MAS-Designer: Learning Task-Adaptive Communication Topologies for LLM-Based Multi-Agent Systems

Junzhi Chen[1] **Ziqi Chen**[2] **Juhao Liang**[2] **Yan Hu**[2]
[1]New York University
[2]The Chinese University of Hong Kong, Shenzhen
jc13140@nyu.edu {ziqichen1,juhaoliang1}@link.cuhk.edu.cn huyan@cuhk.edu.cn

Abstract

Large language model (LLM)-based multi-agent systems can improve reasoning performance through collaboration, but their effectiveness and cost depend critically on the communication topology among agents. Recent topology-design methods such as G-Designer Zhang et al. [2024] generate task-adaptive communication graphs with graph neural networks, yet their REINFORCE-style training objectives often suffer from high gradient variance, slow convergence, and unstable sparsity optimization.

We propose `MAS-Designer`, a streamlined framework for learning task-adaptive multi-agent communication topologies. Instead of using REINFORCE, `MAS-Designer` optimizes sampled graph structures with Group Relative Policy Optimization (GRPO), where multiple candidate topologies are sampled for each query and the group-average reward serves as a zero-cost variance-reduction baseline. We further introduce a success-conditioned composite reward, $R = u + u \cdot R_{\text{edge}}$, that rewards sparse communication only when the final answer is correct. This design discourages degenerate disconnected graphs while still favoring efficient topologies.

Experiments on reasoning and code-generation benchmarks show that `MAS-Designer` preserves the accuracy of strong refinement-based topology designers while improving training stability and communication efficiency. On GSM8K, it achieves 95.0% accuracy, comparable to G-Designer at 95.3%, while reducing total token consumption by 3.2% and lowering gradient variance by more than 60% across seeds. These results indicate that GRPO provides a practical and effective optimization strategy for discrete topology learning in LLM-based multi-agent systems.

1 Introduction

Large language models (LLMs) have reshaped the design of intelligent systems. Although single-agent LLM systems have made substantial progress, their performance on complex, open-ended problems remains constrained by a single context window and sequential generation Brown et al. [2020], Yao et al. [2022], Shinn et al. [2023], Wei et al. [2022], Yao et al. [2023], Besta et al. [2024]. This has motivated **multi-agent systems (MAS)**, where multiple specialized LLM-based agents collaborate through explicit communication Wei et al. [2022], Li et al. [2023], Du et al. [2023], Park et al. [2023]. Despite their success, statically designed MAS suffer from two critical limitations: (1) excessive token cost; (2) lack of task-adaptivity.

Preprint. Under review.

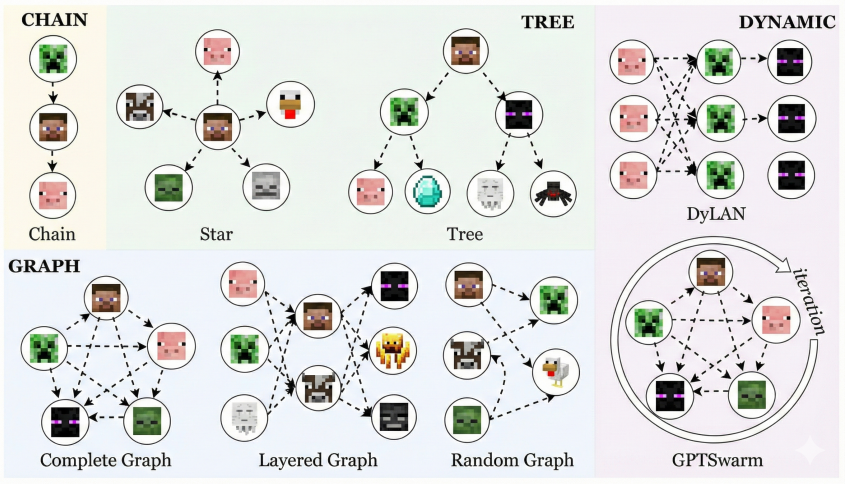

Figure 1: Overview of a multi-agent system.

