

Assemble Your Crew: Automatic Multi-agent Communication Topology Design via Autoregressive Graph Generation

Shiyuan Li¹, Yixin Liu¹, Qingsong Wen², Chengqi Zhang³, Shirui Pan¹

¹Griffith University, ²Squirrel Ai Learning, ³Hong Kong Polytechnic University

li.shiy511@gmail.com; {yixin.liu, s.pan}@griffith.edu.au; qingsongedu@gmail.com; chengqi.zhang@polyu.edu.hk

Abstract

Multi-agent systems (MAS) based on large language models (LLMs) have emerged as a powerful solution for dealing with complex problems across diverse domains. The effectiveness of MAS is critically dependent on its collaboration topology, which has become a focal point for automated design research. However, existing approaches are fundamentally constrained by their reliance on a template graph modification paradigm with a predefined set of agents and hard-coded interaction structures, significantly limiting their adaptability to task-specific requirements. To address these limitations, we reframe MAS design as a conditional autoregressive graph generation task, where both the system composition and structure are designed jointly. We propose **ARG-DESIGNER**, a novel autoregressive model that operationalizes this paradigm by constructing the collaboration graph from scratch. Conditioned on a natural language task query, ARG-DESIGNER sequentially and dynamically determines the required number of agents, selects their appropriate roles from an extensible pool, and establishes the optimal communication links between them. This generative approach creates a customized topology in a flexible and extensible manner, precisely tailored to the unique demands of different tasks. Extensive experiments across six diverse benchmarks demonstrate that ARG-DESIGNER not only achieves state-of-the-art performance but also enjoys significantly greater token efficiency and enhanced extensibility. The source code of ARG-DESIGNER is available at <https://github.com/Shiy-Li/ARG-Designer>.

Introduction

Agents built on large language models (LLMs) have demonstrated impressive capabilities in tackling complex tasks across diverse domains, including code generation, data analysis, decision-making, and question answering (Zhu et al. 2024; Li et al. 2024a; Song et al. 2023; Wang et al. 2024; Zhong, Wang, and Shang 2024). To overcome the limitations of a single agent in tackling more complex tasks, the research interests have increasingly shifted towards multi-agent systems (MAS), which unlock new potential through collaborative interactions among agents with diverse capabilities and roles. Central to MAS is its **collaboration topology**, a graph that defines how agents with various roles are

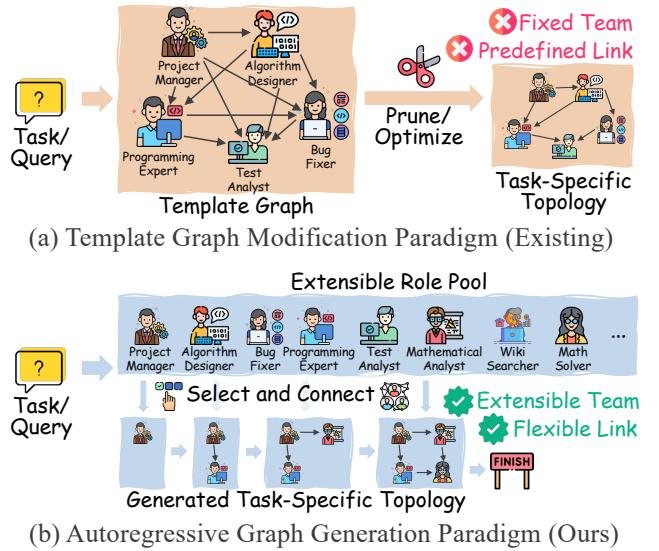


Figure 1: The comparison of two paradigms: (a) template graph modification and (b) autoregressive graph generation.

structured and how they exchange information. A growing body of evidence, spanning from sequential reasoning pipelines to debate-based approaches, demonstrates that MAS performance varies dramatically depending on how inter-agent communication is architected (Zhang et al. 2024; Zhou et al. 2025). Therefore, designing an effective collaboration graph tailored to specific tasks becomes a critical research challenge.

Early research on MAS topology design focused on static and manually designed graphs, such as chains that enforce a sequential workflow (Wei et al. 2022; Hong et al. 2023), trees that enable structured deliberation (Yao et al. 2023), and fully connected graphs that ensure sufficient communication. Although these canonical collaboration topologies can facilitate effective coordination in specific scenarios, the inherent rigidity of these fixed topologies limits their adaptability across diverse tasks, often resulting in sub-optimal performance. To enhance flexibility and efficiency, a more recent line of work focuses on adaptively constructing task-specific communication structures using

graph learning models (Zhuge et al. 2024; Shen et al. 2025). For example, AgentPrune (Zhang et al. 2025a) and Agent-Dropout (Wang et al. 2025) learn to create sparse and task-specific graphs by pruning connections or agents from a predefined template topology. More advanced approaches like G-Designer (Zhang et al. 2025b) follow the paradigm of graph structure learning, which leverages a graph autoencoder to learn efficient collaboration structures in a task-adaptive manner.

Despite their varied designs, existing graph learning-based methods often follow a shared paradigm: *template graph modification* (Fig. 1a). That is, they typically start from a fixed communication template based on a predefined set of agents and hard-coded interaction structures, and apply learnable adjustments, such as edge reweighting or pruning, to adapt the topology to specific tasks (Zhang et al. 2025a,b). Despite offering reasonable adaptability in constrained settings, this paradigm exhibits two inherent limitations. **Limitation 1: Redundant Composition.** To ensure structural flexibility, template graphs are often initialized with numerous agent roles and densely connected edges, many of which are unnecessary for a specific task. Even with pruning mechanisms, irrelevant agents or connections may be retained in the learned task-specific topology, leading not only to reduced efficiency but also to potential sub-optimal decision-making during execution. **Limitation 2: Limited Extensibility.** In the fast-evolving field of LLM-based agents, a massive number of new agent functionalities are emerging with increasing frequency. However, trained on a fixed template graph, the existing methods struggle to generalize to scenarios with dynamic agent sets or evolving collaboration needs. Meanwhile, it would be prohibitively expensive to build a large-scale template graph that covers all possible agent roles and interaction patterns, and then prune it to a suitable task-specific topology. Given the above limitations, a natural question arises: *Going beyond template graph modification, can we design a more flexible and extensible paradigm for collaboration topology construction?*

To seek the answer to the above question, we draw inspiration from real-world practices of recruiting teams for complex tasks. Rather than starting with a fully staffed team where every possible member is onboarded from the beginning, real-world teams are usually formed incrementally, with members added based on expertise, availability, and evolving task needs. This practical pattern inspires us to explore *autoregressive graph generation* (Fig. 1b) as a more promising paradigm for collaboration topology construction. Unlike pruning from a predefined overcomplete structure, the new generation paradigm constructs the collaboration graph from scratch by progressively selecting appropriate agents. Such an incremental procedure naturally avoids redundant agent-role compositions during the design process, which naturally addresses **Limitation 1**. Moreover, by discarding the fixed template, the generative paradigm enables dynamic expansion of the agent pool, with only linear computational cost during the node generation phase. This merit enhances the extensibility of collaboration graph construction and thus alleviates **Limitation 2**.

