DyTopo: Recursive Self-Improvement through Topology Evolution for Multi-Agent Reasoning

Anonymous authors
Paper under double-blind review

Abstract

As agents revise a solution, the feedback they need also changes. A communication graph chosen before reasoning may miss the exchanges needed for later corrections. We introduce **DyTopo**, a framework for recursive self-improvement through communication topology evolution. Each round, workers generate task messages and describe what information they need and can provide. Semantic matching converts these descriptions into a directed graph that selects which private messages enter each worker's next-round context. These contexts shape the next outputs and descriptors, so the system uses its own reasoning to revise both the solution and the routes for further feedback. A Manager updates the shared goal and determines when to stop. This adaptation occurs within a task, with worker models and the matching rule held fixed. Across four coding and mathematical-reasoning benchmarks and four LLM backbones, DyTopo ranks first in all 16 settings, with a mean gain of 5.2 percentage points over the strongest agent baseline per setting. On HumanEval with MiMo-V2-Flash, it uses about half the tokens and runtime of AgentScope and MasRouter. Semantic feedback selection experiments show gains at fixed edge counts; additional rounds can reduce accuracy.

1 Introduction

Multi-agent reasoning requires decisions about both what to compute and which intermediate results to share. An agent's current reasoning determines what information it needs, and the feedback it receives shapes its next step. Role-specialized LLM agents distribute solution generation and checking across interacting participants (Li et al., 2023; Wu et al., 2024; Hong et al., 2023; Du et al., 2023). Their collective performance depends on whether useful contributions reach the agents that can incorporate them. The graph determines which peer contributions inform each agent's revisions.

This structure need not remain appropriate throughout a task. Early coding rounds may require algorithm selection and interface design; later rounds may depend on a short exchange between a Developer and a Tester. Broadcasting every output can burden context with irrelevant material, while a fixed sparse graph can omit a newly needed connection. The choice of information exchange has long been studied in cooperative multi-agent systems (Goldman & Zilberstein, 2003). For LLM agents, changing information needs make round-level coordination a concrete part of the problem (Liu et al., 2024). Surveys place communication alongside collaboration and infrastructure design (Tran et al., 2025; Li et al., 2024). Other work examines tool creation (Wölflein et al., 2025) and agent trustworthiness (Yu et al., 2025).

Recursive self-improvement (RSI) systems such as SICA and the Darwin Gödel Machine revise reusable agent code (Robeyns et al., 2025; Zhang et al., 2026). AIDE^2 revises research-agent implementations (Srikanth et al., 2026); Meta^n constructs solver layers from earlier code and traces (Kim et al., 2026). DyTopo studies RSI through the communication graph within a single task. Its workers describe what they need and can offer, and these descriptions select messages for their next contexts. The received feedback can then change the solution, the descriptors, and the next graph. In this way, the system's own outputs revise the communication structure used for subsequent improvement. We call this within-task structural RSI: roles, model parameters, the encoder, and the

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.

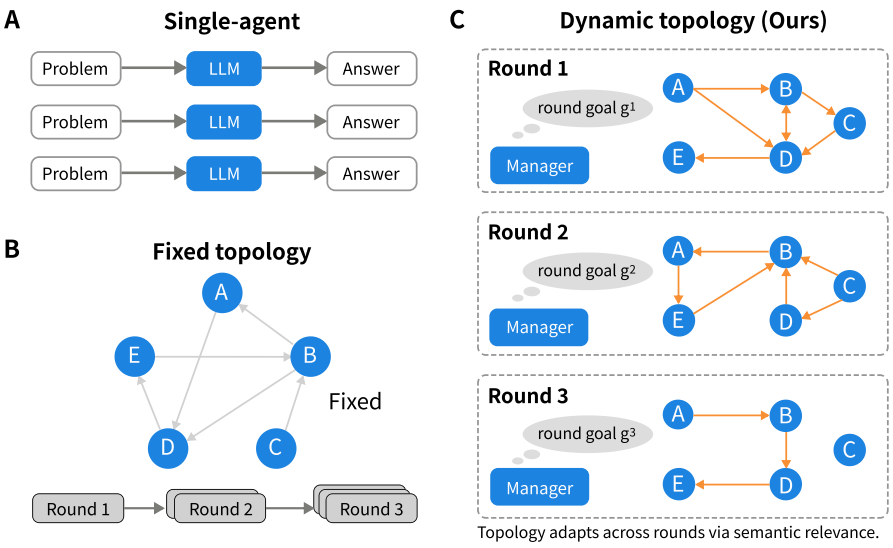


Figure 1: Communication structures for multi-agent reasoning. Single-agent prompting (A) and a fixed multi-agent topology (B) are compared with DyTopo (C), which reconstructs communication links from the agents' current information needs and offers.

graph-construction rule remain fixed. Improvement is measured by final task performance; individual rounds need not improve the answer.

DyTopo implements this loop with semantic query–key matching. In each round, every worker produces task messages and two short descriptors: a *query* for what it needs and a *key* for what it can provide. A fixed encoder embeds these descriptors, and thresholded cosine similarity selects directed provider-to-consumer edges. Messages travel along those edges and enter the recipients' next-round memories. The Manager reads public outputs, updates the shared goal, and decides whether the task is complete. The graphs record changes in information flow as reasoning proceeds.

Prior methods address task-conditioned graph design (Zhang et al., 2024b; Jiang et al., 2025), routing among agent components (Yue et al., 2025; Zhang et al., 2025b), graph optimization (Zhuge et al., 2024), and learned orchestration (Dang et al., 2025). We study how feedback received through one graph changes the reasoning state from which the next graph is constructed. The same workers can thereby change their feedback sources during a task without retraining a router.

We formulate structural RSI as a coupled update of agent memories and communication graphs, separating the fixed routing rule from the graph constructed from current outputs. DyTopo implements this update through semantic need–offer matching, shared goals, and adaptive stopping. Our experiments measure final reasoning performance and test the contribution of feedback selection. DyTopo leads all 16 benchmark–backbone settings, with a mean gain of 5.2 points over the strongest agent baseline per setting. Matched-round comparisons assess efficiency independently of early stopping; semantic feedback selection experiments test message relevance at fixed edge counts.

2 Related Work

Recursive self-improvement. Self-improvement methods differ in what they revise and retain. Self-Refine revises answers (Madaan et al., 2023), Reflexion retains verbal experience (Shinn et al., 2023), and ADAS searches agent programs (Hu et al., 2025). SICA selects its best archived agent to edit its own code, while the Darwin Gödel Machine explores multiple agent lineages (Robeyns et al., 2025; Zhang et al., 2026). AIDE^2 extends code revision to research agents evaluated on AI R&D tasks under fixed budgets (Srikanth et al., 2026). Meta^n holds a meta-operation fixed while recursively constructing solver layers from earlier code and traces (Kim et al., 2026). RSIAgent constructs environment-specific memory through exploration and freezes it for downstream use (Zhu et al., 2026). DyTopo updates the communication graph during an ongoing task. Each graph routes

feedback that conditions subsequent outputs and graph construction. Its RSI process therefore operates on the information available for further solution revision, with fixed model parameters, roles, encoder, and matching rule. The revised graph is used within the current task; we do not evaluate whether it transfers to later tasks.

Multi-agent reasoning. CAMEL studies role-playing dialogue (Li et al., 2023), AutoGen provides programmable conversations (Wu et al., 2024), and MetaGPT organizes agents through operating procedures (Hong et al., 2023). Multi-agent debate uses repeated proposals and critiques to improve reasoning (Du et al., 2023); HuggingGPT delegates subtasks to specialized models (Shen et al., 2023). These approaches specify how agents contribute and exchange feedback. DyTopo reconstructs communication links from the workers' current needs and offers. Their outputs determine both the available messages and their routes to peers.

Selective communication. TarMAC learns targeted communication in cooperative multi-agent reinforcement learning (Das et al., 2019). Conditional expert activation (Fedus et al., 2022) and content-based sparse attention (Roy et al., 2021) similarly select which computations exchange information. For LLM agents, AgentPrune removes low-value connections from a spatio-temporal graph (Zhang et al., 2024a), G-Designer generates task-conditioned topologies (Zhang et al., 2024b), and GTD uses graph diffusion for topology generation (Jiang et al., 2025). DyTopo applies selection repeatedly during inference through semantic matching of natural-language descriptors, without training a graph generator.

Topology design and orchestration. AdaptOrch selects canonical topologies using task dependencies (Yu, 2026); OFA-MAS generates task-specific graphs with a mixture-of-experts model (Li et al., 2026). MasRouter learns collaboration modes and role assignments (Yue et al., 2025), while AgentRouter uses a knowledge graph to guide routing (Zhang et al., 2025b). AgentOrchestra defines a protocol for hierarchical delegation (Zhang et al., 2025a). GPTSwarm optimizes both node prompts and graph connectivity (Zhuge et al., 2024), and Puppeteer learns a centralized policy for successive agent activations (Dang et al., 2025). DyTopo reconstructs its peer communication graph from current outputs inside each reasoning trajectory. The explicit graph lets us replace selected edges while keeping the worker models fixed, as in our semantic feedback selection experiments.

3 METHOD

DyTopo represents a reasoning trajectory as a **Dynamic Computation Graph** (DCG), $\mathcal{G} = \{G^{(t)}\}_{t=0}^{T-1}$, with $T \leq T_{\max}$. Let $X^{(t)} = (C_{task}^{(t)}, \mathcal{H}_1^{(t)}, \ldots, \mathcal{H}_N^{(t)})$ collect the shared goal and worker memories before round t, and let $\mathcal{O}^{(t)} = \{O_i^{(t)}\}_{i=1}^{N}$ collect the worker outputs. Each round consists of three updates:

$$\begin{aligned} \mathcal{O}^{(t)} &\sim \mathcal{P}(\cdot \mid X^{(t)}), \\ G^{(t)} &= \mathcal{R}_\tau\Big(\mathrm{Desc}(\mathcal{O}^{(t)})\Big), \\ \big(X^{(t+1)}, y^{(t)}\big) &\sim \mathcal{T}\Big(\cdot \mid X^{(t)}, \mathcal{O}^{(t)}, G^{(t)}\Big). \end{aligned} \tag{1}$$

Here $\mathcal{P}$ denotes worker generation, Desc extracts query and key descriptors, and $\mathcal{R}_\tau$ is the fixed matching rule. The transition $\mathcal{T}$ combines memory updates with the Manager's goal and stopping decision. The graph selects which private messages enter the next state. This is the recursive step: current outputs determine the feedback available for subsequent reasoning. All workers generate before routing, so $G^{(t)}$ affects worker generation only in subsequent rounds (Figure 2; Appendix E).