Recent efforts have therefore shifted toward **automatic, task-adaptive topology generation**. Notable examples include MetaGPT Hong et al. [2023], MAS-GPT Ye et al. [2025], and G-Designer Zhang et al. [2024], which uses a Variational Graph Auto-Encoder to generate query-specific communication graphs.

However, G-Designer relies on REINFORCE-style policy gradient training with a complex low-rank refinement objective. In practice, this approach introduces severe optimization difficulties:

- **Training Instability:** REINFORCE suffers from high gradient variance, leading to slow convergence and frequent training collapse.
- **Reward Hacking:** The model either collapses to disconnected graphs (sacrificing accuracy) or ignores sparsity signals entirely.

In this work, we present `MAS-Designer`, a simple training framework for adaptive multi-agent topology generation that directly addresses these shortcomings:

- **Simple and effective MLP decoder**: We introduce a lightweight MLP+Gumbel-Sigmoid decoder that yields competitive topologies with greater stability.
- **Reward hacking-resistant composite reward**: We introduce a success-conditioned sparsity reward, where sparsity is *only* rewarded when the task is solved, cleanly separating the goals of correctness and efficiency and eliminating degenerate solutions.
- **Group Relative Policy Optimization (GRPO)**: We replace REINFORCE with GRPO. The group-average reward serves as a zero-cost baseline, reducing gradient variance and enabling stable convergence under sparse rewards.
- **Per-query dynamic topology generation**: At inference, we perform full reparameterization and Gumbel sampling for each query, producing genuinely task-adaptive graphs without any fixed spatial/temporal masks.

Our contributions can be summarized as follows:

1. A simplified, refinement-free decoder that matches strong refinement-based baselines while being easier to train.
2. A theoretically grounded, hacking-resistant reward that significantly reduces deployed MAS cost without accuracy trade-offs.
3. An application of GRPO to discrete structure optimization in the LLM multi-agent setting, establishing it as an effective replacement for REINFORCE in high-variance, sparse-reward regimes.

4. Empirical validation demonstrating improved efficiency-accuracy trade-offs in adaptive multi-agent systems.

By making adaptive topology design stable, inexpensive, and task-specific, `MAS-Designer` supports practical deployment of collaborative LLM agents in real-world applications.

2 Related Work

2.1 LLM-Agent Collaboration

Research on LLM agents has moved from single-agent paradigms Yao et al. [2022], Shinn et al. [2023], Wang et al. [2023] toward multi-agent systems (MAS), where multiple specialized agents collaborate on complex tasks Du et al. [2023], Talebirad and Nadiri [2023], Li et al. [2023], Park et al. [2023].

Early multi-agent systems relied on **static, hand-crafted topologies**. Representative examples include chain-style workflows Wei et al. [2022], star/leader structures Li et al. [2023], debate protocols Du et al. [2023], role-specialized teams Hong et al. [2023], Qian et al. [2024], and fully-connected societies Park et al. [2023]. These approaches consistently outperform single-agent baselines by leveraging specialization, iterative criticism, and collective reasoning Talebirad and Nadiri [2023].

More recent efforts have explored **dynamic or automatically optimized collaboration**. GPTSwarm Zhuge et al. [2024] models agents as optimizable computational graphs and uses evolutionary search or reinforcement learning to refine both node prompts and edge connectivity. MAS-GPT Ye et al. [2025] takes a different approach by fine-tuning a medium-sized LLM to generate executable multi-agent system code in a single forward pass given the user query, achieving broad generality across domains.

Despite these advances, significant challenges remain. Static topologies lack task-adaptivity and often incur unnecessary communication overhead on simple queries. Dynamic or generative approaches (e.g., GPTSwarm, MAS-GPT) either require expensive iterative LLM calls during execution or suffer from high training/inference costs and unstable optimization when learning to generate structures.

2.2 Multi-Agent Systems as Graphs

Graphs are a natural data structure for representing relationships among entities and were widely used before the LLM era to facilitate communication in multi-agent reinforcement learning (MARL). With the rise of LLM-based agents, researchers have similarly recognized that agent interactions can be modeled from a graph-based perspective Wei et al. [2022], Li et al. [2023], Du et al. [2023], Park et al. [2023].

