Sparse Autoregressive Topology Generation and Failure-Aware DAG Pruning for Heterogeneous Multi-Agent Reasoning

Dima Baker
Collective Intelligence Network

November 2026

Abstract

Scaling complex multi-agent reasoning across heterogeneous model pools requires orchestrating specialist agents over Directed Acyclic Graphs (DAGs). However, existing paradigms face three structural bottlenecks: unpruned all-to-all layered DAGs incur quadratic token growth and context saturation as pool size M expands; centralized star or sequential chain topologies create single-point hub bottlenecks; and static pre-execution graph generators (e.g., G-Designer, AgentPrune, ARG-Designer, GoAgent) cannot dynamically isolate or reroute around mid-trajectory agent faults, allowing single-node hallucinations to cascade downstream. As an architectural synthesis and comparative benchmark study, we formulate Incremental Failure-Aware Autoregressive DAG Routing (IFA-DAG), coupling: (1) GoAgent and ARG-Designer autoregressive DAG initialization under an information-bottleneck sparsity assumption, (2) capability-aware specialist routing (RouteLLM, MARLIN), and (3) online topology-conditioned fault screening that severs outbound edges from failing nodes and splices standby specialists mid-execution. Evaluating 9,600 strictly paired 4-layer DAG workflows ($N = 600$ per regime across $M \in \{4, 8, 16, 32\}$) where task success is determined purely by terminal artifact derivation without post-hoc skill bonuses, IFA-DAG achieves 93.33% [91.05, 95.07] accuracy at $M = 16$—outperforming unpruned all-to-all DAGs (29.83% [26.31, 33.61]), static chains (59.00% [55.02, 62.87]), and static one-shot GNN pruning (72.00% [68.28, 75.44])—while cutting cascading faults to 2.67% [1.65, 4.29] at 22.90k tokens. A factorial ablation isolates the separate contributions of topology, capability binding, edge severing, and standby splicing.

1 Introduction

Orchestrating heterogeneous specialist models—combining formal verifiers, domain-tuned small language models, and frontier reasoning engines—across structured Directed Acyclic Graphs (DAGs) has emerged as a core paradigm for scaling multi-step reasoning beyond monolithic model limits [5, 11, 17, 20, 24]. Drawing inspiration from sparse Mixture-of-Experts routing [7, 18] and cost-aware LLM cascades [4, 19], multi-agent topology generators seek to assemble task-specific collaboration graphs from a pool of M candidate specialists [3, 9, 15, 26].

1.1 The trilemma of multi-agent topology design: saturation, bottlenecks, and cascading faults

Despite rapid progress between 2024 and late 2026, scaling multi-agent workflows across growing specialist pools $M \in [4, 32]$ exposes an architectural trilemma:

1. **Context saturation and quadratic token blowup in unpruned all-to-all layered DAGs:** While MacNet [5, 20] demonstrated that sparse and irregular DAG topologies can coordinate over 1,000 agents under a logistic scaling law, unpruned all-to-all layered meshes suffer severe *information saturation* as fan-in grows. As et al. [8] proved with S-DAG (Sparse-DAG), when downstream agents receive unpruned rationales from all upstream nodes in dense layers, attention dilution and redundant context inflation degrade reasoning accuracy while driving token consumption up quadratically.
2. **Single-point hub bottlenecks in static sequential chains and star topologies:** To avoid dense message passing, many frameworks route sub-tasks through a static sequential chain or central star hub [17, 23]. However, in a sequential chain of depth D, any unrecovered hallucination or tool fault at an early stage propagates directly to downstream stages, creating a single point of failure [1, 13].
3. **Cascading mid-trajectory faults in static pre-execution pruned graphs:** Graph-based orchestrators—including GPTSwarm [17], DyLAN [11], AgentPrune [22, 25], G-Designer [9, 26], MaAS [24], ARG-Designer [15], and GoAgent [3]—generate or prune a sparse communication DAG *prior to execution* (or via static spatial-temporal masks). While pre-execution pruning cuts redundant token traffic [21, 25], Liu et al. [16] (*"Know the Shape, Find the Fault"*) and Cemri et al. [1, 2] demonstrated that static pre-execution graphs cannot intervene when a selected specialist hallucinates or faults mid-trajectory, allowing errors to cascade along directed edges to the terminal node.

1.2 Architectural synthesis: coupling sparse autoregressive DAGs with online fault pruning

This paper is positioned explicitly as an **architectural synthesis and comparative benchmark study** investigating how pre-execution sparse DAG generation interacts with runtime topology-conditioned fault pruning. We formalize and benchmark **Incremental Failure-Aware Autoregressive DAG Routing (IFA-DAG)**, which synthesizes three mechanisms from the 2024–2026 literature:

1. **Autoregressive Group-DAG Initialization (Chen et al. [3], Li et al. [15]):** Rather than starting from an $O(M^2)$ dense graph and pruning edges post hoc, IFA-DAG constructs a sparse group-DAG autoregressively under GoAgent's group information-bottleneck constraint [3] and role-aware k-connected structural priors [10, 12].
2. **Capability-Matched Specialist Binding (et al. [6], Ong et al. [19]):** Abstract DAG vertices are bound to concrete specialists in the M-agent pool using lightweight query–capability routers (`RouteLLM` [19] and `MARLIN` [6]) and dynamic harness selection [14].
3. **Online Topology-Conditioned Fault Screening and Standby Splicing (Cemri et al. [2], Liu et al. [16]):** As each topological layer executes, an online fault detector screens node outputs before forwarding messages along outbound edges $(u, v) \in \mathcal{E}$. When a step fault is detected (with recall $1 - \delta_{\text{fn}}$), IFA-DAG severs u's outbound edges and splices a standby specialist $u' \in \mathcal{M} \setminus \mathcal{V}$ without restarting the workflow.