3.1 AGENT STATE AND FEEDBACK GENERATION

Let $\mathcal{A} = \{a_1, \ldots, a_N\}$ be the worker agents. Each worker has a fixed role ρ_i and a local memory $\mathcal{H}_i^{(t)}$ containing information available before round t. A separate Manager broadcasts the round goal $C_{task}^{(t)}$. Public messages are visible to the Manager; private messages are subject to graph-based routing. We initialize local memories as empty and supply the task through $C_{task}^{(0)}$.

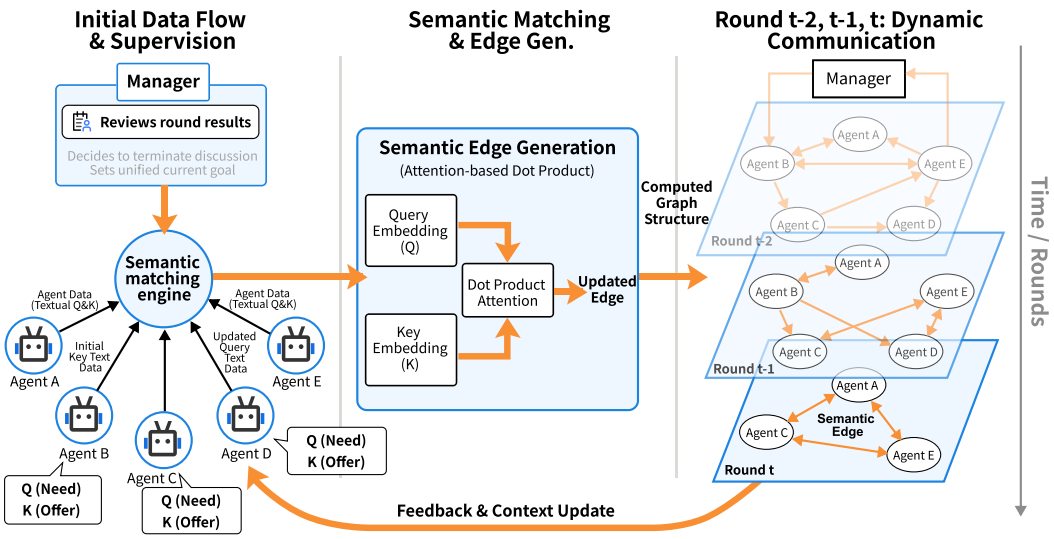


Figure 2: DyTopo's recursive communication loop. Workers generate messages and descriptors at round t. Matching constructs $G^{(t)}$ and routes private messages into $\mathcal{H}_i^{(t+1)}$. The Manager updates the goal; updated contexts condition the next outputs and graph.

3.1.1 Single-pass generation

Each worker makes one generation call per round:

$$S_i^{(t)} = [\rho_i;\, C_{task}^{(t)};\, \mathcal{H}_i^{(t)}], \qquad O_i^{(t)} \sim \pi_{\theta_i}(\cdot \mid S_i^{(t)}). \tag{2}$$

Its output is $O_i^{(t)} = \langle m_{pub,i}^{(t)}, m_{priv,i}^{(t)}, s_{q,i}^{(t)}, s_{k,i}^{(t)} \rangle$: a public contribution, private message content, and query and key descriptors. The query states what information the agent currently needs; the key states what it can offer. Messages and descriptors are generated together, before the round's edges are selected. There is no second generation call conditioned on the newly constructed graph. After graph construction, only selected recipients receive private content.

3.1.2 Manager update and stopping

The Manager maps ordered public outputs to a stopping decision and the next goal:

$$\begin{aligned} S_{global}^{(t)} &= [C_{task}^{(t)};\, \Sigma_{\sigma^{(t)}}(\{m_{pub,i}^{(t)}\}_{i=1}^{N})], \\ \langle y^{(t)}, C_{task}^{(t+1)} \rangle &\sim \Pi_{meta}(\cdot \mid S_{global}^{(t)}). \end{aligned} \tag{3}$$

Here $y^{(t)} \in \{0, 1\}$ indicates termination. The Manager is prompted to require a solution and a successful verification report before stopping. When it continues, the next goal focuses on unresolved subgoals. This decision is based on the information in the agent conversation; it does not by itself certify correctness against the benchmark evaluator. Execution also ends at the round cap.

3.2 Topology evolution through semantic matching

The descriptors expose the workers' current information needs and available contributions. DyTopo uses them to reconstruct $G^{(t)} = (\mathcal{A}, \mathcal{E}^{(t)})$ at every round, including the first.

3.2.1 Descriptor embedding and relevance

A fixed encoder $\mathrm{Emb} : \mathcal{D} \to \mathbb{R}^d$ maps descriptors to vectors. We compute $\mathbf{q}_i^{(t)} = \mathrm{Emb}(s_{q,i}^{(t)})$ and $\mathbf{k}_i^{(t)} = \mathrm{Emb}(s_{k,i}^{(t)})$, normalize them to unit length, and measure provider j's relevance to consumer i:

$$r_{i,j}^{(t)} \;=\; (\hat{\mathbf{q}}_i^{(t)})^{\top} \hat{\mathbf{k}}_j^{(t)} \;\in\; [-1, 1]. \tag{4}$$

Structured descriptors express needs and offers in a common format. For example, an implementation request can match a Developer's description of available code. Cosine similarity is a routing signal, not a test of logical entailment or correctness. Descriptor-template and encoder ablations assess its sensitivity (Appendix A.3.1).

3.2.2 SPARSE DIRECTED GRAPH CONSTRUCTION

The fixed threshold τ_{edge} selects edges:

$$A^{(t)}_{j\to i} \;=\; \mathbb{I}\Big(r^{(t)}_{i,j} > \tau_{\text{edge}}\Big) \cdot (1-\delta_{ij}). \tag{5}$$

The factor $(1-\delta_{ij})$ excludes self-loops. The edge set is $\mathcal{E}^{(t)} = \{(a_j \to a_i) \mid A^{(t)}_{j\to i} = 1\}$, with incoming neighbors $\mathcal{N}^{(t)}_{in}(i) = \{j \mid A^{(t)}_{j\to i} = 1\}$. An edge selects a_j as a provider for a_i; the reverse edge is a separate decision. The threshold remains fixed within a trajectory, while the number and identity of active edges change with the descriptors.

3.3 RECURSIVE STATE UPDATE

3.3.1 ROUTING AND MEMORY UPDATE

The constructed graph routes private messages into the recipients' next-round memories:

$$\mathcal{H}^{(t+1)}_i = \mathcal{H}^{(t)}_i \oplus m^{(t)}_{pub,i} \oplus \Sigma_{\sigma^{(t)}_i}\Big(\{m^{(t)}_{priv,j} \mid j \in \mathcal{N}^{(t)}_{in}(i)\}\Big). \tag{6}$$

Here $\oplus$ denotes concatenation. All workers generate from their pre-round states before these memories are updated. This synchronization avoids a circular dependency between message generation and the graph induced from the messages' descriptors.

Equations 2 and 6 close the RSI loop in Equation 1. Received messages can change a worker's next output and descriptors, which in turn determine its next communication links. For example, after implementing an algorithm, a Developer may request a test report instead of algorithmic guidance. The matching rule stays fixed as the requested feedback changes. The Manager's updated goal coordinates these local revisions.

3.3.2 DETERMINISTIC MESSAGE SERIALIZATION

For each recipient, $\sigma^{(t)}_i$ orders incoming messages by decreasing relevance $r^{(t)}_{i,j}$, with ties broken by agent index. A separate global order $\sigma^{(t)}$ serializes public outputs for the Manager. It uses topological sorting for acyclic graphs and a deterministic minimum-in-degree heuristic for cyclic graphs. These orders specify prompt layout after generation; they do not schedule workers to consume same-round messages. Appendix E.2 gives the full definitions. Deterministic ordering makes serialization reproducible for a given trace, without making stochastic LLM generations deterministic.

4 EXPERIMENTS

4.1 DATASETS AND EVALUATION SCOPE

We evaluate code generation and mathematical reasoning (Table 10). HumanEval tests Python function correctness (Chen et al., 2021); APPS-Competition contains 100 selected competition-level programming problems (Hendrycks et al., 2021). The mathematical experiments use 70-problem subsets of MATH-500 (Lightman et al., 2023) and Omni-MATH (Gao et al., 2024a). The Omni-MATH sample uses seven domains with 10 problems per domain. The tasks require implementation, multi-step deduction, and verification. Results concern these subsets and the stated decoding protocol.

4.2 MODELS, BASELINES, AND SETTINGS

We instantiate DyTopo with MiMo-V2-Flash (Xiao et al., 2026), GPT-oss-120B (Agarwal et al., 2025), Llama-3-8B-Instruct (Grattafiori et al., 2024), and Qwen3-8B (Yang et al., 2025). Models

Table 1: Performance (%) across four benchmarks and four LLM backbones. Parentheses report relative improvement over Vanilla LLM; **bold** and underlining mark the best and second-best scores within each dataset column.

