1.3	92.5↓0.7	89.4↓1.1	76.3↓0.5	71.0↓0.6	61.0↑0.3	89.6↓1.3
	w/ gelu	**89.5**↑0.7	92.8↓0.4	**91.0**↑0.5	**77.2**↑0.4	70.9↓0.7	60.3↓0.4	90.4↓0.5

As presented in the main paper, the embeddings of three channels are progressively fused. In step 1, h_v^d is directly used as h_v. In step 2, the agent embedding is:

$$\begin{aligned} h_v &= a_1 h_1 + a_2 h_2, \\ h_1 = W_1(\text{LN}_1(h_v^d)), h_2 &= W_2(\text{LN}_2(h_v^{s1})), \\ a &= \text{softmax}(W_3[h_1||h_2]) \in R^2, \end{aligned} \tag{18}$$

where $W1$, W_2 and W_3 are learnable matrices and LN denotes the layer norm. And after execution, in step 3, the agent embedding is:

$$\begin{aligned} h_v &= a_1 h_1 + a_2 h_2 + a_3 h_3, \\ h_1 = W_4(\text{LN}_3(h_v^d)), h_2 = W_5(\text{LN}_4(h_v^{s2})), h_3 &= W_6(\text{LN}_5(h_v^z)), \\ a &= \text{softmax}(W_7[h_1||h_2||h_3]) \in R^3. \end{aligned} \tag{19}$$

The above designs could be treated as a variant of the recurrent neural network (RNN), where structural information is the input of the previous steps and the input of the current step is composed of self description and agent output. We reveal that advanced time-series models are unnecessary for few-step multi-agent planning tasks but with significant latency. Moreover, as shown in Table 6, the softmax gate and layer norm contribute to consistent performance improvements after supervised pre-training. However, incorporating additional activation functions yields no consistent and substantial performance gains across benchmarks. The reason is that the language model's embeddings are already in a nonlinear space.

C.2 Leaf and Link Predictor

In traditional link prediction, the embedding of an edge is a combination of two endpoints. Therefore, the leaf predictor is designed as:

$$p_\theta^{agt}(v,u) = \text{sigmoid}(h_l), \text{ where } h_l = W_l(h_v||h_u). \tag{20}$$

Similarly, the link predictor is designed as:

$$p_\theta^{link}(v,u) = \text{sigmoid}(h_e), \text{ where } h_e = W_e(h_v||h_u). \tag{21}$$

In summary, the encoder-decoder architecture only involves several matrices and an embedding language model, which is substantially more lightweight compared to LLM backbones of agents.

The code achievement can be found at /rmap/emb/enc_dec.py in our anonymous repository.

D ANALYSIS OF POST-TRAINING

D.1 CASE STUDIES

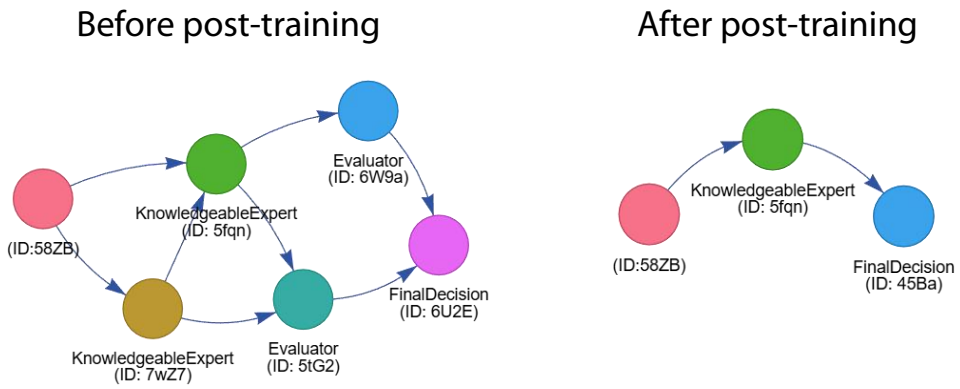


Figure 6: The planned MAS topology of RMAP before and after post-training for a query on MMLU.

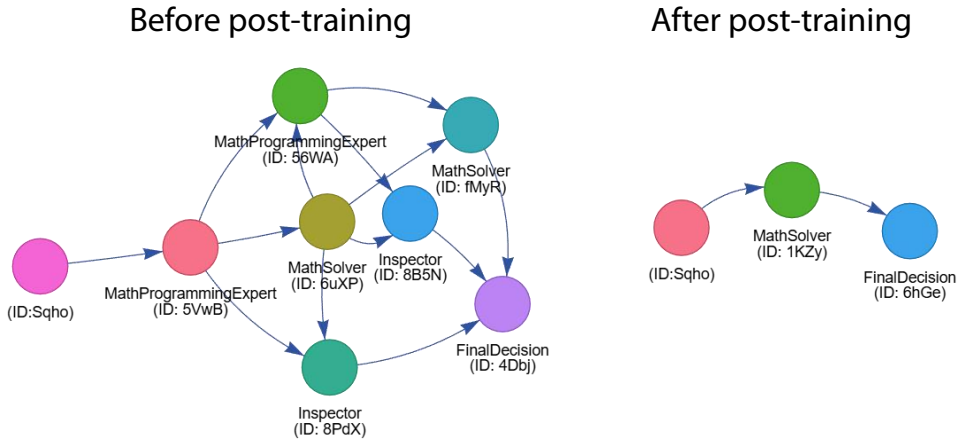


Figure 7: The planned MAS topology of RMAP before and after post-training for a query on SVAMP.

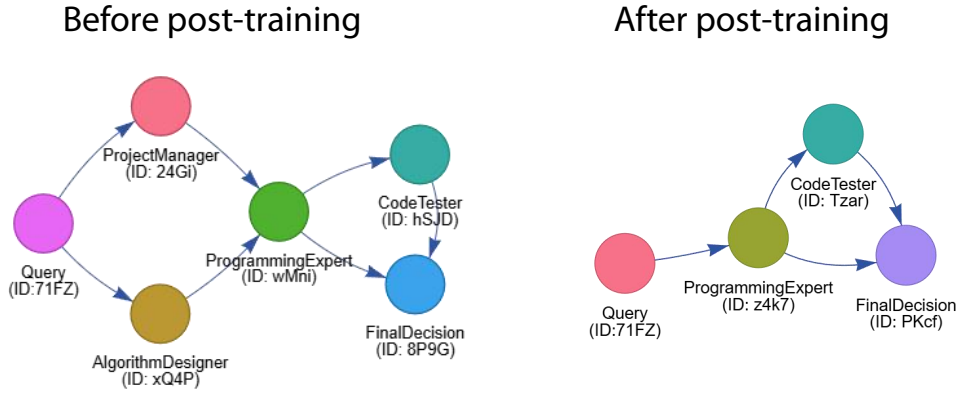


Figure 8: The planned MAS topology of RMAP before and after post-training for a query on HumanEval.