Building upon the new paradigm, in this paper, we pro-

pose **ARG-DESIGNER**, a novel AutoRegressive Graph generation model that acts as a MAS topology **Designer**. Conditioned on a natural language task query, ARG-DESIGNER constructs the entire collaboration graph from scratch by iteratively generating each node (i.e., agent) along with its corresponding edges (i.e., communication links) to previously generated nodes. Compared to prior approaches, ARG-DESIGNER provides enhanced flexibility and scalability with respect to the number of agents, the variety of agent roles, and the richness of potential interactions. To train our generative model, we design a curriculum learning strategy that starts with denser communication topologies to ease the cold-start problem, and gradually transitions to sparser, pruned graphs for fine-tuning, encouraging the model to generalize to minimal yet effective structures. Extensive experiments on six benchmarks demonstrate that our method achieves state-of-the-art effectiveness, communication efficiency, and robustness.

Problem Formulation

MAS as a Collaboration Graph. We model a MAS as a *collaboration graph*, a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that outlines the system architecture and the flow of information among its components. The nodes $\mathcal{V} = \{v_1, v_2, \dots, v_N\}$ represent the set of agents, where each agent v_i is an instance of an LLM endowed with a specific role $r_i \in R$ that dictates its function and expertise. It also maintains an internal state $s_i \in S$, which serves as a memory of its past actions and interactions. The edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ define the directed communication pathways. An edge $e_{ji} = (v_j, v_i)$ signifies that agent v_i is a designated recipient of information from agent v_j . The set of direct predecessors of agent v_i is denoted by $\mathcal{N}_{in}(v_i) = \{v_j \mid (v_j, v_i) \in \mathcal{E}\}$.

MAS Collaboration Protocol. Given a collaboration graph \mathcal{G} , the MAS addresses a user query \mathcal{Q} by executing a multi-step collaboration protocol. This protocol governs how information is processed and passed between agents, unfolding over a series of communication rounds. The operational sequence for agent activation within each round is determined by a topological sort of the nodes, ensuring that an agent is activated only after its prerequisite inputs are available. This process can be executed for K rounds to allow for iterative refinement. In each round $k \in \{1, \dots, K\}$, an agent v_i generates its response $m_i^{(k)}$ by invoking its language model with a dynamically constructed prompt $\mathcal{P}_i^{(k)}$:

$$m_i^{(k)} = \text{LLM}_i(\mathcal{P}_i^{(k)}). \quad (1)$$

where the prompt integrates the intrinsic properties of the agent with the outputs of its predecessors from the previous round:

$$\mathcal{P}_i^{(k)} = f(\underbrace{r_i, s_i}_{\text{System}}, \underbrace{\mathcal{Q}, \{m_j^{(k-1)} \mid v_j \in \mathcal{N}_{in}(v_i)\}}_{\text{User}}). \quad (2)$$

After K rounds, the final output \mathcal{O} is obtained by aggregating the final-round responses from a subset of or all agents:

$$\mathcal{O} = \text{Aggregate}(\{m_i^{(K)} \mid v_i \in \mathcal{V}\}), \quad (3)$$

where the aggregation strategy $\text{Aggregate}(\cdot)$ varies across implementations. Common strategies include majority voting, delegating the final decision to a specific terminal agent, or selecting the output from the last agent in the execution order. The number of communication rounds K can be either predefined or adaptively determined via early-stopping mechanisms.

MAS Topology Design as a Graph Generation Task. The automatic task-specific construction of MAS topologies is a key challenge and research frontier. Traditional automated approaches that start from a large, predefined template graph, analogous to a fully-staffed team with every possible role, suffer from redundancy and limited extensibility. Drawing inspiration from real-world practices of building expert teams incrementally, we reframe the problem from modifying a fixed template to generating a bespoke graph from scratch. Instead of navigating the enormous graph space \mathbb{G} with an expensive utility function $\phi(\text{Execute}(\mathcal{G}, \mathcal{Q}))$, we propose to learn a **conditional generative model**, i.e., $P(\mathcal{G}|\mathcal{Q}, \mathcal{R})$, where \mathcal{R} is an extensive agent role pool. This model directly captures the relationship between a task query and the principles of effective collaboration, aiming to find the optimal communication topology \mathcal{G}^* that is most probable under this learned distribution:

$$\mathcal{G}^* = \arg \max_{\mathcal{G} \in \mathbb{G}} P(\mathcal{G}|\mathcal{Q}, \mathcal{R}). \quad (4)$$

Compared to modifying a predefined template graph, the generative formulation offers a more flexible, extensible, and scalable approach for constructing high-quality MAS topologies.

Autoregressive Graph Generation. To make this topology generation process more tractable, we formulate it as an autoregressive graph generation problem. This formulation decomposes the intractable joint probability of an entire graph into a tractable sequence of conditional probabilities. The graph is constructed incrementally, where each step involves adding a new node and its corresponding edges, conditioned on the partial graph built so far.

Formally, this factorization is expressed as:

$$P(\mathcal{G}|\mathcal{Q}, \mathcal{R}) = \prod_{i=1}^{|\mathcal{V}|} \underbrace{P(v_i|\mathcal{G}_{<i}, \mathcal{Q}, \mathcal{R})}_{\text{Node Generation}} \cdot \underbrace{\prod_{j=1}^{i-1} P(e_{ji}|v_i, \mathcal{G}_{<i}, \mathcal{Q})}_{\text{Edge Generation}}, \quad (5)$$

where $\mathcal{G}_{<i}$ represents the subgraph of the first $i - 1$ nodes. The generation process at each step i thus involves two key actions: **node generation**, predicting the role of the next agent to add, and **edge generation**, establishing its connections from existing agents. This formulation provides significant flexibility, enabling the model to dynamically determine the total number of agents by learning to sample a special END token, and to model complex structural dependencies by conditioning on the generation history.

Discussion. This generative approach provides several key advantages over traditional template-based methods, as summarized in Table 1. **1 Task-Adaptive Construction.** By conditioning on the task query, the model constructs a bespoke collaboration graph from scratch, avoiding the rigidity

Table 1: Comparison of degrees of freedom in MAS Design Paradigms. The icons (full), (partial), and (no support) represent full, partial, and no support for each capability, respectively.