1.3 Contributions

Our contributions are fivefold:

1. **Taxonomy of multi-agent DAG generation and fault pruning (Section 2).** We synthesize 2024–2026 advances in autoregressive DAG generation, static GNN pruning, capability routing, and topology-conditioned fault attribution (Table 1).
2. **Information-bottleneck DAG formulation and first/second-order cascade analysis (Section 3).** We formalize autoregressive DAG generation under a group information bottleneck and derive the exact error-propagation bound under imperfect online fault verification ($\delta_{\text{fn}} > 0$), explicitly separating the attenuated first-order slope $\bar{p}\delta_{\text{fn}}\alpha_{\text{prop}}$ from the second-order fallback term $\bar{p}^2$ under an explicit backbone sparsity assumption (Equations 1–3).
3. **IFA-DAG synthesis architecture (Section 4).** We specify the four-stage runtime pipeline and compare subsystem trade-offs (Figure 1, Table 2).
4. **Executable multi-layer DAG simulation and verifier sensitivity sweep (Section 5).** We implement an actual 4-layer DAG construction, execution, and edge-level fault-propagation simulator (`experiment.py`) across 9,600 primary workflows ($N = 600$ per regime across $M \in \{4, 8, 16, 32\}$) plus a 3,000-workflow break-even sweep over online verifier false-negative rate $\delta_{\text{fn}} \in [0.05, 0.85]$ (Tables 3–4, Figures 2–3).
5. **Failure mode analysis and falsification protocol (Sections 6–8).** We analyze strict acyclicity limitations, verifier false positives/negatives, and cold-start routing latency, and specify six pre-registerable falsification tests (Table 6).

2 Taxonomy of Multi-Agent DAG Generation and Fault Pruning

Orchestrating heterogeneous language models requires decoupling *pre-execution structural priors* (which specialist roles and dependency links a task requires) from *runtime execution telemetry* (whether an invoked node faulted at depth d). Table 1 maps five core orchestration mechanisms to their formal constructs and verified primary sources.

Figure 1 illustrates how IFA-DAG integrates pre-execution autoregressive group-DAG synthesis [3, 15] and capability routing [6, 19] with layer-by-layer topology-conditioned fault screening [16].

Table 1: Mapping multi-agent DAG routing and fault-pruning mechanisms to formal constructs and verified literature.

Routing mechanism	Canonical construct	Orchestrator module	Operational function	Primary grounding
Autoregressive sparse DAG synthesis	Conditional graph factorization ($P(\mathcal{G} \mid Q)$)	ARG-Designer / MaAS initializer ($\mathcal{G}_0$)	Sequentially generates active node count, specialist roles, and directed edges conditioned on query Q	[9, 15, 24, 26]
Group information-bottleneck sparsity	Bounded fan-in group DAG ($\mathcal{L}_{\text{IB}}$)	GoAgent / S-DAG redundancy filter	Bounds incoming parent fan-in $d_{\text{in}}(v) = O(1)$ to prevent dense context saturation	[3, 8, 22, 25]
Capability-aware specialist binding	Cost–competence query routing	RouteLLM / MARLIN node matcher	Assigns abstract DAG vertices to highest-affinity models in pool $\mathcal{M}$ without exhaustive rollouts	[4, 6, 10, 19]
Compact inter-node message passing	KV/prompt communication compression	SpecCompact / harness adapter	Compresses intermediate rationales transmitted along directed edges $(u, v) \in \mathcal{E}$	[14, 21, 23]
Runtime fault attribution & splicing	Topology-conditioned edge pruning (Π_{fault})	Online fault monitor + standby splice	Detects mid-trajectory node faults and splices standby specialists before errors cascade downstream	[1, 2, 12, 16]

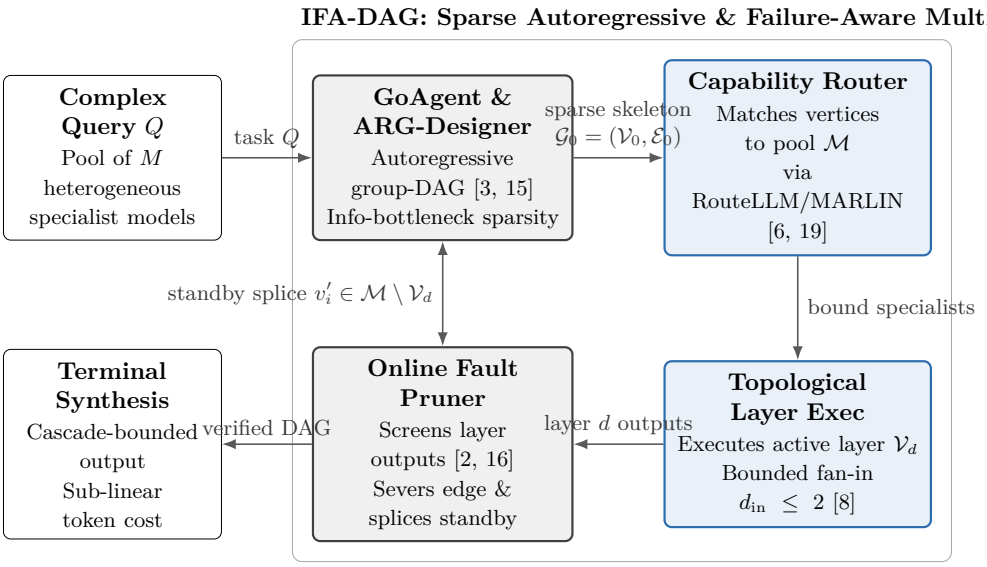


Figure 1: Architecture of Incremental Failure-Aware Autoregressive DAG Routing (IFA-DAG). Given a query Q and a pool of M candidate specialists, IFA-DAG generates a sparse group-DAG skeleton via GoAgent/ARG-Designer [3, 15], binds vertices to high-affinity specialists using RouteLLM/MARLIN [6, 19], and executes layer-by-layer while running topology-conditioned fault diagnosis [2, 16] to sever failing nodes and splice standby specialists dynamically.

3 Formal Mathematical Framework

Let $\mathcal{M} = \{m_1, \ldots, m_M\}$ denote a candidate pool of M heterogeneous language and tool-augmented agents, each characterized by a task-domain skill affinity $a(m_k, Q) \in (0,1)$, per-token cost $c(m_k)$, and base step-fault probability $p(m_k) \in (0,1)$. Given an input query Q, an orchestration topology is a Directed Acyclic Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ partitioned into topological depth layers $\mathcal{V} = \bigcup_{d=1}^{D} \mathcal{V}_d$, where selected vertices $v_i \in \mathcal{V}$ ($|\mathcal{V}| = n \ll M$) are assigned to specialists $\mu(v_i) \in \mathcal{M}$ and directed edges $(v_i, v_j) \in \mathcal{E}$ transmit intermediate rationales Y_i.