Method	MiMo-V2-Flash	GPT-oss-120B	Llama3-8B	Qwen3-8B	Average
HumanEval — Code Generation, Fundamental					
Vanilla LLM	86.59	95.73	23.78	18.29	56.10
Single-turn Agent	88.41 (+2.1%)	93.29 (−2.5%)	47.56 (+100.0%)	46.95 (+156.7%)	69.05 (+23.1%)
Random Topology	88.17 (+1.8%)	91.46 (−4.5%)	42.07 (+77.0%)	80.49 (+340.1%)	75.55 (+34.7%)
AgentScope	90.24 (+4.2%)	92.68 (−3.2%)	42.94 (+80.6%)	80.00 (+337.4%)	76.47 (+36.3%)
MasRouter	89.63 (+3.5%)	92.68 (−3.2%)	44.17 (+85.7%)	80.98 (+342.8%)	76.87 (+37.0%)
AgentRouter	89.02 (+2.8%)	92.07 (−3.8%)	43.56 (+83.2%)	79.45 (+334.4%)	76.03 (+35.5%)
DyTopo	**92.07** (+6.3%)	**98.16** (+2.5%)	**50.61** (+112.9%)	**89.63** (+390.0%)	**82.62** (+47.3%)
APPS-Competition — Code Generation, Advanced					
Vanilla LLM	41.18	31.88	10.13	9.62	23.20
Single-turn Agent	42.84 (+4.0%)	60.55 (+90.0%)	13.28 (+31.1%)	13.47 (+40.0%)	32.54 (+40.3%)
Random Topology	42.01 (+2.0%)	52.21 (+63.8%)	12.44 (+22.8%)	24.24 (+152.0%)	32.73 (+41.1%)
AgentScope	48.42 (+17.6%)	58.71 (+84.2%)	14.29 (+41.1%)	23.12 (+140.3%)	36.14 (+55.8%)
MasRouter	47.19 (+14.6%)	57.62 (+80.7%)	13.72 (+35.4%)	22.78 (+136.8%)	35.33 (+52.3%)
AgentRouter	46.92 (+13.9%)	60.97 (+91.2%)	12.44 (+22.8%)	23.04 (+139.5%)	35.84 (+54.5%)
DyTopo	**49.81** (+21.0%)	**69.66** (+118.5%)	**18.21** (+79.8%)	**25.14** (+161.3%)	**40.71** (+75.5%)
Math-500 — Mathematical Reasoning, Hard					
Vanilla LLM	57.14	60.00	12.86	48.57	44.64
Single-turn Agent	65.71 (+15.0%)	81.43 (+35.7%)	25.71 (+100.0%)	57.14 (+17.6%)	57.50 (+28.8%)
Random Topology	78.57 (+37.5%)	87.14 (+45.2%)	20.00 (+55.6%)	68.57 (+41.2%)	63.57 (+42.4%)
AgentScope	75.71 (+32.5%)	82.86 (+38.1%)	30.00 (+133.3%)	72.86 (+50.0%)	65.36 (+46.4%)
MasRouter	75.71 (+32.5%)	85.71 (+42.8%)	35.24 (+174.0%)	71.58 (+47.4%)	67.06 (+50.2%)
AgentRouter	77.14 (+35.0%)	84.29 (+40.5%)	24.36 (+89.4%)	70.35 (+44.8%)	64.04 (+43.5%)
DyTopo	**87.14** (+52.5%)	**88.57** (+47.6%)	**47.14** (+266.7%)	**75.71** (+55.9%)	**74.64** (+67.2%)
Omni-Math — Mathematical Reasoning, Expert					
Vanilla LLM	32.86	14.29	14.29	21.43	20.72
Single-turn Agent	37.14 (+13.0%)	25.71 (+80.0%)	11.43 (−20.0%)	32.86 (+53.3%)	26.79 (+29.3%)
Random Topology	41.43 (+26.1%)	34.29 (+140.0%)	15.71 (+10.0%)	25.71 (+20.0%)	29.29 (+41.4%)
AgentScope	44.28 (+34.8%)	37.14 (+160.0%)	22.86 (+60.0%)	35.71 (+66.6%)	35.00 (+68.9%)
MasRouter	50.00 (+52.2%)	35.71 (+149.9%)	21.96 (+53.7%)	38.72 (+80.7%)	36.60 (+76.6%)
AgentRouter	45.71 (+39.1%)	38.57 (+169.9%)	18.72 (+31.0%)	36.45 (+70.1%)	34.86 (+68.2%)
DyTopo	**52.86** (+60.9%)	**41.43** (+190.0%)	**30.00** (+110.0%)	**51.43** (+140.0%)	**43.93** (+112.0%)

are accessed through vLLM (Kwon et al., 2023) or hosted endpoints. Semantic matching uses `all-MiniLM-L6-v2` or `gte-large-en-v1.5`. Appendix F.3 lists the roles and prompts.

The comparisons include direct LLM generation, a single-turn agent ensemble, Random Topology, AgentScope (Gao et al., 2024b), MasRouter, and AgentRouter. The latter two use the iterative adaptations described in Appendix B. Random Topology preserves the graph sparsity level while randomizing its connections. These comparisons evaluate complete systems; additional routing controls examine the effect of edge selection.

DyTopo uses Manager-controlled stopping within a maximum round budget. The round ablation disables stopping and runs exactly T rounds. The routing threshold is 0.3 for coding, selected on APPS-Competition and transferred to HumanEval, and 0.4 for mathematics, selected on Omni-MATH and transferred to MATH-500. APPS-Competition and Omni-MATH therefore serve as both selection and evaluation sets. Appendix A.3 reports the selection protocol and sensitivity analyses; Appendix C reports uncertainty under the original decoding protocol.

5 Results

We evaluate the final accuracy of DyTopo's recursive improvement process, then test how feedback selection and the round budget affect its results. The cost analysis compares systems at the same number of rounds before measuring the additional savings from early stopping.

5.1 END-TO-END REASONING PERFORMANCE

DyTopo achieves the highest score in all 16 backbone×dataset settings in Table 1. The mean gain over the strongest agent baseline in each setting is +5.2 points, with individual gains of 0.90–12.71 points. Here the comparison set comprises Single-turn Agent, Random Topology, AgentScope, MasRouter, and AgentRouter. On Math-500, the largest gain is +11.90 points (Llama3-8B: 47.14 vs. 35.24); on Omni-Math it is +12.71 points (Qwen3-8B: 51.43 vs. 38.72). Coding gains include +8.69 points on APPS-Competition with GPT-oss-120B (69.66 vs. 60.97) and +8.65 points on HumanEval with Qwen3-8B (89.63 vs. 80.98). Gains vary by backbone and task in both domains.

Paired-bootstrap intervals exclude zero on each dataset (Table 7), with a macro-average difference of $+6.31$ points (95% CI: $[+4.22, +8.37]$). This comparison averages across four backbones before selecting the strongest baseline for each dataset. It differs from the $+5.2$-point summary, which selects the strongest agent baseline separately in each of the 16 settings. These intervals describe dataset-level and aggregate differences, not uncertainty for individual backbone settings. We use three seeds and paired outcomes, resampling APPS test cases by problem (Appendix C.2).

Random Topology scores above the Single-turn Agent in some settings and below it in others. We next change which agents exchange messages while keeping the number of links fixed, to test whether selecting semantically matched messages improves accuracy. Appendix C reports uncertainty and completion failures under the stated decoding protocol.

5.2 SEMANTIC FEEDBACK SELECTION

We test feedback selection by replacing high-similarity routes with low-similarity routes while preserving the active edge count. High-similarity routing yields 14.02–25.61 points higher accuracy across the four conditions in Table 2. Keeping the same number of links is therefore insufficient to retain accuracy. Feedback selection matters for recursive improvement.

Table 2: High- versus low-similarity routing at matched edge counts (accuracy, %). Parentheses give the change in percentage points.

Dataset	Qwen3-8B	Llama3-8B
HumanEval	$89.63 \rightarrow 64.02$ (-25.61)	$50.61 \rightarrow 36.59$ (-14.02)
Omni-MATH	$51.43 \rightarrow 27.14$ (-24.29)	$30.00 \rightarrow 14.29$ (-15.71)

A separate control holds adaptive halting fixed and replaces random routing with semantic routing. Across its four backbone–dataset conditions, the mean accuracy gain is +12.5 points; changing fixed to adaptive halting under random routing yields a mean change of +0.45 points (Appendix A.1). Semantic routing improves accuracy in all four conditions (+8.57 to +22.78 points), whereas the halting change is positive in only one. The small mean halting effect therefore conceals variation across tasks and backbones. These controls support feedback selection as a source of accuracy gains; they do not imply that stopping has the same effect under semantic routing.

Descriptors and encoder choice. With descriptors held fixed, MiniLM and GTE-large produce edge sets with Jaccard overlap of 85.82–95.41% across the reported rounds (Appendix A.2). The two encoders select largely overlapping routes for the same inputs. Table 3 tests how encoder and descriptor choices affect final accuracy.

Removing structured examples reduces accuracy by 3.65–5.72 points with MiniLM and 1.82–4.29 with GTE-large (Table 3). Structured descriptors improve accuracy under both encoders. The weakest free-form configuration still exceeds MasRouter by 5.00 points on HumanEval and 6.99 on Omni-MATH.

Table 3: Descriptor ablation on Qwen3-8B. Δ compares structured few-shot with free-form descriptors.

Encoder	Structured	Free-form	Δ
HumanEval (MasRouter: 80.98)			
MiniLM	89.63	85.98	+3.65
GTE-large	89.02	87.20	+1.82
Omni-MATH (MasRouter: 38.72)			
MiniLM	51.43	45.71	+5.72
GTE-large	52.86	48.57	+4.29

Encoder rankings also depend on descriptor format. On HumanEval, GTE-large exceeds MiniLM by 1.22 points with free-form descriptors, but MiniLM leads by 0.61

points with structured descriptors. On Omni-MATH, its advantage shrinks from 2.86 to 1.43 points. Thus, changing encoders provides less benefit when descriptors are structured.

5.3 Communication links and improvement rounds

We vary two settings: how many communication links are selected per round, and how many rounds the agents run. Threshold routing exceeds the best tested fixed top-k setting by 6.09 points on HumanEval and 4.29 on Omni-MATH (Appendix A.2). The threshold sweeps peak at $\tau_{\text{edge}} = 0.3$ on APPS and HumanEval, and at 0.4 on Omni-MATH (Table 6). Allowing the number of links to vary outperforms the tested fixed edge budgets.

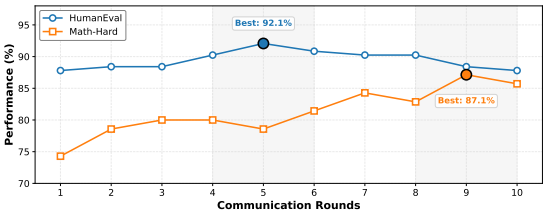


Figure 3: Accuracy versus communication rounds. HumanEval peaks at 5 rounds (92.07%); Math-500 peaks at 9 (87.14%).

Encoder choice matters more away from the best threshold. On HumanEval with Qwen3-8B, GTE-large and MiniLM peak at $\tau = 0.3$ with 89.02% and 89.63% accuracy. At $\tau = 0.6$, their scores diverge to 87.20% and 80.49%, respectively (Table 6).

