ADAPTIVE ORGANIZATION-CENTRIC COLLABORATION FOR LLM-BASED MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM)-based multi-agent systems (MAS) excel at complex tasks, and their effectiveness depends critically on how agents are organized to collaborate. However, most existing methods design collaboration only at the instance level, generating for each query a concrete topology that fixes which agents participate and how they communicate before execution. They leave unmodeled the organizational level of collaboration, which comprises the groups through which agents collaborate, how these groups form and change as the task unfolds, and how context passes between them. As a result, these methods cannot adapt collaboration during execution, and the organizational knowledge they learn stays tied to whole-task topologies. Inspired by the Agent-Group-Role (AGR) organizational model, which separates the organizational structure from the concrete organizations that instantiate it, we lift collaboration design to the organizational level and propose AGR-AR, an organization-centric framework. AGR-AR learns how to organize agents and instantiates groups on demand during execution. At each step, a lightweight autoregressive topology generator constructs the next group conditioned on the task and execution feedback, selecting its members, including overlapping agents that carry context from earlier groups, and determining how they communicate. The organization thus adapts during execution, and a single generator serves tasks across domains. Experiments on six benchmarks show that AGR-AR achieves higher average accuracy than strong baselines while consuming substantially fewer tokens and remaining more robust to adversarial agents; without retraining, it also incorporates unseen agents and transfers to unseen tasks. Code is available at https://anonymous.4open.science/r/AGR-AR-ECF6/.

1 INTRODUCTION

Multi-agent systems (MAS) built on large language models (LLMs) have shown strong capabilities in complex tasks beyond the reach of individual agents, such as code generation (Hong et al., 2023; Qian et al., 2024a), biomedicine (Inoue et al., 2025; Tang et al., 2024), and scientific discovery (Rothfarb et al., 2025; Boiko et al., 2023). Central to their success is the collaboration topology, which specifies how agents interact and how information flows among them (Zhang et al., 2024b; Shen et al., 2025), and the most suitable topology can differ substantially across tasks (Zhang et al., 2024b; Qian et al., 2024b). Effective collaboration thus depends not only on capable agents but also on how they are organized (Guo et al., 2026).

Early systems rely on hand-crafted topologies such as chains, trees, complete graphs, and multi-agent debate (Hong et al., 2023; Yao et al., 2023; Qian et al., 2024b; Du et al., 2024; Liang et al., 2024), which suit the settings they were designed for but transfer poorly across tasks (Qian et al., 2024b) and incur the same communication cost on easy and hard queries (Zhang et al., 2024b; Jiang et al., 2025). To improve generality and communication efficiency, learnable methods adapt the topology to the task or query. AgentPrune (Zhang et al., 2024a) and AgentDropout (Wang et al., 2025) prune redundant edges or agents from a dense template graph, G-Designer (Zhang et al., 2024b) decodes a query-specific topology with a variational graph auto-encoder, and EIB-LEARNER (Shen et al., 2025) balances error suppression against beneficial information propagation by fusing sparse and dense connectivity. More recent designers generate the topology from scratch (Li et al., 2026a; Jiang et al., 2025) or train a single designer for all domains (Li et al., 2026b).

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

Despite their varied designs, these methods share an instance-level paradigm (Figure 1a), designing one concrete collaboration topology for each query and fixing it before execution. What they leave out is the organizational level of collaboration, which comprises the groups through which agents collaborate, how these groups form and change as the task unfolds, and how context passes between them. This omission leads to two limitations. Limitation 1: Limited adaptation during execution. A topology fixed in advance can be tailored to the query, for instance to its expected difficulty (Zhang et al., 2024b; Shen et al., 2025), but not to what happens during execution. Checking and correction must therefore be provisioned whether or not an error occurs, and the planned interactions continue even after the answer has been found. Methods that do adapt during execution rely on an LLM to redesign or patch the concrete topology (Liu et al., 2024; 2026; Wang et al., 2026), so their adaptation still operates on individual agents and links rather than on groups. Limitation 2: Organizational knowledge tied to whole-task topologies. Learnable designers map each query to a whole-task topology and must therefore capture topologies that differ widely across domains, even though these topologies are assembled from recurring collaboration patterns, such as a solver paired with a verifier. As a result, they are trained per domain (Zhang et al., 2024a;b; Shen et al., 2025) or rely on specialized architectures such as mixture-of-experts (Li et al., 2026b). Group-based methods make such patterns explicit as higher-order units (Zhang et al., 2025; Chen et al., 2026), yet still assemble them into a single topology before execution. This raises a natural question: can we model collaboration at the organizational level, learning how agents should be organized while instantiating concrete groups only as the task unfolds?

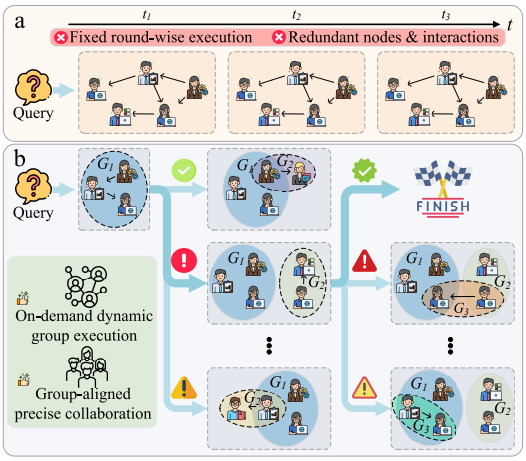


Figure 1: Design paradigms. (a) Instance-level design, shown with an agent-level topology: a concrete topology is fixed before execution and rerun unchanged. (b) AGR-AR instantiates groups on demand during execution; overlapping agents carry context across groups.

Organizational theory for MAS offers a principled answer (Wooldridge et al., 2000; Zambonelli et al., 2003; Hübner et al., 2002). The Agent-Group-Role (AGR) model (Ferber & Gutknecht, 1998; Ferber et al., 2003), in particular, separates an organizational structure, the groups and roles through which agents collaborate, from the concrete organizations that instantiate it. One structure can underlie many concrete organizations, groups are created when needed (much like project teams in a company), and an agent may belong to several groups at once. In these terms, existing topology design methods produce only concrete organizations, that is, instances of an organizational structure that is itself never modeled. Inspired by this separation, we make the organizational level learnable (Figure 1b). Whereas AGR leaves the organizational structure to human designers, we learn from data how agents should be organized and instantiate concrete groups on demand during execution. Instantiating each group only when needed, conditioned on execution feedback, allows collaboration to adapt during execution at the level of groups, which addresses Limitation 1; learning how to form the next group, rather than a whole-task topology, allows recurring patterns to be shared across domains, which alleviates Limitation 2.

