AnyMAC: Cascading Flexible Multi-Agent Collaboration via Next-Agent Prediction

Anonymous ACL submission

Abstract

Recent progress in large language model (LLM)-based multi-agent collaboration highlights the power of structured communication in enabling collective intelligence. However, existing methods largely rely on static or graph-based inter-agent topologies, lacking the potential adaptability and flexibility in communication. In this work, we propose a new framework that rethinks multi-agent coordination through a sequential structure rather than a graph structure, offering a significantly larger topology space for multi-agent communication. Our method focuses on two key directions: (1) Next-Agent Prediction, which selects the most suitable agent role at each step, and (2) Next-Context Selection (NCS), which enables each agent to selectively access relevant information from any previous step. Together, these components construct task-adaptive communication pipelines that support both role flexibility and global information flow. Extensive evaluations across multiple benchmarks demonstrate that our approach achieves superior performance while substantially reducing communication overhead.

1 Introduction

The rise of large language models (LLMs) has revolutionized many domains by enabling powerful agents that can perform complex reasoning, planning, and action execution (Pan et al., 2023; Hong et al., 2023; Zhuge et al., 2024). These LLM-based agents, which integrate language generation with decision-making and external tool use, have demonstrated remarkable capabilities in diverse tasks, such as chain-of-thought reasoning (Yao et al., 2023b; Wang et al., 2023a) and code synthesis (Shinn et al., 2023; Chen et al., 2023). Beyond single-agent settings, recent work has shown that teams of LLM agents can collaboratively solve harder problems than any individual agent (Du et al., 2023; Wang et al., 2023b; Shinn et al., 2023; Zheng et al., 2023; Wu et al., 2023; Zhang et al., 2023), giving rise to an emergent form of collective intelligence. This emergent capability hinges critically on the design of inter-agent communication topologies: how agents are structured, how they exchange messages, and how they integrate information from others.

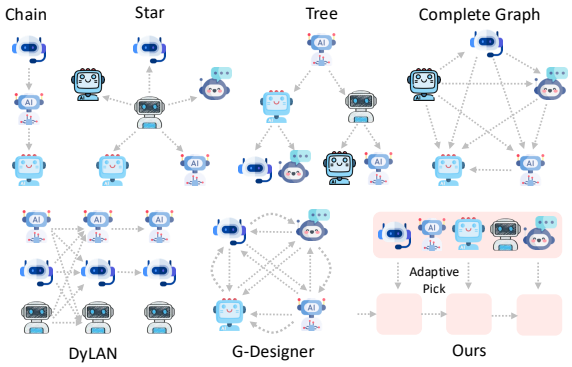


Fig. 1: Comparison of LLM-based multi-agent communication topology design.

To support such collaboration, researchers have investigated a wide range of multi-agent communication structures (Fig. 1), including chains (Wei et al., 2022; Zhang et al., 2022), trees (Yao et al., 2023a), stars (Wu et al., 2023), fully connected or random graphs (Qian et al., 2024), and learned or optimizable topologies (Zhuge et al., 2024; Zhang et al., 2024a). These designs, often tailored to task complexity or communication budgets, aim to balance performance and efficiency in various deployment scenarios. Notably, recent approaches have introduced learning-based topology construction (Hao et al., 2023; Liu et al., 2023; Zhang et al., 2024b), enabling dynamic selection of agent communication graphs conditioned on input tasks and queries. Such adaptive frameworks mark a shift from fixed pipelines to more flexible, input-aware systems that can better exploit the potential of LLM collectives.

Despite these advancements, current graph-based structures still face fundamental limitations. First, they enforce *static communication schemas* within each round: once the topology is learned, all agents operate under the same fixed communication pattern, preventing the reuse of agents or dynamic adaptation during the reasoning process. Additionally, to maintain acyclic message flow, many designs restrict the graph to be a Directed Acyclic Graph (DAG), which further constrains the solution space (of communication topology) and prohibits recursive or repeated consultation of specific agents. For example, in a task where one expert agent (e.g., a Python coder) is particularly useful at multiple stages of reasoning, a DAG-based structure cannot re-query this agent after it is used earlier in the round. This leads to inefficient or suboptimal reasoning, especially in complex tasks where revisiting agents is crucial. Second, most existing works limit information flow strictly to direct graph edges between agents, meaning that each agent can only access messages from its neighbors. For example, in tree-based structures, downstream agents often lack access to parallel branches' outputs, missing potentially useful contextual signals. This makes it hard for the agents to obtain global context for well-informed reasoning.

To address these challenges, we propose a new multi-agent collaboration framework, namely AnyMAC, that formulates multi-agent collaboration through a *sequential communication protocol* rather than a graph-based one. In this way, the construction of the communication topology is formulated as predicting the next agent iteratively. Our framework contains two novel and critical designs: (1) **Next-Agent Prediction**, where the system dynamically determines the next agent to activate in a stepwise manner. This sequential design bypasses the constraints of graph structures, allowing for greater flexibility in agent reuse and order variation across different queries. (2) **Next-Context Selection**, which allows each step to flexibly retrieve outputs from any previously activated agents. This globally accessible mechanism enables richer and more adaptive communication flows, where information is not constrained to propagate through fixed graph edges or sequential orders, but instead can be retrieved through dynamic selection based on task requirements. We conduct extensive experiments across multiple benchmarks, and the results validate the effectiveness of our approach, outperforming state-of-the-art communication topologies in both accuracy and efficiency in terms of token consumption. Our contributions can be summarized as follows:

- **Formulation.** We propose a new formulation of multi-agent communication, where the system predicts the next agent role and selects context from any previous agents. This formulation is proven to subsume the solution space of prior graph-based methods.
- **Framework.** We propose a transformer-based framework to realize our formulation, leveraging the transformer's global attention and sequential modeling capabilities.
- **Experiments.** We conduct extensive experiments across diverse benchmarks. Our method outperforms state-of-the-art multi-agent baselines in both accuracy and efficiency, demonstrating adaptivity, robustness, and favorable cost-performance trade-offs.