To study the effect of additional rounds, we hold the workers and routing mechanism fixed, disable adaptive halting, and execute exactly the specified number of rounds. Figure 3 shows that final accuracy is non-monotonic on both HumanEval and Math-500.

HumanEval peaks at 5 rounds (92.07%), while Math-500 peaks at 9 rounds (87.14%). Additional feedback may supply a missing check or lead an agent to revise a correct answer. The recursive update therefore does not yield monotonic gains across rounds. Appendix A.4 describes other failures.

The two peak locations also show why a single fixed round budget can stop too early for one task and continue beyond the best observed point for another. These are aggregate accuracy curves: they do not reveal whether the same problems remain correct across rounds. Identifying harmful revisions requires paired, per-problem trajectories.

5.4 Cost of recursive improvement

Table 4 compares token usage and end-to-end latency on HumanEval with `mimo-v2-flash`. Random Topology, AgentScope, MasRouter, and AgentRouter execute 5 rounds. We evaluate DyTopo both at the same 5-round horizon and with Manager-controlled stopping, which executes 2.6 rounds on average. Single-pass generation and a parallel ensemble provide reference costs.

Table 4: HumanEval efficiency with MiMo-V2-Flash. Config denotes agents $\times$ rounds; full DyTopo uses adaptive halting (2.6 rounds on average). Token ratios are relative to full DyTopo.

Method	Agentic Config.	Accuracy (%)	Total Token Consumption	Latency (s)	Token Ratio vs. Full DyTopo
LLM Output	1×1	86.59	635	1.5	0.07×
Single-turn Agent	4×1	88.41	2,835	6.7	0.30×
Random Topology	4×5	88.17	15,783	34.2	1.67×
AgentScope	4×5	90.24	19,520	39.8	2.06×
MasRouter	4×5	89.63	20,356	47.5	2.15×
AgentRouter	4×5	89.02	18,741	43.1	1.98×
DyTopo (halting disabled)	5×5	**92.07**	**12,774**	37.2	1.35×
DyTopo (full)	**5×2.6**	**92.07**	**9,453**	**22.3**	**1.00×**

At the matched horizon, DyTopo retains 92.07% accuracy while using fewer tokens than every multi-round comparator. For example, it consumes 12,774 tokens versus AgentScope's 19,520, a 35% reduction. Latency falls from 39.8 to 37.2 seconds, a smaller change because all 5 generation

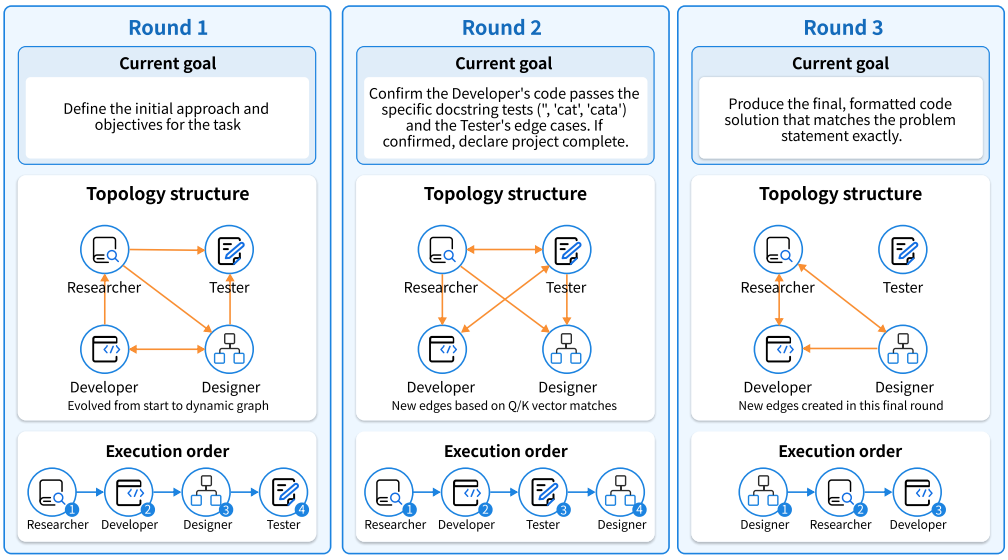


Figure 4: Topology evolution in a coding task. Goals and graphs change from exploration to verification and finalization. "Execution order" denotes message serialization: all workers generate once per round before routing and use incoming feedback in the next round.

rounds still execute. The comparison uses 5 agents for DyTopo and 4 for the baselines. DyTopo uses fewer tokens despite the extra Manager call, though coordination also differs between systems.

Enabling halting reduces DyTopo's token use to 9,453 and latency to 22.3 seconds at the same 92.07% accuracy. Full DyTopo uses 48% of AgentScope's tokens and 46% of MasRouter's, with latency reductions of 44% and 53%, respectively. Within DyTopo, stopping saves a further 26.0% of tokens and 40.1% of latency relative to five-round execution. These reductions differ from the 48.0% reduction in average rounds: round count alone does not capture total cost, since prompt lengths and generation lengths can vary. Equal aggregate accuracy does not establish identical solved problems. Appendix B gives the iterative baseline configurations and detailed comparisons.

5.5 Inspecting topology evolution

Figure 4 shows how selected routes change within one HumanEval trajectory: algorithm discussion gives way to implementation checking and finalization. In the verification round, the Tester's request for an implementation matches the Developer's offered code (similarity 0.77). The Developer's request for a test report matches the Tester's offered checks (0.49; Table 9). These reciprocal routes pass code to the Tester and return checks to the Developer. They illustrate how an intermediate implementation creates the need for a new feedback exchange.

Appendix D provides the full trace. Aggregate edge counts (Appendix D.3) also reflect trajectory termination and do not isolate graph sparsity.

6 Conclusion

DyTopo implements within-task RSI through communication topology evolution. Agents' current outputs select feedback for subsequent reasoning, which in turn determines later graphs. This couples solution improvement with adaptation of the information available to each agent, allowing feedback requests to change as reasoning progresses. With fixed models and routing rules, DyTopo improves final accuracy on the evaluated tasks and reduces HumanEval token use and latency. Semantic feedback selection contributes to accuracy, though additional rounds can reduce it. Isolating the benefit of repeated graph reconstruction requires further controls with frozen topologies.

REPRODUCIBILITY STATEMENT

Appendix E.1 specifies the round lifecycle, Appendix E.2 defines message serialization, Appendix E.3 analyzes complexity, and Appendix F lists role prompts and configurations. Appendices A and C describe the routing controls and reported resampling protocol.

AI USE STATEMENT

For this manuscript revision, an LLM assistant supported conceptual framing of the existing method, interpretation of reported results, literature lookup, and manuscript editing. Appendix H describes its role and the checks performed for this revision.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL `https://arxiv.org/abs/2107.03374`.

Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, Xiaoyin Che, Ye Tian, et al. Multi-agent collaboration via evolving orchestration. *arXiv preprint arXiv:2505.19591*, 2025.

Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on machine learning*, pp. 1538–1546. PMLR, 2019.

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

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-math: A universal olympiad level mathematic benchmark for large language models, 2024a. URL `https://arxiv.org/abs/2410.07985`.

Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, et al. Agentscope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024b.

Claudia V Goldman and Shlomo Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 137–144, 2003.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021.

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.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

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

Zae Myung Kim, Young-Jun Lee, Seungyeon Jwa, and Dongyeop Kang. Meta^n: Recursive Self-Improvement through Emergent Depth. arXiv:2608.24735, 2026. URL https://arxiv.org/abs/2608.24735.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

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

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. *arXiv preprint arXiv:2601.12996*, 2026.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, 2024.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Wei Liu, Chenxi Wang, Yifei Wang, Zihao Xie, Rennai Qiu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, and Chen Qian. Autonomous agents for collaborative task under information asymmetry. *Advances in Neural Information Processing Systems*, 37:2734–2765, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. In *ICLR 2025 Workshop on Scaling Self-Improving Foundation Models*, 2025. URL https://openreview.net/forum?id=rShJCyLsOr.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. Recursive self-improvement of AI research agents. arXiv:2609.26457, 2026. URL https://arxiv.org/abs/2609.26457.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. Llm agents making agent tools. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26092–26130, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Geunbin Yu. Adaptorch: Task-adaptive multi-agent orchestration in the era of llm performance convergence. *arXiv preprint arXiv:2602.16873*, 2026.

Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pan, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, et al. A survey on trustworthy llm agents: Threats and countermeasures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6216–6226, 2025.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15549–15572, 2025.

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. *arXiv preprint arXiv:2410.02506*, 2024a.

Guibin Zhang, Yanwei Yue, et al. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024b.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=pUpzQZTvGY.

Wentao Zhang, Liang Zeng, Yuzhen Xiao, Yongcong Li, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. Agentorchestra: Orchestrating multi-agent intelligence with the tool-environment-agent (tea) protocol. *arXiv preprint arXiv:2506.12508*, 2025a.

Zheyuan Zhang, Kaiwen Shi, Zhengqing Yuan, Zehong Wang, Tianyi Ma, Keerthiram Murugesan, Vincent Galassi, Chuxu Zhang, and Yanfang Ye. Agentrouter: A knowledge-graph-guided llm router for collaborative multi-agent question answering. *arXiv preprint arXiv:2510.05445*, 2025b.

Sibo Zhu, Shicheng Fan, Xinyue Wang, Wenyi Wu, Kun Zhou, and Biwei Huang. RSIAgent: Autonomous Exploration for Recursive Self-improvement in New Environments. arXiv:2609.15364, 2026. URL https://arxiv.org/abs/2609.15364.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*, 2024.

A Additional Feedback Controls

DyTopo changes both the information exchanged during improvement and the number of improvement rounds. Semantic routing selects the private messages entering each agent's context; Manager-controlled halting determines when interaction ends. We vary these choices to test how feedback selection and stopping affect final accuracy.

A.1 Routing and stopping conditions

We compare random routing with fixed five-round execution, random routing with adaptive halting, and semantic routing with adaptive halting. Results are ordered as HumanEval with Qwen3-8B, HumanEval with Llama3-8B, MATH-500 with Qwen3-8B, and MATH-500 with Llama3-8B.