GPTSwarm Zhuge et al. [2024] pioneered the view of language agents as compositional computational graphs, where nodes perform LLM queries or tool operations and edges define information flow. Subgraphs represent individual agents, and the full swarm graph enables hierarchical collaboration. Optimization is performed via evolutionary search or REINFORCE over both node prompts and edge connectivity.

Most closely related to our work is **G-Designer** Zhang et al. [2024], which casts multi-agent topology design as a graph generation problem. G-Designer uses a Variational Graph Auto-Encoder (VGAE) conditioned on agent role embeddings and a virtual task node to decode task-specific directed communication graphs. A sophisticated refinement module with low-rank reconstruction and nuclear-norm regularization further encourages sparsity and smoothness. Training relies on REINFORCE with the refinement objective as an auxiliary reward. G-Designer achieves state-of-the-art performance (95.07% on GSM8K) while significantly reducing token consumption on easy tasks (up to 95.33% savings).

However, as we demonstrate empirically, its REINFORCE-based training with complex refinement objectives suffers from high variance, slow convergence, and reward hacking, which hinder reproducibility and practical deployment.

3 Prior Knowledge

This section establishes the notation for multi-agent topology design, formalizes key concepts from a topology perspective, and defines the setting for `MAS-Designer`.

3.1 Topology Structure

We model the multi-agent system as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ represents the set of nodes (with $N = |\mathcal{V}|$) and $\mathcal{E}$ denotes the set of edges. Each node $v_i \in \mathcal{V}$ corresponds to an agent, formalized as:

$$v_i = \{\text{Base}_i, \text{Role}_i, \text{State}_i, \text{Plugin}_i\} \tag{1}$$

where each agent v_i is composed of four key elements: (1) Base_i, the language model instance powering v_i; (2) Role_i, the agent's pre-assigned role or function; (3) State_i, representing the agent's accumulated knowledge and interaction history; and (4) Plugin_i, a set of external tools or plugins available to v_i, such as web searchers, code compilers, or file readers. Each LLM-based agent v_i receives prompt $\mathcal{P}$ and generates response $\mathcal{R}_i$:

$$\mathcal{R}_i = v_i(\mathcal{P}) = v_i\left(\mathcal{P}_{\text{sys}}, \mathcal{P}_{\text{usr}}\right) \tag{2}$$

where $\mathcal{P}_{\text{sys}} = \{\text{Role}_i, \text{State}_i\}$ represents the system prompt encompassing its role and state, and $\mathcal{P}_{\text{usr}}$ denotes the user prompt, which possibly includes the given tasks, responses/instructions from other agents and externally retrieved knowledge.

The connectivity of $\mathcal{G}$ can also be characterized by a (nonsymmetric) adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$, where $\mathbf{A}[i,j] = 1$ if $e_{ij} = (v_i, v_j) \in \mathcal{E}$, otherwise 0. Each edge $e_{ij} \in \mathcal{E}$ represents the flow of information from v_i to v_j.

3.2 Communication Pipeline

Given a query/problem $\mathcal{Q}$, the multi-agent system engages in K rounds of interactive utterances, which collaboratively drive the agents toward producing the final solution $a^{(K)}$ based on their cumulative dialogue exchanges. At the beginning of the t-th dialogue round, a mapping function ϕ is applied to determine the execution index for each agent:

$$\begin{aligned} \phi : \mathcal{G} \longmapsto \sigma, \sigma = [v_{\sigma_1}, v_{\sigma_2}, \cdots, v_{\sigma_N}] \\ \text{s.t. } \forall i > j, \quad v_{\sigma_i} \notin \mathcal{N}_{\text{in}}\left(v_{\sigma_j}\right) \end{aligned} \tag{3}$$

where σ is the execution sequence of agents, $\mathcal{N}_{\text{in}}\left(v_{\sigma(j)}\right)$ denotes the in-neighborhood of $v_{\sigma(j)}$, and the constraint ensures that an agent $v_{\sigma(i)}$ can only execute after any agent $v_{\sigma(j)}$ from which it receives information. Once the execution order is determined, each agent proceeds to perform input-output operations sequentially:

$$\mathcal{R}_i^{(t)} = v_i\left(\mathcal{P}_{\text{sys}}^{(t)}, \mathcal{P}_{\text{usr}}^{(t)}\right), \mathcal{P}_{\text{usr}}^{(t)} = \left\{\mathcal{Q}, \cup_{v_j \in \mathcal{N}_{\text{in}}(v_i)} \mathcal{R}_j^{(t)}\right\} \tag{4}$$

where $\mathcal{R}_i^{(t)}$ represents the output of v_i, which could be a rationale, an answer, or a partial solution, depending on the specific context. The output $\mathcal{R}_i^{(t)}$ is generated based on the system prompt $\mathcal{P}_{\text{sys}}^{(t)}$ and the context prompt, consisting of the query $\mathcal{Q}$ and messages from other agents. At the end of each dialogue, an aggregation function is adopted to generate the answer/solution $a^{(t)}$:

$$a^{(t)} \leftarrow \text{Aggregate}\left(\mathcal{R}_1^{(t)}, \mathcal{R}_2^{(t)}, \cdots, \mathcal{R}_N^{(t)}\right) \tag{5}$$

The implementation of the Aggregate function is flexible, with possible options including majority voting Zhuge et al. [2024], aggregating all agents' responses and delegating one agent to provide the

final answer, or simply using the output of the last agent $\mathcal{R}_N^{(t)}$. Through K rounds of utterances, the overall system $\mathcal{G}$ produces the final answer $a^{(K)}$ for $\mathcal{Q}$.

4 MAS-Designer

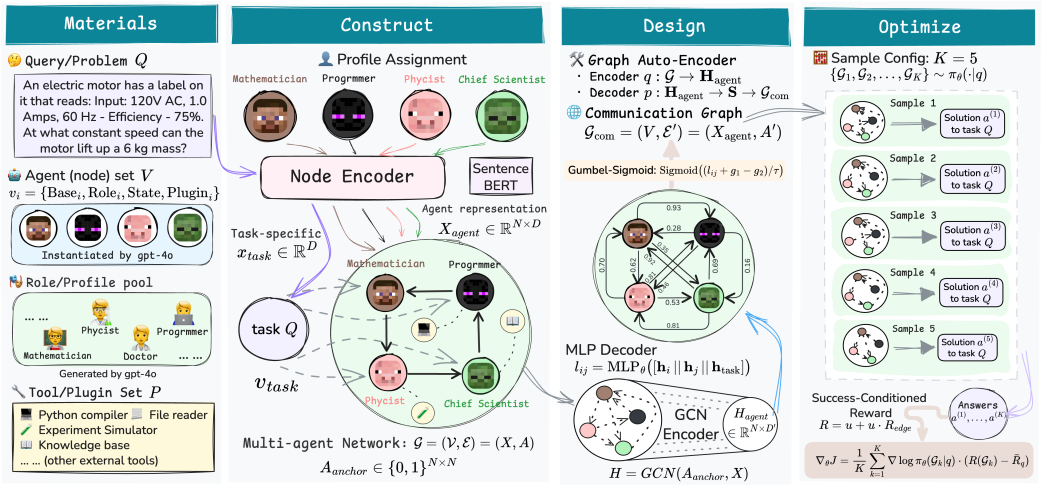

Figure 2: Design workflow of MAS-Designer.

The overall architecture is illustrated in Figure 2. MAS-Designer consists of four components: **Preliminaries** (agent pool and node construction), **Construct & Encoder** (GCN-based contextualization), **Design** (spatial decoding of task-specific topologies), and **Optimize** (GRPO training with success-conditioned rewards).

4.1 Preliminaries: Agent Pool and Node Feature Construction

Given a query q, we embed its full text with Sentence-BERT Reimers and Gurevych [2019] to obtain $\mathbf{x}_{\text{task}} \in \mathbb{R}^D$. This embedding is treated as a virtual task node v_{task} that is implicitly connected to all agents for broadcast-style conditioning. Similarly, each role description is embedded to yield initial agent features $\mathbf{X}_{\text{agent}} \in \mathbb{R}^{N\times D}$.