Method	Task-Adaptive	Variable Size	Flexible Roles
Manual Design			
AgentDropout			
AgentPrune			
G-Designer			
ARG-DESIGNER (ours)			

and one-size-fits-all limitations of predefined graphs. **2 Dynamic and Extensible Composition.** The model dynamically determines the necessary number of agents and selects their roles from an extensible pool, ensuring the MAS composition is precisely tailored to the task needs and can easily incorporate new agent capabilities. **3 Tractable Generation.** The autoregressive factorization transforms the intractable problem of generating a whole graph into a sequence of simple, conditional steps, making the learning process both manageable and scalable.

ARG-DESIGNER for MAS Topology Design

Based on the autoregressive graph generation paradigm, this section instantiates the proposed method, **ARG-DESIGNER**, which is specifically crafted for MAS topology generation. We first introduce the model architecture designed to implement the sequential generation process, and then describe the training and inference strategies to guide ARG-DESIGNER toward generating both functionally correct and structurally efficient collaboration graphs.

Model Architecture

Following the autoregressive generation paradigm, ARG-DESIGNER constructs collaboration graphs step-by-step. ARG-DESIGNER employs a hierarchical architecture based on gated recurrent units (GRUs), which are well-suited for sequence modeling due to their effectiveness in capturing long-range dependencies while maintaining computational efficiency. The architecture separates the generation model into two sub-components: a *node generator* to select agent roles and an *edge generator* to build communication links. An overview of the model architecture is depicted in Fig. 2a. **Input Representation.** Before generation begins, ARG-DESIGNER encodes all textual conditioning information (i.e., the task query and available agent roles) into dense vector representations. Specifically, the task query \mathcal{Q} is mapped into a fixed-dimensional vector $\mathbf{f}_Q \in \mathbb{R}^d$ by a pre-trained BERT-like sentence encoder followed by a feed-forward network (FFN) with Layer Normalization (LN):

$$\mathbf{f}_Q = \text{FFN}(\text{LN}(\text{SentenceEncoder}(\mathcal{Q}))). \quad (6)$$

Similarly, each available agent role $r_k \in \mathcal{R}$ is converted into an embedding \mathbf{z}_{r_k} . These pre-computed embeddings are collected into a role embedding matrix $\mathbf{Z} \in \mathbb{R}^{|\mathcal{R}| \times d}$, serving as the knowledge base of available agents.

Node Generation. At each step i , the node generator selects the role for the next agent, i.e., v_i . ARG-DESIGNER first

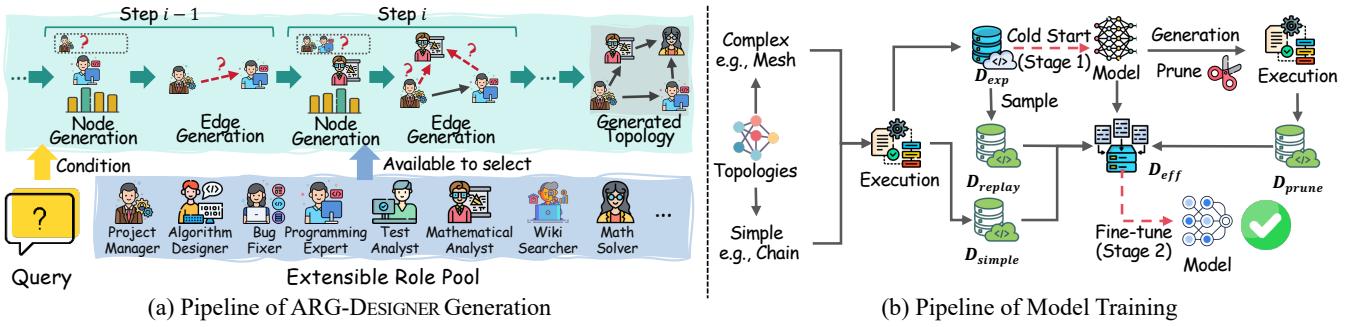


Figure 2: The pipeline of ARG-DESIGNER, including (a) MAS communication topology generation and (b) model training.

models the context information by combining task information with the generation history. A dedicated GRU, GRU_{prev} , is employed to aggregate the role embeddings of all preceding agents to form a historical embedding $\mathbf{f}_{\text{hist}}^{(i)}$:

$$\mathbf{f}_{\text{hist}}^{(i)} = \text{GRU}_{\text{prev}}([\mathbf{z}_{r_1}, \mathbf{z}_{r_2}, \dots, \mathbf{z}_{r_{i-1}}]). \quad (7)$$

Then, we fuse the historical embedding $\mathbf{f}_{\text{hist}}^{(i)}$ with the task embedding $\mathbf{f}_{\mathcal{Q}}$ via a dynamic gate to produce the context embedding $\mathbf{f}_{\text{cont}}^{(i)}$:

$$\mathbf{f}_{\text{cont}}^{(i)} = (1 - \mathbf{g}_i) \cdot \mathbf{f}_{\text{hist}}^{(i)} + \mathbf{g}_i \cdot \mathbf{f}_{\mathcal{Q}}, \quad \mathbf{g}_i = \sigma \left(\frac{\mathbf{f}_{\text{hist}}^{(i)} \cdot \mathbf{f}_{\mathcal{Q}}}{\sqrt{d}} \right), \quad (8)$$

where σ denotes the sigmoid function. This context, along with an edge feature vector $\mathbf{f}_{\text{edge}}^{(i)}$ (i.e., a vector encoding the connectivity pattern of the previously added node v_{i-1}), are concatenated into an input vector $[\mathbf{f}_{\text{cont}}^{(i)}, \mathbf{f}_{\text{edge}}^{(i)}]$. Then, another GRU module, GRU_{node} , updates its hidden state $\mathbf{h}_{\text{node}}^{(i)}$, which captures the full generation condition:

$$\mathbf{h}_{\text{node}}^{(i)} = \text{GRU}_{\text{node}}(\text{MLP}_{\text{node}}([\mathbf{f}_{\text{cont}}^{(i)}, \mathbf{f}_{\text{edge}}^{(i)}]), \mathbf{h}_{\text{node}}^{(i-1)}). \quad (9)$$

To preserve the extensible property of ARG-DESIGNER during agent role selection, we use a metric learning-based module for node generation. Concretely, the hidden state $\mathbf{h}_{\text{node}}^{(i)}$ is projected into a “node intent” embedding. Then, the node prediction scores $\mathbf{s}_{\text{node}}^{(i)}$ will be acquired by a dot-product operation with projected role embeddings:

$$\mathbf{s}_{\text{node}}^{(i)} = \text{MLP}_{\text{pred_n}}(\mathbf{h}_{\text{node}}^{(i)}) \cdot \text{MLP}_{\text{role}}([\mathbf{Z}, \mathbf{z}_{\text{end}}]), \quad (10)$$

where, \mathbf{z}_{end} is a learnable embedding for ending token END, which signals the termination of the generation process. Finally, we can obtain the predicted probability as follows:

$$P(v_i | \mathcal{G}_{<i}, \mathcal{Q}, \mathcal{R}) = \text{Softmax}(\mathbf{s}_{\text{node}}^{(i)}), \quad (11)$$

where the $\text{Softmax}(\cdot)$ function converts the scores into a probability distribution.