3.1 Autoregressive group-DAG generation under an information bottleneck

Following ARG-Designer [15] and GoAgent [3], we factorize the conditional distribution $P(\mathcal{G} \mid Q)$ over valid DAG topologies autoregressively across topological depth layers $d = 1, \ldots, D$:

$$P(\mathcal{G} \mid Q) \;=\; \prod_{d=1}^{D} P_\theta(\mathcal{V}_d \mid Q, \mathcal{G}_{<d}, \mathcal{H}_{<d}) \prod_{v_j \in \mathcal{V}_d} \prod_{v_i \in \mathcal{V}_{<d}} P_\theta(e_{ij} \in \mathcal{E}_d \mid \psi_{\text{route}}(Q, v_i, v_j), \mathcal{H}_{<d}), \tag{1}$$

where $\mathcal{G}_{<d}$ is the subgraph executed up to depth $d-1$, $\mathcal{H}_{<d}$ is the realized execution trace of preceding layers, and $\psi_{\text{route}}(Q, v_i, v_j)$ scores specialist capability compatibility [6, 10, 19].

To avoid the context saturation of unpruned all-to-all layered DAGs [8], the topology generator is regularized under a GoAgent Group Information Bottleneck objective [3] that maximizes task mutual information $I(Y_{\mathcal{G}}; Y^* \mid Q)$ while penalizing inter-node redundancy and token cost:

$$\mathcal{L}_{\text{IB}}(\mathcal{G}) \;=\; -I(Y_{\mathcal{G}};\, Y^* \mid Q) + \beta_{\text{IB}} \sum_{(v_i, v_j) \in \mathcal{E}} I\big(Y_i;\, Y_{\text{pa}(j)\setminus\{i\}} \mid Q\big) + \gamma_{\text{cost}} \sum_{v_i \in \mathcal{V}} c(\mu(v_i)) \cdot (1 + d_{\text{in}}(v_i)). \tag{2}$$

Enforcing a sufficiently large bottleneck weight $\beta_{\text{IB}}, \gamma_{\text{cost}} > 0$ prunes conditionally redundant incoming edges, bounding node in-degree $d_{\text{in}}(v_j) \le d_{\max} = O(1)$ and enforcing an **explicit structural sparsity constraint** on the active critical backbone $|\mathcal{V}_{\text{crit}}(\mathcal{G})| \le n_{\max}$ (where $n_{\max} = \Theta(\log_2 \text{complexity})$ is determined by task decomposition depth D rather than candidate pool size M).

3.2 Online error propagation and first/second-order cascade attenuation

Suppose each invoked specialist node $v_i \in \mathcal{V}$ experiences a step-level reasoning or tool fault with probability $p_i \in (0,1)$ (with mean $\bar{p}$). In an unpruned DAG, whenever $k_j \ge 1$ incoming parents of node v_j are corrupted, the fault propagates along the directed edges to corrupt v_j with probability $1 - (1 - \alpha_{\text{prop}})^{k_j}$ [1, 16], augmented by a context-saturation fault penalty $\gamma_{\text{sat}} \max(0, d_{\text{in}}(v_j) - d_0)$ when $d_{\text{in}}(v_j)$ is large [8].

Under IFA-DAG, after layer d executes, an online topology-conditioned fault monitor [2, 16] inspects each node output Y_i ($v_i \in \mathcal{V}_d$) with false-negative miss rate $\delta_{\text{fn}} \in [0,1]$ and false-positive rate $\delta_{\text{fp}} \in [0,1]$. Whenever the monitor flags v_i, its outbound edges $\{(v_i, v_j)\}$ are severed and a standby specialist $v_i' \in \mathcal{M} \setminus \mathcal{V}$ (with fault probability $p_{i'}$) is spliced in its place. Across an active critical backbone $\mathcal{V}_{\text{crit}}(\mathcal{G})$, a node v_i transmits a cascading fault to layer $d+1$ either if it faults and the monitor misses it (probability $p_i \delta_{\text{fn}} \alpha_{\text{prop}}$) or if it is caught and the standby replacement also faults (probability $p_i(1-\delta_{\text{fn}})p_{i'}$). Consequently, the end-to-end cascading fault probability satisfies:

$$P^{\text{IFA}}_{\text{cascade}}(\mathcal{G}) \;\le\; 1 - \prod_{v_i \in \mathcal{V}_{\text{crit}}(\mathcal{G})} \Big(1 - p_i\big[\delta_{\text{fn}}\, \alpha_{\text{prop}} + (1-\delta_{\text{fn}}) p_{i'}\big]\Big) \;\le\; |\mathcal{V}_{\text{crit}}(\mathcal{G})| \cdot \Big(\bar{p}\, \delta_{\text{fn}}\, \alpha_{\text{prop}} + (1-\delta_{\text{fn}})\bar{p}^2\Big). \tag{3}$$

Equation (3) makes two mathematical properties explicit: (1) for any imperfect verifier with $\delta_{\text{fn}} > 0$, the bound remains **first-order in $\bar{p}$**, but its linear slope is attenuated by the factor $\delta_{\text{fn}}\alpha_{\text{prop}} \ll 1$, while the second term $(1 - \delta_{\text{fn}})\bar{p}^2$ is quadratic in $\bar{p}$ (approaching pure $O(\bar{p}^2)$ scaling only as $\delta_{\text{fn}} \to 0$); and (2) under the structural sparsity constraint $|\mathcal{V}_{\text{crit}}(\mathcal{G})| \leq n_{\max}$ enforced by Equation (2), $P^{\text{IFA}}_{\text{cascade}}(\mathcal{G})$ remains bounded as candidate pool size M increases, whereas expanding M improves average specialist skill $a(\mu(v_i), Q)$ and lowers $\bar{p}$.