2 Related Work

Single Agent Reasoning. Recent research has demonstrated that multi-step reasoning allows large language models (LLMs) to solve complex problems and self-correct along the way. Broadly, single agent multi-step reasoning can be achieved via *training-based* and *prompting-based* methods.

In training-based approaches, reinforcement learning (RL) is used to optimize the model's ability to generate long-form Chain-of-Thought (CoT) reasoning (DeepSeek, 2025). While effective, RL methods typically require substantial data and computational resources. To reduce cost, distillation-based methods (Muennighoff et al., 2025; Ye et al., 2025) collect high-quality reasoning traces and apply supervised fine-tuning to teach models multi-step reasoning behaviors.

Beyond training-based methods, prompting-based techniques enable step-by-step reasoning by prompting procedure. Early approaches include multi-step reasoning exemplars directly in the prompt, as in CoT (Wei et al., 2022) and Automatic CoT (Zhang et al., 2022). Previous work also explicitly enforces multi-step reasoning in prompting procedure, such as ToT (Yao et al., 2023a) and budget-forcing (Muennighoff et al., 2025). Beyond single-agent reasoning, *multi-agent* approaches leverage collaboration among multiple LLMs to further improve accuracy, detailed below.

Multi-Agent Collaboration. Existing multi-agent systems typically predefine role types and fix the number of agents per role based on the task, then design a communication topology for collaboration. Prior work can be categorized by how this topology is generated. (1) Early approaches adopt *static* structures, such as chain (Qian et al., 2023; Hong et al., 2023; Holt et al., 2024), star (Wu et al., 2023; Yan et al., 2024; Zhou et al., 2023), and tree (Ishibashi and Nishimura, 2024), which remain unchanged across tasks. (2) To improve adaptability, recent methods have explored learning static communication graphs from data. GPTSwarm (Zhuge et al., 2024) parameterizes agent interactions using predefined Directed Acyclic Graph (DAG) topologies and optimizes them using reinforcement learning. However, the resulting structures remain fixed across the dataset and are input-independent, lacking the flexibility to adapt communication to individual task instances. (3) Recent efforts explore query-adaptive topology generation, such as DyLAN (Liu et al., 2023) and G-Designer (Zhang et al., 2024b), where agent interactions are dynamically constructed based on the input. While more flexible, they still rely on a manually defined number of agents and are constrained by canonical graph structures (i.e., the anchor structure). In contrast, our formulation allows the network to adaptively determine both the number of agents and the communication structure, without being restricted by a predefined topology. This flexibility enables exploration of a significantly larger topology space, leading to more effective and adaptive collaboration.

3 Problem Formulation

In this section, we define the key concepts for our sequential agent collaboration framework. Unlike prior works that formulate the multi-agent topology as a fixed directed acyclic graph (DAG), we represent the communication pipeline as a **sequence** $\mathcal{S} = [a_1, a_2, \ldots, a_T]$, where each element a_t is an LLM-based agent selected at the t-th step. This design allows agents to be reused multiple times and enables dynamic adjustment of the interaction order based on the task.

Each agent a_t is defined by:

$$a_t = \{\texttt{Base}_t, \texttt{Role}_t, \texttt{State}_t, \texttt{Tool}_t\}, \quad (1)$$

where $\texttt{Base}_t$ is the underlying language model instance, $\texttt{Role}_t$ indicates the agent's role, $\texttt{State}_t$ captures its memory and interaction history, and $\texttt{Tool}_t$ is an optional set of plugins (e.g., calculator, search engine, or file retriever).

Given an initial query $\mathcal{Q}$, the communication sequence unfolds over T steps. At each step t, the system predicts the next agent a_t and composes a prompt $\mathcal{P}^{(t)}$ containing both the original query and selected messages from previous steps:

$$\mathcal{P}_R^{(t)} = \texttt{Select}\left(\{\mathcal{O}^{(1)}, \ldots, \mathcal{O}^{(t-1)}\}\right), \quad (2)$$

where $\mathcal{O}^{(t-1)}$ denotes the response generated by agent a_{t-1}, and $\texttt{Select}(\cdot)$ is a learnable module that chooses relevant past outputs to include in the current prompt. This enables **flexible and global context access**, unlike graph-based models restricted to local neighborhoods.

Each agent executes based on its own system and user prompt:

$$\mathcal{O}^{(t)} = a_t\left(\mathcal{P}_{\text{sys}}^{(t)}, \mathcal{P}_{\text{usr}}^{(t)}, \mathcal{P}_R^{(t)}\right), \quad (3)$$

where $\mathcal{P}_{\text{sys}}^{(t)}$ includes $\texttt{Role}_t$ and $\texttt{State}_t$, and $\mathcal{P}_{\text{usr}}^{(t)}$ is the user prompt, which may include the query and task instructions from the user. After T steps, a final referee agent will aggregate the output and provide the final answer.

4 Methodology

As introduced in Section 3, we formulate the problem of LLM-based multi-agent collaboration as a sequential decision process. Our framework, AnyMAC, dynamically constructs a communication sequence $\mathcal{S} = [a_1, a_2, \ldots, a_T]$ by predicting, at each step, the next agent a_t and the relevant context to be passed as input. This formulation overcomes the rigidity of fixed graph topologies by allowing agent reuse and flexible context routing across steps.