Building on this idea, we propose AGR-AR, an organization-centric framework that instantiates groups on demand during execution. Given a query, AGR-AR proceeds group by group until the task is solved. For each group, a lightweight autoregressive topology generator, conditioned on the task and execution feedback, selects the members and decides how they communicate, and agents selected again from earlier groups carry their context into the new group as overlapping agents. Our main contributions are as follows:

- We identify instance-level design as the common root of both limitations and, inspired by AGR, address them by making the organizational level of collaboration learnable.
- We propose AGR-AR, which instantiates groups on demand during execution with a lightweight autoregressive topology generator and connects successive groups through overlapping agents.

- Experiments on six benchmarks show that a single generator achieves higher average accuracy than fixed topologies and learnable designers such as G-Designer and EIB-LEARNER while consuming substantially fewer tokens and remaining more robust to adversarial agents; without retraining, it also incorporates unseen agents and transfers to unseen tasks.

2 Related Work

2.1 Collaboration Topology Design

The collaboration topology is central to LLM-based MAS (Qian et al., 2024b; Zhang et al., 2024b). Hand-crafted systems, such as standard operating procedures (Hong et al., 2023; Qian et al., 2024a) and multi-agent debate (Du et al., 2024; Liang et al., 2024), fix roles and interactions at design time and generalize poorly across tasks (Qian et al., 2024b). Learnable designers instead learn the topology from data. AgentPrune (Zhang et al., 2024a) and AgentDropout (Wang et al., 2025) prune a dense graph for each task, G-Designer (Zhang et al., 2024b) and EIB-LEARNER (Shen et al., 2025) learn query-dependent structures with graph neural networks, and ARG-Designer (Li et al., 2026a) and GTD (Jiang et al., 2025) generate the graph from scratch, autoregressively or through guided diffusion. OFA-MAS (Li et al., 2026b) trains a single mixture-of-experts designer for all domains. Group-based methods model higher-order collaboration units explicitly, through hyperedges (Zhang et al., 2025) or by composing candidate groups (Chen et al., 2026). Whatever their units, these methods design at the instance level, producing a concrete topology that is fixed before execution. AGR-AR instead learns how to form the next group and instantiates groups during execution.

2.2 Adaptation During Execution

A few methods adapt collaboration while the task is being solved. DyLAN (Liu et al., 2024) selects a team of agents from candidates and lets them collaborate in a dynamic communication structure, MAS-on-the-Fly (Liu et al., 2026) builds a query-specific system from retrieved procedures and supervises it at test time, and AgentConductor (Wang et al., 2026) trains an LLM orchestrator with reinforcement learning to revise a layered topology from execution feedback. These methods rely on an LLM to redesign or patch a concrete topology of individual agents and links. AGR-AR adapts at the level of groups, with a lightweight generator deciding which group to form next, conditioned on execution feedback.

2.3 Organizational Models of MAS

Organization-centric MAS describe collaboration through groups and roles rather than individual agents (Ferber et al., 2003), and organized teams also benefit LLM agents in embodied cooperation (Guo et al., 2026). Gaia (Wooldridge et al., 2000; Zambonelli et al., 2003) provides a methodology for organizational design, MOISE+ (Hübner et al., 2002) specifies the structural, functional, and deontic aspects of an organization, and AGR (Ferber & Gutknecht, 1998; Ferber et al., 2003) separates an organizational structure from the concrete organizations that instantiate it. These models leave the organizational structure to human designers; AGR-AR learns it from data and instantiates concrete groups during execution.

3 Problem Formulation

3.1 Organization-Centric Modeling of MAS

We model multi-agent collaboration with the Agent-Group-Role (AGR) model (Ferber & Gutknecht, 1998; Ferber et al., 2003). Given an agent pool $\mathcal{A}$, collaboration takes place in groups, and the t-th group is a directed acyclic graph

$$G_t = (\mathcal{V}_t,\, \mathcal{E}_t), \qquad \mathcal{V}_t \subseteq \mathcal{A}, \tag{1}$$

where $\mathcal{E}_t$ contains the directed communication edges among its members. Communication is confined within each group, and roles are not assigned in advance. The role of an agent is determined by which members of G_t it receives messages from and which it informs, so the same agent can play different

roles in different groups. For a query Q, the concrete organization of the MAS is the sequence of groups formed during execution, $\mathcal{M}_Q = (G_1, G_2, \ldots, G_T)$, whose length T is determined during execution rather than fixed in advance. As in AGR, an agent may take part in several groups over time, which we record with an overlap vector $\mathbf{m}_t \in \{0,1\}^{|\mathcal{A}|}$:

$$m_{t,k} = \mathbf{1}\big[\exists\, \tau < t : a_k \in \mathcal{V}_\tau\big]. \tag{2}$$

Agents selected again in a later group are called overlapping agents. They carry context from the groups they joined earlier into later groups.

3.2 From Instance-Level to Organizational-Level Design

Existing topology design methods operate at the instance level. Given a query, they directly output a concrete topology for the whole task and, in most cases, fix it before execution; groups, when present, are determined once as part of this topology. We instead model collaboration at the organizational level, learning a group-level conditional distribution from which the concrete organization is instantiated step by step during execution:

$$\underbrace{\mathcal{G} = f_\theta(Q)}_{\text{instance level}} \qquad \longrightarrow \qquad \underbrace{G_t \sim p_\theta\big(G \mid \Phi_t, \mathbf{m}_t\big), \quad t = 1, \ldots, T,}_{\text{organizational level}} \tag{3}$$

where the joint context Φ_t encodes the task and execution feedback. In AGR terms, p_θ plays the role of the organizational structure. Shared across tasks and domains, it specifies which groups to form in which situations and how their members are connected, and the sequence $\mathcal{M}_Q$ instantiated from it is the concrete organization. The unit of generation thus shifts from a whole-task topology to a single group, and generation is conditioned not only on the query but also on feedback that is updated during execution. An ideal organization solves the task while keeping communication cheap,

$$\max_\theta \; \mathbb{E}_Q\Big[\, u(Q, \mathcal{M}_Q) - \beta\, c(\mathcal{M}_Q)\,\Big], \tag{4}$$

where u measures task utility, c measures communication cost, and β trades off the two. Because evaluating u requires executing the MAS, optimizing Eq. (4) directly is too costly; we instead learn p_θ from demonstrations that reflect effectiveness and compactness through progressive training.