We also maintain a diverse role/profile pool generated once by an LLM, together with a tool/plugin set.

4.2 Construct: Multi-agent Network Construction

Given an input query $\mathcal{Q}$ and a set of LLM agents $\mathcal{V}$, MAS-Designer designs a task-adaptive communication topology $\mathcal{G}_{\text{com}}$. We begin by assigning each agent a unique role and profile. Based on these roles, external tools are allocated to agents (e.g., Mathematica for a math analyst or a Python compiler for a programmer). Each agent v_i is initialized as $\{\text{Base}_i, \text{Role}_i, \text{State}_i, \text{Plugin}_i\}$, as defined in Equation 1.

We proceed to construct a structured multi-agent network as input to MAS-Designer, represented as $\mathcal{G} = (\mathbf{X}_{\text{agent}}, \mathbf{A})$, where $\mathbf{X}_{\text{agent}} \in \mathbb{R}^{N\times D}$ is the node (agent) feature matrix and $\mathbf{A} \in \mathbb{R}^{N\times N}$ represents the connectivity matrix. For the feature matrix $\mathbf{X}_{\text{agent}}$, we employ a node encoder to transform each agent's unique profile into a fixed-length embedding representation:

$$\mathbf{x}_i \leftarrow \text{NodeEncoder}\left(\mathcal{T}\left(\text{Base}_i\right), \text{Role}_i, \mathcal{T}\left(\text{Plugin}_i\right)\right) \tag{6}$$

where $\mathcal{T}(\cdot)$ extracts the textual description of the agent's LLM backbone and assigned plugins, and NodeEncoder can be implemented with lightweight text embedding models Reimers and Gurevych [2019]. To incorporate query information, we introduce a task-specific virtual global node v_{task},

which is bidirectionally connected to all agent nodes. This task node is encoded by the NodeEncoder as follows:

$$\mathbf{x}_{\text{task}} \leftarrow \text{NodeEncoder}(\mathcal{Q}) \tag{7}$$

After obtaining the agent node features $\mathbf{X}_{\text{agent}} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\top$ and the task embedding $\mathbf{x}_{\text{task}}$, we provide a simple anchor topology $\mathbf{A}_{\text{anchor}} \in \{0,1\}^{N\times N}$ as the starting point. For example, in a code generation task with manager, programmer, and code reviewer agents, the anchor topology can be a chain: "manager $\rightarrow$ programmer $\rightarrow$ reviewer". The anchor topology can be user-defined or generated by an LLM. Although it is often simple and suboptimal, it provides a useful prior for optimization. We incorporate v_{task} and its corresponding edges to obtain $\tilde{\mathbf{A}}_{\text{anchor}} \in \{0,1\}^{(N+1)\times(N+1)}$. This yields a task-specific multi-agent network $\tilde{\mathcal{G}}$:

$$\begin{aligned}\tilde{\mathcal{G}} &= \left(\begin{bmatrix}\mathbf{X}_{\text{agent}}\\ \mathbf{x}_{\text{task}}^\top\end{bmatrix}, \tilde{\mathbf{A}}_{\text{anchor}}\right) = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})\\ &= \left(\mathcal{V} \cup \{v_{\text{task}}\}, \mathcal{E} \cup \{(v_i, v_{\text{task}}) \mid v_i \in \mathcal{V}\}\right)\end{aligned} \tag{8}$$

where $\begin{bmatrix}\mathbf{X}_{\text{agent}}\\ \mathbf{x}_{\text{task}}\end{bmatrix}$ can also be jointly denoted as $\tilde{\mathbf{X}}$.

4.3 Encoder: GCN Encoder

We construct the initial multi-agent network as a heterogeneous graph containing N agent nodes and one virtual task node. Input node features are $\mathbf{X} = [\mathbf{X}_{\text{agent}}; \mathbf{x}_{\text{task}}] \in \mathbb{R}^{(N+1)\times D}$.