Figure 2 illustrates the workflow of AnyMAC. Given a task query $\mathcal{Q}$, a set of candidate agent roles $\mathcal{R}$, and an optional tool set, our model iteratively builds the communication sequence. At each step t, it performs three stages: **Encoding**, **Prediction**, and **Execution**. In the *Encoding* stage, the $\mathcal{Q}$, $\mathcal{R}$, and historical conversation $\mathcal{H}_{t-1}$ are tokenized. These tokens are then fed into a Transformer to obtain contextual embeddings. In the *Prediction* stage, the contextual embeddings are used for the Next Agent Prediction (NAP) and Next Context Selection (NCS). In the *Execution* stage, we invoke the selected agent (LLM) with the chosen role and

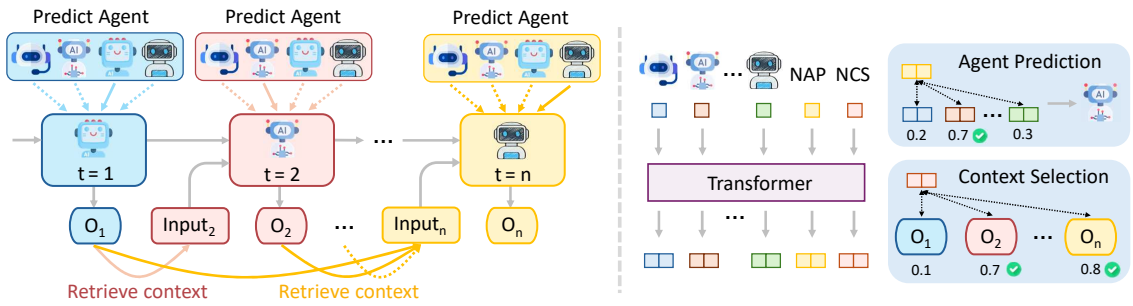


Fig. 2: The overview of our proposed framework ANYMAC. Left-hand side: At each time step, we perform two stages of operations: (1) Next-Agent Prediction (NAP), which aims to select the most suitable agent role from a set of candidate roles. (2) Next-Context Selection (NCS), which aims to retrieve useful context from the outputs of previously activated agents. The retrieved context will act as the input to the selected agent. Right-hand side: Given the embeddings of a series of activated agents, we perform contextual encoding using a transformer-based model to encode them with additional NAP and NCS tokens. The output embeddings of NAP and NCS tokens will be used to select the next agent and retrieve context from the next agent, respectively.

context to generate the response. The response $\mathcal{R}^{(t)}$ is appended to $\mathcal{H}_t$ and used in the next round until a final aggregation step produces the answer $a^{(T)}$.

4.1 Contextual Encoding

Semantic Tokenization. At time step t, given a task query $\mathcal{Q}$, a set of candidate role descriptions $\mathcal{R}_i, i = 1, 2, \ldots, N$, and conversation history $\mathcal{H}_{t-1}$ of previous steps, we begin by encoding these components into embeddings. Specifically, the task query $\mathcal{Q}$ contains textual instructions describing the question. Each agent role $\mathcal{R}_i$ includes a role prompt that instructs the agent to act in a specific role and provides an optional list of tools the agent can access. Each historical conversation in $\mathcal{H}$ corresponds to a previous agent, consisting of the role description of this agent and its associated response. Let $\texttt{Embed}(\cdot)$ denote an encoder function that outputs an embedding for any input text. Formally, the tokenization process is:

$$\mathbf{q} = \text{Embed}(\mathcal{Q}), \tag{4}$$

$$\mathbf{r}_i = \text{Embed}(\mathcal{R}_i), \quad i = 1, 2, \ldots, N \tag{5}$$

$$\mathbf{h}^{(j)} = \mathbf{r}^{(j)} \| \, \text{Embed}(\mathcal{O}^{(j)}), \quad j = 1, 2, \ldots, t-1 \tag{6}$$

Here, $\mathbf{q}$ is the embedding of the task query, $\mathbf{r}_i$ is the embedding of the i-th agent role description, and $\mathbf{h}_j$ is the embedding of the i-th historical conversation, obtained by concatenating the role and response embeddings of agent $a^{(j)}$ at step j. All embeddings are passed through a linear projection layer to match the transformer's input dimension. We use three separate projection layers: one for the task query, one for the role descriptions, and one shared across all historical conversations:

$$\tilde{\mathbf{q}} = f_q(\mathbf{q}), \quad \tilde{\mathbf{r}}_i = f_r(\mathbf{r}_i), \quad \tilde{\mathbf{h}}_j = f_h(\mathbf{h}_j), \tag{7}$$

where f_q, f_r, f_h are learnable linear projections. Moreover, to enable task-adaptive NAP and NCS, we generate the NAP and NCS tokens $\mathbf{t}_{\text{NAP}}$ and $\mathbf{t}_{\text{NCS}}$ using the task query embedding $\mathbf{q}$ by passing it through two separate linear layers:

$$\mathbf{t}_{\text{NAP}} = f_{\text{NAP}}(\mathbf{q}), \quad \mathbf{t}_{\text{NCS}} = f_{\text{NCS}}(\mathbf{q}), \tag{8}$$

where f_{NAP} and f_{NCS} are learnable linear projections that map the query embedding to the input dimension expected by the Transformer.

Contextual Encoding. After obtaining the token embeddings, we concatenate them into a sequence and feed it into an L-layer Transformer encoder to obtain contextualized embeddings $\mathbf{T}$:

$$\mathbf{T}^\star = \text{Trans}_L\left([\tilde{\mathbf{q}}, \tilde{\mathbf{r}}_{1:N}, \tilde{\mathbf{h}}_{1:t-1}, \mathbf{t}_{\text{NAP}}, \mathbf{t}_{\text{NCS}}]\right). \tag{9}$$

This encoding fuses information from task intent, role priors, and dialogue history, producing contextualized embeddings $\mathbf{T}^\star$ of the same length, which are used in the subsequent NAP and NCS.

4.2 Next-Agent Prediction (NAP)

Let N denote the total number of candidate roles. To determine the next agent role, we first compute pairwise compatibility scores between the contextualized NAP token $\mathbf{t}^\star_{\text{NAP}}$ and each contextualized role token $\mathbf{r}^\star_i$ using an inner product:

$$s_i = \langle \mathbf{t}^\star_{\text{NAP}}, \mathbf{r}^\star_i \rangle, \quad i = 1, \ldots, N. \tag{10}$$

we then select the role of the next agent (i.e., a_t) by considering the role $\mathcal{R}_k$ with the highest compatibility score s_i:

$$a_t = \mathcal{R}_k, \quad \text{where} \quad k = \underset{i=1,2,\ldots,N}{\arg\max}\, s_i. \tag{11}$$

During training, to encourage exploration, we apply the Gumbel-Softmax (Jang et al., 2016) over $\{s_i\}$.