With random routing, adaptive halting changes accuracy by -0.49, -0.64, -1.43, and $+4.36$ points across these four conditions, for a mean change of $+0.45$ points. With adaptive halting fixed, semantic routing improves on random routing by $+9.63$, $+9.18$, $+8.57$, and $+22.78$ points, respectively (mean $+12.5$). Feedback selection has the larger effect in these comparisons. The two deltas can be summed along this sequence of changes, but they do not identify independent routing and halting effects.

The direction of the changes is informative beyond their means: semantic routing improves all four conditions, while adaptive halting under random routing reduces accuracy in three and improves it in one. However, this three-condition comparison omits semantic routing with fixed five-round execution under the same protocol. It therefore cannot estimate the routing–halting interaction. The fixed-round DyTopo result in Table 4 uses a different backbone and cannot supply that missing condition.

A.2 Functional Validation of the Induced Edges

The following interventions examine the communication graph at two levels. Edge inversion changes the selected agent pairs at a fixed edge count. Fixed top-k selection constrains the edge count while retaining similarity-based ranking. These controls test feedback source selection and link count.

A.2.1 High-similarity versus low-similarity routing

We invert the semantic selection rule while preserving the number of activated edges. The default condition retains high-similarity query–key pairs, and the inverted condition selects low-similarity pairs. This tests the choice of feedback source at a fixed edge count.

Table 2 shows that selecting low-similarity edges reduces accuracy by 14.02–25.61 points. On HumanEval with Qwen3-8B, low-similarity routing obtains 64.02%, compared with 80.49% for Random Topology. It also falls below Random Topology on HumanEval with Llama3-8B (36.59% versus 42.07%) and on Omni-MATH with Llama3-8B (14.29% versus 15.71%). On Omni-MATH with Qwen3-8B, the inverted condition is slightly above Random Topology (27.14% versus 25.71%) but remains 24.29 points below semantic routing. Thus, routing quality depends on which agents exchange messages, even when the number of active links is unchanged.

A.2.2 Threshold routing versus fixed top-k routing

Threshold routing allows the number of activated edges to change across rounds. To test whether this variable edge budget is useful, we replace the threshold rule with a fixed top-k rule.

Threshold routing exceeds the best fixed top-k result by 6.09 points on HumanEval and 4.29 points on Omni-MATH (Table 5). The fixed rule preserves a specified number of links even when the strength of the descriptor matches changes. Thresholding instead retains the links whose scores exceed the same relevance criterion in each round. The results favor adapting link counts to descriptor scores.

Table 5: Accuracy (%) under threshold and fixed top-k edge selection with Qwen3-8B.

Routing rule	HumanEval	Omni-MATH
Threshold routing	**89.63**	**51.43**
Top-3	80.49	42.86
Top-6	83.54	47.14
Top-9	82.32	45.71
Top-12	79.88	40.00

Table 6: Accuracy (%) across similarity thresholds, backbones, and sentence encoders. Bold indicates the best result in each row. Descriptor-template comparisons are in Table 3.

Backbone	Encoder	Dataset	Similarity threshold τ_{edge}								
			.1	.2	.3	.4	.5	.6	.7	.8	.9
MiMo-V2-Flash	MiniLM	APPS	43.61	44.84	**49.81**	47.35	44.13	40.86	40.47	37.59	34.51
	MiniLM	Omni-MATH	42.86	47.14	50.00	**52.86**	48.57	42.86	41.43	40.00	38.58
Qwen3-8B	MiniLM	HumanEval	79.88	82.32	**89.63**	87.20	83.54	80.49	78.05	75.61	73.17
	GTE-large	HumanEval	81.71	82.93	**89.02**	88.41	87.80	87.20	82.93	78.05	76.22
	MiniLM	Omni-MATH	35.71	47.14	45.71	**51.43**	42.86	40.00	37.14	35.71	31.43
	GTE-large	Omni-MATH	38.57	42.86	45.71	**52.86**	47.14	41.42	38.57	34.29	32.86

A.2.3 Cross-encoder agreement of the induced graphs

We further test whether graph construction depends strongly on the choice of sentence encoder. We apply `all-MiniLM-L6-v2` and `gte-large-en-v1.5` to identical query and key descriptors and compute the Jaccard overlap between their induced edge sets.

The induced edge sets have Jaccard overlap of 85.82–95.41%, with the highest overlap in the later rounds. Most selected routes are therefore shared by these two encoders for the same descriptors. The remaining differences motivate the accompanying accuracy and threshold sensitivity comparisons.

A.3 Threshold, Encoder, and Descriptor Sensitivity

A.3.1 Threshold and descriptor results

We examine how the routing threshold, encoder, and descriptor format affect accuracy. Table 6 reports the threshold sweeps; Table 3 reports the descriptor-template ablation. The following sections describe threshold selection and interpret both comparisons.

A.3.2 Threshold-selection protocol

The similarity threshold is selected at the task-type level. Code benchmarks use $\tau_{\text{edge}} = 0.3$, selected on APPS-Competition, whereas mathematical-reasoning benchmarks use $\tau_{\text{edge}} = 0.4$, selected on Omni-MATH. The selected code threshold is transferred unchanged to HumanEval, and the selected mathematics threshold is transferred unchanged to MATH-500. HumanEval and MATH-500 therefore do not participate in threshold selection.

APPS-Competition and Omni-MATH serve as both selection and evaluation sets. Results on these two datasets should consequently be interpreted with this overlap in mind. Under a strictly global $\tau_{\text{edge}} = 0.3$, the reported Omni-MATH result with MiMo-V2-Flash changes from 52.86% to 50.00%. HumanEval and APPS-Competition are unchanged because they already use 0.3. A corresponding strictly-global result for MATH-500 is not reported because that setting has not been measured.

A.3.3 Encoder and threshold sweep

We repeat the threshold sweep with MiniLM and GTE-large using Qwen3-8B. Encoder effects in Table 6 should be compared within the same backbone and dataset. Differences between its MiMo-V2-Flash and Qwen3-8B rows also include a change of backbone.

Both encoders attain their highest accuracy at the same thresholds: 0.3 on HumanEval and 0.4 on Omni-MATH. Changing the encoder alters peak accuracy by -0.61 points on HumanEval and $+1.43$ points on Omni-MATH. GTE-large has a flatter HumanEval curve; at $\tau = 0.6$, it retains 87.20% accuracy, compared with 80.49% for MiniLM. Thus, these encoders differ in sensitivity away from the best threshold even when the location of the observed peak is unchanged.

A.3.4 DESCRIPTOR-TEMPLATE ABLATION

The default system uses structured few-shot examples to separate a descriptor expressing an information need from one expressing an offered capability. We remove these examples and allow agents to produce free-form query and key descriptors.

Table 3 shows that removing the template reduces accuracy by 3.65 and 5.72 points with MiniLM, and by 1.82 and 4.29 points with GTE-large, on HumanEval and Omni-MATH respectively. The structured format yields more useful descriptors for routing in these comparisons. Even the weakest free-form configuration exceeds the strongest baseline under the same protocol, by 5.00 points on HumanEval and 6.99 points on Omni-MATH. GTE-large reduces the observed loss from free-form descriptors, but does not eliminate it.

The descriptor benefit is 1.83 points larger with MiniLM than with GTE-large on HumanEval, and 1.43 points larger on Omni-MATH. These differences of differences describe the reported scores; without uncertainty estimates for these contrasts, they do not establish a statistically reliable interaction. One plausible explanation is that a consistent need–offer format makes useful matches easier for either encoder, reducing the advantage of the encoder that performs better on free-form text. The ablation changes few-shot examples and output format together, so it cannot attribute the gain to either component separately.

A.4 OBSERVED FAILURE REGIMES

The observed gains vary across tasks and backbones. Accuracy also falls when routing thresholds are too high or structured descriptor examples are removed.

Variation in accuracy gains. On HumanEval with MiMo-V2-Flash, DyTopo improves over Random Topology by 3.90 points; on MATH-500 with Llama3-8B, the difference is 27.14 points. Because the task and backbone both differ, these results do not isolate the contribution of either factor.

Over-sparse communication. Increasing the threshold beyond its task-appropriate range can exclude useful provider–consumer links. On HumanEval with Qwen3-8B and MiniLM, accuracy decreases from 89.63% at $\tau = 0.3$ to 73.17% at $\tau = 0.9$. The same trend appears on Omni-MATH. At high thresholds, agents may not receive the information required for verification or correction.

Ambiguous descriptors. Removing structured descriptor examples reduces accuracy by up to 5.72 points. GTE-large incurs a smaller loss than MiniLM, but both encoders perform better with the structured format. Descriptor format therefore matters for routing with either encoder.

B EFFICIENCY PROTOCOL AND EXTENDED COMPARISON

The efficiency comparison in §5.4 separates two costs of recursive improvement: exchanging feedback within a round and continuing to later rounds.

We adapt AgentRouter and MasRouter to iterative reasoning under fixed round and per-round token budgets. AgentRouter first builds a knowledge graph and runs RouterGNN to obtain an answer and a short summary. Later rounds add the previous answer and summary as entity nodes and rerun the frozen router. MasRouter begins with its three-level selection of mode, roles, and LLMs. Later rounds feed the previous answer and accumulated cost into the latent model to update component selection. These are the iterative baseline implementations used in our comparison.

Setup. We use `mimo-v2-flash` on HumanEval with the following baselines:

- **LLM Output:** a single-agent, single-pass generation (1 agent, 1 round).
- **Single-turn Agent:** Parallel workers without inter-agent messages (4 agents, 1 round).

Table 7: Paired bootstrap comparison between DyTopo and the strongest baseline for each dataset, averaged over four backbones. Results correspond to the original decoding protocol.

Dataset	DyTopo	Strongest baseline	Paired difference	95% CI	p
HumanEval	82.62	76.87 (MasRouter)	$+5.75$	$[+2.34, +9.12]$	0.003
APPS-Competition	40.71	36.14 (AgentScope)	$+4.57$	$[+1.18, +7.93]$	0.008
MATH-500	74.64	67.06 (MasRouter)	$+7.58$	$[+3.09, +12.04]$	0.001
Omni-MATH	43.93	36.60 (MasRouter)	$+7.33$	$[+2.61, +12.08]$	0.002
Macro average	**60.48**	**54.17**	$\mathbf{+6.31}$	$[\mathbf{+4.22}, \mathbf{+8.37}]$	$< \mathbf{0.01}$

- **Random Topology:** 4 agents with random per-round connections, run for 5 rounds.
- **AgentScope, MasRouter, AgentRouter:** multi-round multi-agent frameworks run with the 4-agent, 5-round configurations used in this comparison.