Discussion of Extensibility: The design of the node generator in ARG-DESIGNER allows new agent roles to be added at inference time without retraining. When new roles

are introduced, we can extend the role embedding matrix \mathbf{Z} by appending new role-specific embedding rows. Since $\mathbf{s}_{\text{node}}^{(i)}$ is produced by a metric learning-based retrieval mechanism rather than a fixed-dimensional classifier, the model can flexibly select from an expanded set of roles based on similarity in the shared embedding space. This design ensures that ARG-DESIGNER remains extensible and adaptable to evolving agent pools, which well fits the real-world scenarios where new agents with novel functionalities are frequently introduced to meet emerging task demands.

Edge Generation. Once agent node v_i is chosen, the edge generator determines its incoming connections from existing agents $\{v_1, \dots, v_{i-1}\}$. Here, we use a dedicated GRU, GRU_{edge} to model this sequential process. Its hidden state is initialized from the final state of the node-level GRU, $\mathbf{h}_{\text{node}}^{(i)}$, serving as the condition of edge prediction:

$$\mathbf{h}_{\text{edge}}^{(i,0)} = \text{MLP}_{\text{node2edge}}(\mathbf{h}_{\text{node}}^{(i)}). \quad (12)$$

After that, the model iterates through previously predicted nodes v_j ($j = 1, \dots, i-1$). At each sub-step, the edge GRU updates its hidden state based on the embedding of the previous edge decision:

$$\mathbf{h}_{\text{edge}}^{(i,j)} = \text{GRU}_{\text{edge}}(\text{MLP}_{\text{edge}}(\mathbf{e}^{(j-1,i)}), \mathbf{h}_{\text{edge}}^{(i,j-1)}), \quad (13)$$

where $\mathbf{e}^{(j-1,i)}$ is a one-hot vector representing the previous decision on whether to form an edge from node v_{j-1} to v_i . Following that, the updated state $\mathbf{h}_{\text{edge}}^{(i,j)}$ is passed through an output MLP to predict the score:

$$s_{\text{edge}}^{(i,j)} = \text{MLP}_{\text{pred_e}}(\mathbf{h}_{\text{edge}}^{(i,j)}). \quad (14)$$

We can then obtain the probability of edge $e_{j,i}$ by:

$$P(e_{j,i} = 1 | v_i, \mathcal{G}_{<i}, \mathcal{Q}) = \text{Sigmoid}(s_{\text{edge}}^{(i,j)}). \quad (15)$$

Training and Inference Strategy

Data Construction of Curriculum Learning. To build a powerful graph generator for MAS communication topology, we set up two key objectives: **① Functional correctness**, which ensures that the generated topology enables agents to collaboratively complete the given task, with all necessary roles and interactions properly instantiated; **② Communicational efficiency**, which encourages

the generated topology to be lightweight, sparse, and compact, by minimizing redundant links or agents. To reach these goals, we design a **curriculum learning** strategy that constructs training data for two-stage training. In the first stage, we create an *exploration dataset* to teach the model to generate correct and diverse topologies; then, an *efficiency dataset* is built to guide the model to design simple yet communication-efficient topologies.

Formally, we define a dataset as $\mathcal{D} = \{\mathcal{G}_k, \mathcal{Q}_k\}_{k=1}^M$ that provides strong supervision on what constitutes an effective collaboration graph \mathcal{G}_k for a given task query \mathcal{Q}_k . Since manually authoring such optimal task-graph pairs is infeasible, we propose to construct high-quality datasets in an automatically synthetic manner.

The first phase focuses on creating an exploration dataset (\mathcal{D}_{exp}) for the cold start training of the model, which aims to teach the model to create effective communication graphs. This dataset is formed by pairing tasks from a base set \mathbb{Q}_{base} with resource-rich, complex configurations from a configuration space $\mathbb{C}_{\text{complex}}$, and retaining only empirically successful instances. Formally, this process is defined as:

$$\mathcal{D}_{\text{exp}} = \{(G(c), \mathcal{Q}) \mid S(\mathcal{Q}, G(c)) = 1\}, \quad (16)$$

where $\mathcal{Q} \in \mathbb{Q}_{\text{base}}$, $c \in \mathbb{C}_{\text{complex}}$ is a configuration blueprint specifying the high-level attributes of a graph, such as topology and agent count. For example, a configuration could be defined as $c = (\text{'star'}, 6, \mathcal{R})$. Note that the “agent_num” parameter here does not limit the core capability of ARG-DESIGNER; rather, it constrains the data generation process to ensure a rich diversity of graph structures in the training data, from which the model learns generalizable collaborative patterns, not fixed sizes. $G(\cdot)$ is a deterministic function that maps a configuration c to a specific graph instance \mathcal{G} , and $S(\mathcal{Q}, \mathcal{G}) \in \{0, 1\}$ is an indicator function that verifies the empirical success of the graph for the given task. This initial phase allows the model to learn fundamental collaborative patterns in an unconstrained and resource-abundant environment, which ensures the model has generalizable graph construction abilities.

In the second phase, an efficiency dataset \mathcal{D}_{eff} is built to teach the model to generate more economical graphs. \mathcal{D}_{eff} is a heterogeneous mixture composed of three sources:

$$\mathcal{D}_{\text{eff}} = \mathcal{D}_{\text{simple}} \cup \mathcal{D}_{\text{pruned}} \cup \mathcal{D}_{\text{replay}}. \quad (17)$$

\mathcal{D}_{eff} includes natively efficient graphs from simple configurations $\mathcal{D}_{\text{simple}}$, successful graphs derived from pruning the dense structures generated by the Phase 1 model $\mathcal{D}_{\text{pruned}}$, and a subset of the original exploration data $\mathcal{D}_{\text{replay}}$ to prevent catastrophic forgetting. More specifically: ① $\mathcal{D}_{\text{simple}}$, which contains task-graph pairs generated from a predefined set of minimal, manually-verified configurations known to be efficient; ② $\mathcal{D}_{\text{pruned}}$, created by taking the overly complex but functional graphs from \mathcal{D}_{exp} , systematically removing individual nodes or edges, and retaining any pruned versions that still successfully complete the task; ③ $\mathcal{D}_{\text{replay}}$, a random subset of the initial \mathcal{D}_{exp} dataset, included to prevent the model from forgetting the fundamental patterns learned in the first phase. With the carefully designed dataset for the second training stage, ARG-DESIGNER learns to strike a desirable

