

# G-Designer: Architecting Multi-agent Communication Topologies via Graph Neural Networks

Guibin Zhang<sup>1,2\*</sup>, Yanwei Yue<sup>2\*</sup>, Xiangguo Sun<sup>3†</sup>, Guancheng Wan<sup>4</sup>,

Miao Yu<sup>5</sup>, Junfeng Fang<sup>5</sup>, Kun Wang<sup>5†</sup>, Dawei Cheng<sup>1,2</sup>

<sup>1</sup>Shanghai AI Laboratory <sup>2</sup>Tongji University <sup>3</sup>CUHK <sup>4</sup>Emory University <sup>5</sup>USTC

\*Equal Contribution †Corresponding Author

✉ Primary contact: guibinz@outlook.com

## Abstract

Recent advancements in large language model (LLM)-based agents have demonstrated that collective intelligence can significantly surpass the capabilities of individual agents, primarily due to well-crafted inter-agent communication topologies. Despite the diverse and high-performing designs available, practitioners often face confusion when selecting the most effective pipeline for their specific task: *Which topology is the best choice for my task, avoiding unnecessary communication token overhead while ensuring high-quality solution?* In response to this dilemma, we introduce **G-Designer**, an adaptive, efficient, and robust solution for multi-agent deployment, which dynamically designs task-aware, customized communication topologies. Specifically, **G-Designer** models the multi-agent system as a multi-agent network, leveraging a variational graph auto-encoder to encode both the nodes (agents) and a task-specific virtual node, and decodes a task-adaptive and high-performing communication topology. Extensive experiments on six benchmarks showcase that **G-Designer** is: (1) **high-performing**, achieving superior results on MMLU with accuracy at 84.50% and on HumanEval with pass@1 at 89.90%; (2) **task-adaptive**, architecting communication protocols tailored to task difficulty, reducing token consumption by up to 95.33% on HumanEval; and (3) **adversarially robust**, defending against agent adversarial attacks with merely 0.3% accuracy drop. The code is available at <https://github.com/yanweiyue/GDesigner>.

## Keywords

Multi-agent network, Graph machine learning, LLM-based agent

### ACM Reference Format:

Guibin Zhang<sup>1,2\*</sup>, Yanwei Yue<sup>2\*</sup>, Xiangguo Sun<sup>3†</sup>, Guancheng Wan<sup>4</sup>, Miao Yu<sup>5</sup>, Junfeng Fang<sup>5</sup>, Kun Wang<sup>5†</sup>, Dawei Cheng<sup>1,2</sup>. 2018. G-Designer: Architecting Multi-agent Communication Topologies via Graph Neural Networks. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX')*. ACM, New York, NY, USA, 11 pages. <https://doi.org/XXXXXX.XXXXXXX>

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

Conference acronym 'XX', June 03–05, 2018, Woodstock, NY

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/18/06

<https://doi.org/XXXXXX.XXXXXXX>

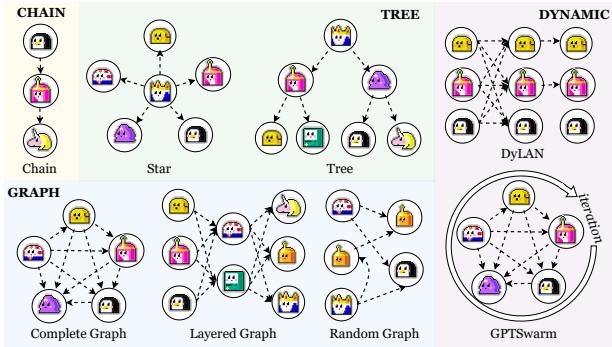


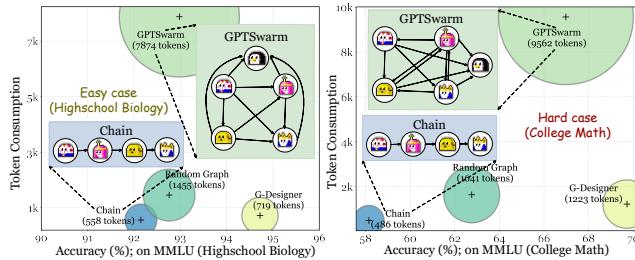
Figure 1: Existing practices for LLM-based multi-agent communication topology design.

## 1 Introduction

Web data, as a naturally occurring data structure, prevails in social networks [22, 63], trade networks [16, 21, 58], transportation systems [1, 18], and recommendation platforms [17, 68], etc. Web data can inherently be represented as graphs, where nodes and edges capture the topological relationships between numerous instances. Recently, there has been a surge of interest in the academic community toward optimizing the topology design for Large Language Model-based multi-agent (LLM-MA) systems, essentially, how to *weave the web of agents* [11].

An LLM-based agent, which integrates the language generation capabilities of LLMs with decision-making and action-execution functionalities [45, 53, 54], has exhibited impressive performance across a wide range of tasks, from reasoning [77] and code generation [59] to even more complex applications like video gaming [65] and autonomous driving [32]. Even more exciting, researchers have discovered that combining multiple LLM-based agents—whether implicitly or explicitly—into a team can outperform individual agents when tackling complex tasks [15, 31, 39, 59, 70, 73, 85], demonstrating a form of collaborative intelligence reminiscent of human teamwork in multi-agent systems [81]. This emergence of humanesque collective intelligence is fundamentally driven by the design of their topology, i.e., how multi-agents are *connected*, and how they *transmit, exchange, and assimilate* information reciprocally.

In practice, prior research has extensively explored how multiple instances of LLMs, referred to as agents [12, 20, 44, 66, 74], should be structured and organized to converse, collaborate, debate, or even compete. Various topological designs have been investigated, such as chain [26, 71], tree [73, 76], star [73], complete graphs [51], random graphs [51], optimizable graphs [80, 88], and



**Figure 2: The token consumption and accuracy of different multi-agent protocols on two subsets of MMLU dataset, “Highschool Biology” and “College Math”, tested with four gpt-4-based agents.**

LLM-based networks [23, 42]. These elaborately designed communication topologies have demonstrated remarkable performance with minimal human supervision, bridging the gap between individual intelligence and collective intelligence. Faced with numerous structures available, an inquisitive practitioner might ask: *how should I select or design a topology that best suits my task at hand?*

The question posed above is *non-trivial* and, at times, *perplexing*. A piece of experimental evidence is presented in Figure 2, where we evaluated the performance of different multi-agent structures on the MMLU dataset [24], a collection of multiple-choice questions across various subjects. The results reveal that even within the same dataset, the suitability of different communication topologies varies.

**① Simpler Case:** in the simpler “High School Biology” subset, the chain structure performs comparably to the complex GPTSwarm, while consuming significantly fewer tokens (0.5k versus 7.8k). In this case, the chain structure is clearly a more economical choice. **② Harder Case:** However, for the more challenging “College Mathematics” subset, GPTSwarm outperforms the chain structure by 8.75% ↑, primarily attributed to its intricate topology and prompt optimization. In summary, practitioners often find it challenging to *effortlessly identify the most efficient and complexity-adaptive multi-agent topology for a given task*.

In light of this dilemma, we propose the **LLM-based Multi-agent Communication Protocol (MACP)**, which aims to establish standardized guidance for future LLM-MA topology design:

**Multi-agent Communication Protocol (MACP):** Given a task/query  $q$ , an optimal LLM-MA communication topology for  $q$  should satisfy the following protocol logics: (1) **Effectiveness**: The communication structure must effectively produce the qualified solution for task  $q$ ; (2) **Complexity-adaptiveness**: The topology should dynamically adjust to the complexity of the task, minimizing communication overhead; (3) **Adversarial robustness**: The topology should maintain reliable under adversarial attacks.

The formal definition of MACP is provided in Section 3.3. To design a communication topology that ideally adheres to the MACP principles, we propose *an effective, adaptive, and robust LLM-powered multi-agent communication graph designer*, termed **G-Designer**. Technically, **G-Designer** first architectures a multi-agent graph, where each agent, along with its specific properties (e.g., profile [36], external API tools [87], or knowledge base [7]), is represented as a node, and communication between agents forms the edges. **G-Designer** employs a variational graph auto-encoder to encode the nodes (agents) along with task-specific information, and to decode the resulting collaboration network between agents. This

input-dependent paradigm allows **G-Designer** to design **task-adaptive, high-performing communication topology**, which is, at the same time, assured of efficiency and robustness with sparsity regularization. Unlike previous LLM-based multi-agent topology designs, which rely on a static structure for all queries/tasks, **G-Designer** adaptively crafts customized topologies for different domains and tasks, serving as a fully autonomous and flexible assistant for multi-agent system establishment and deployments.