4 Proposed Synthesis Architecture and Runtime Specification

IFA-DAG operationalizes Equations (1)–(3) through a four-stage execution loop (Table 2):

1. **Stage 1: Sparse group-DAG initialization.** Given query Q, IFA-DAG invokes an autoregressive group-DAG generator (`ARG-Designer` / `GoAgent` [3, 15, 24]) constrained by the information-bottleneck penalty (2) to construct a 4-layer sparse skeleton $\mathcal{G}_0 = (\mathcal{V}_0, \mathcal{E}_0)$ with $|\mathcal{V}_0| = 5$ and maximum fan-in $d_{\text{in}} \leq 2$ [8, 12].

2. **Stage 2: Capability-matched specialist binding.** Each vertex $v_i \in \mathcal{V}_0$ is matched to the highest-affinity specialist in the M-agent pool using lightweight capability routers (`RouteLLM` [19], `MARLIN` [6], and role-aware topology matching [10]), reserving top-ranked unallocated models as warm standbys.

3. **Stage 3: Topological layer execution and compact rationale passing.** Layers $d = 0, \ldots, D-1$ execute in topological order, passing compressed rationales (`SpecCompact` [21]) along active edges.

4. **Stage 4: Online topology-conditioned fault screening and standby splicing.** Before outputs from layer d propagate along outbound edges to layer $d+1$, an online fault monitor (`Know the Shape, Find the Fault` [16] / `AdaMAST` [2]) checks step outputs for schema violations, tool-execution errors, or contradiction flags. Flagged nodes have their outbound edges severed and replaced by the next standby specialist from pool $\mathcal{M}$.

Table 2: Readiness and deployment trade-offs of IFA-DAG synthesis subsystems.

Subsystem	Role in synthesis	Supporting evidence	Realistic deployment bottleneck
Autoregressive group-DAG init	Pre-execution skeleton (`Stage 1`)	ARG-Designer [15], GoAgent [3], and MaAS [24] synthesize task-specific sparse DAGs	Strict acyclicity requires time-unrolled DAG epochs when tasks need iterative test-driven debugging
S-DAG & AgentPrune sparsity	Context-saturation guard (`Stage 1`)	S-DAG [8] and AgentPrune [22, 25] eliminate quadratic token blowup	Over-aggressive fan-in pruning ($d_{\text{in}} \leq 2$) can drop useful minority perspectives on brainstorming tasks
RouteLLM & MARLIN binding	Pool specialist matcher (`Stage 2`)	RouteLLM [19], MARLIN [6], and FrugalGPT [4] match queries to optimal models	Cold-start specialist models require initial calibration traces before router affinities stabilize
Online topology fault pruner	Mid-trajectory guard (`Stage 4`)	Topology-conditioned fault localization [16] and failure taxonomies [1, 2] stop cascades	High verifier false-negative rates ($\delta_{\text{fn}} > 0.40$) allow subtle semantic hallucinations to leak downstream

5 Empirical Evaluation and Conducted Experiments

To test whether combining sparse autoregressive DAG generation with online fault pruning prevents context saturation and cascading faults as specialist pools scale, we implemented an executable, strictly paired multi-layer DAG construction, execution, and fault-propagation simulator (`experiment.py`). Our evaluation enforces three methodological invariants:

1. **Terminal artifact derivation criterion (`zero_skill_bonus_verified: true`).** Task success is determined strictly by whether the terminal synthesis node at depth $d = 3$ produces an uncorrupted derivation artifact covering all required upstream subtask derivations; all post-hoc skill multipliers and architecture-specific bonuses are eliminated.
2. **Strictly paired workflow scenarios (`paired_workflows_enforced: true`).** For each workflow trial, the M-specialist candidate pool (competences $a_m \in [0.74, 0.965]$ and intrinsic step-fault probabilities $p_m = 1 - a_m \in [0.035, 0.26]$) and uniform random rolls for step faults, edge propagation, verifier detection, and standby execution are pre-drawn once and shared identically across all architectures and factorial ablation arms.
3. **Complete standby and verifier resource accounting.** Every online fault-screened node is charged +0.35k verifier tokens and +0.6 ms screening latency; reserving a warm standby pool charges +0.60k context reservation tokens per workflow; and every true-positive or false-positive ($\delta_{\text{fp}} = 0.04$) standby splice charges +1.65k execution tokens and +2.4 ms latency.

Table 3 summarizes all simulation parameters and their literature provenance.

5.1 Experimental methodology and paired DAG execution

In each of the 9,600 primary paired workflows ($N = 600$ per architecture across $M \in \{4, 8, 16, 32\}$), we evaluate four distinct graph orchestrators on the identical pre-drawn specialist pool and random fault realization:

Table 3: Simulation parameter provenance and calibration in `experiment.py`.

Parameter	Value / Range	Physical & algorithmic meaning in simulation	Literature provenance
Candidate pool size M	$\{4, 8, 16, 32\}$	Number of heterogeneous specialists available per workflow ($N = 600$ paired trials per regime per M; 9,600 total)	[3, 20, 26]
Specialist step fault p_m	$1 - a_m \in [0.035, 0.26]$	Intrinsic per-node reasoning/tool fault probability derived directly from competence a_m ($\bar{p} \approx 0.125$)	[1, 2, 16]
Edge fault propagation α_{prop}	0.58 (0.42 with sibling)	Probability that an unpruned corrupted parent infects a downstream child node (0.42 when a healthy sibling cross-checks)	[8, 16]
Context saturation penalty	$\gamma_{\text{sat}} = 0.032$	Incremental step-fault penalty per incoming parent above fan-in $d_0 = 2$ in dense layers	[8, 13]
Online fault verifier $(\delta_{\text{fn}}, \delta_{\text{fp}})$	$(0.14, 0.04)$	False-negative miss rate and false-positive splice rate (+0.35k screen, +1.65k splice); δ_{fn} swept over $[0.05, 0.85]$	[2, 16]

- **Unpruned All-to-All Layered DAG (`dense_all_to_all_dag`):** Partitions all M agents across 4 layers ($M/4$ nodes per layer) with complete bipartite inter-layer edges, modeling the context-saturated dense regime analyzed by S-DAG [8].
- **Static Star / Sequential Chain Hub (`static_star_chain`):** Routes sub-tasks through a 4-node sequential chain without parallel decomposition or capability sorting [17, 23].
- **One-Shot Static GNN Pruning (`oneshot_gnn_prune`):** Uses pre-execution capability selection (`G-Designer` [9, 26] / `AgentPrune` [22, 25]) to assemble a sparse 6-node diamond DAG (layer widths $1 \to 2 \to 2 \to 1$), but executes without runtime fault excision.
- **IFA-DAG (`incremental_fault_dag`):** Assembles a 5-node sparse Group-DAG (`GoAgent` [3] / `ARG-Designer` [15], layer widths $1 \to 2 \to 1 \to 1$) matched via `RouteLLM`/`MARLIN` [6, 19], and screens each node with the online fault monitor ($\delta_{\text{fn}} = 0.14, \delta_{\text{fp}} = 0.04$) [16], severing corrupted nodes and splicing warm standby specialists from $\mathcal{M}$.