Algorithm 1: The Inference algorithm of ARG-DESIGNER

```

1: Input: Task query  $\mathcal{Q}$ , trained model  $P_\theta$ 
2: Output: Collaboration graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ 
3: Initialize  $\mathcal{V} \leftarrow \emptyset, \mathcal{E} \leftarrow \emptyset, i \leftarrow 1$ 
4: loop
5:   Sample agent role  $r_i \sim P_\theta(v_i | \mathcal{G}_{<i}, \mathcal{Q})$ 
6:   if  $r_i = \text{END}$  or  $i > N_{\max}$  then
7:     break
8:   end if
9:    $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_i\}$ 
10:  for  $j = i - 1$  down to 1 do
11:    Sample edge existence  $b_{ji} \sim P_\theta(e_{ji} | v_i, \mathcal{G}_{<i}, \mathcal{Q})$ 
12:    if  $b_{ji} = 1$  then
13:       $\mathcal{E} \leftarrow \mathcal{E} \cup \{e_{ji}\}$ 
14:    end if
15:  end for
16:   $i \leftarrow i + 1$ 
17: end loop
18: return  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ 

```

Table 2: Dataset descriptions and statistics.

Category	Dataset	Answer Type	Metric	#Test	License
General reasoning	MMLU	Multi-choice	Acc.	153	MIT License
Math reasoning	GSM8K	Number	Acc.	1,319	MIT License
	MultiArith	Number	Acc.	600	Unspecified
	SVAMP	Number	Acc.	1,000	MIT License
Code generation	AQuA	Multi-choice	Acc.	254	Apache-2.0
	HumanEval	Code	Pass@1	164	MIT License

balance between correctness and simplicity, producing high-quality topologies with minimal redundancy.

Model Training. The training objective is to maximize the conditional log-likelihood of the ground-truth graphs in a given dataset \mathcal{D} . The model parameters θ are optimized by minimizing the negative log-likelihood (NLL) loss:

$$\mathcal{L}(\theta) = - \sum_{(\mathcal{G}, \mathcal{Q}) \in \mathcal{D}} \log P_\theta(\mathcal{G} | \mathcal{Q}). \quad (18)$$

Following the autoregressive factorization, the above loss is decomposed into a node generation term and an edge generation term. The final training loss is a weighted sum of these two terms:

$$\mathcal{L}_{\text{total}} = \alpha \cdot \mathcal{L}_{\text{node}} + (1 - \alpha) \cdot \mathcal{L}_{\text{edge}}, \quad (19)$$

where $\alpha \in [0, 1]$ is a hyperparameter balancing the two objectives. The individual loss terms are defined as the NLL over all node and edge generation steps, respectively:

$$\mathcal{L}_{\text{node}} = - \sum_{(\mathcal{G}, \mathcal{Q}) \in \mathcal{D}} \sum_{i=1}^{|\mathcal{V}|} \log P_\theta(v_i | \mathcal{G}_{<i}, \mathcal{Q}), \quad (20)$$

$$\mathcal{L}_{\text{edge}} = - \sum_{(\mathcal{G}, \mathcal{Q}) \in \mathcal{D}} \sum_{i=1}^{|\mathcal{V}|} \sum_{j=1}^{i-1} \log P_\theta(e_{ji} | v_i, \mathcal{G}_{<i}, \mathcal{Q}). \quad (21)$$

Throughout training, we employ a teacher forcing strategy, feeding the model ground-truth structures at each step to

Table 3: Performance comparison (%) on six benchmarks. The best results are highlighted in **bold**.

Method	MMLU	GSM8K	AQuA	MultiArith	SVAMP	HumanEval	Average
Vanilla	80.39	82.30	71.06	93.09	86.55	71.39	80.80
CoT SC (CoT)	81.69 $\uparrow_{1.30}$ 83.66 $\uparrow_{3.27}$	86.50 $\uparrow_{4.20}$ 81.60 $\downarrow_{0.70}$	73.58 $\uparrow_{2.52}$ 75.63 $\uparrow_{4.57}$	93.25 $\uparrow_{0.16}$ 94.12 $\uparrow_{1.03}$	87.36 $\uparrow_{0.81}$ 88.59 $\uparrow_{2.04}$	74.67 $\uparrow_{3.28}$ 79.83 $\uparrow_{8.44}$	82.84 $\uparrow_{2.04}$ 83.91 $\uparrow_{3.11}$
Chain	83.01 $\uparrow_{2.62}$	88.30 $\uparrow_{6.00}$	74.05 $\uparrow_{2.99}$	93.27 $\uparrow_{0.18}$	87.17 $\uparrow_{0.62}$	81.37 $\uparrow_{9.98}$	84.53 $\uparrow_{3.73}$
Tree	81.04 $\uparrow_{0.65}$	85.20 $\uparrow_{2.90}$	71.23 $\uparrow_{0.17}$	93.68 $\uparrow_{0.59}$	88.91 $\uparrow_{2.36}$	80.53 $\uparrow_{9.14}$	83.43 $\uparrow_{2.63}$
Complete	82.35 $\uparrow_{1.96}$	80.10 $\downarrow_{2.20}$	72.95 $\uparrow_{1.89}$	94.53 $\uparrow_{1.44}$	84.01 $\downarrow_{2.54}$	79.03 $\uparrow_{7.64}$	82.16 $\uparrow_{1.36}$
Random	84.31 $\uparrow_{3.92}$	86.90 $\uparrow_{4.60}$	76.48 $\uparrow_{5.42}$	94.08 $\uparrow_{0.99}$	87.54 $\uparrow_{0.99}$	82.66 $\uparrow_{11.27}$	85.33 $\uparrow_{4.53}$
LLM-Debate	84.96 $\uparrow_{4.57}$	91.40 $\uparrow_{9.10}$	77.65 $\uparrow_{6.59}$	96.36 $\uparrow_{3.27}$	90.11 $\uparrow_{3.56}$	84.70 $\uparrow_{13.31}$	87.53 $\uparrow_{6.73}$
AgentPrune	85.07 $\uparrow_{4.57}$	91.10 $\uparrow_{8.80}$	80.51 $\uparrow_{9.45}$	94.65 $\uparrow_{1.56}$	90.58 $\uparrow_{4.03}$	86.75 $\uparrow_{15.36}$	88.09 $\uparrow_{7.29}$
AgentDropout	85.62 $\uparrow_{5.23}$	91.70 $\uparrow_{9.40}$	80.94 $\uparrow_{9.88}$	95.60 $\uparrow_{2.51}$	91.04 $\uparrow_{4.49}$	85.98 $\uparrow_{14.59}$	88.48 $\uparrow_{7.68}$
G-Designer	86.92 $\uparrow_{6.53}$	93.80 $\uparrow_{11.50}$	81.60 $\uparrow_{10.54}$	96.50 $\uparrow_{3.41}$	93.10 $\uparrow_{6.55}$	88.33 $\uparrow_{16.94}$	90.04 $\uparrow_{9.24}$
ARG-DESIGNER	89.54 $\uparrow_{9.15}$	94.37 $\uparrow_{12.07}$	86.45 $\uparrow_{15.39}$	98.93 $\uparrow_{5.84}$	95.63 $\uparrow_{9.08}$	91.74 $\uparrow_{20.35}$	92.78 $\uparrow_{11.98}$

stabilize and accelerate learning. We train ARG-DESIGNER following a two-phase process, see Fig. 2b. We begin with a **cold start** on \mathcal{D}_{exp} , followed by **efficiency fine-tuning** on \mathcal{D}_{eff} with a lower learning rate.