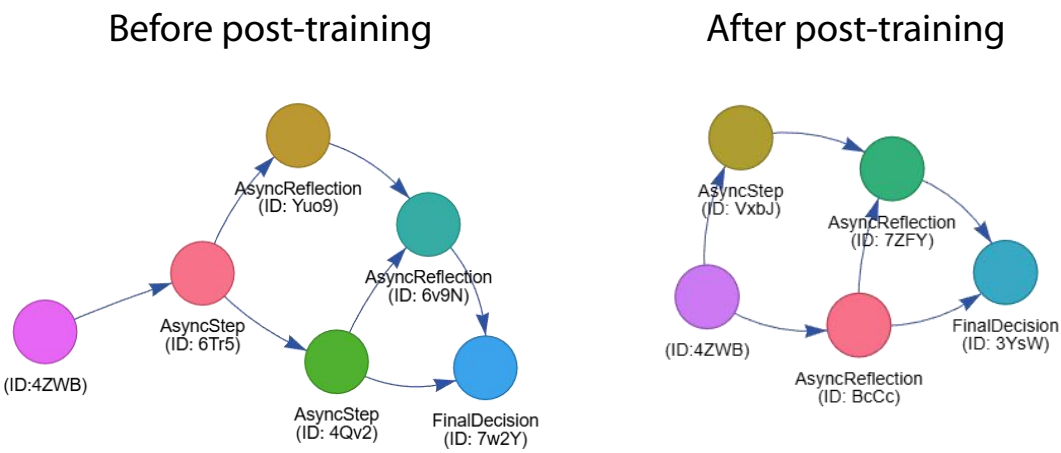


Figure 9: The planned MAS topology of RMAP before and after post-training for a query on AsyncHow.

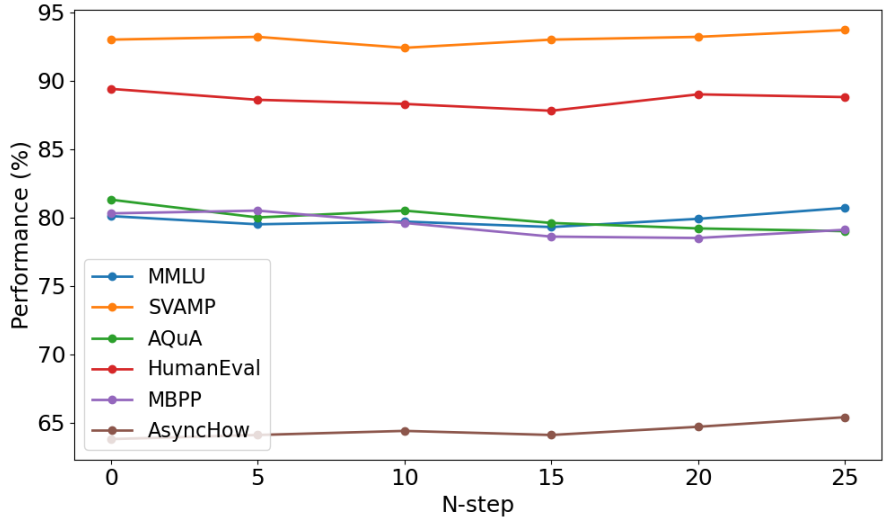


Figure 10: The performance (in %) as the number of post-training steps increases across tasks.

D.2 Report of Systematic Statistics

In this section, we report performance, time costs, token consumption, the number of edges, and the number of agents across tasks. The LLM backbone for agents is GPT-4o-mini. As shown in Figure 10 to Figure 14, RL-based post-training yields substantial efficiency gains for the planned MAS, while maintaining competitive performance relative to the pre-trained baseline.

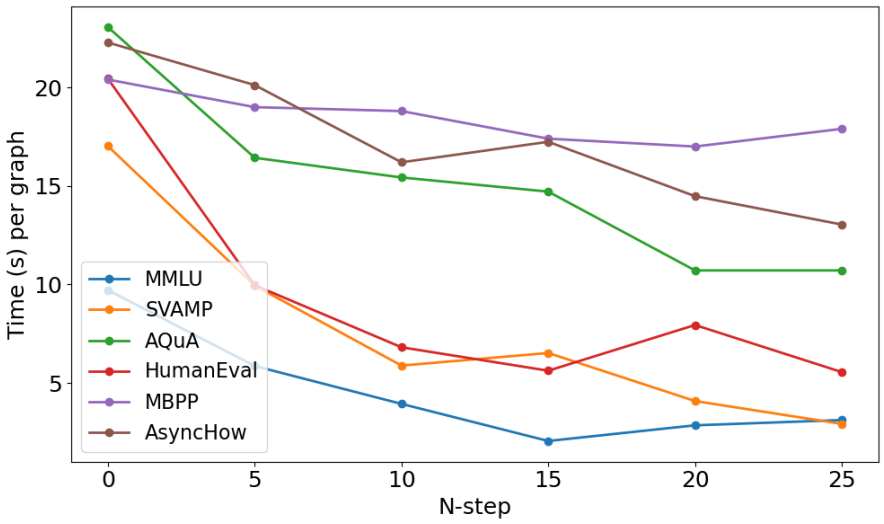


Figure 11: The average time cost (in seconds per graph) as the number of post-training steps increases across tasks.

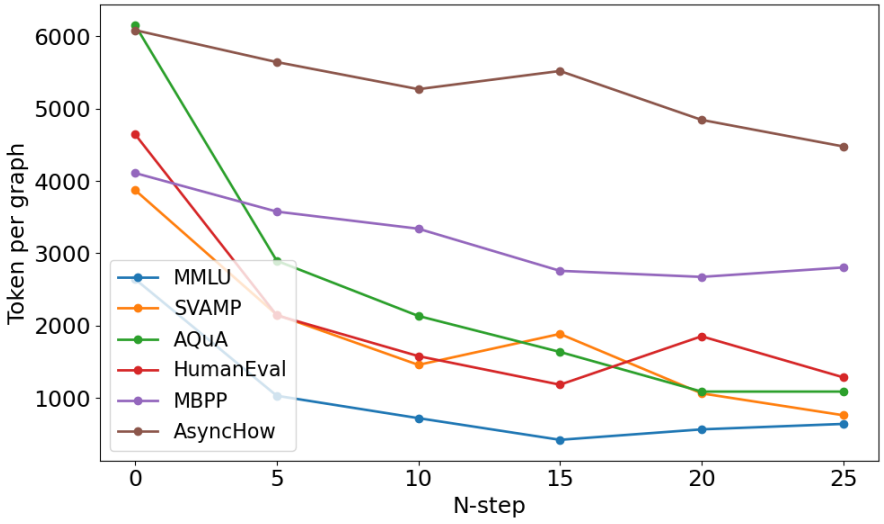


Figure 12: The average token usage (in tokens per graph) as the number of post-training steps increases across tasks.