DyTopo uses 5 agents, comprising four workers and a Manager. Manager-controlled halting (§3.1.2) terminates execution after 2.6 rounds on average.

B.1 Matched-round efficiency comparison

We first disable Manager-controlled early stopping and force DyTopo to run five rounds, matching the interaction budget of the multi-round baselines. All measurements use HumanEval with the MiMo-V2-Flash backbone under identical serving conditions. The baseline systems use four agents, whereas DyTopo includes four workers and one additional Manager. The additional Manager therefore increases, rather than reduces, DyTopo's number of model calls.

At the matched five-round budget, DyTopo retains 92.07% accuracy while using 35% fewer tokens than AgentScope, 37% fewer than MasRouter, 32% fewer than AgentRouter, and 19% fewer than Random Topology. Early stopping is disabled, so fewer executed rounds cannot explain these differences. The results are consistent with shorter contexts under sparse routing, although differences in coordination implementations remain.

At five rounds, DyTopo is 6.5% faster than AgentScope despite using 35% fewer tokens. Reducing context length does not remove an entire generation round. Halting reduces latency from 37.2 to 22.3 seconds at the same accuracy. The larger latency reduction thus comes from executing fewer rounds.

Most of the token savings remain at five rounds, while adaptive stopping accounts for the larger latency reduction. Execution length is fixed, though coordination differs.

C Statistical uncertainty under the original decoding protocol

C.1 Decoding and completion behavior

The low Vanilla LLM scores for Llama3-8B and Qwen3-8B on HumanEval reflect completion failures under the fixed context and decoding configuration (Appendix F). Outputs that do not terminate within the token budget are counted as incorrect. The scores therefore measure both answer quality and completion under this protocol. The agent configurations reduce the observed failure rate; Table 8 reports invalid or non-terminating outputs alongside run-to-run variation.

C.2 Multi-seed uncertainty analysis

We estimate run-to-run variation using three seeds. For APPS-Competition, treating all 3,820 test cases as independent would underestimate uncertainty because the cases are nested within 100 programming problems. We therefore use a problem-cluster bootstrap. For all datasets, the DyTopo and baseline outcomes are resampled as paired observations so that the confidence interval is computed directly for their difference.

Table 7 selects the strongest baseline after averaging over backbones within each dataset. Its macro-average gain of 6.31 points differs from the main-text 5.2-point summary, which selects the strongest

Table 8: Run-to-run standard deviation and the percentage of invalid or non-terminating outputs under the original decoding protocol.

Dataset	Method	SD (%)	Invalid/non-term. (%)
HumanEval	MasRouter	2.9	3.1
	DyTopo	**2.2**	**1.2**
APPS	AgentScope	4.2	6.8
	DyTopo	**3.4**	**4.5**
MATH-500	MasRouter	4.8	7.5
	DyTopo	**3.9**	**4.9**
Omni-MATH	MasRouter	5.4	9.2
	DyTopo	**4.5**	**6.1**

agent baseline separately in each backbone–dataset setting. All paired confidence intervals exclude zero under the reported resampling protocol. APPS-Competition has the smallest lower bound, $+1.18$ points. Problem-level, seed-level, and nested resampling yield the same direction of effect; the smallest observed lower bound is $+0.96$ points for APPS-Competition under nested resampling.

Table 8 shows that DyTopo has lower run-to-run standard deviation and a lower invalid or non-terminating output rate than the strongest baseline on all four datasets. The 70-problem MATH-500 and Omni-MATH subsets nevertheless produce relatively wide uncertainty intervals, so their point estimates should be interpreted together with the paired confidence intervals.

D Communication Traces

D.1 Examples of Semantic Key-Query Matching

Table 9: Real-world examples of Key (s_k) and Query (s_q) descriptors generated during the `is_palindrome` task and the resulting edge formation.

Round	Query Descriptor (s_q)	Key Descriptor (s_k)	Outcome
1	**Agent A (Researcher):** I need the function signature for make_palindrome implementation details.	**Agent B (Developer):** I provide the implementation using KMP failure function on the concatenated string...	**Edge Created** ($B \to A$) Score: 0.52
2	**Agent C (Tester):** I need the Developer's implementation of make_palindrome to verify against these test cases.	**Agent B (Developer):** I provide the complete Python implementation of make_palindrome and is_palindrome.	**Edge Created** ($B \to C$) **Score: 0.77 (High)**
2	**Agent B (Developer):** I need the Tester's report on the docstring tests and edge cases.	**Agent C (Tester):** I provide comprehensive test suite covering empty, single char, palindromes...	**Edge Created** ($C \to B$) Score: 0.49

Table 9 lists descriptor pairs extracted from the experiment logs and their matching scores. Each edge records a selected information route under the threshold rule; the similarity score is not a calibrated probability that the message will improve the answer.

D.2 Topology Evolution Trace: A Case Study

We examine a HumanEval trace involving `is_palindrome` and `make_palindrome`. Its 4 worker agents are Researcher, Developer, Tester, and Designer. The trace follows their changing goals and communication links during solution improvement.

Round 1: Initial Exploration & Algorithm Selection.

- **Goal:** Define initial approach.
- **Dynamics:** The Researcher (A) proposed KMP/Manacher algorithms. The Developer (B) drafted an initial implementation.
- **Topology:** Figure 4 includes a **Researcher** $\rightarrow$ **Developer** route. Table 9 records the reverse **Developer** $\rightarrow$ **Researcher** route with score 0.52.
- **Reasoning:** The Developer's query ("I need an efficient algorithmic approach and complexity guidance") matched the Researcher's key ("I provide candidate algorithms and complexity considerations"), routing algorithmic guidance to implementation.

Round 2: Implementation & Verification.

- **Goal:** Verify code against specific test cases.
- **Dynamics:** The focus shifted to correctness. The Tester (C) generated test cases.
- **Topology:** The graph included **Developer** $\rightarrow$ **Tester** (Score: 0.77).
- **Reasoning:** The Tester requested the Developer's implementation for verification, and the Developer offered the implementation. Their descriptors selected the code-delivery route.

Round 3: Finalization & Termination.

- **Goal:** Final formatting and docstring compliance.
- **Dynamics:** Successful verification prompted the Manager to request the formatted output.
- **Topology:** The graph became sparse. The Manager aggregated the final outputs, and the system terminated with `is_complete=True`.

The trace illustrates structural RSI within one task: intermediate work changes the feedback that agents request, and those requests determine later communication links. The trace illustrates the update rule; Appendix A.2 tests how semantic feedback selection affects final accuracy.

D.3 Aggregate edge counts on HumanEval

We analyze the communication graphs on HumanEval with the MIMO-v2-Flash model to examine how topology changes across a reasoning trajectory. The workers have four fixed roles: **Agent A (Researcher)**, **Agent B (Developer)**, **Agent C (Tester)**, and **Agent D (Designer)**. The role definitions remain unchanged while the communication edges evolve.

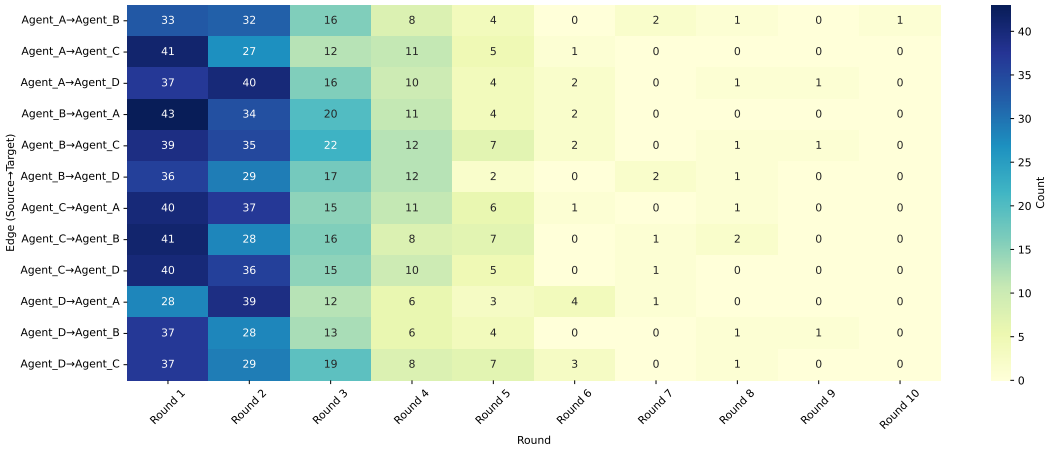


Figure 5: **Macro-topology heatmap.** Aggregate edge counts across 10 communication rounds on HumanEval with the MIMO-v2-Flash backbone. The displayed rounds 1–10 correspond to method indices $t = 0, \ldots, 9$. Counts decline across all agent pairs; the plot does not normalize by the number of trajectories still active in each round.

Figure 5 reports raw counts for each directed worker pair over 10 rounds. All pairs appear in the early rounds, including routes from the Researcher to the other workers and exchanges between the Developer and Tester. These aggregate counts describe how often links occur across trajectories; they do not imply that every task begins with a fully connected graph.

Counts decrease across all pairs in later rounds. The Developer–Tester routes follow this general decline rather than forming persistent late-round bands. Because the number of active trajectories is not shown, the plot cannot distinguish fewer links per active task from fewer tasks continuing to later rounds. Nor does it establish the content or utility of the routed messages. The case study in Appendix D provides message-level examples, while the edge-selection controls in Appendix A.2 test effects on final performance.

E Algorithm, Serialization, and Complexity

The recursive memory update changes the information available to a worker, rather than its parameters. A revised answer can also revise the worker's stated needs, redirecting later feedback even when the shared task is unchanged. Thus, answer revision and communication adaptation are coupled through memory. A fixed graph would retain answer revision but remove this dependence of future routes on current descriptors.

E.1 Algorithm for DyTopo

Algorithm 1 collects the generation, routing, memory, and Manager updates defined in Section 3.