Our contribution can be summarized as follows:

- ❶ **Protocol Proposal.** We propose the *first* communication protocol tailored for LLM-powered multi-agent systems, MACP, which comprehensively regulates multi-agent topology design across three dimensions: *performance, adaptability, and robustness*, and incisively highlights the shortcomings of existing designs.
- ❷ **Practical Solution.** We present **G-Designer**, an effective, adaptive, and robust designer of LLM-powered multi-agent communication graphs. By leveraging a variational graph auto-encoder to construct and process the multi-agent network, **G-Designer** decodes task-adaptive and high-performing agent communication, which is also equipped with strong robustness against agent-rooted adversarial attacks via dynamic topology adjustment.
- ❸ **Experimental Validation.** Extensive experiments across six benchmarks show that **G-Designer** is: (1) **high-performing**, achieving superior results on MMLU with accuracy at 84.50% and on HumanEval with pass@1 at 89.90%, surpassing state-of-the-art topologies by 0.20% ~ 4.10%; (2) **task-adaptive**, dynamically adjusting topology complexity with task awareness, outperforming state-of-the-art methods on MMLU with a cost of merely  $1.5e+5$  compared to their  $2.6e+6$ , reducing token consumption by up to 92.24%; and (3) **adversarially robust**, defending against agent adversarial attacks with merely 0.3% accuracy drop.

## 2 Related Works

### 2.1 LLM-agent Collaboration

While the academic community has widely recognized the success of single LLM-based agents in reasoning [2, 71, 76] and planning [29, 57, 62], collaboration among multiple LLM-based agents has swiftly emerged as a powerful approach for integrating the specialized capabilities of different agents, even exceeding the performance of individual LLMs [5, 6, 10, 14, 28, 46]. A basic form of collaboration is majority voting [8], where agents operate independently. However, more effective multi-agent collaboration should construct an interconnected system and iterative topology that encourages interdependent interactions and deliberate decision-making [8, 49]. Building on this insight, pioneering research has explored various multi-agent communication topologies, including: (1) **Non-interactive**, where agents operate independently without inter-agent communication, as employed in systems like LATM [82] and LLM-Debate [15]; (2) **Chain**, where agents are arranged in a sequential structure, each receiving the output from its predecessor and passing information to its successor, utilized by ChatDev [50], MetaGPT [26], and L2MAC [25]; (3) **Star**, where a central administrative agent (often referred to as a commander, manager, teacher, etc.) directs subordinate agents, seen in AutoGen [73], SecurityBot [75], and MiniGrid [86]; (4) **Tree**, where a root or parent agent hierarchically manages multiple child agents, as in SoA [30];

and (5) **Graph**, encompassing complete graphs [51, 88], layered graphs [23, 42, 51], and random graphs [51], among others.

## 2.2 Multi-agents as Graphs

Graphs, as a fundamental data structure for organizing and representing relationships between entities [3, 83], are widely adopted in the pre-LLM era as a powerful tool to facilitate effective communication in multi-agent reinforcement learning (MARL) [27, 41, 48]. With the rise of LLMs and the proliferation of LLM-based agents [5, 6, 10, 14, 28, 46], researchers have similarly recognized that interactions among multiple agents can naturally be modeled from a graph-based perspective [10, 42, 51, 88]. Early attempts are implicit, where ChatEval [5] and AutoGen [73] employ fixed graphs to facilitate information exchange among agents. Subsequent works, such as STOP [79] and DSPy [33], further explore joint optimization of prompts and inference structures. More recent practices including ChatLLM [23], DyLAN [42], GPTSwarm [88], and MacNet [51], have explicitly represented the organization of multiple agents as a graph. Specifically, both ChatLLM [23] and DyLAN [42] utilize a multilayer perception (MLP)-like layered graph, while MacNet [51] systematically evaluates various predefined topologies. GPTSwarm [88] parameterizes and optimizes the fully-connected graph distribution. However, all these attempts, whether predefined static topologies or those iteratively optimized, remain *input-independent*. Consequently, they fail to be task-aware and adaptively design topologies that suit the complexity of the specific task.

## 3 Formalization

This section establishes the notation, formalizes key concepts in multi-agent systems from a topology perspective, and formally defines our proposed multi-agent communication protocol.

### 3.1 Topology Structure

We model the multi-agent system as a directed graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V} = \{v_1, v_2, \dots, v_N\}$  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, which can be formalized as:

$$v_i = \{\text{Base}_i, \text{Role}_i, \text{State}_i, \text{Plugin}_i\}, \quad (1)$$

where each agent  $v_i$  is composed of four key elements: (1)  $\text{Base}_i$ , the language model instance powering  $v_i$ ; (2)  $\text{Role}_i$ , the agent's pre-assigned role or function; (3)  $\text{State}_i$ , representing the agent's accumulated knowledge and interaction history; and (4)  $\text{Plugin}_i$ , a set of external tools or plugins available to  $v_i$ , such as web searchers [43], code compilers [4, 26, 30, 54, 73], or file readers [54, 88]. Each LLM-based agent  $v_i$  receives prompt  $\mathcal{P}$  and generates response  $\mathcal{R}_i$ :

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

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 (non-symmetric) 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 agent  $v_i$  to agent  $v_j$ .

## 3.2 Communication Pipeline

Given a query/problem  $Q$ , 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  $\phi$  is applied to determine the execution index for each agent:

$$\phi : \mathcal{G} \longmapsto \sigma, \quad \sigma = [v_{\sigma_1}, v_{\sigma_2}, \dots, v_{\sigma_N}], \quad (3)$$

s. t.  $\forall i > j, \quad v_{\sigma_i} \notin \mathcal{N}_{\text{in}}(v_{\sigma_j})$ ,

where  $\sigma$  is the execution sequence of agents,  $\mathcal{N}_{\text{in}}(v_{\sigma(j)})$  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:

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

where  $\mathcal{R}_i^{(t)}$  represents the output of agent  $v_i$ , 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  $Q$  and messages from other agents. At the end of each dialogue round, a certain aggregation function is adopted to generate the answer/solution  $a^{(t)}$  based on the dialogue history:

$$a^{(t)} \leftarrow \text{Aggregate}(\mathcal{R}_1^{(t)}, \mathcal{R}_2^{(t)}, \dots, \mathcal{R}_N^{(t)}). \quad (5)$$

The implementation of the Aggregate function is flexible, with possible options including majority voting [8, 37, 88], aggregating all agents' responses and delegating one agent to provide the final answer [31, 42, 73, 80], or simply using the output of the last agent  $\mathcal{R}_{\sigma_N}^{(t)}$  [51]. Through  $K$  rounds of utterances, either predefined [51] or determined by an early-stopping mechanism [42], the overall system  $\mathcal{G}$  produces the final answer  $a^{(K)}$  in response to  $Q$ .

## 3.3 MACP Protocol

We give the formal definition of MACP Protocol as follows:

**DEFINITION 1 (MULTI-AGENT COMMUNICATION PROTOCOL).** *Given an LLM-based multi-agent system  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , we establish the following objective as an optimization principle or protocol:*

$$\min_{\mathcal{G} \in \mathbb{G}} \text{MACP}_{\beta}(\mathcal{G}) \triangleq \left[ -u(\mathcal{G}(Q)) + \beta_1 \cdot ||\mathcal{G}|| + \beta_2 \cdot |\hat{\mathcal{G}}(\hat{Q}) - \mathcal{G}(Q)| \right], \quad (6)$$

where  $\mathbb{G}$  represents the feasible parameter space of  $\mathcal{G}$ ,  $\beta = \{\beta_1, \beta_2\}$ ,  $u(\cdot)$  is the utility evaluator,  $||\mathcal{G}||$  measures the computational and communication overhead of the entire graph, and  $\hat{Q}$  and  $\hat{\mathcal{G}}$  denote the query description and the multi-agent system after adversarial perturbation, respectively. The first term in Equation (6) corresponds to **high performance**, aiming to maximize the utility of the system's output; the second term addresses **task-adaptiveness**, seeking to minimize system complexity to reduce power consumption and economic cost; and the third term focuses on **robustness**, constraining the deviation of system output under adversarial attacks.