3.3 Autoregressive Group Generation

We factorize the generation of each group autoregressively into a sequence of node and edge decisions (You et al., 2018):

$$p_\theta(G_t \mid \Phi_t, \mathbf{m}_t) = \prod_{i=1}^{|\mathcal{V}_t|+1} p_\theta\big(v_i \mid v_{<i}, \Phi_t, \mathbf{m}_t\big) \prod_{i=2}^{|\mathcal{V}_t|} \prod_{j=1}^{i-1} p_\theta\big(e_{j\to i} \mid v_{\le i}, e_{<(j,i)}, \Phi_t, \mathbf{m}_t\big). \tag{5}$$

Nodes are selected from the agent pool one at a time until a special `END` token is produced, and each new node is followed by predictions of its incoming edges from earlier nodes. The size, membership, and communication topology of a group are therefore determined jointly by a single model, and since edges only point from earlier to later nodes, every generated group is acyclic.

4 Method

Figure 2 gives an overview of AGR-AR. As shown in Figure 2a, AGR-AR instantiates groups one at a time during execution rather than generating a complete topology for the whole task. At each step, the autoregressive topology generator reads the joint context and the overlap vector and generates the members of the next group and their communication topology, and the execution feedback of this group drives the next step until the task is solved. The generator is trained with the three-stage strategy shown in Figure 2b.

4.1 On-Demand Group Instantiation

(1) Joint context. The generation of group G_t is conditioned on the joint context

$$\Phi_t = \big[\, Q;\; F_t;\; H_{<t}\,\big], \tag{6}$$

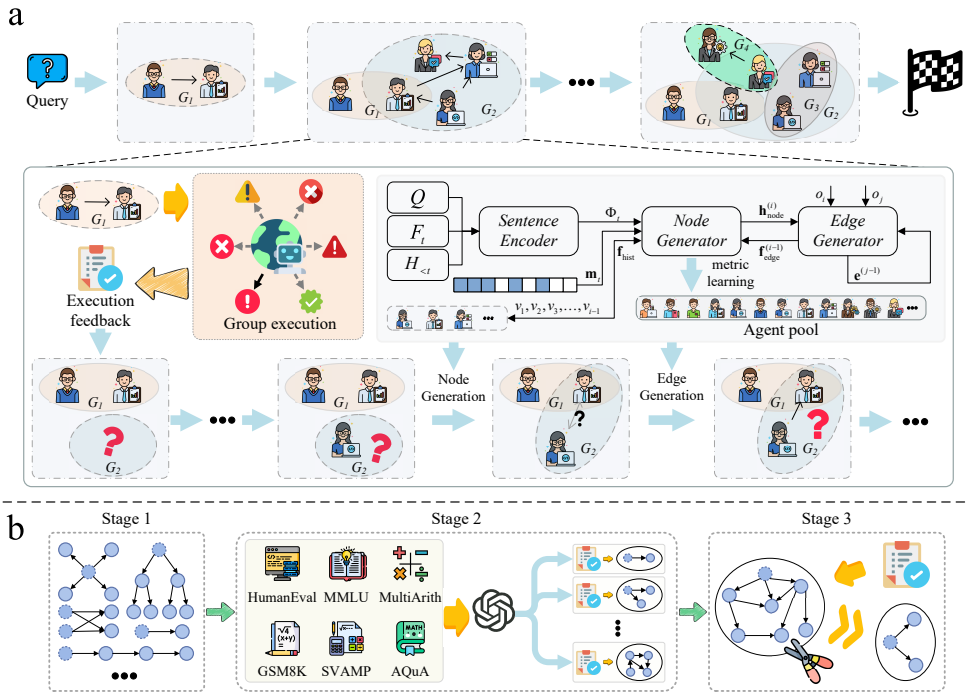


Figure 2: Overview of AGR-AR. (a) On-demand group instantiation: conditioned on the task and execution feedback, the autoregressive topology generator instantiates each group, and overlapping agents carry context across groups. (b) Three-stage progressive training.

where Q is the task query, F_t is the execution feedback, which reflects the progress of the task and what remains to be done, and $H_{<t}$ is the execution history of previous groups. A sentence encoder maps the three components to a dense representation. Execution feedback allows the generator to form each group according to what actually happens during execution. When the feedback reveals an erroneous intermediate result, the generator can form a group for correction and verification, and when the feedback indicates that the task is solved, the process terminates.

(2) Overlap mechanism. Context passes across groups through overlapping agents. When an agent a_k with $m_{t,k} = 1$ is selected again for G_t, it carries cross-group context, namely its last output in each group it joined earlier together with the summary of that group. Context passing is thus selective. An overlapping agent receives only the context of the groups it took part in, not the full execution history, so irrelevant context does not accumulate.

4.2 Autoregressive Topology Generator

The generator constructs each group following Eq. (5), alternating between node and edge generation with a node-level GRU and an edge-level GRU.

(1) Node generation. At step i, the node generator fuses a global feature $\mathbf{f}_{\text{global}}$, which encodes the joint context and the overlap vector, with a history feature $\mathbf{f}_{\text{hist}}$ aggregated from previously generated nodes by a dedicated GRU. A gate balances the two, and the fused feature, together with an edge feature $\mathbf{f}_{\text{edge}}^{(i-1)}$ that summarizes the connections of the previous node, updates the node GRU:

$$g_i = \sigma\left(\frac{\mathbf{f}_{\text{hist}}^{\top}\mathbf{f}_{\text{global}}}{\sqrt{d}}\right), \qquad \mathbf{f}_{\text{fused}} = (1 - g_i)\,\mathbf{f}_{\text{hist}} + g_i\,\mathbf{f}_{\text{global}}, \tag{7}$$

$$\mathbf{h}_{\text{node}}^{(i)} = \text{GRU}_{\text{node}}\big([\mathbf{f}_{\text{fused}};\, \mathbf{f}_{\text{edge}}^{(i-1)}],\ \mathbf{h}_{\text{node}}^{(i-1)}\big), \tag{8}$$

where σ is the sigmoid function and d is the feature dimension. The next agent is selected by matching the hidden state against agent embeddings,

$$p_\theta(v_i = a_k) = \text{softmax}\Big(\text{MLP}_{\text{out}}\big(\mathbf{h}_{\text{node}}^{(i)}\big) \cdot \text{MLP}_{\text{proj}}(\mathbf{Z}_{\mathcal{A}})^{\top}\Big)_k, \tag{9}$$

where the agent embedding matrix $\mathbf{Z}_{\mathcal{A}}$ also contains the `START` and `END` tokens. This metric-learning formulation decouples the classifier from the size of the agent pool, so new agents can be incorporated at inference time, without retraining, by appending their embeddings to $\mathbf{Z}_{\mathcal{A}}$.