4.3 Next-Context Selection (NCS)

To decide which parts of the historical conversation should be fed into the next agent as context, we compute cosine similarity scores between the contextualized NCS token $\mathbf{t}^\star_{\text{NCS}}$ and each contextualized history token $\mathbf{h}^\star_j$:

$$c_j = \frac{\langle \mathbf{t}^\star_{\text{NCS}}, \mathbf{h}^\star_j \rangle}{\|\mathbf{t}^\star_{\text{NCS}}\| \cdot \|\mathbf{h}^\star_j\|}, \quad c_j \in [-1, 1], \tag{12}$$

These scores are then passed through a sigmoid function, resulting in values that lie in $(0, 1)$:

$$g_j = \sigma(c_j) = \frac{1}{1 + e^{-c_j}}, \quad j = 1, 2, \ldots, t-1, \tag{13}$$

to obtain gate values g_j, which determine the inclusion probability for each context. At inference time, we select the context from previous agents by applying a threshold:

$$\mathcal{P}_O^{(t)} = \{(\mathcal{R}^{(j)}, \mathcal{O}^{(j)}) | g_j \geq \eta, j = 1, 2, \ldots, t-1\}, \tag{14}$$

where $\eta \in \mathbb{R}$ is a threshold used for selecting. During training, we sample Bernoulli masks with probabilities given by g_j to enable exploration.

Sparsity Penalty. Ideally, we would like the agent to access as much historical context as possible. However, excessively long context can lead to two issues: the agent may become overwhelmed by irrelevant information, and the API cost increases with longer context. To address this, we propose a sparsity penalty during training:

$$\mathcal{L}_{\text{sparse}} = \lambda \sum_{j=1}^{t-1} |g_j|. \tag{15}$$

Here, g_j is the gating score for the j-th history. By penalizing the magnitude of g_j, the model learns to selectively include only the most relevant context, avoiding unnecessary prompt tokens.

4.4 Execution

Once the next agent role and its associated context are determined, we perform *Execution*. We prompt the selected agent a_t with the system prompt $\mathcal{P}^{(t)}_{\text{sys}}$, the user prompt $\mathcal{P}^{(t)}_{\text{usr}}$, and selected context $\mathcal{P}^{(t)}_R$, to generate the response $\mathcal{O}^{(t)}$:

$$\mathcal{O}^{(t)} = a_t(\mathcal{P}^{(t)}_{\text{sys}}, \mathcal{P}^{(t)}_{\text{usr}}, \mathcal{P}^{(t)}_O). \tag{16}$$

The output is appended to the historical conversation buffer, enabling iterative reasoning and coordination in subsequent rounds. Formally, we update the conversation history as:

$$\mathcal{H}_t = \mathcal{H}_{t-1} \cup \{(a_t, \mathcal{O}^{(t)})\}, \tag{17}$$

where a_t denotes the selected agent based on NAP scores at time step t, and $\mathcal{O}^{(t)}$ is the generated response after executing the role a_t using the selected context $\mathcal{P}^{(t)}_O$. This process continues iteratively until a final decision agent is selected or the maximum number of iterations is reached.

4.5 Optimizing AnyMAC with RL

To improve routing quality with awareness of task difficulty and efficiency, we formulate NAP training as a reinforcement learning (RL) problem.

Efficiency-Aware Reward. Given a final aggregated answer a from a communication sequence with length l, we define the reward r as:

$$r = \gamma^l \cdot \mathbb{I}[\texttt{is\_correct}(a)], \tag{18}$$

where $\mathbb{I}[\cdot]$ is the indicator function that returns 1 if the answer a is correct, and 0 otherwise. The exponential decay term γ^l, where $\gamma \in (0, 1]$, penalizes longer routing paths. A smaller γ places greater emphasis on efficiency by punishing large l more.

Difficulty-Aware Advantage Estimation. Since questions vary in difficulty, directly comparing rewards across different queries is misleading. This is because a high reward may result from an easy question rather than a good routing policy. To address this, we apply a standard reward normalization technique (Gu et al., 2016; Schulman et al., 2017), which takes the average reward of a question into account and compute advantage A_t of each trajectory, as described in Appendix A.2. The policy gradient loss is then computed as follows:

$$\nabla_\Theta \mathcal{L}_{\text{PG}} = -\sum_{t=1}^{T} A_t \nabla_\Theta \log P_\Theta(\tau_t), \tag{19}$$

where $P_\Theta(\tau_t)$ denotes the probability of the t-th sampled trajectory under the current model parameters Θ. This includes both the NAP selection probability (Section 4.2) obtained via Gumbel-Softmax, and the NCS selection probability (Section 4.3) derived from the sigmoid gating function. The full objective combines the policy gradient and the sparsity regularization term (Section 4.3):

$$\mathcal{L} = \mathcal{L}_{\text{PG}} + \lambda \mathcal{L}_{\text{sparse}}, \tag{20}$$

where λ is a hyper-parameter to control the importance of the sparsity regularization term.

5 Experiments

5.1 Experimental Setup

Benchmarks & Tasks. We evaluate AnyMAC across a diverse suite of tasks spanning general reasoning, mathematical problem solving, and code generation. Specifically, we use `MMLU` (Hendrycks et al., 2021) for general reasoning; `GSM8K` (Cobbe et al., 2021), `MultiArith` (Roy and Roth, 2016), `SVAMP` (Patel et al., 2021), and `AQuA` (Ling et al., 2017) for math reasoning; and `HumanEval` (Chen et al., 2021) for code generation. We follow the standard train/test splits provided by each dataset.

Variants. We provide two variants of our method: AnyMAC and AnyMAC-Eff The first one disables all efficiency-related constraints by setting the sparsity loss weight $\lambda = 0$, the reward decay factor $\gamma = 1$, allowing the model to focus solely on accuracy. In contrast, AnyMAC-Eff sets $\lambda = 1\text{e}{-3}$, $\gamma = 0.9$, and constrains the context selection to be within the newest 2 responses, encouraging shorter routing trajectories (i.e., fewer agents) and compact context selection.