Inference. During inference, given a new task query \mathcal{Q} , the trained ARG-DESIGNER model generates a graph autoregressively without ground-truth guidance. The process, detailed in Algo. 1, starts with the encoded task embedding and enters a generation loop. It samples a new agent role at each step, and if it is not the special END token, it then sequentially samples the existence of incoming edges from all previously generated nodes. This loop continues until an END token is sampled or a maximum node count is reached.

Experiments

Experimental Setting

Datasets and Metrics. Following (Zhang et al. 2025b), we evaluated ARG-DESIGNER on three categories of datasets: **① General Reasoning:** MMLU (Hendrycks et al. 2020); **② Mathematical Reasoning:** GSM8K (Cobbe et al. 2021), MultiArith (Roy and Roth 2016), SVAMP (Patel, Bhattacharya, and Goyal 2021), and AQuA (Ling et al. 2017); **③ Code Generation:** HumanEval (Chen et al. 2021). The statistics of the datasets are shown in Table 2.

Baselines. We compare ARG-DESIGNER against various baselines, which can be grouped into four main categories: **① Single-agent methods**, including CoT (Wei et al. 2022) and Self-Consistency (Wang et al. 2022); **② MAS with fixed topologies**, such as Chain, Tree, Complete Graph, and Random Graph (Qian et al. 2024); **③ MAS with Debate** like LLM-Debate (Du et al. 2023), where multiple agents iteratively critique and refine responses in a structured process; **④ MAS with Learnable topologies**, which include AgentPrune (Zhang et al. 2025a), AgentDropout (Wang et al. 2025), and G-Designer (Zhang et al. 2025b).

Implementation Details. We access GPT models via the OpenAI API, primarily using gpt-4o-2024-08-06 (GPT-4o). We employ a summarizer agent to aggregate the history of dialogue and produce the final solution $a^{(K)}$, with

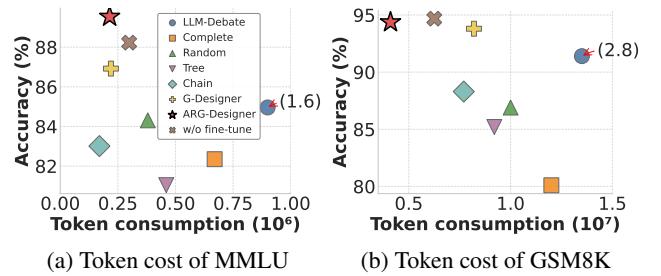


Figure 3: The prompt token cost comparison.

$K = 3$ for all baselines across all experiments. The node encoder is implemented using all-MiniLM-L6-v2 (Wang et al. 2020), with the embedding dimension set to $D = 384$. The hyperparameter α is set to 0.2 for all experiments. Following classical configurations in LLM-MAS (Zhuge et al. 2024; Yin et al. 2023; Zhang et al. 2025b), we provide explicit agent profiles for multi-agent methods and use GPT-4 to generate these profile pools. For all datasets, we use $B \in \{40, 60\}$ queries for model training.

Experimental Results

Performance Comparison. The comparison results are presented in Table 3, from which we have the following observations. **①** ARG-DESIGNER achieves the best performance across all six benchmarks, consistently outperforming a wide range of baselines. The superior performance demonstrates the effectiveness of the autoregressive graph generation paradigm in MAS topology design. **②** Compared to the strongest learning-based baseline, G-Designer, ARG-DESIGNER shows a significant performance gain. For instance, on AQuA, ARG-DESIGNER achieves an accuracy of 86.45%, surpassing G-Designer by a substantial margin of 4.85%. **③** When compared to debate-based methods like LLM-Debate, ARG-DESIGNER demonstrates a remarkable improvement of 8.8% on AQuA and 2.66% on GSM8K. This highlights the inefficiency of the fixed and all-to-all communication protocol.

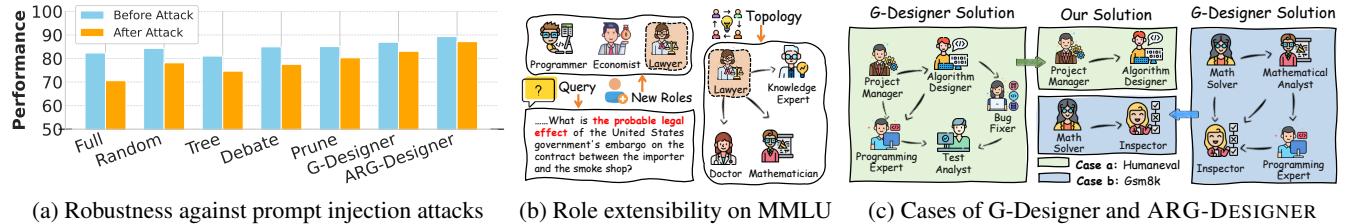


Figure 4: The robustness, extensibility of ARG-DESIGNER and case studies.

Table 4: Results of ablation study.

Method	MMLU	GSM8K	HumanEval	Average
Vanilla	80.39	82.30	71.39	78.02
ARG-DESIGNER	89.54	94.37	91.74	91.88
w/o fine-tune	88.23	94.68	90.91	91.27
w/o task embedding	86.93	93.12	89.26	89.77
w/o hist. embedding	88.23	93.59	90.08	90.63

Token Efficiency. A key benefit of ARG-DESIGNER is its ability to generate tailored topologies for different tasks, which prevents unnecessary complexity and thus minimizes token consumption. Fig. 3a and 3b illustrate the trade-off between performance and token cost. We can observe that: ① ARG-DESIGNER elegantly balances efficiency and performance. On the GSM8K dataset, ARG-DESIGNER is the most token-efficient method, using only 4.1e6 tokens, while achieving a top-tier accuracy of 94.37%. It surpasses the strong G-Designer baseline in accuracy while using approximately 50% fewer tokens. ② A general trend where more complex communication structures, such as the dialogue-based LLM-Debate, achieve relatively good performance but at an extremely high token cost. ③ A direct comparison between ARG-DESIGNER and its **w/o fine-tune** variant further underscores the value of the efficiency fine-tuning phase. For instance, on MMLU, fine-tuning improves accuracy from 88.23% to 89.54% while simultaneously cutting token usage by nearly 30%. On GSM8K, it reduces token consumption by a massive 34% (from 6.25e6 to 4.1e6). This demonstrates that our two-phase training strategy is highly effective at optimizing for both performance and efficiency.