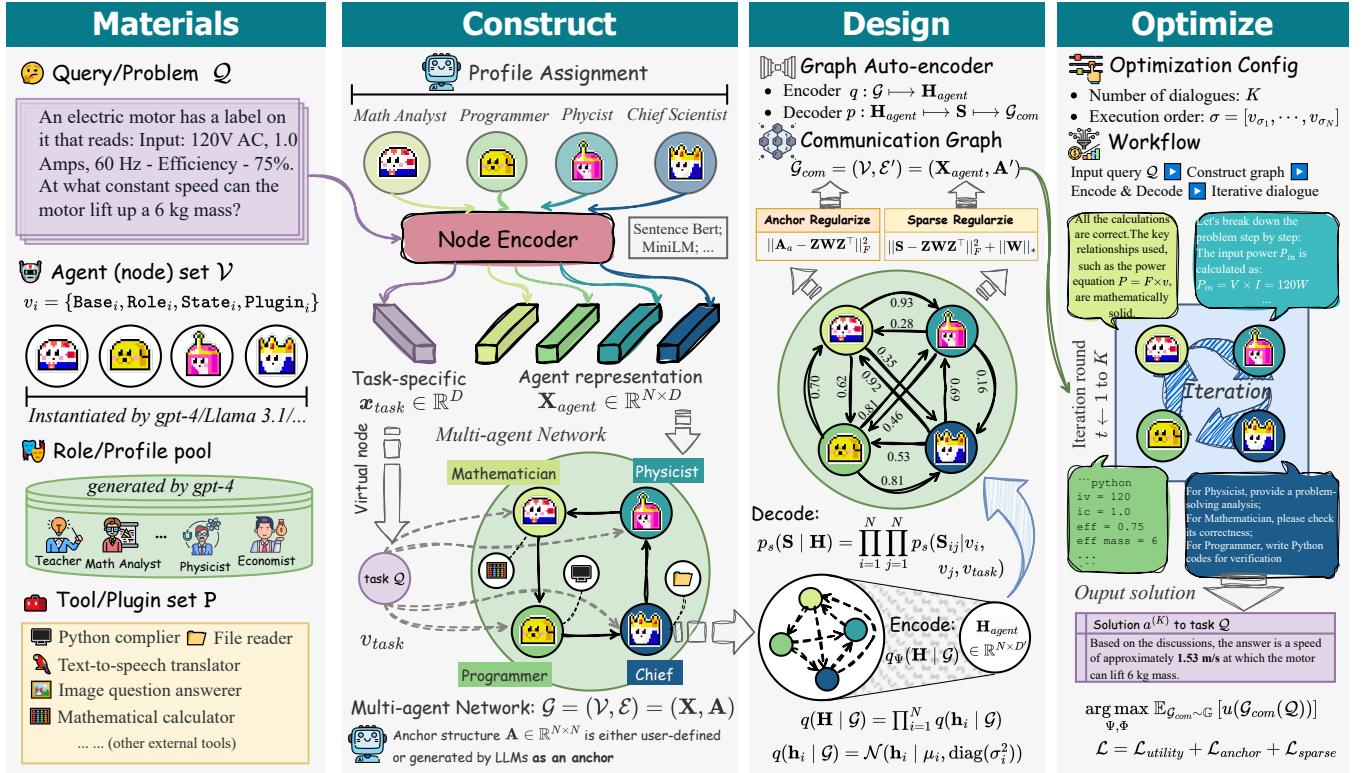


Figure 3: The designing workflow of our proposed G-Designer.

## 4 G-Designer

Figure 3 illustrates how **G-Designer** adaptively designs communication topologies for any given query. Specifically, the process begins with a few “raw materials”: the input query  $Q$ , the agent set  $\mathcal{V}$ , the profile pool, and the toolset. In the *Construct* stage, **G-Designer** leverages a node encoder to construct a multi-agent network along with a task-specific virtual node. In the *Design* stage, a graph auto-encoder is employed to decode the communication graph topology  $\mathcal{G}_{com}$ , which is leveraged for multi-round inter-agent collaboration in the *Optimize* stage.

### 4.1 Multi-agent Network Construction

Given an input query  $Q$  and a set of LLM-agents  $\mathcal{V}$ , **G-Designer** aims to design a task-adaptive and effective communication topology  $\mathcal{G}_{com}$ . We begin by assigning each agent a unique role and profile, as previous research [70] has shown that assigning distinct personas or roles to LLM-based agents can enhance cognitive synergy. Based on these roles, different external tools are allocated to the agents (e.g., Mathematica for a math analyst, Python compiler for a programmer). Thus, we successfully initialize each agent  $v_i$  as  $\{Base_i, Role_i, State_i, Plugin_i\}$ , as defined in Equation (1).

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

into a fixed-length embedding representation:

$$x_i \leftarrow \text{NodeEncoder}(\mathcal{T}(Base_i), Role_i, \mathcal{T}(Plugin_i)), \quad (7)$$

where  $\mathcal{T}(\cdot)$  extracts the textual description of the agent’s LLM backbone and its assigned plugins, and NodeEncoder can be realized using small and lightweight text embedding models such as SentenceBERT [52] or MiniLM [67]. After encoding the individual agents, we aim to ensure that the multi-agent network incorporates information related to the query  $Q$ , as this query-dependent approach enables **G-Designer** to be task-aware and adaptive. To this end, we introduce an additional *task-specific virtual global node*  $v_{task}$ , which is bidirectionally connected to all agent nodes, enabling a global “storage sink” and facilitating smoother information flow among agents [55, 60, 64]. This task node is encoded by the NodeEncoder as follows:  $x_{task} \leftarrow \text{NodeEncoder}(Q)$ .

After obtaining the agent node features  $\mathbf{X}_{agent} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^T$  and the task-specific embedding  $\mathbf{x}_{task}$ , we provide a simple *anchor topology*  $\mathbf{A}_{anchor} \in \{0, 1\}^{N \times N}$ , which serves as a starting point for **G-Designer**’s topology design process. For instance, given a code generation task with three agents: manager/programmer/code reviewer, the anchor topology could be configured as a chain structure, i.e., “manager → programmer → reviewer”, reflecting the typical workflow of code completion. The anchor topology, being either user-defined or automatically generated by LLMs, is often simple and sub-optimal. However, it provides a foundational reference and prior knowledge for **G-Designer**’s subsequent optimization process. We incorporate the task-specific vertex  $v_{task}$  and its corresponding edges and obtain  $\tilde{\mathbf{A}}_{anchor} \in \{0, 1\}^{(N+1) \times (N+1)}$ .

Consequently, we establish a task-specific multi-agent network  $\tilde{\mathcal{G}}$ :

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

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

## 4.2 Designing Communication Topology

Building upon the task-specific multi-agent network  $\tilde{\mathcal{G}}$ , **G-Designer** seeks to establish a more fine-grained and precise communication topology  $\mathcal{G}_{com}$ . Drawing inspiration from the variational graph auto-encoder (VGAE) framework [34, 84], **G-Designer** employs a VGAE-based encoder-decoder  $f_v$  to generate the multi-agent interaction topology, which is formulated as:

$$\mathcal{G}_{com} = f_v(\tilde{\mathcal{G}}; \Theta_v) = p(\mathcal{G}_{com} | \mathbf{H})q(\mathbf{H} | \tilde{\mathbf{X}}, \tilde{\mathbf{A}}_{anchor}), \quad (9)$$

where  $f_v$  is the encoder-decoder architecture with parameters  $\Theta_v$ ,  $q(\cdot)$  is the encoder module,  $p(\cdot)$  is the decoder module. The encoder utilizes posterior probabilities to encode the node embeddings into low-dimensional latent vector representations  $\mathbf{H}_{agent}$ , which can be formulated as:

$$\begin{aligned}q(\mathbf{H}_{agent} | \tilde{\mathbf{X}}, \tilde{\mathbf{A}}_{anchor}) &= \prod_{i=1}^N q(\mathbf{h}_i | \tilde{\mathbf{X}}, \tilde{\mathbf{A}}_{anchor}), \\ q(\mathbf{h}_i | \tilde{\mathbf{X}}, \tilde{\mathbf{A}}_{anchor}) &= \mathcal{N}(\mathbf{h}_i | \boldsymbol{\mu}_i, \text{diag}(\boldsymbol{\sigma}_i^2)),\end{aligned}\quad (10)$$

where  $\boldsymbol{\mu} = \text{GNN}_\mu(\tilde{\mathbf{X}}, \tilde{\mathbf{A}}_{anchor}; \Theta_\mu)$  is the matrix of mean vectors  $\boldsymbol{\mu}_i$ ; similarly  $\log(\boldsymbol{\sigma}) = \text{GNN}_\sigma(\tilde{\mathbf{X}}, \tilde{\mathbf{A}}_{anchor}; \Theta_\sigma)$ . The choice of GNN backbone can be customized as needed; here, we utilize a simple two-layer GCN [35].  $\mathbf{h}_i$ ,  $\boldsymbol{\mu}_i$ , and  $\boldsymbol{\sigma}_i$  denote the  $i$ -th column of  $\mathbf{H}$ ,  $\boldsymbol{\mu}$ , and  $\boldsymbol{\sigma}$ , respectively. The encoder  $q(\cdot)$  is parameterized by  $\Theta_e = \{\Theta_\mu, \Theta_\sigma\}$ . Following the encoding phase, the decoder employs the latent representations to generate a comprehensive blueprint for multi-agent communication. More specifically, the decoder  $q(\cdot) = q_c \circ q_s$  first constructs a parameterized, sketched graph  $\mathbf{S}$ , which is then refined into the final multi-agent communication topology:

$$p(\mathcal{G}_{com} | \mathbf{H}_{agent}) = \int_{\mathbf{S}} p_c(\mathcal{G}_{com} | \mathbf{S}) p_s(\mathbf{S} | \mathbf{H}_{agent}) d\mathbf{S}. \quad (11)$$