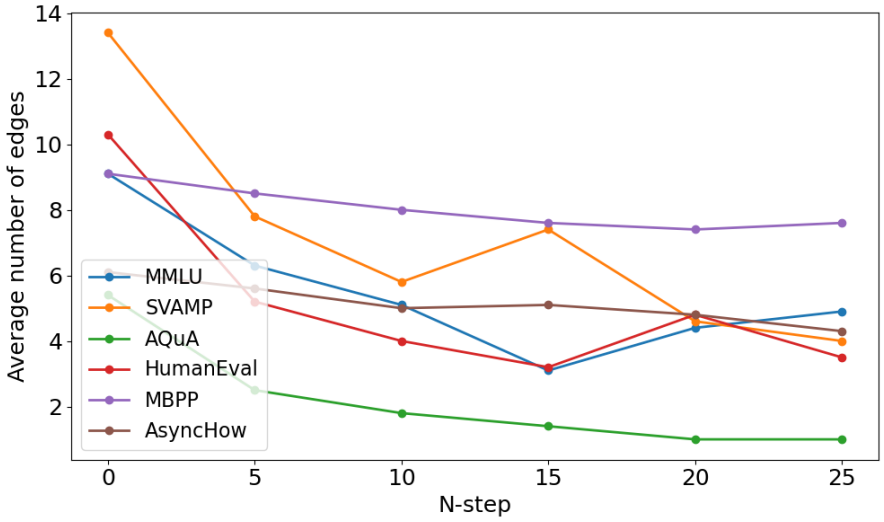


Figure 13: The average number of edges (in edges per graph) as the number of post-training steps increases across tasks.

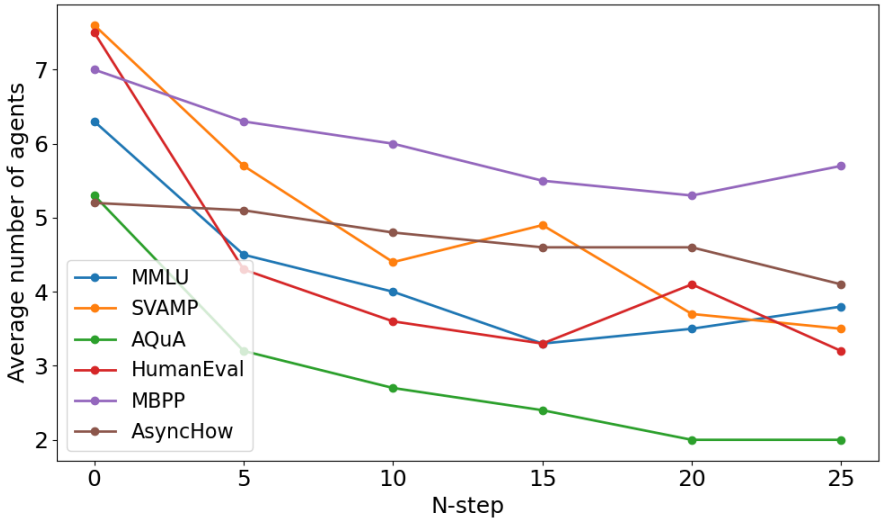


Figure 14: The average number of agents (in agents per graph) as the number of post-training steps increases across tasks.

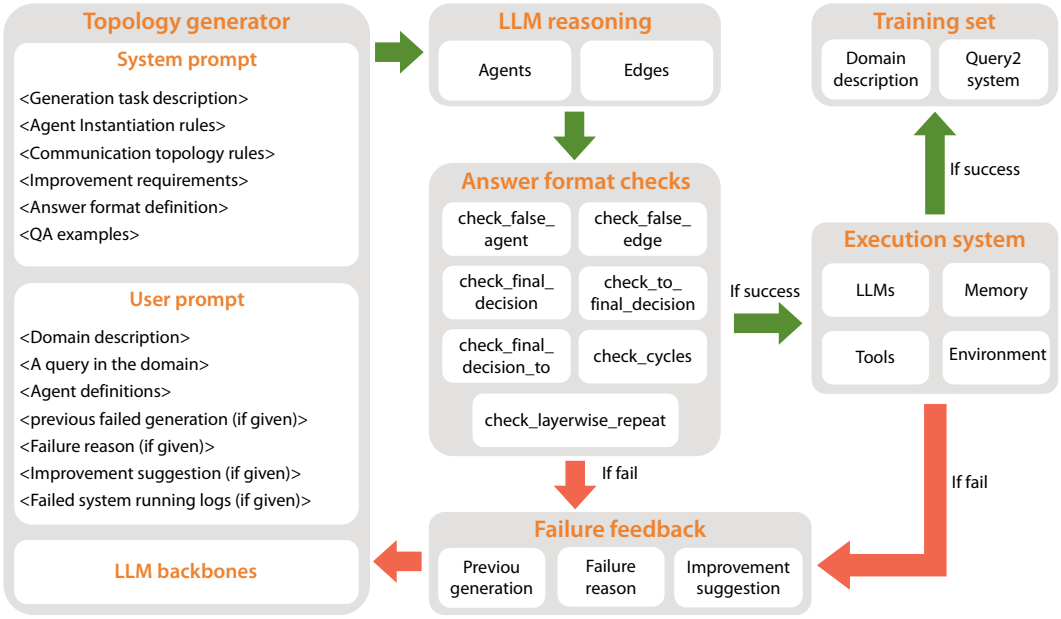


Figure 15: The structure of our data generator.