Baselines. To ensure a comprehensive comparison of various methods, we compare AnyMAC against both single-agent prompting and multi-agent collaboration frameworks. Single-agent baselines include COT (Chain-of-Thought) (Wei et al., 2022), ComplexCoT (Fu et al., 2022), Self-Consistency (Wang et al., 2023a), and PHP (Zheng et al., 2023). Multi-agent baselines include topology-based methods, including Chain, Star, Tree (Qian et al., 2024), Complete Graph, and Random Graph. For learning-based frameworks, we consider AutoGen (Wu et al., 2023), LLM-Debate (Du et al., 2023), DyLAN (Liu et al., 2023), GPTSwarm (Zhuge et al., 2024), and G-Designer (Zhang et al., 2024b).

Implementation. We implement all agents using the OpenAI model, `gpt-4-1106-preview`. For all single-agent and multi-agent baselines, we follow the official configurations used in G-Designer (Zhang et al., 2024b)[1], which adopt a temperature of 0 for single-agent and 1 for multi-agent methods. For sentence embeddings, we use `all-MiniLM-L6-v2` (Wang et al., 2020) as $\texttt{Embed}(\cdot)$ to encode queries, role descriptions, and historical responses into 384-dimensional vectors.

We also make specific design choices in AnyMAC. We use a default temperature of 1, except on multiple-choice benchmarks (`MMLU` and `AQuA`), where we set the temperature to 0 as we find it helps reduce hallucinations. The maximum number of agents is fixed to 5 across all tasks. Once this limit is reached, the next agent role is forced to be the decision model, which produces the final answer and terminates the reasoning process.

We use `GPT-2 Small` (Radford et al., 2019) as the routing model and initialize it with pretrained weights. The computational overhead of the routing model is negligible compared to LLMs: it is lightweight (117M parameters) and only predicts two tokens per step (NAP and NCS). In contrast, LLMs are over 1000× larger (e.g., `GPT-3` has 175B parameters (Brown et al., 2020)) and generate hundreds of tokens per response. To train the model, we sample 80 questions from each dataset and collect 1000 routing trajectories along with their corresponding rewards.

Training questions selection. If a dataset provides a train/test split, we sample 80 task instances from the training set; otherwise, we use the first 80 tasks in test set. The model is optimized separately for each dataset.

Adaptive Sampling. We collect 1000 trajectory samples (attempted answers) per dataset. To make effective use of the training sample budget, we design an adaptive sampling strategy that allocates more samples to difficult questions. Specifically, when iterating through all questions in a dataset, we do not set a fixed number of trials per question. Instead, we define a required number of correct answers for each question before moving on. This strategy naturally allocates more sampling budget to harder examples, as easy questions tend to reach the threshold with fewer samples, while difficult ones require more. After each

[1] Since the official implementations of baselines are not fully open-source, we use results provided in G-Designer.

Table 1: Accuracy comparison across single-agent and multi-agent baselines. The best results are shown in **bold**, and the runner-ups are underlined. We highlight the accuracy gain (+ / −) relative to the vanilla baseline.

Method	MMLU	GSM8K	MultiArith	SVAMP	AQuA	HumanEval	Avg.
Vanilla	82.14	85.40	93.15	87.18	70.34	71.68	81.65
CoT	82.65 (+0.51)	87.17 (+1.77)	94.79 (+1.64)	88.32 (+1.14)	73.91 (+3.57)	75.52 (+3.84)	83.73
ComplexCoT	83.78 (+1.64)	87.62 (+2.22)	95.86 (+2.71)	90.17 (+2.99)	77.58 (+7.24)	74.94 (+3.26)	84.99
SC (CoT)	82.66 (+0.52)	87.93 (+2.53)	96.88 (+3.73)	88.69 (+1.51)	75.08 (+4.74)	77.30 (+5.62)	84.75
SC (Complex)	83.65 (+1.51)	86.14 (−0.74)	96.94 (+3.79)	89.72 (+2.54)	77.69 (+7.35)	77.94 (+6.26)	85.35
PHP	83.45 (+1.31)	95.50 (+10.10)	98.10 (+2.84)	90.02 (+3.44)	79.00 (+8.66)	82.96 (+11.36)	88.17
Chain	82.35 (+0.21)	85.57 (+0.17)	94.38 (+1.23)	83.41 (−3.77)	70.94 (+0.60)	80.88 (+9.20)	82.92
Star	80.79 (−1.35)	85.55 (+0.15)	93.79 (−0.64)	88.09 (+0.91)	68.57 (−1.77)	75.65 (+3.97)	82.07
Tree	81.89 (−0.25)	84.56 (−0.84)	94.60 (+1.45)	89.25 (+2.07)	72.84 (+2.50)	77.38 (+5.70)	83.42
Complete Graph	83.15 (+1.01)	86.49 (+1.09)	97.20 (+4.05)	89.48 (+2.30)	79.21 (+8.87)	83.75 (+12.07)	86.55
Random Graph	83.76 (+1.62)	86.14 (+0.74)	95.46 (+2.31)	85.41 (−1.77)	74.07 (+3.73)	82.66 (+10.98)	84.58
AutoGen	82.13 (−0.01)	90.06 (+7.92)	93.80 (+0.65)	88.44 (−1.26)	73.65 (+3.31)	85.41 (+13.73)	85.58
LLM-Debate	83.69 (+1.55)	90.23 (+4.83)	96.27 (+3.12)	90.56 (+3.38)	77.52 (+7.18)	83.79 (+12.11)	87.01
DyLAN	80.16 (−1.98)	88.16 (+2.76)	94.27 (+1.12)	87.40 (+0.22)	74.16 (+3.82)	89.70 (+18.02)	85.64
GPTSwarm	83.98 (+1.84)	89.74 (+4.34)	97.84 (+4.69)	86.42 (−0.76)	78.16 (+7.82)	88.49 (+16.81)	87.32
G-Designer	**84.50** (+2.36)	95.07 (+9.67)	98.30 (+5.15)	91.85 (+4.67)	79.47 (+9.13)	89.90 (+18.22)	89.84
AnyMAC	84.30 (+2.16)	**95.66** (+10.26)	**99.44** (+6.29)	**92.67** (+5.49)	**81.50** (+11.16)	**90.12** (+18.44)	**90.62**

epoch, if there is remaining trajectory budget, we proceed to the next epoch and repeat the process until the total sampling budget is exhausted. For AnyMAC, we set the threshold to 1 correct answer; for AnyMAC-Eff, we increase it to 4 to encourage more stable training.