Ablation Study. To validate the effectiveness of key components in ARG-DESIGNER, we investigate three major variants of ARG-DESIGNER: ① **w/o fine-tune**, where efficiency fine-tuning phase is removed. ② **w/o task embedding**, where the influence of task embedding f_Q is set to be 0. ③ **w/o hist. embedding**, where the historical embeddings are removed from the generation model.

The results in Table 4 demonstrate the contribution of each component: ① ARG-DESIGNER model achieves the highest average score of 91.88, significantly outperforming a vanilla baseline (78.02) that lacks these sophisticated mechanisms. ② The absence of task-specific guidance (**w/o task embedding**) causes the most significant performance degradation, which highlights that conditioning on the task is crucial for generating a bespoke and effective collaboration

topology. ③ The removal of historical embeddings (**w/o hist. embedding**) also leads to a noticeable decline to 90.63, confirming the value of modeling dependencies between agents. ④ Interestingly, the performance of the **w/o fine-tune** variant is highly competitive. This indicates that the model can learn fundamental collaboration patterns from the first stage. Nevertheless, the fine-tuning can bring significant benefits in terms of efficiency (see Fig. 3), while also slightly improving the performance.

Robustness Analysis. Following Zhuge et al. (2024), we evaluate the robustness of ARG-DESIGNER by simulating a system prompt attack, where an adversarial prompt is injected into a single agent to disrupt its function. As illustrated in Fig. 4a, this attack causes significant performance degradation in MAS with fixed and naive topologies. In contrast, ARG-DESIGNER shows excellent robustness against attacks with the least performance degradation (2.15%). This resilience emerges from our training objective, which discourages brittle structures and guides the model to construct fault-tolerant topologies with distributed risk and redundant communication paths.

Extensibility Analysis. Furthermore, we examine the model’s extensibility. As depicted in Fig. 4b, we introduce several new roles, to the pre-trained model without any re-training. When presented with a legal question from the MMLU regarding a contract embargo, ARG-DESIGNER demonstrates its adaptability. Correctly identifies the high relevance of the newly added “Lawyer” role and dynamically generates a collaboration graph placing the lawyer at its core, coordinating with other experts. This case vividly illustrates that ARG-DESIGNER can seamlessly scale its capabilities by integrating new knowledge, creating effective specialized team structures on the fly.

Case Study. To further illustrate the advantages of ARG-DESIGNER over learning-based baselines (e.g., G-Designer (Zhang et al. 2024)), we conduct a comparative case study on representative cases in HumanEval and GSM8K. As shown in Fig. 4c. The key difference lies in the flexibility of composition versus the static design. G-Designer requires a predefined and fixed set of agents and a fixed agent count. Its solution graphs remain within this rigid template, regardless of task complexity. ARG-DESIGNER, in contrast, dynamically generates both the roles and their communication links from an extensible role pool. It adapts the number of agents and connections based on task needs. Therefore, ARG-DESIGNER constructs more ef-

ficient collaboration graphs with fewer agents and messages, cutting token usage without sacrificing accuracy.

Related Work

Autoregressive Graph Generation

Autoregressive models are a cornerstone of graph generation, tackling the intractable joint probability of a graph by factorizing it into a tractable sequence of conditional probabilities for nodes and edges (You et al. 2018; Liao et al. 2019). The design of these models, however, is critically dependent on the choice of node ordering. Initial models like GraphRNN (You et al. 2018) pioneered this sequential approach but relied on simple, fixed node orderings like Breadth-First Search (BFS). Subsequent work like GRAN (Liao et al. 2019) showed that exploring various ad-hoc ordering schemes could improve performance, but highlighted the theoretical challenge that these orderings can lead to a loose variational bound. To resolve this, Graph-GEN (Goyal, Jain, and Ranu 2020) proposed using a single, canonical order for a graph, though this creates a new mismatch problem as the generation process itself may not follow this canonical path. Beyond the pivotal node-ordering problem, other research has focused on improving scalability, with methods like BiGG (Dai et al. 2020) reducing generation time complexity and others proposing more scalable graph representations (Jang, Lee, and Ahn 2023). Another key direction is conditional generation, where models like CCGG (Ommi et al. 2022) learn to produce graphs based on a given class label, and hybrid architectures have also been explored, for instance by combining autoregressive models with diffusion models (Kong et al. 2023).

LLM-based Multi-Agent System

The advent of powerful Large Language Models (LLMs) has catalyzed a shift from single-agent systems to Multi-Agent Systems (MAS) for tackling complex problems (Qian et al. 2024; Zhu et al. 2024). The success of these systems hinges on their collaboration topology, the design of which has thus become a central research problem (Zhuge et al. 2024; Zhang et al. 2024).

Research into topology design has evolved from static to adaptive structures. Initial approaches adopted fixed, manually-designed topologies, such as chains that enforce a sequential workflow (Wei et al. 2022; Hong et al. 2023) or trees that facilitate structured exploration and deliberation (Yao et al. 2023). While foundational, the inherent rigidity of these static structures limits their adaptability across diverse tasks, often leading to sub-optimal performance. To address this, a recent line of work has focused on learning adaptive communication graphs. For instance, some approaches start with a dense, fully-connected graph and learn to prune it in a task-aware manner. AgentPrune (Zhang et al. 2025a) learns to remove redundant communication links, while AgentDropout (Wang et al. 2025) applies a dynamic dropout technique to both agents and edges. To move beyond simple pruning, more advanced methods leverage the expressive power of graph neural networks (GNNs), which have become a standard for learning on graph-structured

data (Li et al. 2024b; Chen et al. 2025). G-Designer (Zhang et al. 2025b), for instance, employs a GNN-based autoencoder to directly generate a query-dependent communication structure. While these methods achieve task-adaptivity in structuring communication, they are still fundamentally constrained by the initial template. This rigidity prevents the MAS from being truly bespoke, leading to the critical issues of *redundant composition* and *limited extensibility*.

Drawing inspiration from real-world practices of recruiting teams incrementally, our work departs fundamentally from modifying a predefined template. We instead propose *autoregressive graph generation*, a paradigm that constructs the collaboration graph from scratch. This allows our method, ARG-DESIGNER, to jointly determine both the system’s composition and structure, dynamically selecting agents from an extensible pool and establishing optimal communication links in a truly task-adaptive manner.