(2) Edge generation. After node v_i ($i \geq 2$) is generated, the edge generator decides for each earlier node v_j whether v_j sends messages to v_i. The edge GRU is initialized from the node hidden state and an overlap indicator $o_i \in \{0, 1\}$ that marks whether v_i is an overlapping agent, and is then updated with the previous edge decision $\mathbf{e}^{(j-1)}$:

$$\mathbf{h}_{\text{edge}}^{(0)} = \text{MLP}_{\text{init}}\big([\mathbf{h}_{\text{node}}^{(i)};\, o_i]\big), \qquad \mathbf{h}_{\text{edge}}^{(j)} = \text{GRU}_{\text{edge}}\big(\mathbf{e}^{(j-1)},\, \mathbf{h}_{\text{edge}}^{(j-1)}\big), \tag{10}$$

$$p_\theta(e_{j\to i} = 1) = \text{softmax}\big(\text{MLP}_{\text{pred}}([\mathbf{h}_{\text{edge}}^{(j)};\, o_j;\, o_i])\big)_1. \tag{11}$$

The overlap indicators let the generator learn how overlapping agents connect, for example as sources that pass earlier context to the other members.

4.3 Progressive Training

Because task-aligned topologies must be designed by an LLM and are thus costly, we train the generator in three stages of increasing difficulty (Figure 2b; Appendix D). Stage 1 (structural prior learning) learns valid collaboration graphs from randomly generated classic topologies, with an overlap constraint loss $\mathcal{L}_{\text{over}}$ that places overlapping agents at source positions. Stage 2 (task–topology alignment) maps task semantics and execution feedback to group topologies that an LLM designs for 40 training tasks per benchmark, disjoint from the test sets. Stage 3 (efficiency-guided refinement) learns from Stage 2 topologies that are progressively pruned and validated by an LLM. All stages minimize

$$\mathcal{L} = \alpha\, \mathcal{L}_{\text{node}} + (1 - \alpha)\, \mathcal{L}_{\text{edge}} + \lambda\, \mathcal{L}_{\text{over}}, \tag{12}$$

where $\mathcal{L}_{\text{node}}$ and $\mathcal{L}_{\text{edge}}$ are the negative log-likelihoods of node and edge predictions, and $\mathcal{L}_{\text{over}}$ is used only in Stage 1.

5 Experiments

5.1 Experimental Setup

Datasets. We evaluate on six benchmarks spanning general knowledge (MMLU (Hendrycks et al., 2020)), mathematical reasoning (GSM8K (Cobbe et al., 2021), AQuA (Ling et al., 2017), MultiArith (Roy & Roth, 2015), SVAMP (Patel et al., 2021)), and code generation (HumanEval (Chen et al., 2021)), and test transfer to unseen tasks on MBPP (Austin et al., 2021) and to a new domain with new agents on GAIA (Mialon et al., 2023); statistics are in Appendix A.

Baselines. We compare against single-agent methods (Vanilla, CoT (Wei et al., 2022), Self-Consistency (Wang et al., 2022)), fixed topologies (chain, tree, complete graph, random graph, LLM-Debate (Du et al., 2024)), and learnable designers (AgentPrune (Zhang et al., 2024a), AgentDropout (Wang et al., 2025), G-Designer (Zhang et al., 2024b), EIB-LEARNER (Shen et al., 2025)), described in Appendix B.

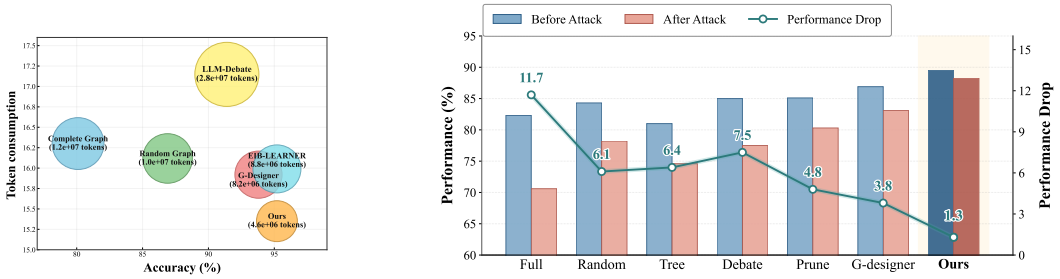
Implementation. Learnable designers are trained separately for each benchmark, whereas AGR-AR trains a single generator on the training tasks of all benchmarks and uses it for every benchmark. All methods are evaluated under the same settings, with GPT-4o as the backbone LLM except on GAIA, where all use GPT-4o-mini; further details are in Appendix C.

5.2 Main Results

Table 1 reports the results, from which we make two observations. ❶ A single generator outperforms designers trained per domain. AGR-AR achieves the highest average accuracy of 93.12%, exceeding the best fixed topology, LLM-Debate, by 5.59 points and the strongest learnable designer, EIB-LEARNER, by 1.74 points; it is best on five of the six benchmarks and ties EIB-LEARNER on GSM8K. Whereas these learnable designers are trained per domain (Zhang et al., 2024a;b; Shen

Table 1: Performance comparison (%) across six benchmarks. Superscripts denote gain (↑) or loss (↓) relative to the Vanilla baseline. Best results are **bold**; second-best are underlined.