5.2 Comparative Results

AnyMAC outperforms other baselines. Table 1 compares the accuracy of AnyMAC against a range of baselines. Notably, AnyMAC achieves the highest average accuracy across all benchmarks, outperforming both single-agent and multi-agent methods. Specifically, it achieves state-of-the-art performance on `GSM8K`, `MultiArith`, `SVAMP`, `AQuA`, and `HumanEval`, and ranks as the runner-up on `MMLU`.

The superior performance of AnyMAC stems from multiple aspects. First, compared to single-agent methods, multi-agent collaboration allows agents to check and correct each other's reasoning, leading to higher accuracy. Second, compared to multi-agent methods with fixed routing structures, AnyMAC generates task-specific multi-agent reasoning trajectories, enabling more adaptive and effective collaboration. Finally, compared to other learning-based routers, AnyMAC is not constrained by manually defined anchor structures in G-Designer and fixed role distributions in DyLAN and GPTSwarm, offering greater flexibility and a larger routing solution space to explore optimal communication sequences.

5.3 Robustness Evaluation

AnyMAC is robust to malicious agents. An important advantage of learning-based multi-agent methods like AnyMAC is their robustness to malicious agents. Through training, the router can learn to identify and downweight or bypass agents that provide misleading or harmful responses. To evaluate this, we conduct an experiment by injecting a malicious agent into the candidate agent pool. This agent is intentionally designed to produce consistently incorrect or distracting content.

Figure 3 shows the robustness evaluation results on `MMLU` dataset. We observe that other methods exhibit a significant accuracy drop (up to -11.0%) when a malicious agent is present. In contrast, AnyMAC maintains high accuracy (-1.3%), highlighting its robustness under adversarial conditions. We also observe that other learning-based baselines such as GPTSwarm and G-Designer exhibit similar robustness, indicating that learned routing policies generally confer a degree of resilience against adversarial inputs.

5.4 Efficiency Evaluation

AnyMAC is efficient. Beyond accuracy, token consumption is also a critical factor that affects practicality, as it reflects the actual cost incurred by users when solving tasks. Figure 4 compares the trade-off between prompt token usage and accuracy across different methods on the `GSM8K` dataset. Notably, AnyMAC achieves the highest accuracy but with relatively higher token usage. In contrast,

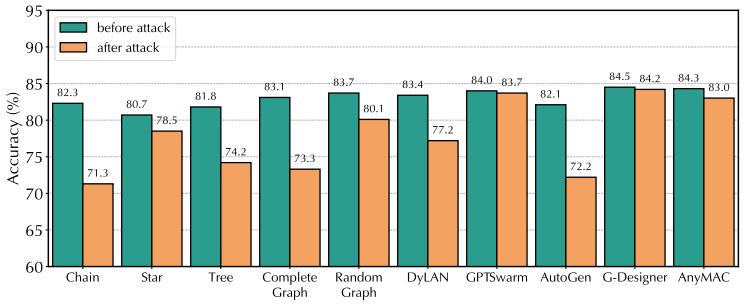


Fig. 3: Robustness analysis of different methods on the MMLU dataset. We compare accuracy *before* and *after* the attack. Learning-based methods exhibit strong resilience to malicious agents.

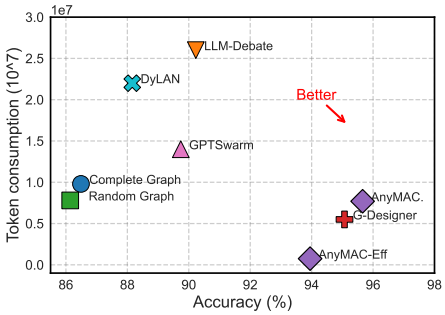


Fig. 4: The performance and token consumption of various multi-agent communication topologies on GSM8K.

ANYMAC-EFF sacrifices a small amount of accuracy in exchange for significantly improved efficiency, achieving the lowest token consumption among all methods. Compared to the most efficient baseline G-DESIGNER, ANYMAC-EFF reduces prompt token usage by 5×, while maintaining competitive performance. This demonstrates that our method is highly flexible: it can be configured to prioritize either accuracy or efficiency, and achieves state-of-the-art performance in both aspects.

5.5 Qualitative Case Studies

We further conduct qualitative analyses to evaluate the adaptability and behavior of ANYMAC. First, we investigate whether ANYMAC can adapt to different types of questions using the MMLU dataset. Figure 5 in Appendix shows the routing results of ANYMAC on two different questions. We observe that ANYMAC selects different roles, showing its ability to adapt based on the question.

Furthermore, we also compare ANYMAC and ANYMAC-EFF using the same question from GSM8K to assess the trade-off between accuracy and efficiency. Figure 6 in Appendix shows that ANYMAC selects more context and produces the correct answer, while ANYMAC-EFF uses less context and fails. This shows that more computation can lead to improved accuracy.

5.6 Ablation Studies

In this subsection, we perform ablations on ANYMAC and conduct experiments on the GSM8K dataset to evaluate the effectiveness of each module in our design. We consider the following variants: (1) Removing task adaptiveness. We replace the query embedding with a single learnable vector shared across all queries. (2) Removing the Next-Agent Prediction (NAP) module. We replace the learned agent selection with a random choice at each step. (3) Removing the Next-Context Selection (NCS) module. We replace the context selection mechanism with random sampling from the context pool with 50% probability for each response. The results are shown in Table 2. We observe that removing any of these components leads to an accuracy drop, highlighting their importance.

Table 2: Ablation study on ANYMAC-ACC evaluated on the GSM8K dataset.