A two-layer Graph Convolutional Network (GCN) Kipf [2016] encodes contextualized representations:

$$\mathbf{H} = \text{GCN}(\hat{\mathbf{A}}_{\text{anchor}}, \mathbf{X}),$$

where $\hat{\mathbf{A}}_{\text{anchor}}$ is the symmetrically normalized adjacency with self-loops. The resulting $\mathbf{H} \in \mathbb{R}^{(N+1)\times D}$ contains richly contextualized agent embeddings $\mathbf{H}_{\text{agent}} \in \mathbb{R}^{N\times D}$ and task embedding $\mathbf{h}_{\text{task}} \in \mathbb{R}^D$ that capture both agent capabilities and query-specific requirements.

4.4 Design: Spatial Decoder (Refinement-Free)

The decoder defines a conditional distribution $\pi_\theta(\mathcal{G} \mid q)$ over directed communication graphs. We use a **simple, refinement-free** MLP+Gumbel-Sigmoid decoder that directly predicts edge probabilities from encoded representations.

For each ordered pair (i, j) with $i \neq j$, the logit for a directed edge $v_i \rightarrow v_j$ is computed as

$$l_{ij} = \text{MLP}_\theta\big([\mathbf{h}_i \,||\, \mathbf{h}_j \,||\, \mathbf{h}_{\text{task}}]\big) \in \mathbb{R},$$

where $||$ denotes concatenation and MLP is a 3-layer network with ReLU activations and hidden dimension $2D$.

During training, edges are sampled via the continuous Gumbel-Softmax relaxation:

$$s_{ij} = \text{Sigmoid}\big((l_{ij} + g_1 - g_2)/\tau\big), \quad g_1, g_2 \sim \text{Gumbel}(0,1).$$

At inference, we use straight-through Gumbel-Sigmoid sampling or take $\arg\max$ over multiple samples. This produces query-specific topologies without fixed spatial or temporal execution masks.

4.5 Optimize: Group Relative Policy Optimization with Success-Conditioned Reward

We train the entire encoder-decoder using policy gradient on a dataset $\mathcal{D}$ of queries with ground-truth answers. For each query $q \in \mathcal{D}$, we sample $K = 5$ independent graphs $\mathcal{G}_1, \ldots, \mathcal{G}_K \sim \pi_\theta(\cdot \mid q)$ and execute the multi-agent system on q with each graph (parallel execution when API limits allow).

The reward for graph k is our **success-conditioned composite reward**:

$$R(\mathcal{G}_k) = u_k + u_k \cdot R_{\text{edge},k},$$

where

- $u_k \in \{0, 1\}$ is 1 if the final answer produced under $\mathcal{G}_k$ is correct,
- $R_{\text{edge},k} = \frac{E_{\text{full}} - E_{\text{current},k}}{E_{\text{full}}} \in [0, 1]$ measures relative sparsity (or token savings) against a dense complete-graph baseline.

Crucially, sparsity is **only rewarded when the task is solved correctly** ($u_k = 1$), which avoids degenerate disconnected solutions caused by prior sparsity regularization schemes.

The policy gradient is computed using **Group Relative Policy Optimization (GRPO)**:

$$\nabla_\theta J(\theta) = \mathbb{E}_{q\sim\mathcal{D}} \left[\frac{1}{K} \sum_{k=1}^{K} \nabla_\theta \log \pi_\theta(\mathcal{G}_k \mid q) \cdot \big(R(\mathcal{G}_k) - \bar{R}_q\big) \right],$$

where $\bar{R}_q = \frac{1}{K} \sum_{k=1}^{K} R(\mathcal{G}_k)$ is the per-query group mean reward. This zero-cost baseline reduces variance relative to standard REINFORCE or reward-to-go, enabling stable convergence under sparse correctness signals.

At inference, for a new query q, we perform full reparameterized sampling (typically $K = 5$) and either (i) select the graph yielding the highest approximate reward when self-evaluation is reliable or (ii) execute all K graphs in parallel and aggregate via voting or confidence-weighted selection. This supports both strong performance and task-adaptivity with modest overhead.