E Data Generator

E.1 Generation Workflow

Our data generator is inherently a ReAct Yao et al. (2022)-style agent. It generates a potential graph for a query, and the systematic environment evaluates the answer format and correctness. If the generated answer is correct, it will be added to the training set. In contrast, the generator will receive a failure report w.r.t. reasons, previous answers, and suggestions for the next generation until the budget is reached. The above description is shown in Figure 15. **The code achievement can be found at /rmap/generator in our anonymous repository.**

E.2 Analysis of LLM Choices for Generator

We compare various LLMs as the backbone for data generation, which is used for RMAP training. Subsequently, trained RMAP is evaluated across tasks and underlying agent LLMs. We report performance, time cost, token consumption, money cost on Figure 16 to Figure 19 when GPT-4o-mini is the underlying reasoning model. We report performance, time cost, token consumption, money cost on Figure 20 to Figure 23 when Gemini-3.1-flash is the underlying reasoning model. We report performance, time cost, token consumption, money cost on Figure 24 to Figure 27 when Llama-4-scout is the underlying reasoning model. In summary, the following conclusions can be drawn:

Widely used LLMs can generate effective training datasets. Specifically, no particular LLM provides a substantial advantage for RMAP in any specific scenario.

Generally, for data generation, Gemini-3.1-flash has considerable advantages in time efficiency. Moreover, it consistently produces high-quality datasets across tasks. Therefore, we adopt it as the default data generation model.

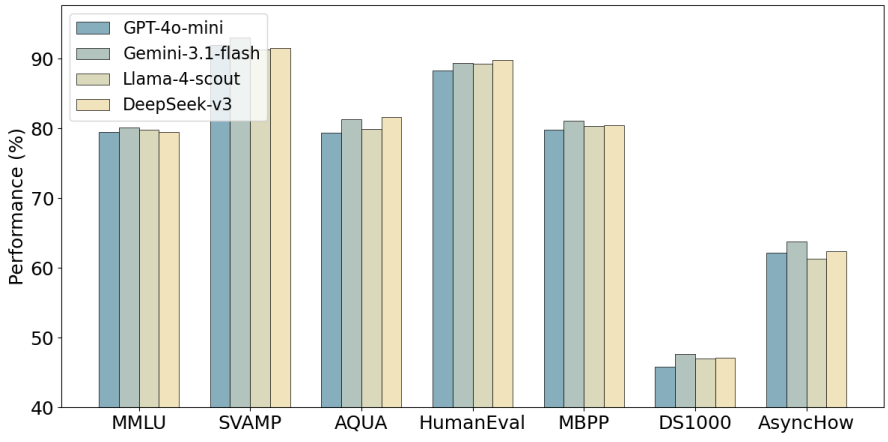


Figure 16: The performance (in %) comparison among LLM backbones for data generation. The LLM for agent reasoning is GPT-4o-mini.

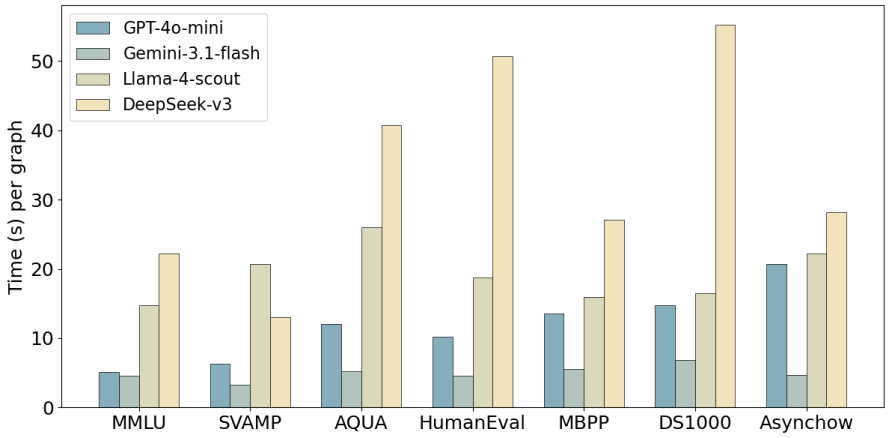


Figure 17: The average time cost comparison (in seconds per graph) among LLM backbones for data generation. The LLM for agent reasoning is GPT-4o-mini.

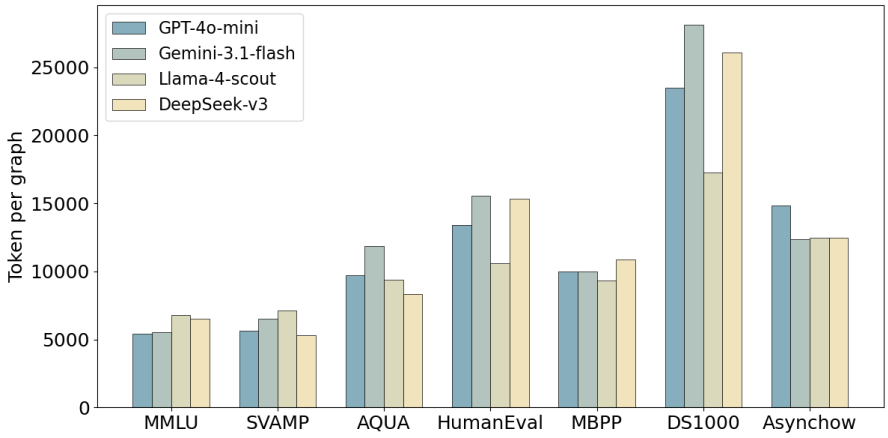


Figure 18: The average token usage comparison (in tokens per graph) among LLM backbones for data generation. The LLM for agent reasoning is GPT-4o-mini.

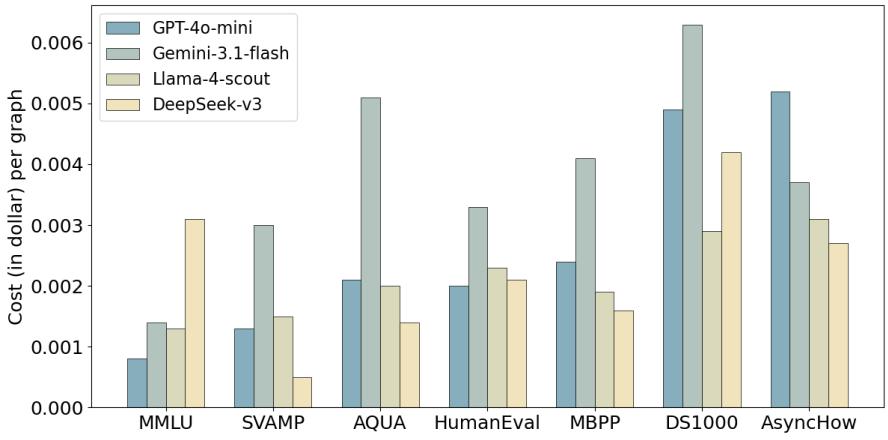


Figure 19: The average money cost comparison (in $ per graph) among LLM backbones for data generation. The LLM for agent reasoning is GPT-4o-mini.