At the first step,  $p_s(\cdot)$  constructs the fully-connected sketched adjacency matrix  $\mathbf{S}$  from the latent representation  $\mathbf{H}_{agent}$ :

$$p_s(\mathbf{S} | \mathbf{H}_{agent}) = \prod_{i=1}^N \prod_{j=1}^N p_s(S_{ij} | \mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{task}; \Theta_d), \quad (12)$$

whose detailed derivation is as follows:

$$\begin{aligned}p_s(S_{ij} = 1 | \mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{task}) &= g(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{task}), \\ &= \text{Sigmoid}((\log(\epsilon) - \log(1 - \epsilon) + \varpi_{ij}) / \tau),\end{aligned}\quad (13)$$

where  $\varpi = \text{FFN}_d([\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{task}])$  with  $\text{FFN}_d$  parameterized by  $\Theta_d$ ,  $\epsilon \sim \text{Uniform}(0, 1)$ , and  $\tau$  denotes the temperature coefficient. When  $\tau$  approaches zero, Equation (13) essentially return the Bernoulli sampling result for  $S_{ij}$ . The resulting matrix  $\mathbf{S} \in [0, 1]^{N \times N}$  represents a densely-connected, non-negative graph distribution, indicating an overly complex and resource-intensive pair-wise communication structure, which is not yet suitable for guiding multi-agent collaboration. To align with **G-Designer**'s objectives of task adaptiveness

and minimizing costs, we apply a refinement decoder  $p_c(\cdot)$  to refine the sketched  $\mathbf{S}$  into a compact, sparse, and highly informative communication graph, instantiated by a regularization objective:

$$\begin{aligned}p_c : \arg \max_{\tilde{\mathbf{S}} \in \mathbb{S}} 1/2\|\mathbf{S} - \mathbf{Z}\mathbf{W}\mathbf{Z}^\top\|_F^2 + \zeta\|\mathbf{W}\|_* + \\ 1/2\|\mathbf{A}_{anchor} - \mathbf{Z}\mathbf{W}\mathbf{Z}^\top\|_F^2, \text{ s.t. } \tilde{\mathbf{S}} = \mathbf{Z}\mathbf{W}\mathbf{Z}^\top,\end{aligned}\quad (14)$$

where  $\mathbf{Z} \in \mathbb{R}^{N \times r}$  is the top- $r$  columns of left singular matrix  $\mathbf{S}$ ,  $\zeta$  is a coefficient hyperparameter,  $\mathbf{W} \in \mathbb{R}^{r \times r}$  is an optimizable weight matrix,  $\|\cdot\|_F$  denotes the Frobenius norm and  $\|\mathbf{W}\|_* = \sum_i \lambda_i$  where  $\lambda_i$  is the  $i$ -th singular value of  $\mathbf{W}$ .  $\tilde{\mathbf{S}} \in \mathbb{R}^{N \times N}$  is the desired sparse topology, which is decomposed as  $\mathbf{Z}\mathbf{W}\mathbf{Z}^\top$ . In Equation (14), the first and second terms are jointly denoted as *anchor regularization*, which encourage the learned  $\tilde{\mathbf{S}}$  to maintain similarity with both the original  $\mathbf{S}$  and the anchor topology. The third term, denoted as *sparsity regularization*, though appearing to minimize the nuclear norm of  $\mathbf{W}$ , essentially sparsifies  $\tilde{\mathbf{S}}$ , since  $\|\tilde{\mathbf{S}}\|_* = \|\mathbf{W}\|_*$  holds due to  $\mathbf{Z}^\top \mathbf{Z} = \mathbb{I}_{r \times r}$ . Therefore, Equation (14) achieves two key goals: (1) producing a sparse, refined communication topology, and (2) constraining the design to remain grounded in practical intuition. The resulting communication design can be represented as follows:

$$\mathcal{G}_{com} = (\mathcal{V}, \mathcal{E}_{com}), \mathcal{E}_{com} = \{(i, j) | \tilde{S}_{ij} \neq 0 \wedge (i, j) \in \mathcal{E}\}. \quad (15)$$

At this stage, we have successfully distilled a lightweight and informative collaboration network  $\mathcal{G}_{com}$  from the roughly constructed task-specific multi-agent network  $\tilde{\mathcal{G}}$ , which is now ready to guide inter-agent message passing in the following process.

## 4.3 Optimizing G-Designer

Upon obtaining  $\mathcal{G}_{com}$ , the multi-agent utterances and dialogues can proceed as usual using  $\mathcal{G}_{com}$ , as detailed in Section 3.2. After  $K$  rounds of interaction, the agents converge to a final solution  $a^{(K)} = \mathcal{G}_{com}(Q)$ . We then give the following optimization objective:

$$\arg \min_{\Theta_e, \Theta_d} \mathbb{E}_{\Theta_e, \Theta_d \sim \Omega} [u(\mathcal{G}_{com}(Q))], \quad (16)$$

where  $\Theta_e$  and  $\Theta_d$  are the parameters of the encoder  $q(\cdot)$  and decoder  $p(\cdot)$ , respectively,  $\Omega$  is the parameter space and  $\mathbb{E}(\cdot)$  denotes the mathematical expectation. Equation (16) aims to maximize the utility of the generated solution, but it is inherently intractable and non-differentiable, as  $u(\cdot)$  often depends on external API calls [24, 38]. To address this, following standard approaches in multi-agent structure design [80, 88], we apply policy gradient [72] to approximate and optimize Equation (16):

$$\nabla_{\Theta} \mathbb{E}_{\Theta \sim \Omega} [u(\mathcal{G}_{com}(Q))] \approx \frac{1}{M} \sum_{k=1}^M u(a_m^{(K)}) \nabla_{\Theta} (P(\mathcal{G}_k)), \quad (17)$$

where  $\Theta = \{\Theta_e, \Theta_d\}$ ,  $\{\mathcal{G}_k\}_{m=1}^M$  are independently samples from  $\mathcal{G}_{com}$ , and  $\{a_m^{(K)}\}_{m=1}^M$  are the corresponding output.  $P(\mathcal{G}_k)$  calculates the probability of  $\mathcal{G}_k$  being sampled, which can be expressed as  $P(\mathcal{G}_k) = \prod_{i=1}^N \prod_{j=1}^N \tilde{S}_{ij}$ . Through iterative optimization guided by Equations (14) and (16) over a limited set of queries as the “training set”, **G-Designer** efficiently develops task-awareness and the capability to strategically design the agent network, achieving truly task-customized multi-agent topology design.

**Table 1: Performance comparison with three types of baselines, including single-agent execution, spatial communication and temporal communication. The best results are highlighted in bold, and the runner-ups are underlined. All methods, except for the single-agent category, utilize five gpt-4-based agents. “Mul.”, “Ada.”, and “Rob.” indicate whether the method supports a multi-agent setting, whether it is task-adaptive, and whether it is adversarially robust, respectively. ✗, ✚ and ✓ signifies no/partial/full support in these aspects.**