E.2 Message ordering

Recipient-specific ordering. For each recipient a_i, $\sigma_i^{(t)}$ orders its incoming messages $\{m_{priv,j}^{(t)} \mid j \in \mathcal{N}_{in}^{(t)}(i)\}$ by *decreasing* relevance $r_{i,j}^{(t)}$, with ties broken by agent index. The most semantically aligned provider therefore appears first inside a_i's next-round prompt, and the entire ordering is fully determined by the matching scores already computed in §3.2.

Global aggregation order. For trace-level aggregation, including the manager's $S_{global}^{(t)}$ in Equation 3 and the public-channel record used for analysis, we additionally define a global order $\sigma^{(t)} = (\sigma_1^{(t)}, \ldots, \sigma_N^{(t)})$, a permutation of $\{1, \ldots, N\}$. For convenience, let

$$\mathrm{pos}_{\sigma^{(t)}}(i) \;=\; \min\{\ell \in \{1, \ldots, N\} \mid \sigma_\ell^{(t)} = i\} \tag{7}$$

denote the position of agent i in the sequence. Because $G^{(t)}$ is induced dynamically, it may be acyclic or contain directed cycles, so we construct $\sigma^{(t)}$ in two cases.

Case I: Directed acyclic graph. If $G^{(t)}$ contains no directed cycles, $\sigma^{(t)}$ is computed via standard topological sort and satisfies

$$\forall\,(a_j \to a_i) \in \mathcal{E}^{(t)} \;\Rightarrow\; \mathrm{pos}_{\sigma^{(t)}}(j) < \mathrm{pos}_{\sigma^{(t)}}(i), \tag{8}$$

so every selected provider precedes its consumer in the induced linearization. When multiple valid topological orders exist, we break ties by agent index for reproducibility.

Case II: Cyclic graph. When $G^{(t)}$ contains cycles, no topological order exists, and we use a greedy minimum-in-degree heuristic. Let $\mathcal{U} \subseteq \{1, \ldots, N\}$ denote the set of unplaced agent indices, and define the restricted in-degree

$$d_{in}^{(t)}(i; \mathcal{U}) \;=\; \big|\{j \in \mathcal{U} \mid (a_j \to a_i) \in \mathcal{E}^{(t)}\}\big|. \tag{9}$$

We iteratively select

$$i^* \;=\; \arg\min_{i \in \mathcal{U}} d_{in}^{(t)}(i; \mathcal{U}), \tag{10}$$

append i^* to $\sigma^{(t)}$, and remove it from $\mathcal{U}$ until all indices are placed. Nodes with smaller restricted in-degree are placed earlier, with ties broken by agent index. This is a deterministic serialization heuristic; it does not guarantee an optimal ordering or change the synchronous generation schedule. Both cases yield a permutation that orders $\Sigma_{\sigma^{(t)}}(\cdot)$ consistently for the Manager and trace.

Algorithm 1 DyTopo: same-round graph induction and recursive state update

Input: Agent set $\mathcal{A}$, initial task context $C_{task}^{(0)}$, max rounds $T_{\max}$
Initialize: $\mathcal{H}_i^{(0)} \leftarrow \emptyset$ for all $a_i \in \mathcal{A}$; $t \leftarrow 0$; $y \leftarrow 0$
while $t < T_{\max}$ **and** $y = 0$ **do**
  Phase 1: Single-Pass Agent Inference (Sec. 3)
  for all $a_i \in \mathcal{A}$ **do**
    $S_i^{(t)} \leftarrow [\rho_i;\, C_{task}^{(t)};\, \mathcal{H}_i^{(t)}]$
    $O_i^{(t)} = \langle m_{pub,i}^{(t)},\, m_{priv,i}^{(t)},\, s_{q,i}^{(t)},\, s_{k,i}^{(t)} \rangle \sim \pi_{\theta_i}(\cdot \mid S_i^{(t)})$
  end for
  Phase 2: Topology Induction (Sec. 3.2)
  for all $a_i \in \mathcal{A}$ **do**
    $\mathbf{q}_i^{(t)} \leftarrow \mathrm{Emb}(s_{q,i}^{(t)}), \quad \mathbf{k}_i^{(t)} \leftarrow \mathrm{Emb}(s_{k,i}^{(t)})$
    $\hat{\mathbf{q}}_i^{(t)} \leftarrow \mathbf{q}_i^{(t)} / \|\mathbf{q}_i^{(t)}\|_2, \quad \hat{\mathbf{k}}_i^{(t)} \leftarrow \mathbf{k}_i^{(t)} / \|\mathbf{k}_i^{(t)}\|_2$
  end for
  Compute relevance matrix: $R^{(t)} \in \mathbb{R}^{N \times N}$ where $R_{i,j}^{(t)} \leftarrow (\hat{\mathbf{q}}_i^{(t)})^\top \hat{\mathbf{k}}_j^{(t)}$
  Threshold to adjacency: $A_{j \to i}^{(t)} \leftarrow \mathbb{I}(R_{i,j}^{(t)} > \tau_{\text{edge}}) \cdot (1 - \delta_{ij})$ for all i, j
  $G^{(t)} \leftarrow (\mathcal{A}, \mathcal{E}^{(t)})$ where $\mathcal{E}^{(t)} = \{(a_j \to a_i) \mid A_{j \to i}^{(t)} = 1\}$
  Phase 3: Deterministic Message Ordering (Sec. 3.3.2)
  Construct global $\sigma^{(t)}$ and recipient orders $\{\sigma_i^{(t)}\}$ (Appendix E.2)
  Phase 4: Routing & Memory Update (Sec. 3.3.1)
  for all $a_i \in \mathcal{A}$ **do**
    $\mathcal{N}_{in}^{(t)}(i) \leftarrow \{j \mid (a_j \to a_i) \in \mathcal{E}^{(t)}\}$
    $\mathcal{H}_i^{(t+1)} \leftarrow \mathcal{H}_i^{(t)} \oplus m_{pub,i}^{(t)} \oplus \Sigma_{\sigma_i^{(t)}}(\{m_{priv,j}^{(t)} \mid j \in \mathcal{N}_{in}^{(t)}(i)\})$
  end for
  Phase 5: Manager Control (Sec. 3.1.2)
  $S_{global}^{(t)} \leftarrow [C_{task}^{(t)};\, \Sigma_{\sigma^{(t)}}(\{m_{pub,i}^{(t)} \mid a_i \in \mathcal{A}\})]$
  $\langle y,\, C_{task}^{(t+1)} \rangle \sim \Pi_{meta}(\cdot \mid S_{global}^{(t)})$
  $t \leftarrow t + 1$
end while
Output: final solution extracted from the last global state

E.3 Complexity Analysis

We compare DyTopo with a static, fully connected graph of N agents.

Communication cost. A fully connected system routes $O(N^2)$ messages per round. DyTopo also computes N^2 pairwise similarities, so graph construction remains $O(N^2)$ for a fixed embedding dimension. The routed message volume depends on the active edges: with sparsity factor ρ determined by threshold τ, $|\mathcal{E}^{(t)}| \approx \rho N^2$, where $\rho \ll 1$ in the sparse regime. Each agent receives private messages only from $\mathcal{N}_{in}^{(t)}$. This reduces the private-message context relative to full connectivity, subject to message length and graph density; it does not remove the quadratic matching step or guarantee a bounded total memory.

Inference overhead. Semantic matching requires descriptor encoding and pairwise dot products in addition to LLM generation. The `all-MiniLM-L6-v2` encoder has approximately 22M parameters. Its encoding operations take milliseconds in the reported setting, whereas generation by backbones such as MiMo-V2-Flash or GPT-oss-120B takes seconds. This suggests a small routing overhead relative to generation in that setting. End-to-end costs are measured in Table 4; the relative overhead may change with the encoder, team size, and serving configuration.

F Evaluation Implementation

F.1 Datasets

Table 10: Dataset domains and difficulty. Mathematics uses benchmark subsets.

Dataset	Domain	Difficulty	Dataset Description
HumanEval	Code	Fundamental	Evaluation of functional correctness in Python via docstrings.
APPS-Competition	Code	Advanced	Competitive programming problems requiring algorithmic design.
MATH-500	Math	Hard	A balanced subset (7 domains) of MATH, focusing on derivation.
Omni-MATH	Math	Expert	Focused on deep reasoning with Olympiad-level complexity.

F.2 Configurations

Embedding model. Semantic matching uses `all-MiniLM-L6-v2` or `gte-large-en-v1.5`.

Experimental configuration and evaluation environment. Table 11 lists agent settings, routing and generation parameters, and the serving environment. Unless specified otherwise, configurations use 4 to 6 agents and at most $T_{\max} = 10$ rounds. The selected encoder embeds each query and key descriptor. The routing threshold is $\tau_{\text{edge}} = 0.3$ for coding and $\tau_{\text{edge}} = 0.4$ for mathematics. Generation uses temperature $T_{\text{gen}} = 0.3$ and a response limit of 3,000–5,000 tokens, depending on the task and role. Outputs use JSON where structured parsing is required. The reported runtimes cover one complete evaluation pass per benchmark under the listed environment. They document evaluation cost and are not normalized for comparison across benchmarks.

Baselines.

- **Vanilla LLM (LLM Output):** The backbone LLM directly generates a final answer in a single pass, without role prompts, multi-agent collaboration, or multi-round improvement. This is a direct-generation reference for each backbone under the evaluation protocol.
- **Single-turn Agent:** Multiple role-specialized worker agents generate responses in parallel within a single round, but no inter-agent messages are exchanged. This provides a reference for role specialization and parallel generation without multi-round communication.
- **Random Topology:** Implemented by enforcing the same sparsity level as DyTopo but randomizing the edge connections at each round. This compares semantic and random edge selection at the same sparsity level.
- **Static topology:** A predefined graph is reused across rounds. A fully matched frozen-topology control remains to be evaluated (see the limitations).
- **AgentScope:** Our comparison uses a pipeline configured with a fixed sequential order and a central hub pattern. This is our evaluation configuration, not a restriction of AgentScope.
- **MasRouter:** A learned multi-agent routing baseline that performs task-level component selection before interaction, including the macro-collaboration mode, participating roles, and LLM components. In our multi-round comparison, we extend MasRouter to iterative reasoning: after each round, the previous answer and accumulated cost are fed back into its latent routing model to bias subsequent component selection, while the total number of rounds and per-round token budget remain fixed. MasRouter thus adapts mainly at the task level, without explicitly rewiring peer-to-peer message edges each round.
- **AgentRouter:** A knowledge-graph-guided routing baseline that constructs a query–entity knowledge graph and uses a graph neural network router to select collaborator agents and produce an answer. In the multi-round setting, we inject the previous answer and its short summary as additional entity nodes into the graph and re-run the frozen router in subsequent rounds. This baseline tests whether a frozen, graph-based task-level router can substitute for round-wise dynamic topology adaptation.