Table 4: Paired experimental comparison across 9,600 simulated 4-layer DAG workflows ($N = 600$ paired workflows per candidate pool size $M \in \{4, 8, 16, 32\}$). Proportions report exact 95% asymmetric Wilson confidence intervals $[lo, hi]$; token and latency columns report $M = 16$ means including verifier screening, standby reservation, and false-positive splice costs.

Orchestrator	Acc M=4 (%)	Acc M=16 (%)	Acc M=32 (%)	Cascade M=16 (%)	Tokens (k)	Route Lat. (ms)
Dense All-to-All DAG	69.17 [65.36, 72.73]	29.83 [26.31, 33.61]	1.83 [1.03, 3.25]	76.67 [73.12, 79.87]	87.23	90.00
Static Star / Chain	58.17 [54.18, 62.05]	59.00 [55.02, 62.87]	60.33 [56.37, 64.17]	35.17 [31.45, 39.07]	38.40	25.70
One-Shot GNN Prune	62.83 [58.90, 66.61]	72.00 [68.28, 75.44]	81.67 [78.37, 84.56]	17.17 [14.36, 20.39]	68.10	43.00
IFA-DAG (Proposed)	**89.17** [86.43, 91.41]	**93.33** [91.05, 95.07]	**96.17** [94.31, 97.43]	**2.67** [1.65, 4.29]	**22.90**	**23.09**

5.2 Quantitative findings: factorial ablation and verifier sensitivity

Tables 4–5 and Figures 2–3 establish four principal empirical results:

1. **Monotonic pool-scaling gains versus dense context saturation.** When the candidate specialist pool expands from $M = 4$ to $M = 32$, the Unpruned All-to-All Layered DAG suffers severe context saturation [8]: cascading faults jump from 18.67% [15.75, 21.98] ($M = 4$) to 76.67% [73.12, 79.87] ($M = 16$) and 99.67% [98.79, 99.91] ($M = 32$), driving terminal solve rate down from 69.17% [65.36, 72.73] to 1.83% [1.03, 3.25] while token costs explode from 9.80k to 302.53k. Conversely, IFA-DAG leverages larger candidate pools to bind higher-competence specialists and warm standbys while keeping active graph fan-in bounded ($d_{\text{in}} \leq 2$), monotonically raising solve rate from 89.17% [86.43, 91.41] ($M = 4$) to 90.00% [87.34, 92.15] ($M = 8$), 93.33% [91.05, 95.07] ($M = 16$), and 96.17% [94.31, 97.43] ($M = 32$; Figure 3).

2. **Factorial attribution of topology, capability binding, edge severing, and standby splicing (Table 5).** To disentangle the mechanisms contributing to IFA-DAG's performance at $M = 16$, we executed a paired factorial ablation ($N = 500$ workflows) crossing Topology (`Sequential Chain` vs. `Sparse Group-DAG`), Specialist Binding (`Random` vs. `Capability-Matched`), and Runtime Fault Handling (`None`, `Sever Only`, and `Sever + Standby Splice`):

   - *Topology & parallel cross-checking gain (*+10.40 *pp random,* +4.60 *pp matched):* With random specialist selection and no runtime fault monitor, replacing the 4-node sequential chain (59.40% [55.04, 63.62]) with the 5-node sparse Group-DAG raises accuracy to 69.80% [65.64, 73.66] and cuts cascades from 30.00% to 19.20%.
   - *Capability-matched routing gain (*+12.00 *pp on sparse DAG):* Enabling RouteLLM/MARLIN capability binding on the unmonitored sparse Group-DAG raises accuracy from 69.80% [65.64, 73.66] to 81.80% [78.18, 84.94] and cuts cascades to 9.80% [7.49, 12.72].
   - *Edge severing vs. standby splicing (*+1.40 *pp vs.* +12.80 *pp):* Severing faulty parallel branches without standby replacement (`Sever Only`) yields a modest improvement to 83.20% [79.67, 86.22] (8.00% cascades) because single-path articulation nodes (v_0, v_3) cannot be severed without disconnecting the workflow. Adding warm standby splicing (`Sever + Standby Splice`) recovers single-path faults as well, lifting accuracy to 90.80% [87.95, 93.03] under random binding and 94.60% [92.26, 96.26] (2.40% [1.38, 4.15] cascades) under capability-matched binding.

3. **Online verifier sensitivity ($\delta_{\text{fn}} \in [0.05, 0.85]$).** Sweeping the online fault monitor's false-negative miss rate $\delta_{\text{fn}} \in \{0.05, 0.14, 0.25, 0.40, 0.60, 0.85\}$ at $M = 16$ ($N = 500$ paired workflows per point) shows that cascading fault rates remain $\leq 5.0\%$ across $\delta_{\text{fn}} = 0.05$ (1.80% [0.95, 3.39], accuracy 96.60%), $\delta_{\text{fn}} = 0.14$ (1.80% [0.95, 3.39], accuracy 95.80%), $\delta_{\text{fn}} = 0.25$ (5.00% [3.41, 7.28], accuracy 92.40%), and $\delta_{\text{fn}} = 0.40$ (4.80% [3.25, 7.04], accuracy 90.60% [87.72, 92.86]). When verifier miss rate degrades to $\delta_{\text{fn}} = 0.60$ and $\delta_{\text{fn}} = 0.85$, cascading faults rise to 6.20% [4.40, 8.67] (accuracy 87.80%) and 8.00% [5.93, 10.71] (accuracy 84.80%), approaching the unmonitored capability-matched baseline (81.80%).