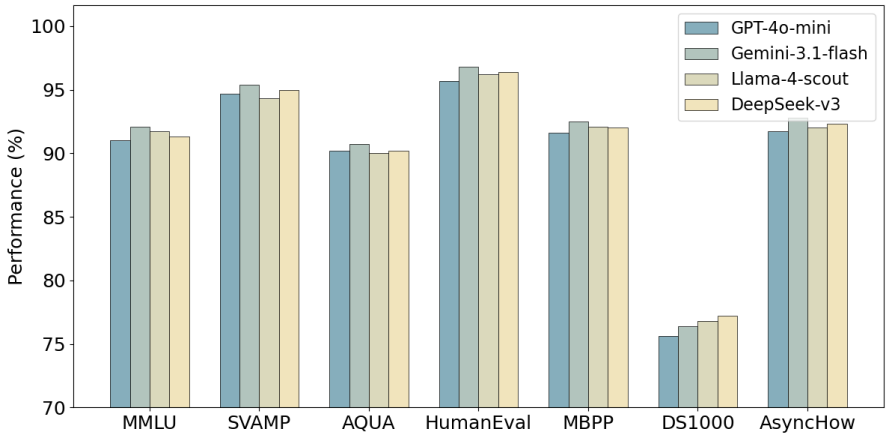


Figure 20: The performance (in %) comparison among LLM backbones for data generation. The LLM for agent reasoning is Gemini-3.1-flash.

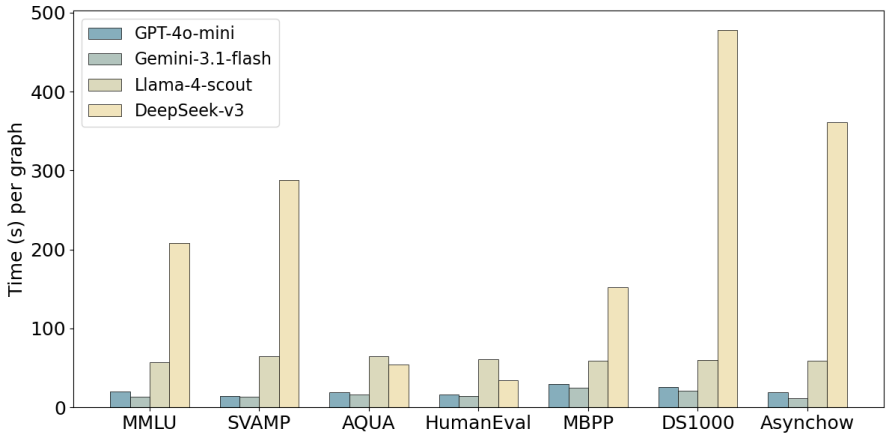


Figure 21: The average time cost comparison (in seconds per graph) among LLM backbones for data generation. The LLM for agent reasoning is Gemini-3.1-flash.

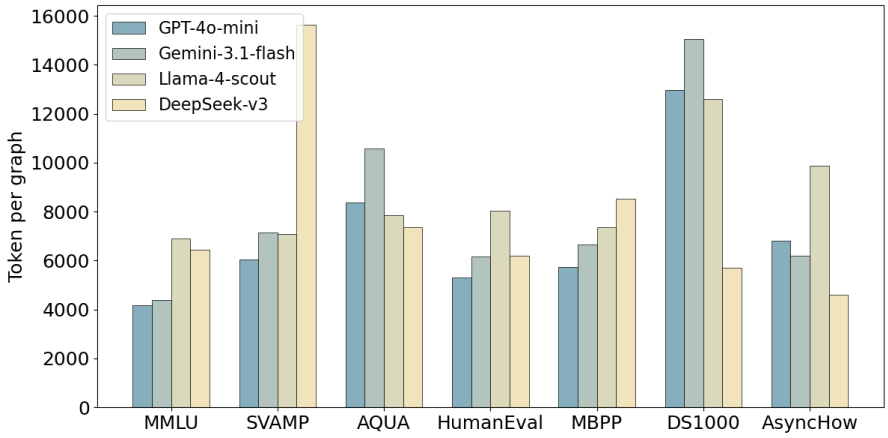


Figure 22: The average token usage comparison (in tokens per graph) among LLM backbones for data generation. The LLM for agent reasoning is Gemini-3.1-flash.

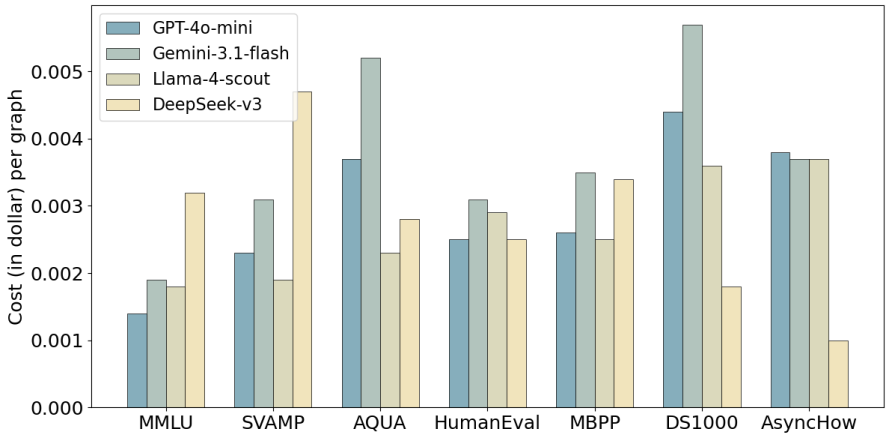


Figure 23: The average money cost comparison (in $ per graph) among LLM backbones for data generation. The LLM for agent reasoning is Gemini-3.1-flash.

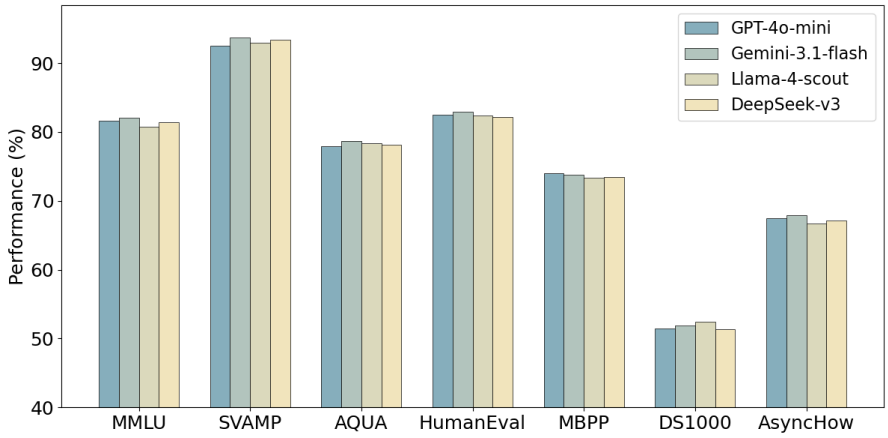


Figure 24: The performance (in %) comparison among LLM backbones for data generation. The LLM for agent reasoning is Llama-4-scout.