Method	MMLU	GSM8K	AQuA	MultiArith	SVAMP	HumanEval	Average
Direct Answer							
Vanilla	80.39	82.30	71.06	93.09	86.55	71.39	80.80
Single-Agent Prompting							
CoT	81.69 ↑1.30	86.50 ↑4.20	73.58 ↑2.52	93.25 ↑0.16	87.36 ↑0.81	74.67 ↑3.28	82.84 ↑2.04
Self-Consistency	83.66 ↑3.27	81.60 ↓0.70	75.63 ↑4.57	94.12 ↑1.03	88.59 ↑2.04	79.83 ↑8.44	83.91 ↑3.11
Fixed Multi-Agent Topologies							
Chain	83.01 ↑2.62	88.30 ↑6.00	74.05 ↑2.99	93.27 ↑0.18	87.17 ↑0.62	81.37 ↑9.98	84.53 ↑3.73
Tree	81.04 ↑0.65	85.20 ↑2.90	71.23 ↑0.17	93.68 ↑0.59	88.91 ↑2.36	80.53 ↑9.14	83.43 ↑2.63
Complete Graph	82.35 ↑1.96	80.10 ↓2.20	72.95 ↑1.89	94.53 ↑1.44	84.01 ↓2.54	79.03 ↑7.64	82.16 ↑1.36
Random Graph	84.31 ↑3.92	86.90 ↑4.60	76.48 ↑5.42	94.08 ↑0.99	87.54 ↑0.99	82.66 ↑11.27	85.33 ↑4.53
LLM-Debate	84.96 ↑4.57	91.40 ↑9.10	77.65 ↑6.59	96.36 ↑3.27	90.11 ↑3.56	84.70 ↑13.31	87.53 ↑6.73
Learnable Topology Design							
AgentPrune	85.07 ↑4.68	91.10 ↑8.80	80.51 ↑9.45	94.65 ↑1.56	90.58 ↑4.03	86.75 ↑15.36	88.09 ↑7.29
AgentDropout	85.62 ↑5.23	91.70 ↑9.40	80.94 ↑9.88	95.60 ↑2.51	91.04 ↑4.49	85.98 ↑14.59	88.48 ↑7.68
G-Designer	86.92 ↑6.53	93.80 ↑11.50	81.60 ↑10.54	96.50 ↑3.41	93.10 ↑6.55	88.33 ↑16.94	90.04 ↑9.24
EIB-LEARNER	88.90 ↑8.51	95.20 ↑12.90	83.49 ↑12.43	96.83 ↑3.74	94.70 ↑8.15	89.15 ↑17.76	91.38 ↑10.58
Organization-Centric Topology Design (Ours)							
AGR-AR	**89.54** ↑9.15	**95.20** ↑12.90	**85.05** ↑13.99	**99.11** ↑6.02	**94.79** ↑8.24	**95.04** ↑23.65	**93.12** ↑12.32



(a) Token cost on GSM8K.

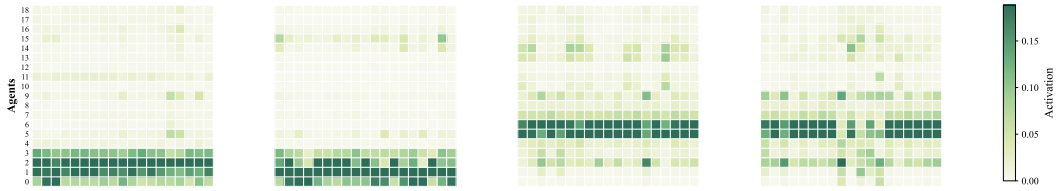
(b) Robustness under adversarial attack.

Figure 3: Token efficiency and adversarial robustness of AGR-AR. Bubble size in (a) reflects token consumption; bars in (b) compare accuracy before and after adversarial agent injection, with the line showing performance drop.

et al., 2025), AGR-AR uses one generator for all benchmarks, suggesting that knowledge of how to form groups can be shared across domains rather than tied to whole-task topologies. ❷ The largest gain arises on code generation. On HumanEval, AGR-AR exceeds EIB-LEARNER by 5.89 points, more than twice its margin on any other benchmark. Code generation benefits from checking and correcting a draft, and AGR-AR instantiates such groups only when execution feedback calls for them, rather than provisioning them in advance for every query.

5.3 Efficiency Analysis

As shown in Figure 3a, AGR-AR achieves the highest accuracy at the lowest token cost among all compared methods, consuming substantially fewer tokens than both fixed topologies and learnable designers. Token counts include all tokens consumed by LLM calls during execution, and all methods share the same backbone LLM and experimental settings. We attribute this efficiency to two properties of the organizational design. First, group boundaries isolate information, so each agent processes only intra-group communication, not the full interaction history, which reduces redundant context. Second, instantiating groups on demand allows AGR-AR to resolve simple tasks with a single compact group and to instantiate additional groups only when execution feedback calls for them, rather than running every interaction planned in advance.



(a) HumanEval (b) MBPP (unseen) (c) GSM8K (d) MultiArith

Figure 4: Agent activation in the first group generated for each task. Rows are agents, columns are sampled test tasks, and color is the selection probability averaged over generation steps.

5.4 Ablation Study

We evaluate four variants: w/o group places all agents in one group that sees the full execution history, reducing AGR-AR to instance-level design with a single topology fixed before execution; w/o overlap and w/o feedback keep the groups but, respectively, pass no context between them or generate them all before execution; and w/o pretrain skips structural prior learning. Table 2 leads to four observations. ❶ Groups contribute most. Removing groups causes the largest drop, 4.66 points on average and 7.44 on HumanEval, because without group boundaries each agent receives the messages of all others and the information relevant to its role is diluted. ❷ Context passed between groups is nearly as important. Removing overlap costs 4.48 points and is the most harmful variant on AQuA, where multi-step solutions build on intermediate results that later groups can no longer access. Together, groups and overlap cause the two largest drops, supporting our claim that the organizational level is what distinguishes AGR-AR from instance-level design. ❸ Adapting groups during execution brings a further gain. Generating all groups before execution costs 1.76 points on average but 3.30 on HumanEval, the benchmark on which AGR-AR gains most in Table 1. Groups planned in advance can only anticipate errors, whereas feedback lets AGR-AR instantiate a correction group for the error that actually arises, as in Figure 5b. ❹ Structural prior learning provides a better starting point. Skipping it costs 3.48 points on average and 6.61 on HumanEval. Learning what valid multi-group structures look like before task alignment thus benefits the generator, and this benefit comes from synthetic structures that require no LLM calls.

Table 2: Ablation study (%).

Method	MMLU	AQuA	HumanEval	Average
Vanilla	80.39	71.06	71.39	74.28
AGR-AR	**89.54**	**85.05**	**95.04**	**89.88**
w/o group	86.27	81.78	87.60	85.22
w/o overlap	86.93	80.84	88.43	85.40
w/o feedback	87.58	85.05	91.74	88.12
w/o pretrain	87.58	83.18	88.43	86.40

5.5 Robustness Analysis

Following Zhang et al. (2024b), we inject adversarial agents that deliberately introduce misleading information, applying the same attack to all methods under identical settings. As shown in Figure 3b, fixed topologies lose 6.1 to 11.7 points under this attack and learnable designers 3.8 to 4.8, whereas AGR-AR loses only 1.3. We attribute this resilience to organizing agents into groups. Group boundaries confine an adversarial agent's influence to a single group, and overlapping agents pass on only the context of their own groups, which limits error propagation across the system.