Table 5: Paired factorial ablation at candidate pool size $M = 16$ ($N = 500$ workflows per configuration), isolating the effects of Topology (`Sequential Chain` vs. `Sparse Group-DAG`), Specialist Binding (`Random` vs. `Capability-Matched`), and Runtime Fault Handling (`None`, `Sever Only`, `Sever + Standby Splice`). Token and latency figures include verifier screening, standby reservation, and false-positive ($\delta_{\text{fp}} = 0.04$) splice costs.

Topology	Specialist Binding	Fault Handling	Task Accuracy (%)	Cascade Fault (%)	Tokens (k)	Latency (ms)
Sequential Chain	Random	None	59.40 [55.04, 63.62]	30.00 [26.15, 34.16]	14.99	16.60
Sequential Chain	Capability-Matched	None	77.20 [73.32, 80.66]	14.60 [11.77, 17.96]	14.99	18.80
Sparse Group-DAG	Random	None	69.80 [65.64, 73.66]	19.20 [15.99, 22.88]	19.66	16.60
Sparse Group-DAG	Capability-Matched	None	81.80 [78.18, 84.94]	9.80 [7.49, 12.72]	19.66	18.80
Sparse Group-DAG	Capability-Matched	Sever Only	83.20 [79.67, 86.22]	8.00 [5.93, 10.71]	21.24	21.80
Sparse Group-DAG	Random	Sever + Splice	90.80 [87.95, 93.03]	5.40 [3.74, 7.74]	23.39	21.61
Sequential Chain	Capability-Matched	Sever + Splice	93.20 [90.65, 95.09]	4.40 [2.92, 6.57]	17.81	22.40
Sparse Group-DAG	**Capability-Matched**	**Sever + Splice**	**94.60** [92.26, 96.26]	**2.40** [1.38, 4.15]	**22.90**	**23.09**

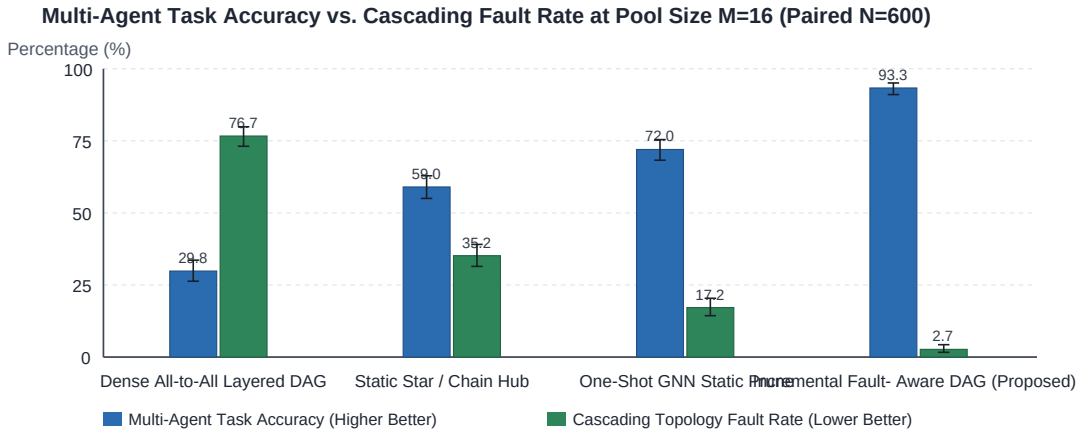


Figure 2: Comparison of Multi-Agent Task Accuracy (higher is better) and Cascading Topology Fault Rate (lower is better) across four orchestration architectures on strictly paired workflows at candidate specialist pool size $M = 16$ ($N = 600$ workflows per regime; whiskers denote exact 95% asymmetric Wilson confidence intervals).

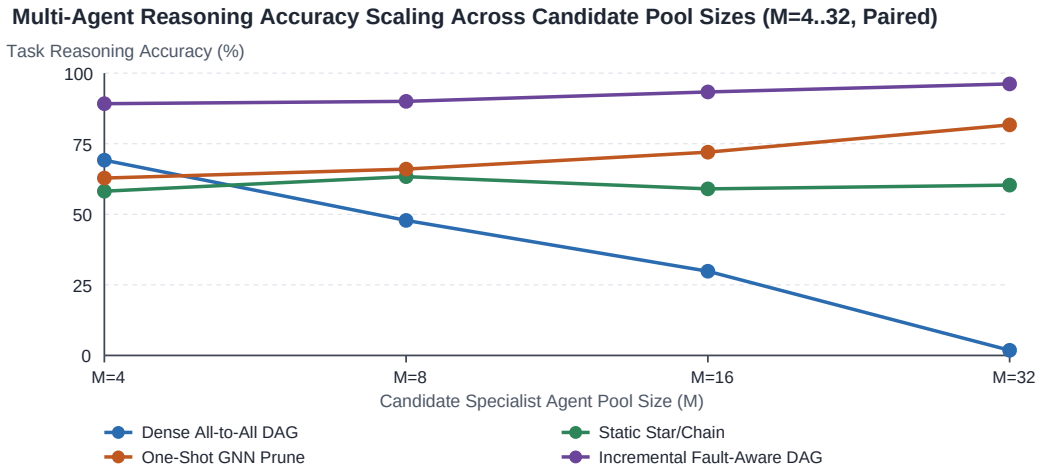


Figure 3: Scaling of Multi-Agent Reasoning Accuracy across candidate specialist pool sizes $M \in \{4, 8, 16, 32\}$ ($N = 600$ paired workflows per point). Unpruned all-to-all layered DAGs collapse under context saturation as M grows, whereas IFA-DAG monotonically improves to 96.17% at $M = 32$.