Method	Mul.	Ada.	Rob.	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 <sup>↑0.51</sup>	87.17 <sup>↑1.77</sup>	94.79 <sup>↑1.64</sup>	88.32 <sup>↑1.14</sup>	73.91 <sup>↑3.57</sup>	75.52 <sup>↑3.84</sup>	83.73
ComplexCoT	✗	✗	✗	83.78 <sup>↑1.64</sup>	87.62 <sup>↑2.22</sup>	95.86 <sup>↑2.71</sup>	90.17 <sup>↑2.99</sup>	77.58 <sup>↑7.24</sup>	74.94 <sup>↑3.26</sup>	84.99
SC (CoT)	✗	✗	✗	82.66 <sup>↑0.52</sup>	87.93 <sup>↑2.53</sup>	96.88 <sup>↑3.73</sup>	88.69 <sup>↑1.51</sup>	75.08 <sup>↑4.74</sup>	77.30 <sup>↑5.62</sup>	84.75
SC (ComplexCoT)	✗	✗	✗	83.65 <sup>↑1.51</sup>	86.14 <sup>↓0.74</sup>	96.94 <sup>↑3.79</sup>	89.72 <sup>↑2.54</sup>	77.69 <sup>↑7.35</sup>	77.94 <sup>↑6.26</sup>	85.35
PHP	✓	✗	✗	83.45 <sup>↑1.31</sup>	<b>95.50</b> <sup>↑10.1</sup>	<u>98.10</u> <sup>↑2.84</sup>	90.02 <sup>↑3.44</sup>	79.00 <sup>↑8.66</sup>	82.96 <sup>↑11.36</sup>	88.17
Chain	✓	✗	✗	82.35 <sup>↑0.21</sup>	85.57 <sup>↑0.17</sup>	94.38 <sup>↑1.23</sup>	83.41 <sup>↓3.77</sup>	70.94 <sup>↑0.60</sup>	80.88 <sup>↑9.20</sup>	82.92
Star	✓	✗	✗	80.79 <sup>↓1.35</sup>	85.55 <sup>↑0.15</sup>	93.79 <sup>↓0.64</sup>	88.09 <sup>↑0.91</sup>	68.57 <sup>↓1.77</sup>	75.65 <sup>↑3.97</sup>	82.07
Tree	✓	✗	✗	81.89 <sup>↓0.25</sup>	84.56 <sup>↓0.84</sup>	94.60 <sup>↑1.45</sup>	89.25 <sup>↑2.07</sup>	72.84 <sup>↑2.50</sup>	77.38 <sup>↑5.70</sup>	83.42
Complete Graph	✓	✗	✗	83.15 <sup>↑1.01</sup>	86.49 <sup>↑1.09</sup>	97.20 <sup>↑4.05</sup>	89.48 <sup>↑2.30</sup>	<u>79.21</u> <sup>↑8.87</sup>	83.75 <sup>↑12.07</sup>	86.55
Random Graph	✓	✗	✗	83.76 <sup>↑1.62</sup>	86.14 <sup>↑0.74</sup>	95.46 <sup>↑2.31</sup>	85.41 <sup>↓1.77</sup>	74.07 <sup>↑3.73</sup>	82.66 <sup>↑10.98</sup>	84.58
AutoGen	✓	✗	✗	82.13 <sup>↓0.01</sup>	90.06 <sup>↑7.92</sup>	93.80 <sup>↑0.65</sup>	88.44 <sup>↓1.26</sup>	73.65 <sup>↑3.31</sup>	85.41 <sup>↑13.73</sup>	85.58
MetaGPT	✓	✗	✗	-	-	-	-	-	85.90 <sup>↑14.22</sup>	84.90
LLM-Blender	✓	✗	✗	81.22 <sup>↓0.92</sup>	89.17 <sup>↑3.77</sup>	94.27 <sup>↑1.12</sup>	88.77 <sup>↑1.59</sup>	77.05 <sup>↑6.71</sup>	-	86.09
LLM-Debate	✓	✗	✓	83.69 <sup>↑1.55</sup>	90.23 <sup>↑4.83</sup>	96.27 <sup>↑3.12</sup>	90.56 <sup>↑3.38</sup>	77.52 <sup>↑7.18</sup>	83.79 <sup>↑12.11</sup>	87.01
DyLAN	✓	✗	✓	80.16 <sup>↓1.98</sup>	88.16 <sup>↑2.76</sup>	94.27 <sup>↑1.12</sup>	87.40 <sup>↑0.22</sup>	74.16 <sup>↑3.82</sup>	<u>89.70</u> <sup>↑18.02</sup>	85.64
GPTSwarm	✓	✗	✓	83.98 <sup>↑1.84</sup>	89.74 <sup>↑4.34</sup>	97.84 <sup>↑4.69</sup>	86.42 <sup>↓0.76</sup>	78.16 <sup>↑7.82</sup>	88.49 <sup>↑16.81</sup>	87.32
<b>G-Designer</b>	✓	✓	✓	<b>84.50</b> <sup>↑2.36</sup>	95.07 <sup>↑9.67</sup>	<b>98.30</b> <sup>↑5.15</sup>	<b>91.85</b> <sup>↑4.67</sup>	<b>79.47</b> <sup>↑9.13</sup>	<b>89.90</b> <sup>↑18.22</sup>	<b>89.84</b>

*Optimization configuration.* The overall training objective of our method is formulated as  $\mathcal{L}_{\text{G-Designer}} = \mathcal{L}_{\text{utility}} + \mathcal{L}_{\text{anchor}} + \mathcal{L}_{\text{sparse}}$ , where  $\mathcal{L}_{\text{utility}}$  represents the optimization target from Equation (16),  $\mathcal{L}_{\text{anchor}}$  corresponds to the first and third terms in Equation (14), and  $\mathcal{L}_{\text{sparse}}$  is the second term. Given a benchmark  $\{Q_i\}_{i=1}^D$  consisting of  $D$  queries, **G-Designer** begins by optimizing with a small subset of  $D'$  queries and then fixes the learned parameters for testing on the remaining  $(D - D')$  queries. The whole algorithm workflow of **G-Designer** is depicted in Algorithm 1.

## 5 Experiments

In this section, we conduct extensive experiments to answer the following research questions:

- **(RQ1)** Can **G-Designer** design *effective* and *high-performing* multi-agent communication topologies?
- **(RQ2)** Can **G-Designer** generate more *task-adaptive* topologies, resulting in less token consumption?
- **(RQ3)** Is **G-Designer** more *robust* against adversarial attacks?
- **(RQ4)** How sensitive is the proposed **G-Designer** sensitive to its key components and parameters?

### 5.1 Experimental Setup

**5.1.1 Datasets and Metrics.** We evaluate **G-Designer**'s ability to enhance LLM-MA collaborative intelligence using three major categories of datasets: ■ **General Reasoning:** We utilize MMLU [24],

which provides a comprehensive set of logical reasoning assessments across diverse subjects in the form of multiple-choice questions. The performance is evaluated using *accuracy* on the generated solutions. ■ **Mathematical Reasoning:** To assess mathematical reasoning capabilities, we use GSM8K [13], MultiArith [56], SVAMP [47], and AQuA [40], with *accuracy* as the evaluation metric across all datasets. ■ **Code Generation:** We opt for HumanEval [9], a widely recognized benchmark for function-level code generation designed to evaluate fundamental programming skills. Performance is measured using *pass@1*, which reflects the correctness of generated functions across multiple test cases.

**5.1.2 Baselines.** We comprehensively select representative baselines from both single-agent enhancement and multi-agent collaboration methods. For single-agent approaches, we select:

- **COT** [71] equips an individual LLM with the capability to generate coherent intermediate reasoning steps.
- **ComplexCoT** [19] is built on CoT, introducing a complexity-based criterion that spans both the prompting (input selection) and decoding (output selection) stages.
- **Self-Consistency** [69] is a reasoning chain ensemble method, working in conjunction with COT or ComplexCoT to ensemble multiple reasoning chains. We use 5-way for experiments.
- **PHP** [85] is a progressive-hint prompting technique, and current state-of-the-art single agent reasoning plugin.

For multi-agent collaboration topologies, we select the following:

- **Chain, Star, and Tree** are simple and straightforward topological configurations, formally defined in [51].
- **Complete Graph** and **Random Graph** are also delineated in [51], whose execution order is defined by topological ordering.