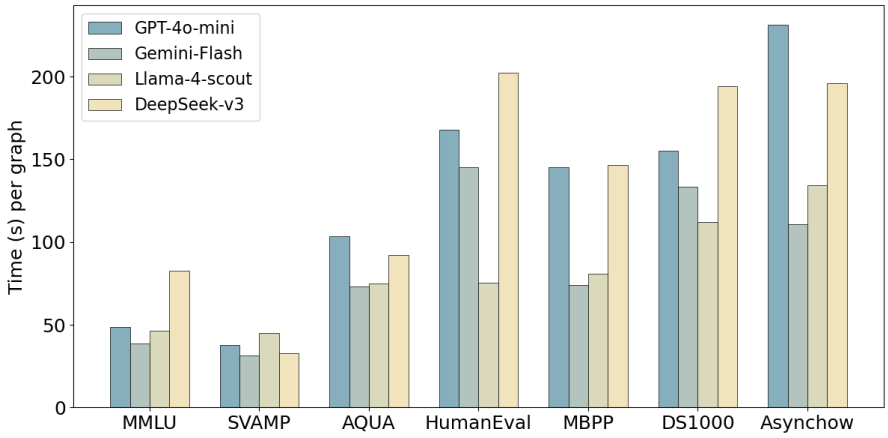


Figure 25: The average time cost comparison (in seconds per graph) among LLM backbones for data generation. The LLM for agent reasoning is Llama-4-scout.

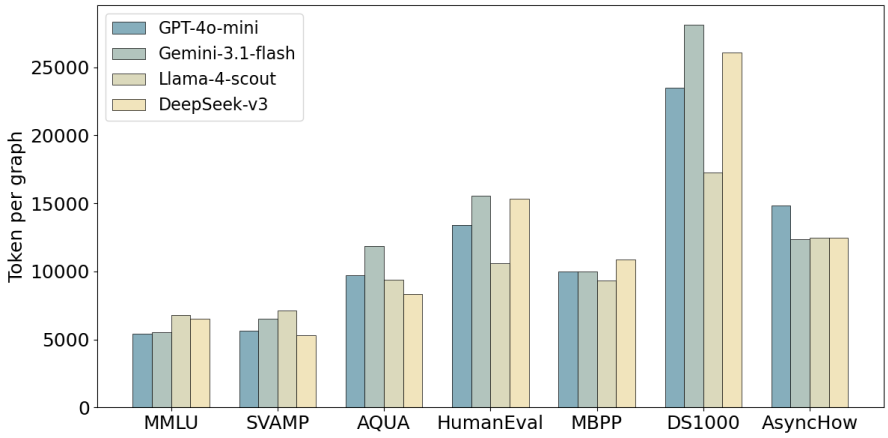


Figure 26: The average token usage comparison (in tokens per graph) among LLM backbones for data generation. The LLM for agent reasoning is Llama-4-scout.

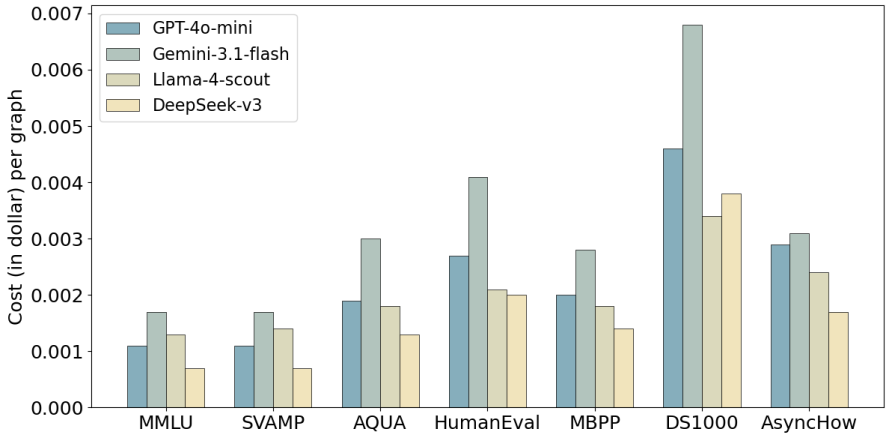


Figure 27: The average money cost comparison (in $ per graph) among LLM backbones for data generation. The LLM for agent reasoning is Llama-4-scout.

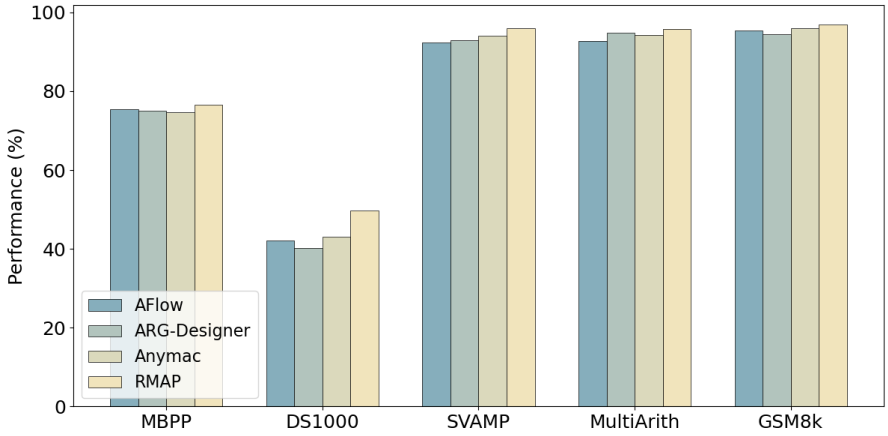


Figure 28: Performance (in %) comparison among designers after domain-specific transferring. The LLM backbone for agent reasoning is GPT-4o-mini.