5.6 What Organization Does AGR-AR Learn?

In AGR, an organizational structure persists as members enter or leave and characterizes a class of concrete organizations (Ferber et al., 2003). Three properties follow: the concrete organizations it underlies can differ in members yet share collaboration patterns; it extends to new tasks of the same kind; and it admits new members without change. We test whether the trained generator, which serves as the organizational structure in AGR-AR, exhibits them. For the first property, Figure 4 shows agent activation in the first group of each task, which is conditioned on the task alone. Activation patterns nearly coincide within a domain (cosine similarity 0.99 between GSM8K and MultiArith) but diverge across domains (0.20 and 0.30 between HumanEval and the arithmetic benchmarks). Yet the generated groups share collaboration patterns across domains. Whenever both are selected, a planner

precedes the solver in code and arithmetic groups (e.g., Algorithm Designer before Programming Expert), and a verifier follows the solver in code and knowledge groups. Sharing thus occurs at the organizational level and specialization at the instance level, letting a single generator serve all domains without per-domain training or specialized architectures. For the second, we use MBPP, a code generation benchmark unseen during training, on which the activation pattern has a cosine similarity of 0.91 with HumanEval but only 0.14 and 0.21 with the arithmetic benchmarks. The generator thus organizes collaboration by the kind of task rather than by benchmark identity, carrying its organization over to unseen tasks without retraining.

For the third, we turn to GAIA (Mialon et al., 2023). Its real-world assistant questions are unseen during training and require four new agents, which are added to the pool without retraining since agents are selected by embedding matching. To isolate the effect of collaboration from that of tool use, all methods run without tools, so the results are not directly comparable with tool-augmented systems. As shown in Table 3, AGR-AR achieves the highest average accuracy of 8.40%, compared with 7.38% for Chain, indicating that the learned structure accommodates new agents in an unseen domain. In contrast, all three learnable designers fall below Chain, at 5.47% to 5.86%, suggesting that their whole-task topologies do not carry over to a new domain.

Table 3: Results on GAIA (%).

Method	Level-1	Level-2	Level-3	Average
Complete Graph	5.66	8.14	0.00	4.60
Chain	11.32	6.98	3.85	7.38
AgentPrune	9.43	8.14	0.00	5.86
G-Designer	5.66	6.98	3.85	5.50
EIB-LEARNER	9.43	6.98	0.00	5.47
AGR-AR (Ours)	**13.21**	**8.14**	**3.85**	**8.40**

5.7 Case Study

Figure 5 shows two organizations generated by AGR-AR. In Figure 5a, a single group of two agents solves a simple problem, and execution stops once the answer is found. In Figure 5b, an error is found in the answer of the first group. Conditioned on this feedback, AGR-AR instantiates a reasoning group, where the Math Solver from the first group, now an overlapping agent, passes its earlier attempt to a Programming Expert, and then a review group, where the Critic, also from the first group, informs a new Inspector to verify the fix. Checking and correction are thus added only after an error occurs, whereas instance-level designs must provision them in advance.

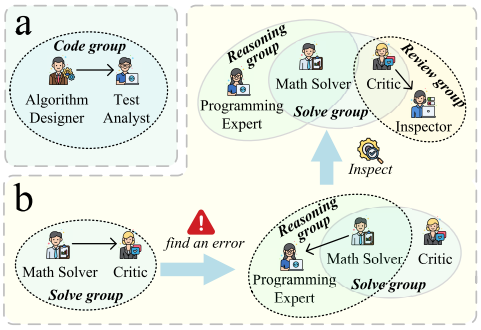


Figure 5: Case studies of AGR-AR.

6 Conclusion

We found that two limitations of existing topology design, limited adaptation during execution and organizational knowledge tied to whole-task topologies, can be traced to a common root, the design of collaboration only at the instance level. Building on this finding and inspired by the Agent-Group-Role model, we made the organizational level learnable and proposed AGR-AR, in which a lightweight autoregressive generator instantiates groups on demand during execution, conditioned on the task and execution feedback, while overlapping agents carry context across groups. Experiments on six benchmarks show that a single generator outperforms fixed topologies and learnable designers with fewer tokens and greater robustness, and that the organizational structure it learns shares collaboration patterns across domains and transfers to unseen tasks and agents without retraining. This work moves topology design from fixing concrete topologies in advance toward learning an organizational structure that is shared across domains and instantiated into groups as a task unfolds.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data: GPT-4o designs the training topologies of Stage 2 and an LLM validates the pruned topologies of Stage 3 (Section 4.3 and Appendix D), which is part of the proposed method. We also used generative AI tools to assist with the translation of parts of this paper. We have not used generative AI tools to implement methods or to clean or reformat datasets, and formulating or proving mathematical claims and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to edit the paper for readability. We have reviewed all AI-assisted work: all text translated or edited with AI assistance was reviewed, revised, and finalized by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only publicly available benchmarks (Appendix A) and involves no human subjects and no personal or sensitive data. The adversarial agents in Section 5.5 inject misleading information into benchmark tasks only to evaluate robustness, following Zhang et al. (2024b). AGR-AR decides how LLM agents are organized but adds no capabilities beyond those of the backbone LLM.

REPRODUCIBILITY STATEMENT

Section 4 describes AGR-AR, including its three-stage training (Section 4.3), and Section 5.1 describes the experimental setup. Appendix A lists the benchmarks and data splits, with 40 training tasks per benchmark that are disjoint from the test sets; Appendix B describes the baselines; Appendix C reports the architecture, training hyperparameters, backbone LLMs, inference procedure (Algorithm 1), and computational resources; and Appendix D explains how the training data of each stage are constructed. The source code, including the prompts, is available at https://anonymous.4open.science/r/AGR-AR-ECF6/ and in the supplementary material.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Daniil A Boiko, Robert MacKnight, and Gabe Gomes. Emergent autonomous scientific research capabilities of large language models. *arXiv preprint arXiv:2304.05332*, 2023.

Hongjiang Chen, Xin Zheng, Yixin Liu, Pengfei Jiao, Shiyuan Li, Huan Liu, Zhidong Zhao, Ziqi Xu, Ibrahim Khalil, and Shirui Pan. Goagent: Group-of-agents communication topology generation for llm-based multi-agent systems. *arXiv preprint arXiv:2603.19677*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*, 2024.

Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings international conference on multi agent systems (Cat. No. 98EX160)*, pp. 128–135. IEEE, 1998.

Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: an organizational view of multi-agent systems. In *International workshop on agent-oriented software engineering*, pp. 214–230. 2003.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L Griffiths, and Mengdi Wang. Embodied llm agents learn to cooperate in organized teams. *IEEE Transactions on Computational Social Systems*, 2026.