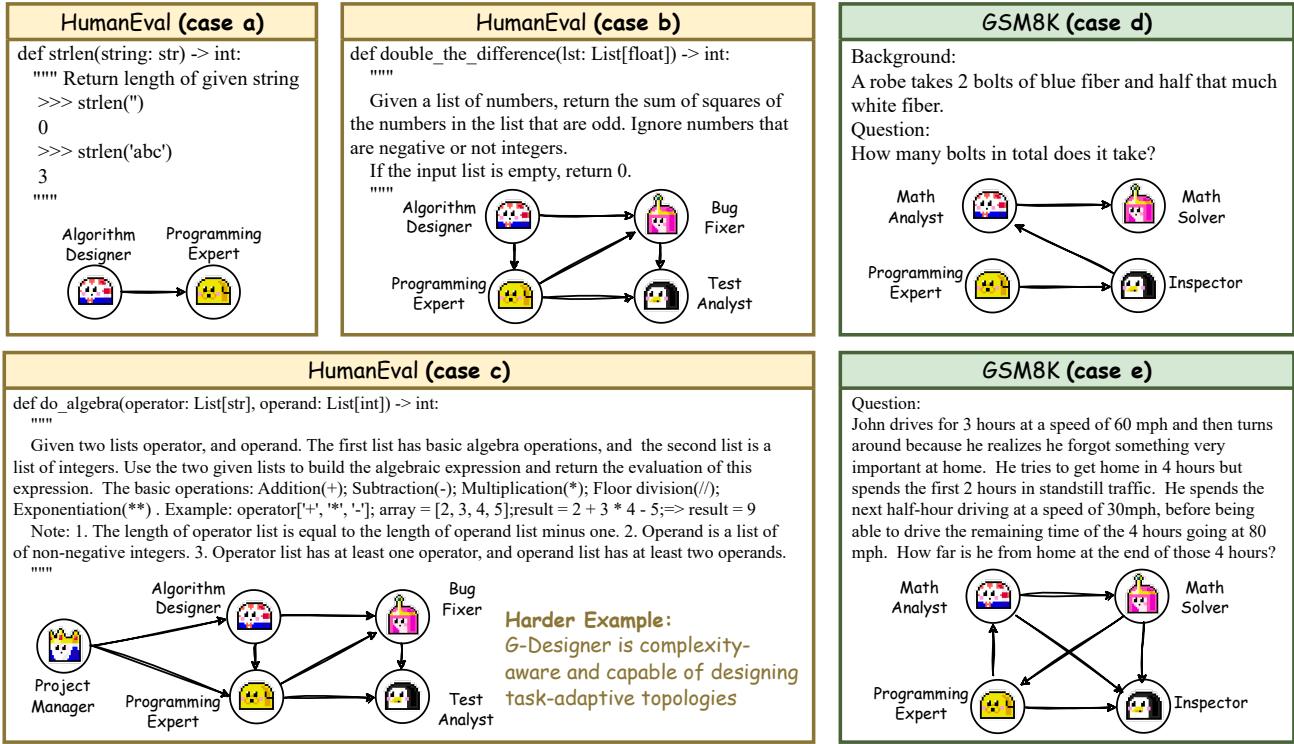


Figure 4: Case study of the communication topologies designed by G-Designer on HumanEval and GSM8K benchmarks.

- **AutoGen** [73] is one of the earliest frameworks for multi-agent collaboration. We primarily employ “A1: Math Problem Solving” for mathematical tasks, “A5: Dynamic Group Chat” for general reasoning, and “A4: Multi-Agent Coding” for code generation.
- **MetaGPT** [26] is a pioneering multi-agent framework specifically designed for software engineering. Therefore, we report its performance only on code generation tasks.
- **LLM-Debate** [15] enhances system reasoning capabilities by facilitating debates among multiple agents.
- **LLM-Blender** [31] utilizes a GenFuser agent to aggregate solutions from independently operating agents.
- **DyLAN** [42] optimizes agent teams by dynamically assessing the agent importance scores and selecting the most valuable ones.
- **GPTSwarm** [88] conceptualizes a swarm of LLM agents as computational graphs and continuously optimizes its distribution, albeit at a relatively high training cost.

5.1.3 *Implementation Details.* We access the GPT models via the OpenAI API, and mainly test on gpt-4-1106-preview (gpt-4). we set temperature to 0 for the single execution and single agent baselines and 1 for multi-agent methods. For decision-making in the multi-agent system, we set a summarizer agent to aggregate the dialogue history and produce the final solution  $a^{(K)}$ , with  $K = 3$  across all experiments. The NodeEncoder( $\cdot$ ) is implemented using all-MiniLM-L6-v2 [67], with the embedding dimension set to  $D = 384$ . The anchor topology  $A_{\text{anchor}}$  is predefined as a simple chain structure. The sampling times  $M$  are set as 10, and  $\tau = 1e - 2$  and  $\zeta = 1e - 1$  are set for all experiments. We provide explicit agent profiling for multi-agent methods, following the classical configurations in LLM-MA systems [42, 61, 78, 88], and use gpt-4

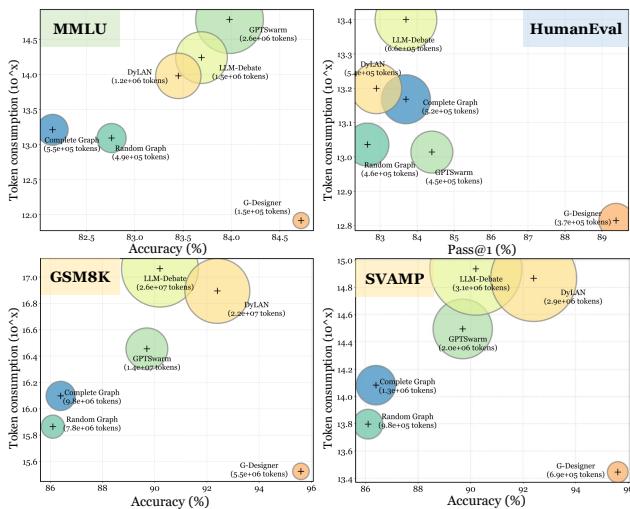
to generate agent profile pools. For all benchmarks, we merely use  $D' = 40$  queries for the optimization process.

## 5.2 Performance Evaluation (RQ1)

To assess the effectiveness of G-Designer in designing powerful LLM-MA topologies, we conducted evaluations using five instances of gpt-4, with results outlined in Table 1. We can draw two key observations (Obs.):

**Obs. ① Meticulously designed multi-agent topology is crucial for collective intelligence.** As demonstrated in Table 1, not all multi-agent topologies outperform single-agent reasoning approaches. In some cases, such as the Star and Tree structures, performance even falls short of vanilla gpt-4, with accuracy drops of 1.35% and 0.25%, respectively, on the MMLU dataset. However, more customized and adaptive topology designs, like AutoGen, DyLAN, and GPTSwarm, consistently show improvements of 16.81% ~ 18.02% on HumanEval, significantly outperforming single-agent baselines such as PHP and CoT. This clearly demonstrates the emergent power of collective intelligence.

**Obs. ② G-Designer is effective in designing powerful LLM-MA topologies.** As shown in Table 1, G-Designer achieves the best performance in five out of six benchmarks, and on the GSM8K benchmark, it trails only PHP with a 9.67% ↑ accuracy improvement. On the HumanEval benchmark, G-Designer surpasses MetaGPT, a specialized multi-agent code generation framework, by 4.0% at  $\text{pass}@1$ , and outperforms state-of-the-art multi-agent collaboration frameworks like GPTSwarm and DyLAN by margins of 0.20% ~ 1.41%. Overall, G-Designer demonstrates exceptional performance in topology design across a wide range of tasks.



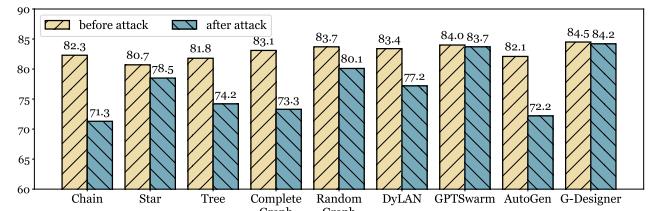
**Figure 5:** Visualization of the performance metrics and prompt token consumption of different multi-agent communication topologies across MMLU, HumanEval, GSM8K, and SVAMP benchmarks. The diameter of each point is proportional to its  $y$ -axis value.

### 5.3 Adaptiveness Evaluation (RQ2)

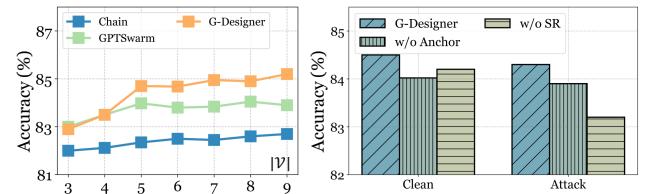
Task-adaptive multi-agent network design not only improves task performance but also regulates the complexity of the topology according to the task’s difficulty. A key benefit of this adaptivity is that it prevents the use of overly complex structures for simple tasks, thus minimizing unnecessary communication costs—in the case of LLM-MA, reducing token consumption. Figure 4 visualizes the different topologies designed by **G-Designer** for varying query difficulties on the HumanEval and GSM8K benchmarks, while Figure 5 compares the token consumption of **G-Designer** against various baselines. We summarize several interesting observations:

**Obs. ③ G-Designer is highly task-aware.** As shown in Figure 4, the multi-agent topologies generated by **G-Designer** are highly dependent on the specific task context and its difficulty. In *Case a*, despite having five gpt-4 agents available as design resources, **G-Designer** identified the task of designing a `strlen(string)` function as relatively simple. It streamlined the topology by removing unnecessary agents, such as “bug fixer” and “test analyst”, and retained only a minimal “Algorithm Designer → Programmer” structure to solve the problem. In contrast, for the more complex *Case c* and *Case e*, **G-Designer** crafted a more intricate communication graph. These cases highlight the strong task-aware and task-adaptive capabilities of **G-Designer**.