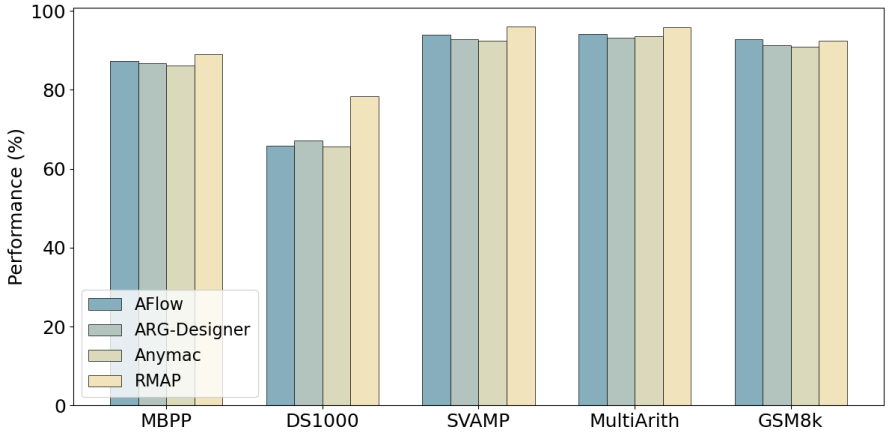


Figure 29: Performance (in %) comparison among designers after domain-specific transferring. The LLM backbone for agent reasoning is Gemini-3.1-flash.

F Analysis of Transferability

F.1 Transferability across datasets

To evaluate the transferability of RMAP across datasets, we train designers on "Dataset A" and then evaluate them on "Dataset B". In real-world applications, transfer learning is typically conducted within the same domain to maintain performance. Therefore, we also evaluate the transferability of RMAP and baselines within domains. Specifically, for code generation, we train designers on HumanEval and evaluate them on MBPP and DS1000. For math reasoning, we train designers on AQuA and evaluate them on SVAMP, MultiArith Roy & Roth (2015), and GSM8k Cobbe et al. (2021). The experimental results are shown in Figure 28, Figure 29, and Figure 30. In summary, RMAP outperforms SOTA baselines in all evaluated scenarios and exhibits strong transferability across different datasets.

F.2 Transferability across LLMs

To evaluate the transferability of RMAP across LLM backbones, we train designers on "LLM A" and then evaluate them on "LLM B" within a specific dataset. Specifically, we train designers on GPT-4o-mini and then evaluate them on other LLMs. The experimental results are shown in

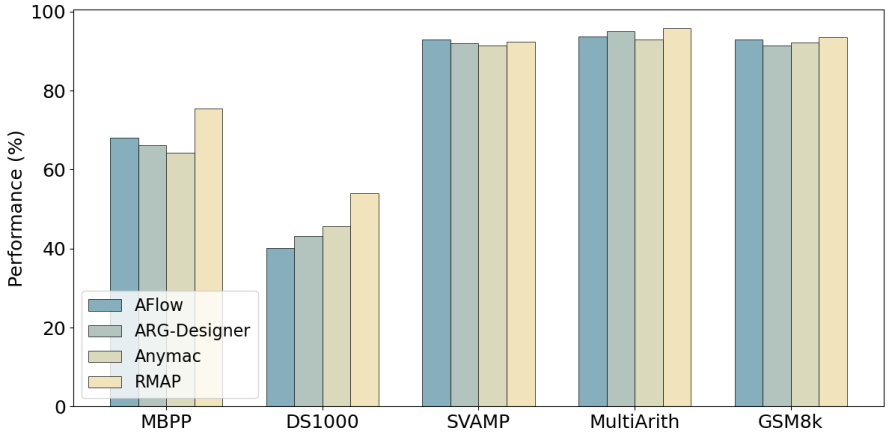


Figure 30: Performance (in %) comparison among designers after domain-specific transferring. The LLM backbone for agent reasoning is Llama-4-scout.

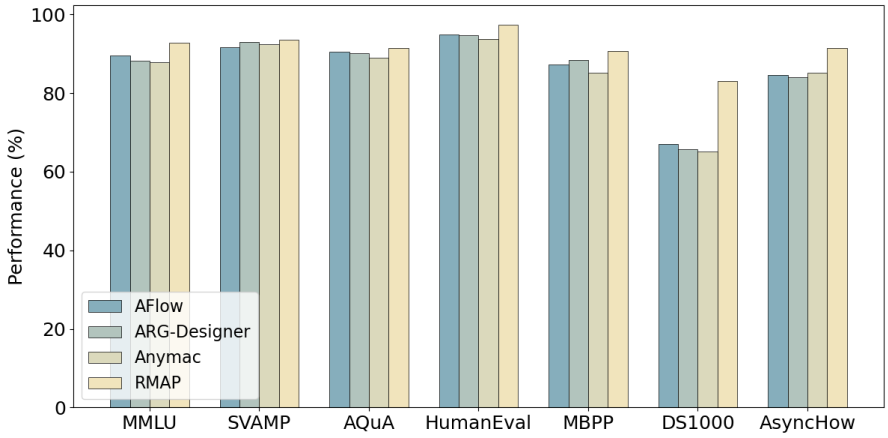


Figure 31: Performance (in %) comparison among designers after transferring from GPT-4o-mini to Gemini-3.1-flash.

Figure 31, Figure 32 and Figure 33. In summary, RMAP outperforms the SOTA baselines in 19 of the 21 evaluated scenarios. Notably, AFlow achieves the best performance in the remaining two scenarios. This is because AFlow adopts an LLM-driven search paradigm, whereas the other baselines primarily rely on parameterized agent contexts. Therefore, AFlow is less affected by context shifts across LLM backbones, resulting in more stable performance in certain scenarios.

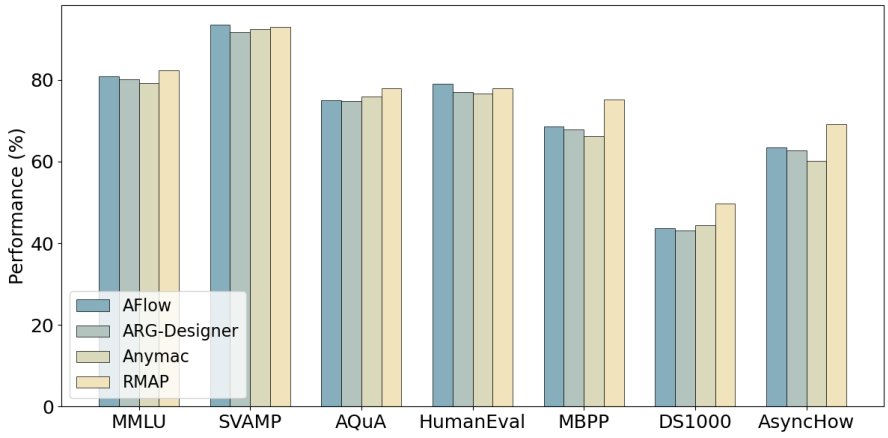


Figure 32: Performance (in %) comparison among designers after transferring from GPT-4o-mini to Llama-4-scout.