Table 11: Experimental configuration, compute environment, and approximate evaluation runtime. Runtime values denote the wall-clock duration of one complete evaluation pass for each benchmark.

Setting	Value	Description
General configuration		
Agent pool size (N)	4–6	Number of agents instantiated for each multi-agent configuration.
Maximum interaction rounds ($T_{\max}$)	10	Upper bound on the number of communication rounds.
Semantic routing configuration		
Threshold sweep range (τ_{edge})	0.1–0.9	Range evaluated in the similarity-threshold sensitivity analysis.
Task-type threshold (τ_{edge})	0.3 for code; 0.4 for math	Selected on APPS-Competition and Omni-MATH, respectively, and transferred unchanged to HumanEval and MATH-500.
Generation configuration		
Temperature (T_{gen})	0.3	Decoding temperature used for agent generation.
Maximum output length ($L_{\max}$)	3,000–5,000 tokens	Upper bound on the number of generated tokens per response, depending on the task and agent role.
Structured output	Enabled	Agent outputs are constrained to JSON when structured parsing is required.
Compute and serving environment		
CPU	2× Intel Xeon Gold 6242R	3.10 GHz, 20 cores per socket, 80 logical cores with hyper-threading, and a maximum frequency of 4.10 GHz.
Cache	32 KB L1d/L1i; 1 MB L2; 36 MB L3	Cache capacity reported for the local compute node.
System memory	503 GB	Approximately 377 GB was available during evaluation, with 4 GB of swap space.
GPU	8× NVIDIA GeForce RTX 3090	24 GB per GPU and 192 GB aggregate GPU memory.
CUDA	12.2 (V12.2.140)	CUDA version used by the local serving environment.
Serving paths	vLLM and OpenRouter	Models were served either locally through vLLM or through hosted OpenRouter endpoints.
Approximate wall-clock runtime per complete evaluation pass		
HumanEval	1.2 h	Code-generation evaluation.
APPS-Competition	1.2 h	Competition-level code-generation evaluation.
MATH-500	0.8 h	Mathematical-reasoning evaluation.
Omni-MATH	1.0 h	Olympiad-level mathematical-reasoning evaluation.

F.3 System Prompts and Templates

We provide the recorded system prompt templates for the Manager and worker agents in both domains. Their JSON fields expose public outputs, private messages, and query (s_q) and key (s_k) descriptors for parsing. These templates define the fixed roles used throughout the recursive improvement process.

F.3.1 Code Generation Agents

Code generation uses a Manager and four worker roles: Developer, Researcher, Tester, and Designer.

Manager Agent (Code Generation). The Manager updates the shared goal and decides when to stop based on the code and verification reports in the conversation.

Table 12: Agent Role Descriptions for Code Generation Tasks.

Role	Description
Developer	Implement complete, runnable code. If using classes, provide independent functions as entry points.
Researcher	Identify standard algorithms and time complexity. Output conclusions without derivation.
Tester	Provide critical test cases and expected results. Describe testing logic rather than full execution logs.
Designer	Design API interfaces and class structures. Only show method signatures and type hints.

Manager Agent Prompt (Code Gen)

```
You are Manager, the Workflow Orchestrator. Decide if the task is
COMPLETE.
STRICT RESPONSE FORMAT - MANDATORY
You MUST output ONLY a valid JSON object with these exact fields:
{
 "public_content": "String. Status summary and next directives.",
 "private_content": {}, // No successors for manager
 "q_desc": "String. REQUIRED. What you need next (query
descriptor).",
 "k_desc": "String. REQUIRED. What you provide (key
descriptor).",
 "is_complete": Boolean,
 "next_goal": "String"
}
CRITICAL RULES:
1. Primary focus: Workflow Orchestrator. Decide if the task is
COMPLETE.
2. Strict constraint: Only set is_complete=True if Code exists
AND Tests pass.
3. If you see Python code, analyze it from your role's
perspective.
```

Worker Agents (Code Generation). Workers share a prompt template with distinct role descriptions.

Generic Worker Agent Prompt Template

```
You are [Agent Name], the [Agent Role].
Role Description: [Insert Role Specific Description]
STRICT RESPONSE FORMAT - MANDATORY
You MUST output ONLY a valid JSON object with these exact fields:
{
 "public_content": "String. Your contribution to the task.",
 "private_content": {"TargetRole": "Instruction"},
 "q_vector": "String. REQUIRED. What you need next (Query).",
 "k_vector": "String. REQUIRED. What you provide (Key)."
}
```

Role Descriptions. Table 12 lists the responsibilities assigned to each worker in the template.

F.3.2 Mathematical Reasoning Agents

Mathematical reasoning uses ProblemParser, Solver, and Verifier workers with a separate Manager.

Role Definitions. Table 13 lists the mathematical agents' responsibilities and constraints.

System Prompts. The recorded math-agent prompt templates are given below.

Table 13: Agent Roles and Responsibilities for Mathematical Reasoning.

Role	Primary Responsibility	Core Constraints
ProblemParser	Decompose the problem statement.	Output must include analysis, known conditions, target, and a step-by-step plan.
Solver	Execute mathematical derivation.	Provide detailed steps, symbolic reasoning, and the final answer.
Verifier	Logic and calculation check.	Verify the rationality of each step; identify logical loopholes or calculation errors.
Manager	Workflow orchestration.	Halt only when a solution exists AND verification passes.

ProblemParser Agent Prompt

```
You are [Agent Name], the Math Problem Analyst. Analyze the
problem statement, identify known conditions, define the solving
target, and break down the solving steps.
STRICT RESPONSE FORMAT - MANDATORY
You MUST output ONLY a valid JSON object with these exact fields:
{
 "public_content": "String. Analysis, conditions, target, and
plan.",
 "private_content": {"Role": "Instruction"},
 "q_vector": "String. REQUIRED. What you need next.",
 "k_vector": "String. REQUIRED. What you provide."
}
CRITICAL RULES:
1. Primary focus: Analyze problem, identify conditions, define
target, break down steps.
2. Constraint: MAX 1000 words. Output MUST include clear
analysis and plan.
3. Analyze mathematical expressions from your role's perspective.
```

Solver Agent Prompt

```
You are [Agent Name], the Mathematical Solver. Execute specific
mathematical calculations, symbolic reasoning, or theorem calls.
Can use SymPy for symbolic computations.
STRICT RESPONSE FORMAT - MANDATORY
You MUST output ONLY a valid JSON object with these exact fields:
{
 "public_content": "String. Detailed solutions with
derivations.",
 "private_content": {"Role": "Instruction"},
 "answer": "String. The final answer.",
 "q_vector": "String. REQUIRED.",
 "k_vector": "String. REQUIRED."
}
CRITICAL RULES:
1. Primary focus: Calculations, symbolic reasoning, theorem
calls.
2. Constraint: MAX 2000 words. Detailed step-by-step solutions
required.
```

Verifier Agent Prompt

```
You are [Agent Name], the Logic Verifier. Check the rationality of
each derivation step, identify logical loopholes or calculation
errors.
```

```
STRICT RESPONSE FORMAT - MANDATORY
You MUST output ONLY a valid JSON object with these exact fields:
{
 "public_content": "String. Verification results and
conclusion.",
 "private_content": {"Role": "Instruction"},
 "q_vector": "String. REQUIRED.",
 "k_vector": "String. REQUIRED."
}
CRITICAL RULES:
1. Primary focus: Check rationality, find loopholes/errors.
2. Constraint: MAX 1000 words. Output verification for each
step.
```

Manager Agent Prompt (Math)

```
You are [Agent Name], the Workflow Orchestrator. Decide if the
task is COMPLETE.
STRICT RESPONSE FORMAT - MANDATORY
You MUST output ONLY a valid JSON object with these exact fields:
{
 "public_content": "String. Status summary.",
 "private_content": {"Role": "Instruction"},
 "q_vector": "String.", "k_vector": "String.",
 "is_complete": Boolean, "next_goal": "String"
}
CRITICAL RULES:
1. Strict constraint: Only set is_complete=True if Solution
exists AND Verification passes.
```

Semantic Descriptors (Key/Query). Each math agent generates query (s_q) and key (s_k) text describing its current information need and offered contribution. The semantic encoder maps these descriptors to vectors for routing.

Table 14: Mathematical agents' query (s_q) and key (s_k) descriptors before embedding.

Role	Typical Query Descriptor (s_q)	Typical Key Descriptor (s_k)
ProblemParser	I need the problem statement to analyze.	I provide problem analysis and solving plan.
Solver	I need a problem analysis and solving plan.	I provide detailed mathematical solutions with step-by-step derivations.
Verifier	I need a detailed solution to verify.	I provide verification of mathematical solutions.
Manager	I need status updates from all team members.	I coordinate team efforts and set priorities.

G LIMITATIONS

DyTopo's RSI process changes communication within one task; model parameters, worker roles, and routing rules remain fixed. We do not evaluate whether any improvement is retained across tasks. The semantic feedback selection controls show that message selection matters, but do not isolate the benefit of reconstructing the graph every round. A frozen-topology control with the same agents, prompts, and budgets would test that benefit; shuffled-feedback controls would test its dependence on the current reasoning state. Small mathematical subsets, threshold selection on two evaluation datasets, and completion failures under the original decoding protocol also limit the empirical scope.

H AI ASSISTANCE DETAILS

For this manuscript revision, an LLM assistant helped articulate the within-task structural RSI framing and formalize the existing algorithm, interpret reported results, locate relevant literature, revise contribution and result wording, organize the manuscript, and adjust LaTeX formatting. It did not run the experiments or generate experimental measurements for this revision. Derived numerical summaries were checked against the existing tables; table contents, prompt templates, figure files, and citation keys were checked for preservation. The source package was recompiled and the rendered PDF inspected. These checks do not reproduce the experiments or resolve the limitations (Appendix G).