**Obs. ④ G-Designer is a token-saving and economical assistant.** Figure 5 illustrates the differences in prompt token consumption between **G-Designer** and several representative multi-agent designs. We observe that simpler topologies, such as complete graphs and random graphs, consume fewer tokens but show significantly weaker performance. More complex communication structures, like GPTSwarm and DyLAN, achieve superior performance, albeit at the cost of excessive token consumption. For instance, DyLAN’s cost on GSM8K is  $2.82\times$  that of the random graph, reaching a substantial  $2.2e+7$ . In contrast, **G-Designer** elegantly balances



**Figure 6:** We compare the accuracy (%) of various multi-agent frameworks before and after prompt attacks on MMLU.



**Figure 7:** (Left) Sensitivity analysis on the number of agents  $N$ . (Right) Ablation study of two regularizations under clean and adversarial attack settings, tested on MMLU benchmark.

both efficiency and task performance, achieving the highest performance across all four benchmarks while maintaining the lowest token cost. For example, on SVAMP, **G-Designer** surpasses DyLAN by 4% while using only 23.7% of DyLAN’s token consumption.

### 5.4 Robustness Verification (RQ3)

In this section, we compare the robustness of different topology designs when subjected to adversarial attacks. Following [88], we simulate a system prompt attack on one of the five agents, with the results shown in Figure 6. We observe the following:

**Obs. ⑤ G-Designer is a robust defender against adversarial attacks.** As seen in Figure 6, many trivial structures, such as chain or complete graph, experience significant performance degradation under partial system attacks, with drops as high as 11.0%. Among more sophisticated structures, GPTSwarm, benefiting from its specialized node optimization mechanism, only suffers a minor 0.3% accuracy decline. However, other methods fare less well, with DyLAN and AutoGen showing accuracy drops of 6.2% and 9.9%, respectively. Remarkably, **G-Designer** demonstrates exceptional robustness against adversarial attacks, maintaining nearly identical performance pre- and post-attack. This resilience can be attributed to its agent encoding capability, which, during optimization, can detect malicious inputs and prune the corresponding edges.

### 5.5 Ablation Study & Sensitivity Analysis (RQ4)

**Sensitivity Analysis.** We compare the performance of the chain structure, GPTSwarm, and **G-Designer** across varying numbers of agents  $N$ . As shown in Figure 7 (Left), with the increase in agent count, the simple chain-style structure exhibits marginal performance improvements and poor scaling capacity. In contrast, **G-Designer** demonstrates a stronger emergent capability, where the involvement of more agents leads to notable performance gains.

**Ablation Study.** We report results for two variants of our method: (1) **G-Designer** w/o SR, which removes the sparsity regularization in Equation (14), and (2) **G-Designer** w/o Anchor, which excludes the anchor structure  $A_{\text{anchor}}$ . As shown in Figure 7 (Right), the

removal of  $A_{anchor}$  consistently leads to performance degradation, while the absence of sparsity regularization makes the system more vulnerable to adversarial attacks in adversarial settings.

## 6 Conclusion

In this paper, we first present the LLM-based Multi-agent Communication Protocol (MACP), which aims to provide insightful guidance for designing complex multi-agent systems. Furthermore, we propose an effective, adaptive, and robust LLM-powered multi-agent communication graph designer, termed **G-Designer**, to facilitate the automated design of collaborative AI systems. **G-Designer** is highly task-aware, dynamically crafting compact and robust communication topologies based on the complexity of the task at hand. We hope that **G-Designer** will inspire future research on the emergence of self-organizing and self-evolving collective intelligence.

## References

- [1] Michael GH Bell, Yasunori Iida, et al. 1997. Transportation network analysis. (1997).
- [2] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Cajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefer. 2023. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. , arXiv:2308.09687 pages.
- [3] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. 1976. *Graph theory with applications*. Vol. 290. Macmillan London.
- [4] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. 2024. Repairagent: An autonomous, llm-based agent for program repair. *arXiv preprint arXiv:2403.17134* (2024).
- [5] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate. *arXiv e-prints* (Aug. 2023). arXiv:2308.07201
- [6] Dake Chen, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Haoyang Zhang. 2023. Gamegpt: Multi-agent collaborative framework for game development. *arXiv preprint arXiv:2310.08067* (2023).
- [7] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17754–17762.
- [8] Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. 2024. Are more llm calls all you need? towards scaling laws of compound inference systems. *arXiv preprint arXiv:2403.02419* (2024).
- [9] 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, Heidiy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Foteios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. , arXiv:2107.03374 pages.
- [10] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023. AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors in Agents. arXiv:2308.10848
- [11] Weize Chen, Ziming You, Ran Li, Yitong Guan, Chen Qian, Chenyang Zhao, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2024. Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence. *arXiv preprint arXiv:2407.07061* (2024).
- [12] Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, and Xiuqiang He. 2024. Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects. *CoRR abs/2401.03428* (2024).
- [13] 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 preprint abs/2110.14168* (2021).
- [14] Roi Cohen, May Hamri, Mor Geva, and Amir Globerson. 2023. Lm vs lm: Detecting factual errors via cross examination. *arXiv preprint arXiv:2305.13281* (2023).
- [15] 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* (2023).
- [16] Giorgio Fagiolo, Javier Reyes, and Stefano Schiavo. 2010. The evolution of the world trade web: a weighted-network analysis. *Journal of Evolutionary Economics* 20 (2010), 479–514.
- [17] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The world wide web conference*. 417–426.
- [18] Reza Zanjirani Farahani, Elnaz Miandoabchi, Wai Yuen Szeto, and Hannaneh Rashidi. 2013. A review of urban transportation network design problems. *European journal of operational research* 229, 2 (2013), 281–302.
- [19] 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*.
- [20] Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. 2023. Large Language Models Empowered Agent-based Modeling and Simulation: A Survey and Perspectives. *CoRR abs/2312.11970* (2023).
- [21] Diego Garlaschelli, Ticiana Di Matteo, Tomaso Aste, Guido Caldarelli, and Maria I Loffredo. 2007. Interplay between topology and dynamics in the World Trade Web. *The European Physical Journal B* 57 (2007), 159–164.
- [22] Derek Greene, Donal Doyle, and Padraig Cunningham. 2010. Tracking the evolution of communities in dynamic social networks. In *2010 international conference on advances in social networks analysis and mining*. IEEE, 176–183.
- [23] Rui Hao, Linmei Hu, Weijian Qi, Qinglin Wu, Yirui Zhang, and Liqiang Nie. 2023. ChatLLM Network: More brains, More intelligence. , arXiv:2304.12998 pages.
- [24] 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)* (2021).
- [25] 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*.
- [26] 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. , arXiv:2308.00352 pages.
- [27] Shengchao Hu, Li Shen, Ya Zhang, and Dacheng Tao. 2024. Learning multi-agent communication from graph modeling perspective. *arXiv preprint arXiv:2405.08550* (2024).
- [28] Wenyue Hua, Lizhou Fan, Lingyao Li, Kai Mei, Jianchao Ji, Yingqiang Ge, Libby Hemphill, and Yongfeng Zhang. 2023. War and peace (waragent): Large language model-based multi-agent simulation of world wars. *arXiv preprint arXiv:2311.17227* (2023).
- [29] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*. PMLR, 9118–9147.
- [30] 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* (2024).
- [31] Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Toronto, Canada, 14165–14178.
- [32] Ye Jin, Xiaoxi Shen, Huiling Peng, Xiaoan Liu, Jingli Qin, Jiayang Li, Jintao Xie, Peizhong Gao, Guyue Zhou, and Jiangtao Gong. 2023. SurrealDriver: Designing Generative Driver Agent Simulation Framework in Urban Contexts based on Large Language Model. arXiv:2309.13193
- [33] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714* (2023).
- [34] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
- [35] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations*. OpenReview.net.
- [36] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbulin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *NeurIPS*.
- [37] Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. More Agents Is All You Need. *CoRR abs/2402.05120* (2024).
- [38] Minghao Li, Yingxin Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244* (2023).
- [39] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. 2023. Encouraging Divergent Thinking