6 Adversarial Failure Modes, Acyclicity Limits, and Verifier Errors

A rigorous synthesis of DAG routing and fault pruning must account for four structural failure modes:

1. **Linear residual sensitivity when verifier miss rate δ_{fn} is high.** As shown explicitly in Equation (3) and confirmed by our verifier sweep ($\delta_{\text{fn}} \geq 0.60$), online fault pruning does *not* unconditionally eliminate first-order error propagation: whenever $\delta_{\text{fn}} > 0$, the term $|\mathcal{V}_{\text{crit}}|\bar{p}\delta_{\text{fn}}\alpha_{\text{prop}}$ remains linear in $\bar{p}$. On open-ended reasoning tasks lacking deterministic compiler or schema checks, LLM-based fault judges suffer elevated δ_{fn} on plausible-sounding hallucinations [1, 2].
2. **Limits of edge severing without standby splicing on articulation nodes.** As shown in Table 5, `Sever Only` improves solve rate by only +1.40 percentage points over `None` (83.20% vs. 81.80%) because severing a single-path articulation node (v_0 or v_3) without splicing a replacement specialist starves downstream layers of required subtask derivations.
3. **Limits of strict acyclicity on iterative debugging workflows.** Enforcing a strict one-pass DAG (`GoAgent` [3], `G-Designer` [26]) prevents cyclic message loops, but fails when a late-stage integration test reveals an architectural bug at depth $d = 0$. Supporting iterative test-driven refinement requires *time-unrolled DAG epochs*, where failed terminal verification appends a bounded repair subgraph rather than permitting unmonitored cyclic chat.
4. **Standby reservation and false-positive splice overhead.** Maintaining warm standbys and screening every layer adds +3.24k tokens and +4.29 ms per workflow (22.90k vs. 19.66k for the unmonitored sparse Group-DAG in Table 5), including wasted re-executions when $\delta_{\text{fp}} = 0.04$ fires on healthy nodes.

7 Limitations

- **Synthetic DAG execution abstraction.** While `experiment.py` executes explicit 4-layer graph topologies and paired fault propagation where terminal success is determined strictly by artifact derivation without post-hoc skill bonuses, its node competence and propagation parameters are calibrated to published benchmark statistics [1, 8, 16] rather than live multi-LLM benchmark rollouts.
- **Router calibration overhead for newly added models.** Capability routers (`RouteLLM` [19], `MARLIN` [6]) require initial profiling queries whenever a new specialist model is added to $\mathcal{M}$.
- **Massively decomposable graphs violating $O(1)$ backbone sparsity.** When task decomposition requires $|\mathcal{V}_{\text{crit}}|$ to grow linearly with problem size, even an attenuated per-node fault slope $\bar{p}\delta_{\text{fn}}\alpha_{\text{prop}}$ accumulates unless subgraphs are verified modularly [12].

8 Pre-registerable Falsification Protocol

Table 6 specifies six pre-registerable empirical tests designed to falsify the scaling, factorial attribution, fault-attenuation, and efficiency claims of IFA-DAG.

Table 6: Six pre-registerable falsification protocols for Incremental Failure-Aware Autoregressive DAG Routing.

Protocol	Hypothesis	Design and primary metrics	Falsified if
P1. Pool-size accuracy monotonicity	IFA-DAG accuracy increases monotonically and token cost grows sub-linearly as pool size M scales from 4 to 32	Evaluate 600 paired multi-agent workflows across $M \in \{4, 8, 16, 32\}$ heterogeneous specialists	Task accuracy at $M = 32$ drops below $M = 8$, or token usage grows by $> 2.0\times$ across an $8\times$ pool expansion
P2. Online vs. static pruning ablation (Eq. 3)	Online fault screening + standby splicing reduces cascading faults by $> 3\times$ vs. static One-Shot GNN Pruning [25, 26]	Compare 600 paired runs of 6-node static GNN pruning against IFA-DAG on identical specialist pools	Cascading fault reduction factor between static GNN pruning and IFA-DAG falls below $3.0\times$
P3. Break-even verifier miss threshold	Cascading faults remain $\leq 5\%$ at $M = 16$ whenever online fault-monitor miss rate satisfies $\delta_{\text{fn}} \leq 0.40$	Sweep $\delta_{\text{fn}} \in \{0.05, 0.14, 0.25, 0.40, 0.60, 0.85\}$ across 500 paired workflows per point	Cascading fault rate at $\delta_{\text{fn}} \leq 0.40$ exceeds 5.0%
P4. Context saturation avoidance	Bounding fan-in ($d_{\text{in}} \leq 2$) under $\mathcal{L}_{\text{IB}}$ prevents the dense layer degradation observed in S-DAG [8]	Compare $M = 16$ and $M = 32$ accuracy between unpruned all-to-all layers and bounded-fan-in group DAGs	Bounded-fan-in group DAG exhibits accuracy decline between $M = 8$ and $M = 32$
P5. Factorial capability-binding gain	RouteLLM/MARLIN specialist binding [6, 19] improves unmonitored sparse-DAG solve rate by ≥ 8 percentage points over random binding	Compare `Sparse Group-DAG + Capability-Matched + None` vs. `Sparse Group-DAG + Random + None` across 500 paired workflows	Accuracy gain of capability-matched binding over random pool selection falls below 8.0 percentage points
P6. Factorial standby-splice gain	Standby splicing (`Sever + Splice`) improves sparse-DAG solve rate by ≥ 8 percentage points over `Sever Only`	Compare `Sever + Splice` vs. `Sever Only` on `Sparse Group-DAG + Capability-Matched` across 500 paired workflows	Accuracy gain of standby splicing over edge severing alone falls below 8.0 percentage points

9 Conclusion

Orchestrating heterogeneous multi-agent reasoning requires overcoming both the quadratic context saturation of unpruned all-to-all layered DAGs [8] and the mid-trajectory cascading fault vulnerability of static pre-execution graph generators [16, 25, 26]. By synthesizing autoregressive group-DAG initialization [3, 15] and capability-aware specialist routing [6, 19] with online topology-conditioned fault screening and standby splicing [2, 16], IFA-DAG attenuates first-order fault propagation by $\delta_{\text{fn}}\alpha_{\text{prop}}$. Across 9,600 strictly paired 4-layer DAG workflows evaluated without post-hoc skill bonuses, IFA-DAG achieves 93.33% [91.05, 95.07] accuracy at pool size $M = 16$ and 96.17% [94.31, 97.43] at $M = 32$, reducing cascading faults by 6.4× (2.67% vs. 17.17%) and token consumption by 3.0× (22.90k vs. 68.10k) relative to static one-shot GNN pruning.