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

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

Jomi Fred Hübner, Jaime Simao Sichman, and Olivier Boissier. Moise+ towards a structural, functional, and deontic model for mas organization. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pp. 501–502, 2002.

Yoshitaka Inoue, Tianci Song, Xinling Wang, Augustin Luna, and Tianfan Fu. Drugagent: Multi-agent large language model-based reasoning for drug-target interaction prediction. *ArXiv*, pp. arXiv–2408, 2025.

Eric Hanchen Jiang, Guancheng Wan, Sophia Yin, Mengting Li, Yuchen Wu, Xiao Liang, Xinfeng Li, Yizhou Sun, Wei Wang, Kai-Wei Chang, et al. Dynamic generation of multi-llm agents communication topologies with graph diffusion models. *arXiv preprint arXiv:2510.07799*, 2025.

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

Shiyuan Li, Yixin Liu, Yu Zheng, Mei Li, Quoc Viet Hung Nguyen, and Shirui Pan. Ofa-mas: One-for-all multi-agent system topology design based on mixture-of-experts graph generative models. pp. 1333–1344, 2026b.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 17889–17904, 2024.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 158–167, 2017.

Guangyi Liu, Haojun Lin, Huan Zeng, Heng Wang, and Quanming Yao. Mas-on-the-fly: Dynamic adaptation of llm-based multi-agent systems at test time. *arXiv preprint arXiv:2602.13671*, 2026.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 2080–2094, 2021.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024a.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, et al. Scaling large language model-based multi-agent collaboration. 2024b.

Samuel Rothfarb, Megan C Davis, Ivana Matanovic, Baikun Li, Edward F Holby, and Wilton JM Kort-Kamp. Hierarchical multi-agent large language model reasoning for autonomous functional materials discovery. *arXiv preprint arXiv:2512.13930*, 2025.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1743–1752, 2015.

Xu Shen, Yixin Liu, Yiwei Dai, Yili Wang, Rui Miao, Yue Tan, Shirui Pan, and Xin Wang. Understanding the information propagation effects of communication topologies in llm-based multi-agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 12358–12372, 2025.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 599–621, 2024.

Siyu Wang, Ruotian Lu, Zhihao Yang, Yuchao Wang, Yanzhou Zhang, Lei Xu, Qimin Xu, Guojun Yin, Cailian Chen, and Xinping Guan. Agentconductor: Topology evolution for multi-agent competition-level code generation. *arXiv preprint arXiv:2602.17100*, 2026.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. 2022.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24013–24035, 2025.

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

Michael Wooldridge, Nicholas R Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, 3(3):285–312, 2000.

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

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

Franco Zambonelli, Nicholas R Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.

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

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

Heng Zhang, Yuling Shi, Xiaodong Gu, Zijian Zhang, Haochen You, Lubin Gan, Yilei Yuan, and Jin Huang. Hyperagent: Leveraging hypergraphs for topology optimization in multi-agent communication. *arXiv preprint arXiv:2510.10611*, 2025.

A Dataset Details

Table 4 summarizes the benchmarks. For each of the six in-distribution benchmarks, 40 tasks are reserved for training, and the test tasks are drawn from the remaining ones. MBPP (Austin et al., 2021), used only in the analysis of the learned organization (Section 5.6), and GAIA are never seen during training.

General knowledge. MMLU (Hendrycks et al., 2020) spans 57 subjects; we evaluate on 153 of its four-option multiple-choice questions.

Mathematical reasoning. GSM8K (Cobbe et al., 2021), MultiArith (Roy & Roth, 2015), and SVAMP (Patel et al., 2021) consist of arithmetic word problems with numerical answers, with SVAMP altering the wording of simple problems to probe robustness. AQuA (Ling et al., 2017) poses algebraic word problems with five answer options.

Code generation. Each HumanEval (Chen et al., 2021) task provides a function signature and a docstring; the generated function is checked by unit tests and scored by pass@1.

GAIA. GAIA (Mialon et al., 2023) collects real-world questions for general AI assistants at three difficulty levels, with 53, 86, and 26 questions, respectively.

Table 4: Dataset statistics and details.

Category	Dataset	Answer Type	Metric	#Train	#Test	License
General knowledge	MMLU	Multi-choice	Acc.	40	153	MIT License
Math reasoning	GSM8K	Number	Acc.	40	1,000	MIT License
	MultiArith	Number	Acc.	40	560	Unspecified
	SVAMP	Number	Acc.	40	960	MIT License
	AQuA	Multi-choice	Acc.	40	214	Apache-2.0
Code generation	HumanEval	Code	Pass@1	40	121	MIT License
Unseen	GAIA (L1)	Multi-modal	Acc.	–	53	CC BY 4.0
	GAIA (L2)	Multi-modal	Acc.	–	86	CC BY 4.0
	GAIA (L3)	Multi-modal	Acc.	–	26	CC BY 4.0

B Baseline Details

We compare against 12 baselines organized in three categories:

Single-agent methods. (1) **Vanilla**: direct LLM prompting without multi-agent collaboration; (2) **CoT** (Wei et al., 2022): chain-of-thought prompting for step-by-step reasoning; (3) **Self-Consistency** (Wang et al., 2022): majority voting over multiple CoT samples.

Fixed topologies. (4) **Chain**: linear sequential pipeline where each agent passes its output to the next; (5) **Tree**: hierarchical structure where agents form branching layers with a single root agent aggregating outputs; (6) **Complete Graph**: fully connected topology where every agent communicates with every other; (7) **Random Graph**: randomly generated sparse topology with random role assignments; (8) **LLM-Debate** (Du et al., 2024): multi-round debate among agents with full connectivity.

Learnable designers. (9) **AgentPrune** (Zhang et al., 2024a): prunes redundant communication edges from a predefined graph; (10) **AgentDropout** (Wang et al., 2025): dynamically eliminates redundant agents and edges; (11) **G-Designer** (Zhang et al., 2024b): learns task-adaptive topologies via graph neural networks; (12) **EIB-LEARNER** (Shen et al., 2025): balances error suppression and information diffusion through causal analysis.

C Implementation Details

C.1 Model Architecture

The generator follows the architecture in Section 4.2, with node-level and edge-level GRUs of hidden size 256. The task, the execution feedback, the execution history, and the agent descriptions are encoded by Sentence-BERT (`all-MiniLM-L6-v2`, 384 dimensions), and the agent embeddings are frozen during training. The generator has about 2.9M trainable parameters.