- in Large Language Models through Multi-Agent Debate. *CoRR* abs/2305.19118 (2023).
- [40] 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* (2017).
- [41] Yuntao Liu, Yong Dou, Yuan Li, Xinhai Xu, and Donghong Liu. 2022. Temporal dynamic weighted graph convolution for multi-agent reinforcement learning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 44.
- [42] 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 (2023).
- [43] Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, Wenhao Yu, and Dong Yu. 2023. Laser: Llm agent with state-space exploration for web navigation. *arXiv preprint arXiv:2309.08172* (2023).
- [44] Qun Ma, Xiao Xue, Deyu Zhou, Xiangning Yu, Donghua Liu, Xuwen Zhang, Zihan Zhao, Yifan Shen, Peilin Ji, Juanjuan Li, Gang Wang, and Wanpeng Ma. 2024. Computational Experiments Meet Large Language Model Based Agents: A Survey and Perspective. *CoRR* abs/2402.00262 (2024).
- [45] Yohei Nakajima. 2023. BabyAGI. <https://github.com/yoheinakajima/babyagi>.
- [46] Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. In *UIST*. ACM, 2:1–2:22.
- [47] Arkil Patel, Satwik Bhattacharya, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191* (2021).
- [48] Emanuele Pesce and Giovanni Montana. 2023. Learning multi-agent coordination through connectivity-driven communication. *Machine Learning* 112, 2 (2023), 483–514.
- [49] Giorgio Piatti, Zhijing Jin, Max Kleiman-Weiner, Bernhard Schölkopf, Mrinmaya Sachan, and Rada Mihalcea. 2024. Cooperate or Collapse: Emergence of Sustainability Behaviors in a Society of LLM Agents. *arXiv preprint arXiv:2404.16698* (2024).
- [50] Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative Agents for Software Development. *arXiv:2307.07924* pages. 25 pages, 9 figures, 2 tables.
- [51] 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* (2024).
- [52] N Reimers. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv preprint arXiv:1908.10084* (2019).
- [53] Reworkd. 2023. AgentGPT. <https://github.com/reworkd/AgentGPT>.
- [54] Toran Bruce Richards and et al. 2023. Auto-GPT: An Autonomous GPT-4 Experiment. <https://github.com/Significant-Gravitas/Auto-GPT>.
- [55] Eran Rosenbluth, Jan Tönshoff, Martin Ritzert, Berke Kisin, and Martin Grohe. 2024. Distinguished In Uniform: Self Attention Vs. Virtual Nodes. *arXiv preprint arXiv:2405.11951* (2024).
- [56] Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413* (2016).
- [57] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Hangyu Mao, Ziye Li, Xingyu Zeng, Rui Zhao, et al. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
- [58] Ma Angeles Serrano and Marián Boguná. 2003. Topology of the world trade web. *Physical Review E* 68, 1 (2003), 015101.
- [59] Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint abs/2303.11366* (2023). <https://doi.org/10.48550/arXiv.2303.11366>
- [60] Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. 2023. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*. PMLR, 31613–31632.
- [61] Qiushi Sun, Zhiyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. 2023. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration. *arXiv preprint arXiv:2310.00280* (2023).
- [62] Simeng Sun, Yang Liu, Shuhang Wang, Chenguang Zhu, and Mohit Iyyer. 2023. Pearl: Prompting large language models to plan and execute actions over long documents. *arXiv preprint arXiv:2305.14564* (2023).
- [63] Xiangguo Sun, Hong Cheng, Bo Liu, Jia Li, Hongyang Chen, Guandong Xu, and Hongzhi Yin. 2023. Self-supervised hypergraph representation learning for sociological analysis. *IEEE Transactions on Knowledge and Data Engineering* 35, 11 (2023), 11860–11871.
- [64] Zhen Tan, Ruocheng Guo, Kaize Ding, and Huan Liu. 2023. Virtual node tuning for few-shot node classification. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2177–2188.
- [65] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv e-prints*, Article arXiv:2305.16291 (May 2023), arXiv:2305.16291 pages. arXiv:2305.16291
- [66] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. 2024. A Survey on Large Language Model based Autonomous Agents. *Front. Comput. Sci.* 18 (2024).
- [67] 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 (2020), 5776–5788.
- [68] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 950–958.
- [69] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*.
- [70] Zhenhai Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2023. Unleashing Cognitive Synergy in Large Language Models: A Task-Solving Agent through Multi-Persona Self-Collaboration. , arXiv:2307.05300 pages. work in progress.
- [71] 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.
- [72] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8 (1992), 229–256.
- [73] 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.
- [74] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhiyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huan, and Tao Gui. 2023. The Rise and Potential of Large Language Model Based Agents: A Survey. *arXiv preprint abs/2309.07864* (2023).
- [75] 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* (2024).
- [76] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izahk Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models.
- [77] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izahk Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.
- [78] Zhiyue Yin, Qiushi Sun, Cheng Chang, Xipeng Guo, Junqi Dai, Xuan-Jing Huang, and Xipeng Qiu. 2023. Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 15135–15153.
- [79] Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2023. Self-taught optimizer (stop): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304* (2023).
- [80] Guibin Zhang, Yanwei Yue, Zhiyun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2024. Cut the Crap: An Economical Communication Pipeline for LLM-based Multi-Agent Systems. *arXiv preprint arXiv:2410.02506* (2024).
- [81] Jintian Zhang, Xin Xu, and Shumin Deng. 2023. Exploring collaboration mechanisms for llm agents: A social psychology view. *arXiv preprint arXiv:2310.02124* (2023).
- [82] Jintian Zhang, Xin Xu, and Shumin Deng. 2023. Exploring collaboration mechanisms for llm agents: A social psychology view. *arXiv preprint arXiv:2310.02124* (2023).
- [83] Ping Zhang and Gary Chartrand. 2006. Introduction to graph theory. *Tata McGraw-Hill* 2 (2006), 2–1.
- [84] Kesen Zhao and Liang Zhang. 2024. Causality-Inspired Spatial-Temporal Explanations for Dynamic Graph Neural Networks. In *The Twelfth International Conference on Learning Representations*.
- [85] Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. Progressive-Hint Prompting Improves Reasoning in Large Language Models. , arXiv:2304.09797 pages. Tech Report.
- [86] 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* (2023).
- [87] Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A Rossi, Somdeb Sarkhel, and Chao Zhang. 2023. Toolchain\*: Efficient action space navigation in large language models with a\* search. *arXiv preprint arXiv:2310.13227* (2023).
- [88] 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 Algorithm Workflow

---

**Algorithm 1:** Designing workflow of G-Designer

---

**Input :** Input query  $Q$ , Graph auto-encoder  $f_v$  composed of encoder  $q(\cdot)$  and decoder  $p(\cdot)$  (parameterized by  $\Theta_e$  and  $\Theta$ ), learning rate  $\alpha$

1 **for** query  $d$  in  $\{1, 2, \dots, D'\}$  **do**

2     /\* Establish multi-agent network \*/

3     **for** node  $i$  in  $\{1, 2, \dots, N\}$  **do**

4          $x_i \leftarrow \text{NodeEncoder}(\mathcal{T}(\text{Base}_i), \text{Role}_i, \mathcal{T}(\text{Plugin}_i))$

5         Obtain agent embeddings  $X_{\text{agent}} \leftarrow [x_1, x_2, \dots, x_N]^T$

6         Obtain task-specific node  $x_{\text{task}} \leftarrow \text{NodeEncoder}(Q_d)$

7         Set an anchor topology  $A_{\text{anchor}}$  // In our experiments,  
            the anchor topology is simply set as the chain structure

8         Obtain a task-specific multi-agent network

9          $\tilde{\mathcal{G}} = \left( \begin{bmatrix} X_{\text{agent}} \\ x_{\text{task}}^T \end{bmatrix}, A_{\text{anchor}} \right)$  // Note that  $A_{\text{anchor}}$  here  
            contains bidirectional edges added by the task node  
             $v_{\text{task}}$

10        /\* Design communication topology \*/

11        Encode  $\tilde{\mathcal{G}}$  into latent agent representations  $H_{\text{agent}}$ :  
             $q(H_{\text{agent}} | \tilde{\mathcal{G}}, A_{\text{anchor}}) = \prod_{i=1}^N q(h_i | \tilde{\mathcal{G}}, A_{\text{anchor}})$

12        Decode (phase 1) and obtain the sketched graph  $S$ :  
             $p_s(S | H_{\text{agent}}) = \prod_{i=1}^N \prod_{j=1}^N p_s(s_{ij} | h_i, h_j, h_{\text{task}}; \Theta_d)$

13        Decode (phase 2) and obtain the communication graph

14         $\mathcal{G}_{\text{com}} = (\mathcal{V}, \mathcal{E}_{\text{com}}), \mathcal{E}_{\text{com}} = \{(i, j) | \tilde{s}_{ij} \neq 0 \wedge (i, j) \in \mathcal{E}\}$

15        /\* Guide multi-agent system collaboration \*/

16        **for** iteration  $t$  in  $\{1, 2, \dots, K\}$  **do**

17            **for** node  $i$  in  $\phi(\mathcal{G}_{\text{com}})$  **do**

18                 Agent  $v_i$  generates  $\mathcal{R}_i^{(t)} \leftarrow$

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

20                 /\* Aggregate solution \*/

21                  $a^{(t)} \leftarrow \text{Aggregate}(\mathcal{R}_1^{(t)}, \mathcal{R}_2^{(t)}, \dots, \mathcal{R}_N^{(t)})$

22                 /\* Update G-Designer parameters \*/

23                  $\Theta^{d+1} \leftarrow \Theta^d - \alpha \nabla_{\Theta^d} \mathcal{L}_{\text{G-Designer}}$

---