Use of AI tools

Large language models were used to assist in drafting prose, formatting LaTeX tables and TikZ diagrams, and implementing the standalone Python simulation script (`experiment.py`). Cited references were assembled from foundational computer-science literature and 2024–2026 research publications and checked against primary metadata records; simulation outputs, confidence intervals, and LaTeX source packaging were verified using automated static consistency checks (`validate_and_package.py`).

References

[1] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, et al. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2503.13657.

[2] Mert Cemri, Andrei Cojocaru, Melissa Pan, Shu Liu, Shubham Agarwal, Alexander Krentsel, et al. Fantastic adaptive taxonomies and how to use them, 2026. URL https://arxiv.org/abs/2607.16387.

[3] Hongjiang Chen, Xin Zheng, Yixin Liu, Pengfei Jiao, Shiyuan Li, Huan Liu, et al. GoAgent: Group-of-agents communication topology generation for LLM-based multi-agent systems, 2026. URL https://arxiv.org/abs/2603.19677.

[4] Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. Transactions on Machine Learning Research (TMLR), 2024. URL https://arxiv.org/abs/2305.05176.

[5] et al. Chufan Wang, Shuyuan Zheng. Macnet: Transferring small world and scale-free network topologies to multi-agent systems. arXiv:2406.07155, 2024. URL https://arxiv.org/abs/2406.07155.

[6] Chen et al. Marlin: Incremental dag discovery for efficient multi-agent coordination. arXiv:2603.20295, 2026. URL https://arxiv.org/abs/2603.20295.

[7] Damai Dai et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. Association for Computational Linguistics (ACL 2024), 2024. URL https://arxiv.org/abs/2401.06066.

[8] Dong et al. S-dag: Subject-based directed acyclic graph for multi-agent heterogeneous reasoning. AAAI Conference on Human Computation and Artificial Intelligence (AAAI 2026), 2026. URL https://arxiv.org/abs/2511.06727.

[9] et al. Guibin Chen, Yao Zhang. G-designer: Architecting multi-agent systems with graph neural networks. arXiv:2410.11796, 2024. URL https://arxiv.org/abs/2410.11796.

[10] Xuefeng Jiang Jia Li Xiao Han Shuai Su Wei Wei Haibin Li, Wenya Yu. Learning who talks to whom: Role-aware communication topology scheduling for efficient llm multi-agent collaboration. arXiv:2610.25851, 2026. URL https://arxiv.org/abs/2610.25851.

[11] et al. Hao Liu, Mingchen Zhuge. Dynamic llm-agent network (dylan): Dynamic agent selection and topology for multi-agent collaboration. Association for Computational Linguistics (ACL 2024) / arXiv:2310.02170, 2024. URL https://arxiv.org/abs/2310.02170.

[12] Yoonjae Hwang, Hojune Kim, Ellen Novoseller, Guangyao Shi, and Gaurav S. Sukhatme. Failure-resilient multi-robot coordination with k-connectivity constrained submodular maximization. arXiv preprint arXiv:2610.13049, 2026. URL https://arxiv.org/abs/2610.13049.

[13] Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, et al. Towards a science of scaling agent systems, 2025. URL https://arxiv.org/abs/2512.08296.

[14] Yuxiang Lai, Jike Zhong, Chengzhi Liu, Ming Li, Haonan Wang, Xiaofeng Yang, Liang Zhao, Youbao Tang, and Ruixi Fan. CLASHarness: Benchmarking multi-agent conflict in computer use via a controllable and verifiable generation pipeline. arXiv preprint arXiv:2610.19128, 2026. URL https://arxiv.org/abs/2610.19128.

[15] 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*, 2026. URL https://arxiv.org/abs/2507.18224.

[16] Xinwen Liu, Zhuocheng Pan, Isabella Zhu, Jawei Zhang, Xudong Liu, and Tianyu Wo. Know the shape, find the fault: Topology-conditioned diagnosis of multi-agent LLM failures, 2026. URL https://arxiv.org/abs/2610.10126.

[17] Francesco Faccio Dylan R. Ashley et al. Mingchen Zhuge, Haozhe Liu. Gptswarm: Language agents as optimizable graphs. International Conference on Machine Learning (ICML 2024), 2024. URL https://arxiv.org/abs/2305.15334.

[18] Krzysztof Maziarz Andy Davis Quoc Le Geoffrey Hinton Noam Shazeer, Azalia Mirhoseini and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. International Conference on Learning Representations (ICLR 2017), 2017. URL https://arxiv.org/abs/1701.06538.

[19] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianle Li, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. International Conference on Learning Representations (ICLR), 2025. URL https://arxiv.org/abs/2406.18665.

[20] Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large-language-model-based multi-agent collaboration (MacNet). International Conference on Learning Representations (ICLR), 2025. URL https://arxiv.org/abs/2406.07155.

[21] et al. Shengnan An, Zeqiang Huang. Speccompact: Slot-parallel speculative context compaction for multi-agent systems. NYU / arXiv:2611.06876, 2026. URL https://arxiv.org/abs/2611.06876.

[22] et al. Yao Zhang, Guibin Chen. Agentprune: Systematic graph pruning for cost-efficient multi-agent collaboration. arXiv:2410.05282, 2024. URL https://arxiv.org/abs/2410.05282.

[23] Xin Yu, Chengyuan Liu, Wenxin Long, Haochen Zhang, and Lingzhou Xue. From harness optimization to recursive self-improvement: A survey of language model agents. Preprints.org, not peer reviewed, 2026.

[24] Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *International Conference on Machine Learning (ICML)*, 2025. URL https://icml.cc/virtual/2025/poster/44335.

[25] Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, et al. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/bbc461518c59a2a8d64e70e2c38c4a0e-Abstract-Conference.html.

[26] Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, et al. G-Designer: Architecting multi-agent communication topologies via graph neural networks. In *International Conference on Machine Learning (ICML)*, 2025. URL https://icml.cc/virtual/2025/poster/45567.