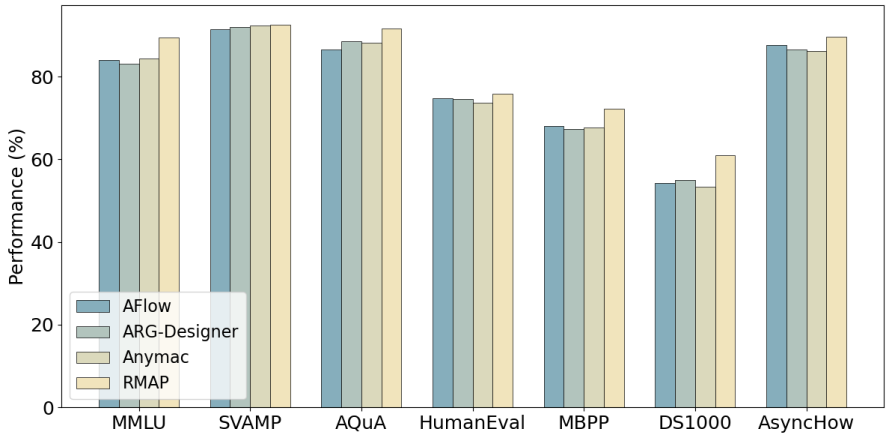


Figure 33: Performance (in %) comparison among designers after transferring from GPT-4o-mini to DeepSeek-v3.

G ALGORITHM

G.1 SUPERVISED PRE-TRAINING

Algorithm 1 Supervised Training of Agent and Link Predictors

Require: Agent encoder f_θ^{enc}, link predictor of agent generation p_θ^{agt}, link predictor of cross-layer link generation p_θ^{link}, the number of epochs e, batch size b, learning rate lr;

Ensure: Optimized agent encoder $f_{\theta'}^{enc}$, link predictor of agent generation $p_{\theta'}^{agt}$, link predictor of cross-layer link generation $p_{\theta'}^{link}$;

Convert the training graphs into embedding graphs;
Initialize optimizer $\mathcal{O}$ with learning rate lr;
for epoch=1 to e **do**
  for each batch $\mathcal{B}$ of size b **do**
    Initialize loss $\mathcal{L}_{agt} = 0$ and loss $\mathcal{L}_{link} = 0$;
    for each graph $G \in \mathcal{B}$ **do**
      Initialize the current agent representations from the query embedding;
      for each agent layer t **do**
        Predict agents in layer t based on Eq.7;
        Encode agents in layer t using f_θ^{enc} based on Eq.8;
        if $t > 1$ **then**
          Predict links from layers before $t-1$ to layer t using p_θ^{link} based on Eq.9;
          Encode agents in layer t using f_θ^{enc} based on Eq.10;
        end if
        Encode agents in layer t using f_θ^{enc} based on Eq.11;
      end for
      Predict links from agent layers to the final decision;
      Compute the prediction loss $\mathcal{L}_{agt}^g$ and $\mathcal{L}_{link}^g$ based on Eq.12;
      $\mathcal{L}_{agt} \leftarrow \mathcal{L}_{agt} + \mathcal{L}_{agt}^g$;
      $\mathcal{L}_{link} \leftarrow \mathcal{L}_{link} + \mathcal{L}_{link}^g$;
    end for
    $\mathcal{L}_{agt} \leftarrow \mathcal{L}_{agt}/N_{agt}$;
    $\mathcal{L}_{link} \leftarrow \mathcal{L}_{link}/N_{link}$;
    $\mathcal{L} \leftarrow \mathcal{L}_{agt} + \mathcal{L}_{link}$;
    $\mathcal{O}$.zero_grad();
    $\mathcal{L}$.backward();
    $\mathcal{O}$.step();
  end for
end for
return $f_{\theta'}^{enc}$, $p_{\theta'}^{agt}$, $p_{\theta'}^{link}$;

G.2 RL-BASED POST-TRAINING

Algorithm 2 RL-Based Post Training of Agent and Link Predictors

Require: Agent encoder f_θ^{enc}, link predictor of agent generation p_θ^{agt}, link predictor of cross-layer link generation p_θ^{link}, the number of steps s, group size g, the number of trajectories n, the execution system;

Ensure: Optimized agent encoder $f_{\theta'}^{enc}$, link predictor of agent generation $p_{\theta'}^{agt}$, link predictor of cross-layer link generation $p_{\theta'}^{link}$;

Initialize optimizer $\mathcal{O}$;
for step=1 to s **do**
  Sample a batch $\mathcal{B}$ of size g;
  for each record $r \in \mathcal{B}$ **do**
    Generate n trajectories $\{\tau_i\}_{i=1}^n$ based on Eq.13 and record their sampling masks;
    Evaluate each trajectory and obtain its accuracy, number of agents, and number of links;
    Compute trajectory rewards base on Eq.14;
    Normalize the trajectory rewards to obtain advantages $\{\hat{A}_i\}_{i=1}^n$;
    Record the log probabilities $\log \pi_{old}(\tau_i)$ as the old distribution;
  end for
  for each update **do**
    Re-evaluate the sampled trajectories using the recorded masks;
    Compute the clipped loss $\mathcal{L}_{clip}$;
    $\mathcal{O}.\text{zero_grad}()$;
    $\mathcal{L}_{clip}.\text{backward}()$;
    $\mathcal{O}.\text{step}()$;
  end for
end for
return $f_{\theta'}^{enc}, p_{\theta'}^{agt}, p_{\theta'}^{link}$;

G.3 INFERENCE

Algorithm 3 Inference of RMAP for MAS Workflow

Require: Optimized Agent encoder f_θ^{enc}, optimized link predictor of agent generation p_θ^{agt} and optimized link predictor of cross-layer link generation p_θ^{link}, the set of candidates $\hat{V}$, maximal length budget t_{max} and the execution system;

Ensure: Execution output z_o;

Initialize a new task and its agent, link, and ancestor information;
for $t = 1$ to $t_{max} - 1$ **do**
  Predict links from layer $t-1$ to $\hat{V}$;
  if no candidate is sampled **then**
    break;
  end if
  Predict the cross-layer links;
  Execute the current agent layer;
end for
Predict links from layer $t-1$ to the final decision o;
Predict the cross-layer links;
Execute the final decision layer;
return z_o;