C.2 Training Configuration

Training follows the three-stage progressive strategy described in Section 4.3. All stages use the Adam optimizer with the same loss formulation Eq. (12); the stages differ in data source, learning rate, and whether the overlap constraint $\mathcal{L}_{\text{over}}$ is active. The scheduled sampling rate ϵ is annealed linearly within each stage. A higher ϵ in early stages encourages the exploration of diverse structures, and a lower ϵ in later stages stabilizes generation toward compact topologies. Table 5 lists the complete hyperparameters.

Table 5: Training hyperparameters for each stage.

	Stage 1 ($\mathcal{D}_{\text{prior}}$)	Stage 2 ($\mathcal{D}_{\text{syn}}$)	Stage 3 ($\mathcal{D}_{\text{eff}}$)
Learning rate	1×10^{-3}	1×10^{-4}	1×10^{-4}
Batch size	16	16	32
Max epochs	50	50	100
Early stopping	10 epochs	10 epochs	–
ϵ (scheduled sampling)	$0.8 \rightarrow 0.5$	$0.5 \rightarrow 0.1$	$0.3 \rightarrow 0.1$
α (loss weight)	0.2	0.2	0.2
Optimizer	Adam	Adam	Adam
Additional loss	$\mathcal{L}_{\text{over}}$ ($\lambda = 0.1$)	–	–

C.3 Inference Configuration

For the six in-distribution benchmarks and all ablation and robustness experiments, the backbone LLM is `gpt-4o` with temperature 0. For GAIA, all methods use `gpt-4o-mini` as the backbone to ensure a fair comparison. At each step, the backbone LLM reads the execution history and produces the execution feedback F_t, a short description of the progress and the remaining work, and it signals termination once the task is solved; the prompts are released with the code. The agent pool consists of 19 agents; for GAIA, four additional agents are added to the pool, which is shared by all methods, and AGR-AR incorporates them without retraining (Section 5.6). Algorithm 1 summarizes the inference procedure.

C.4 Computational Resources

Experiments are conducted on a machine with an NVIDIA GeForce RTX 3060 Laptop GPU (6 GB), Python 3.10, PyTorch 2.3.0, and CUDA 12.1. The primary computational cost lies in LLM API calls for data generation (Stage 2 synthesis) and evaluation.

The topology generation model is lightweight ($\sim$2.9M parameters, 117 MB peak GPU memory). Code is available at `https://anonymous.4open.science/r/AGR-AR-ECF6/` and is also provided in the supplementary material.

C.5 Scalability of Topology Generation

Table 6 reports the generation latency and GPU memory consumption of the autoregressive topology generator as a function of agent count, measured on the RTX 3060. GPU memory remains nearly constant across all group sizes because it is dominated by the model weights (2.9M parameters). Latency grows quadratically with the number of agents due to the $O(n^2)$ pairwise edge predictions,

Algorithm 1 AGR-AR inference.

Require: Task query Q, trained generator p_θ, backbone LLM, maximum number of groups $T_{\max}$
Ensure: Final answer a^*
Initialize the overlap vector $\mathbf{m}_1 \leftarrow \mathbf{0}$ and the history $H_{<1} \leftarrow \emptyset$
for $t = 1$ **to** $T_{\max}$ **do**
  $F_t \leftarrow \text{Feedback}(Q, H_{<t})$ {Execution feedback from the backbone LLM}
  if F_t indicates that the task is solved **then**
    break
  end if
  $\Phi_t \leftarrow \text{Encode}([Q;\ F_t;\ H_{<t}])$ {Joint context, Eq. (6)}
  $G_t \sim p_\theta(G \mid \Phi_t, \mathbf{m}_t)$ {Node and edge generation}
  Collect the cross-group context of the overlapping agents in G_t
  Execute G_t on Q and append its outputs to the history
  Update $\mathbf{m}_{t+1}$ by marking the members of G_t
end for
$a^* \leftarrow$ final answer aggregated from the outputs of the executed groups
return a^*

but remains well under one second even for 20 agents. In practice, AGR-AR generates groups of 2–5 agents, where topology generation takes 8–22 ms per group, which is negligible compared with the execution time of LLM agents (seconds per agent).

Table 6: Topology generation scalability on NVIDIA RTX 3060 (averaged over 20 runs).

#Agents	2	3	5	8	10	15	20
Latency (ms)	8.6	9.3	21.6	50.1	65.5	135.5	227.0
Peak GPU (MB)	117.4	117.4	117.4	117.5	117.5	117.5	117.5

D Training Data Construction

Each training stage uses its own dataset.

Stage 1: Structural priors ($\mathcal{D}_{\text{prior}}$). We randomly generate multi-group sequences using six classic topology patterns: chain, star, tree, layered, mesh, and fully connected. Each sequence contains 2–4 groups, where each group has 2–5 nodes with agents randomly sampled from the agent pool. No task context is used (the context vector is set to zero). To enforce the overlap constraint (Section 4.3), source nodes in later groups are assigned agents that appeared in earlier groups, making them the entry points of earlier context. The default configuration generates 2,000 sequences.

Stage 2: Task–topology alignment ($\mathcal{D}_{\text{syn}}$). Training tasks are sampled from the six in-distribution benchmarks (40 per benchmark). For each task, GPT-4o (temperature 0.7) designs a sequence of about four single-group topologies, one for each step of execution. For each step, it writes a feedback message that describes what the group should accomplish, selects the agents, and specifies the directed edges among them. The first topology corresponds to the first group and has no overlapping agents. For each later step, GPT-4o also writes a hypothetical summary of the previous group's output, selects one or two agents of the previous group to continue as overlapping agents, and describes the context they carry; overlapping agents are placed as information sources. The joint context of each topology is obtained by encoding the task query, the feedback message, and the previous output with Sentence-BERT. The prompts are released with the code.

Stage 3: Efficiency refinement ($\mathcal{D}_{\text{eff}}$). Starting from Stage 2 topologies, we apply two-phase progressive pruning. Phase 1 (edge pruning): edges are ranked by priority (non-overlap→overlap edges first, overlap→non-overlap edges last) and tested via a lightweight LLM call that assesses whether the communication link is necessary given the group's objective. Phase 2 (node pruning): non-overlap nodes are ranked by degree (leaves first, intermediates last) and tested by full group

execution followed by LLM-as-judge validation against the ground-truth answer. Overlap nodes are never removed. Only pruned topologies that pass validation enter $\mathcal{D}_{\text{eff}}$. The pruning budget is capped per graph size to control API cost.