Conclusion

In this work, we addressed two limitations in the design of multi-agent systems: the redundant composition and limited extensibility inherent in template graph modification-based approaches. We formulate the collaboration topology design problem as determining both the system composition (i.e., the set of agents and their roles) and its communication structure through autoregressive graph generation. To operationalize this paradigm, we introduced ARG-DESIGNER, a novel autoregressive model that constructs a collaboration graph from scratch, conditioned on a natural language task query. Our approach empowers the model to dynamically determine the necessary number of agents, select appropriate roles from an extensible pool, and establish optimal communication links between them, resulting in a bespoke MAS topology precisely tailored to the specific demands of each task. Extensive experiments across six diverse benchmarks demonstrate that ARG-DESIGNER consistently outperforms existing methods, achieving new state-of-the-art performance while maintaining superior token efficiency.

References

- Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Chen, Q.; Li, S.; Liu, Y.; Pan, S.; Webb, G. I.; and Zhang, S. 2025. Uncertainty-Aware Graph Neural Networks: A Multi-hop Evidence Fusion Approach. *IEEE Transactions on Neural Networks and Learning Systems*.
- Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Dai, H.; Nazi, A.; Li, Y.; Dai, B.; and Schuurmans, D. 2020. Scalable deep generative modeling for sparse graphs. In *International conference on machine learning*, 2302–2312. PMLR.

- Du, Y.; Li, S.; Torralba, A.; Tenenbaum, J. B.; and Mordatch, I. 2023. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning*.
- Goyal, N.; Jain, H. V.; and Ranu, S. 2020. Graphgen: A scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, 1253–1263.
- Hendrycks, D.; Burns, C.; Basart, S.; Zou, A.; Mazeika, M.; Song, D.; and Steinhardt, J. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.
- Hong, S.; Zheng, X.; Chen, J.; Cheng, Y.; Wang, J.; Zhang, C.; Wang, Z.; Yau, S. K. S.; Lin, Z.; Zhou, L.; et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4): 6.
- Jang, Y.; Lee, S.; and Ahn, S. 2023. A simple and scalable representation for graph generation. *arXiv preprint arXiv:2312.02230*.
- Kong, L.; Cui, J.; Sun, H.; Zhuang, Y.; Prakash, B. A.; and Zhang, C. 2023. Autoregressive diffusion model for graph generation. In *International conference on machine learning*, 17391–17408. PMLR.
- Li, B.; Luo, Y.; Chai, C.; Li, G.; and Tang, N. 2024a. The Dawn of Natural Language to SQL: Are We Fully Ready? *Proceedings of the VLDB Endowment*, 17(11): 3318–3331.
- Li, S.; Liu, Y.; Chen, Q.; Webb, G. I.; and Pan, S. 2024b. Noise-resilient unsupervised graph representation learning via multi-hop feature quality estimation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 1255–1265.
- Liao, R.; Li, Y.; Song, Y.; Wang, S.; Hamilton, W.; Duvenaud, D. K.; Urtasun, R.; and Zemel, R. 2019. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32.
- Ling, W.; Yogatama, D.; Dyer, C.; and Blunsom, P. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.
- Ommi, Y.; Yousefabadi, M.; Faez, F.; Sabour, A.; Soleymani Baghshah, M.; and Rabiee, H. R. 2022. Ccgg: A deep autoregressive model for class-conditional graph generation. In *Companion Proceedings of the Web Conference 2022*, 1092–1098.
- Patel, A.; Bhattacharya, S.; and Goyal, N. 2021. Are NLP models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.
- Qian, C.; Xie, Z.; Wang, Y.; Liu, W.; Dang, Y.; Du, Z.; Chen, W.; Yang, C.; Liu, Z.; and Sun, M. 2024. Scaling large-language-model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*.
- Roy, S.; and Roth, D. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.
- Shen, X.; Liu, Y.; Dai, Y.; Wang, Y.; Miao, R.; Tan, Y.; Pan, S.; and Wang, X. 2025. Understanding the Information Propagation Effects of Communication Topologies in LLM-based Multi-Agent Systems. *arXiv preprint arXiv:2505.23352*.
- Song, C. H.; Wu, J.; Washington, C.; Sadler, B. M.; Chao, W.-L.; and Su, Y. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2998–3009.
- Wang, W.; Wei, F.; Dong, L.; Bao, H.; Yang, N.; and Zhou, M. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33: 5776–5788.
- Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; and Zhou, D. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Wang, Z.; Wang, Y.; Liu, X.; Ding, L.; Zhang, M.; Liu, J.; and Zhang, M. 2025. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. *arXiv preprint arXiv:2503.18891*.
- Wang, Z.; Zhang, H.; Li, C.-L.; Eisenschlos, J. M.; Perot, V.; Wang, Z.; Miculicich, L.; Fujii, Y.; Shang, J.; Lee, C.-Y.; et al. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*.
- Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.
- Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36: 11809–11822.
- Yin, Z.; Sun, Q.; Chang, C.; Guo, Q.; Dai, J.; Huang, X.; and Qiu, X. 2023. Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. *arXiv preprint arXiv:2312.01823*.
- You, J.; Ying, R.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Graphrnn: Generating realistic graphs with deep autoregressive models. In *International conference on machine learning*, 5708–5717. PMLR.
- Zhang, G.; Yue, Y.; Li, Z.; Yun, S.; Wan, G.; Wang, K.; Cheng, D.; Yu, J. X.; and Chen, T. 2025a. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *International Conference on Learning Representations*.
- Zhang, G.; Yue, Y.; Sun, X.; Wan, G.; Yu, M.; Fang, J.; Wang, K.; Chen, T.; and Cheng, D. 2025b. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *International Conference on Machine Learning*.
- Zhang, J.; Xiang, J.; Yu, Z.; Teng, F.; Chen, X.; Chen, J.; Zhuge, M.; Cheng, X.; Hong, S.; Wang, J.; et al. 2024. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.
- Zhong, L.; Wang, Z.; and Shang, J. 2024. Debug like a Human: A Large Language Model Debugger via Verifying

Runtime Execution Step by Step. In *Findings of the Association for Computational Linguistics ACL 2024*, 851–870.

Zhou, H.; Wan, X.; Sun, R.; Palangi, H.; Iqbal, S.; Vulić, I.; Korhonen, A.; and Arik, S. Ö. 2025. Multi-Agent Design: Optimizing Agents with Better Prompts and Topologies. *arXiv preprint arXiv:2502.02533*.

Zhu, Y.; Du, S.; Li, B.; Luo, Y.; and Tang, N. 2024. Are Large Language Models Good Statisticians? *arXiv preprint arXiv:2406.07815*.

Zhuge, M.; Wang, W.; Kirsch, L.; Faccio, F.; Khizbulin, D.; and Schmidhuber, J. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.