By removing refinement modules, using success-conditioned rewards, and leveraging GRPO, `MAS-Designer` improves the accuracy-efficiency trade-off, convergence speed, and training stability compared with refinement-based graph designers.

5 Experiments

5.1 Experiment Setup

Datasets and metrics. We evaluate `MAS-Designer` on three categories of datasets: (1) **general reasoning**: MMLU Hendrycks et al. [2020], (2) **mathematical reasoning**: GSM8K Cobbe et al. [2021], and (3) **code generation**: HumanEval Chen [2021].

Implementation details. We access GPT models through the OpenAI API and mainly evaluate `gpt-4o-2024-11-20`. We set `temperature` to 1 for multi-agent methods. A summarizer agent aggregates dialogue history and produces the final solution $a^{(K)}$, with $K = 5$ across all experiments. `NodeEncoder(·)` is implemented with `all-MiniLM-L6-v2` Wang et al. [2020], with embedding dimension $D = 384$. The anchor topology $\mathbf{A}_{\text{anchor}}$ is predefined as a simple chain. We set the sampling count M to 20, $\tau = 10^{-2}$, and $\zeta = 10^{-1}$ for all experiments. We use `gpt-4` to generate agent profile pools. For each benchmark, we use only $B' \in \{40, 80\}$ queries for optimization.

Baselines. We focus on state-of-the-art parametric topology optimization methods and select G-Designer Zhang et al. [2024] as the primary baseline.

- **G-Designer (reproduced)**: Faithful reproduction of the original method with refinement module, nuclear-norm regularization, and REINFORCE training.
- **G-Designer-R**: Ablation removing the refinement module, using hard Gumbel-Softmax + unconditional reward $R = u + \lambda R_{\text{edge}}$.

5.2 Main Results

We conduct experiments across three benchmarks to evaluate the efficiency and performance of `MAS-Designer`.

Efficiency. Table 1 shows that `MAS-Designer` designs effective LLM-MAS topologies. It achieves **95.0% accuracy**, closely matching our G-Designer reproduction (95.3%), while reducing evaluation token usage by 3.2%. Training is also more stable, with gradient variance reduced by more than 60% across seeds and convergence in roughly half the iterations, as shown in Figure 3.

Benchmark performance. On the standard benchmarks (full test sets, same 40-question meta-training), `MAS-Designer` preserves G-Designer's strong performance while being easier to train.

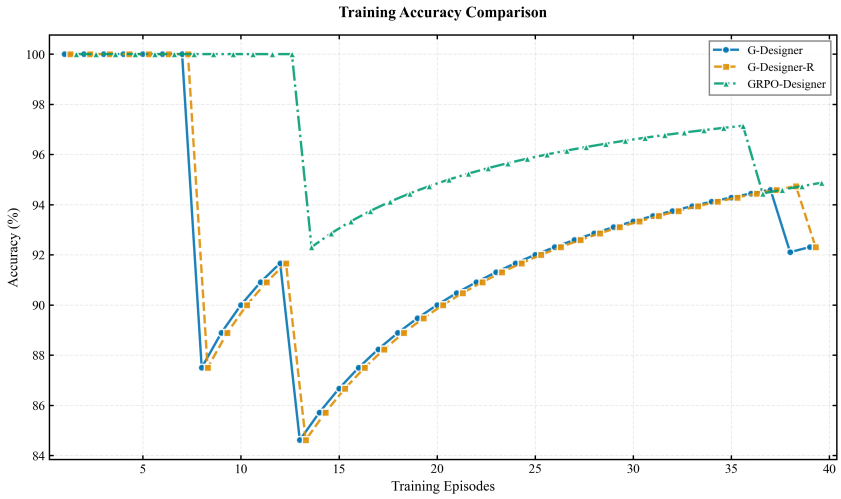

Figure 3: Training accuracy comparison over 40 examples.

Table 1: Main results on GSM-8K (meta-train on 40 questions, evaluate on 300 held-out questions).