Variant	Accuracy (%)
ANYMAC (Full Model)	95.66
w/o Task Embed. (Task Adaptive)	93.87
w/o Next-Agent Prediction (NAP)	94.75
w/o Next-Context Selection (NCS)	94.35

6 Conclusion

In this work, we investigate multi-agent collaboration through the lens of communication topology design. We propose a novel sequential formulation that dynamically constructs communication topologies by predicting the next agent at each step. Our framework introduces two key components: Next-Agent Prediction for flexible, task-aware role allocation, and Next-Context Selection for globally informed information routing. Together, these components enable a broader solution space than traditional graph structures, supporting adaptive, efficient, and robust multi-agent reasoning. Extensive experiments across multiple benchmarks show that our approach consistently outperforms existing methods in both accuracy and efficiency. We believe this sequential communication protocol opens new directions for future research in scalable and generalizable multi-agent LLM systems.

7 Limitations

While ANYMAC achieves the highest overall accuracy, its performance on the `HumanEval`, a code generation dataset, is slightly below the state-of-the-art as shown in Table 1. After investigating the reason, we find that ANYMAC tends to select overly long contexts, which may overwhelm the LLM, leading to incorrect code generation.

We believe the above issue is caused by an insufficient number of training samples (1,000) for reinforcement learning (RL), which may lead to convergence to a suboptimal solution. We hypothesize that this can be mitigated by scaling up RL training. However, scaling RL sampling data for LLMs is both computationally and financially expensive in practice. Therefore, we leave this as future work and plan to explore it using smaller language models, which offer significantly lower data collection costs. Moreover, smaller language models may benefit more from multi-agent collaboration due to their weaker individual capabilities.

8 Ethical Consideration

This work builds upon large language models (LLMs) for multi-agent collaboration and reasoning. All models and datasets used in our experiments are publicly available and widely adopted in the research community. We do not introduce any new data collection, nor do we engage in sensitive information. During our experiments, we did not observe any explicit ethical concerns, harmful behaviors, or misuse cases. Nevertheless, we acknowledge that LLM-based systems, especially when deployed in multi-agent configurations, can potentially amplify biases or generate misleading information if not properly controlled. Our method focuses on improving multi-agent communication effectiveness and efficiency and does not alter the core generative behavior of the underlying LLMs.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2023. Codet: Code generation with generated tests. In *The Eleventh International Conference on Learning Representations*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv prepring*, abs/2110.14168.

DeepSeek. 2025. Deepseek r1: Towards deep reinforcement learning for language models. *arXiv preprint arXiv:2501.12948*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *CoRR*, abs/2305.14325.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2022. Complexity-based prompting for multi-step reasoning. In *The Eleventh International Conference on Learning Representations*.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. 2016. Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pages 2829–2838. PMLR.

Rui Hao, Linmei Hu, Weijian Qi, Qingliu Wu, Yirui Zhang, and Liqiang Nie. 2023. Chatllm network: More brains, more intelligence.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Samuel Holt, Max Ruiz Luyten, and Mihaela van der Schaar. 2024. L2mac: Large language model automatic computer for extensive code generation. In *The Twelfth International Conference on Learning Representations*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. 2023. Metagpt: Meta programming for multi-agent collaborative framework.

Yoichi Ishibashi and Yoshimasa Nishimura. 2024. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *CoRR*, abs/2310.02170.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. Work in Progress.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. 25 pages, 9 figures, 2 tables.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2024. Scaling large-language-model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

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

Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint*, abs/2303.11366.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023a. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2023b. Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. Work in progress.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework.

Yikuan Yan, Yaolun Zhang, and Keman Huang. 2024. Depending on yourself when you should: Mentoring llm with rl agents to become the master in cybersecurity games. *arXiv preprint arXiv:2403.17674*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models.

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

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2024a. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. *arXiv preprint arXiv:2410.02506*.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. 2024b. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*.

Jintian Zhang, Xin Xu, and Shumin Deng. 2023. Exploring collaboration mechanisms for llm agents: A social psychology view. *arXiv preprint arXiv:2310.02124*.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. Progressive-hint prompting improves reasoning in large language models. Tech Report.

Zihao Zhou, Bin Hu, Chenyang Zhao, Pu Zhang, and Bin Liu. 2023. Large language model as a policy teacher for training reinforcement learning agents. *arXiv preprint arXiv:2311.13373*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.

A RL Training Details

A.1 Exploration and Exploitation

An important challenge in reinforcement learning is balancing the trade-off between *Exploration* and *Exploitation*. Exploration refers to discovering new possibilities by sampling uncertain actions, while exploitation focuses on selecting the best-known decision based on current knowledge.

In our framework, we encourage exploration by injecting noise into the output decision logits during training. By tuning the relative magnitude of the noise and decision score, we can effectively balance exploration and exploitation.

Next agent prediction (NAP). For NAP, it is essential to balance the compatibility scores s_i and the injected Gumbel noise (Section 4.2). To achieve this, we first compute the mean and standard deviation of compatibility scores s_i across N roles:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} s_i', \quad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(s_i' - \mu)^2}. \tag{21}$$

We then normalize the scores to have zero mean and scale them to have a standard deviation of α:

$$s_i = \alpha \cdot \frac{s_i' - \mu}{\sigma}. \tag{22}$$

By tuning α, we can control the determinism of the routing behavior: higher values of α lead to more deterministic (exploitation-oriented) decisions, while lower values encourage exploration.

Empirically, we set $\alpha = 1.5$, as it provides a good balance between exploration and exploitation.

Next context selection (NCS). We apply a similar scaling strategy to the cosine similarities in NCS. Specifically, each similarity score c_j is multiplied by a scaling factor β to obtain c_j':

$$c_j' = c_j, \quad c_j' \in [-\beta, \beta], \tag{23}$$

which is passed through a sigmoid function (Equation (13)) to compute the sampling probability g_j.

Note that contexts are selected based on g_j using Bernoulli sampling. By adjusting β, we control the determinism of context selection: a larger β makes the selection more deterministic (for exploitation), while a smaller β promotes exploration. Empirically, β is set to 3, as it provides a good balance between exploration and exploitation.

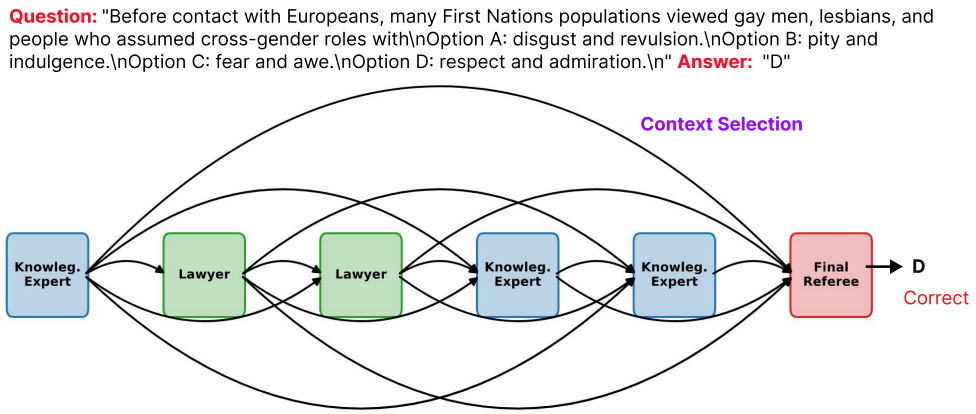
A.2 Reward Normalization

As discussed in Section 4.5, directly comparing rewards across different questions is inappropriate, as their difficulty levels may vary. A high reward on an easy question may not reflect the effectiveness of the routing strategy but rather the simplicity of the task itself. To address this, we adopt a standard reward normalization trick (Gu et al., 2016; Schulman et al., 2017), which normalizes the reward of each trajectory based on a baseline estimated from the same question:

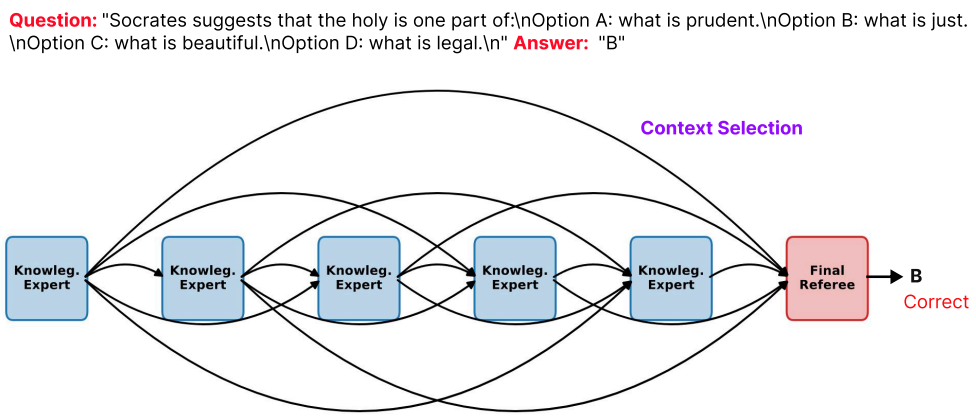
$$A_t = \frac{r_t - \mu}{\sigma}, \quad \mu = \frac{1}{T}\sum_{k=1}^{T} r_k, \quad \sigma = \sqrt{\frac{1}{T}\sum_{k=1}^{T}(r_k - \mu)^2}. \tag{24}$$

Here, r_t denotes the reward of the current trajectory, and T is the number of sampled trajectories for the same question. Each r_k represents the reward of the k-th sampled trajectory. The resulting A_t is the advantage value of current trajectory in policy gradient optimization (Equation (19)), reflecting how much better (or worse) the current trajectory performs compared to the average baseline.

This normalization trick mitigates the impact of question difficulty and enables the reinforcement learning process to treat rewards from different queries more fairly and consistently.



(a) Routing results and final answer of ANYMAC-ACC. for Question 1 in MMLU.



(b) Routing results and final answer of ANYMAC-ACC. for Question 2 in MMLU.

Fig. 5: Qualitative illustration of ANYMAC's routing decisions on two different questions. In (a), the model selects a combination of *Lawyer* and *Knowledgeable Expert* for Question 1. In (b), it assigns all agents as *Knowledgeable Experts* for Question 2.

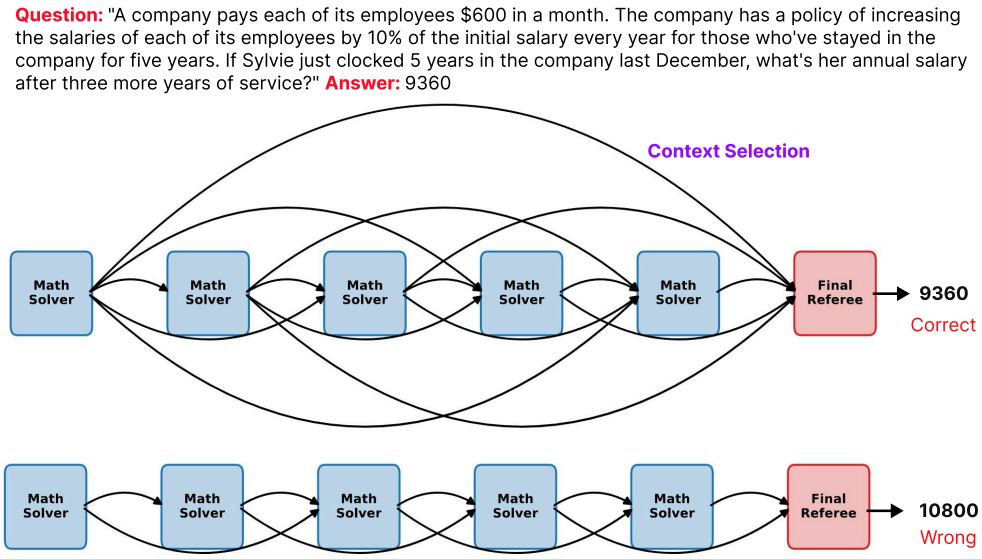


Fig. 6: Qualitative comparison between ANYMAC-ACC. (top) and ANYMAC-EFF. (bottom) on the same question. They use different computation budgets, and the increased computation in ANYMAC-ACC. leads to the correct answer.