Method	Accuracy (%)	Total Tokens	Cost ($)	Tokens vs G-Designer
G-Designer (repro)	95.3	1,652,939	0.409	0%
G-Designer-R (no refine)	94.3	2,275,273	0.533	+38%
`MAS-Designer` **(Ours)**	**95.0**	**1,598,960**	**0.403**	**-3.2%**

`MAS-Designer` slightly outperforms G-Designer on both benchmarks, with more stable training dynamics and no NaN/crash issues from refinement-module training.

The results also show that `G-Designer-R` underperforms in both accuracy and total tokens. This indicates that simply removing the refinement module is insufficient; GRPO is needed to maintain efficiency without the complex architecture.

Table 2: Performance on standard MMLU and HumanEval test sets (meta-trained on 40 examples).

Method	MMLU (%)	HumanEval (%)
G-Designer (repro) Zhang et al. [2024]	84.50	89.90
MAS-Designer (Ours)	**84.62** $\pm$ 0.31	**90.12** $\pm$ 0.45

5.3 Result Analysis

While `MAS-Designer` improves stability and efficiency, several limitations remain. (1) Our experiments relied on a small meta-training set ($B' \in \{40, 80\}$), and it remains unclear if the method scales effectively to larger datasets without overfitting; (2) Our optimization initializes from a simple "chain" anchor; future work must determine if random or fully-connected initializations yield different optimal topologies.

6 Conclusion

In this paper, we introduce `MAS-Designer`, a streamlined framework for optimizing task-adaptive multi-agent topologies. By replacing high-variance REINFORCE gradients with Group Relative Policy Optimization (GRPO) and using a refinement-free decoder with a success-conditioned composite reward, we address the stability and reward-hacking issues of prior methods.

Empirically, `MAS-Designer` matches the accuracy of complex refinement-based baselines (95.0% on GSM8K) while reducing token consumption by 3.2% and gradient variance by more than 60%. Our

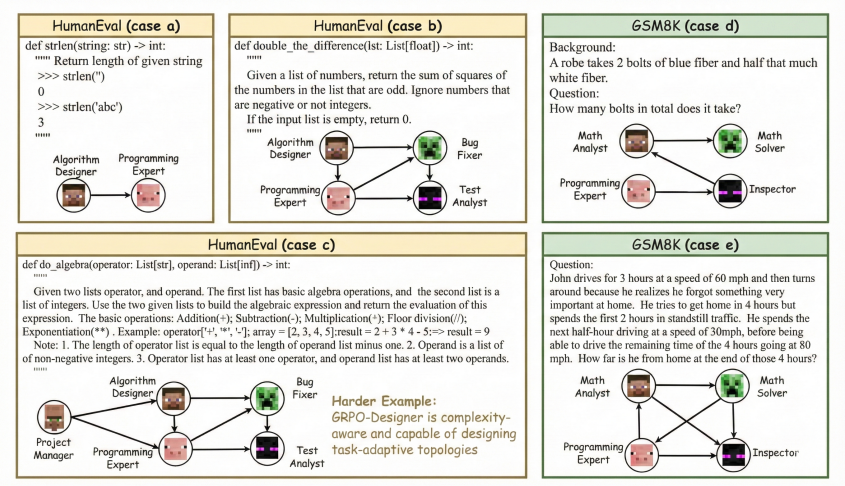

Figure 4: Case study of the communication topologies designed by `MAS-Designer` on HumanEval and GSM8K benchmarks.

findings suggest that a group-based optimization strategy combined with strict success conditioning is sufficient to learn efficient, interpretable communication structures. We identify GRPO as an effective replacement for REINFORCE in discrete structure optimization for LLM-based multi-agent systems.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690, 2024.

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

Mark Chen. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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*, 2023.

TN Kipf. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, 2024.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

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

Yashar Talebirad and Amirhossein Nadiri. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv:2306.03314*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Rui Ye, Shuo Tang, Rui Ge, Yaxin Du, Zhenfei Yin, Siheng Chen, and Jing Shao. Mas-gpt: Training llms to build llm-based multi-agent systems. *arXiv preprint arXiv:2503.03686*, 2025.

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. *arXiv preprint arXiv:2410.11782*, 2024.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.