EvoHyper: Evolving Hypergraph Topologies for Unified Collaboration in Multi-Agent Communication

Heng Zhang[⋆] **Yihao Zhong**[†] **Lubin Gan**[‡] **Zhihe Chen**[§] **Jiajun Wu**[♣]
Yuling Shi[§] **Xiaodong Gu**[§] **Hao Zhang**[♦] **Haochen You**[♡] **Jin Huang**[⋆]
[⋆]SCNU [†]NYU [‡]USTC [§]SJTU [♣]CSU [♦]UCAS [♡]CU
Primary Contact: 2024025450@m.scnu.edu.cn

Abstract

Multi-agent systems powered by large language models have achieved strong performance on complex tasks, yet naive collaboration topologies often cause high communication costs and redundant context. Existing methods usually use a fixed communication graph and manage collaboration structure and shared memory in separate modules. Our log analysis of several representative systems shows that this separation leads to multiple copies of the same key facts in dialogue, memory and model inputs. We address this issue with **EvoHyper**, a framework based on an evolving hypergraph topology for multi-agent collaboration. In **EvoHyper**, a single hypergraph represents agents and shared memory, and each hyperedge serves as a collaboration unit that binds a group of agents to that shared memory. During execution a controller edits the hypergraph through a small set of predefined evolution operations, so collaboration units can spawn, update and merge as tasks unfold. Experiments on four benchmarks covering mathematical reasoning and code generation show that **EvoHyper** is (I) **high-performing**, achieving 3.2% to 7.8% accuracy gains over state-of-the-art methods, (II) **efficient**, reducing token consumption by up to 23.5%, and (III) **adaptive**, adjusting topology complexity according to task requirements.

1 Introduction

Recent advances in Large Language Model (LLM) based agents (OpenAI et al., 2024; Anthropic, 2024) have achieved remarkable success across various tasks. Building on this progress, LLM-based Multi-Agent Systems (MAS) harness collective intelligence through agent collaboration (Qian et al., 2025; Chen et al., 2024b; Qian et al., 2025). At the core of such collaboration lies the communication topology (Zhuge et al., 2024; Zhang et al., 2025c; Liu et al., 2024). It governs agent interactions and directly affects both performance and resource consumption. To ensure reliable coordination, most developers adopt dense topologies with lengthy shared contexts (Wu et al., 2023; Hong et al., 2024). Yet this conservative strategy causes token consumption to grow rapidly on complex tasks. Designing topologies that balance performance and efficiency remains an open problem.

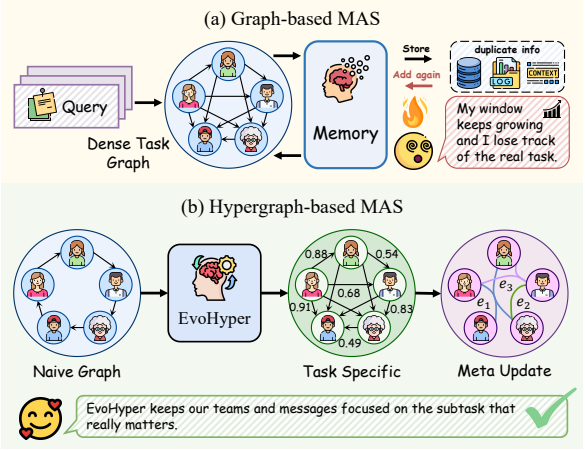


Figure 1: Paradigm comparison between graph-based and hypergraph-based multi-agent systems. **(a)** Graph-based MAS separates topology and memory, causing duplicate information to accumulate in context windows. **(b)** **EvoHyper** evolves a unified hypergraph where each hyperedge binds collaborating agents with shared memory, keeping focus on task-relevant content.

To address this challenge, various topology design methods have been proposed. Early attempts employ predefined static structures such as Chain (Wei et al., 2022), Star (Wu et al., 2023), and Tree (Yao et al., 2023a), which are simple but cannot adapt to varying task demands. Learning-based methods like DyLAN (Liu et al., 2024), GPTSwarm (Zhuge et al., 2024), and G-Designer (Zhang et al., 2025c) overcome this rigidity through dynamic topology generation. Meanwhile, another line explores agent memory for knowledge accumulation, including HG-Mem (Zhou et al., 2025) for structured memory updates and MemEvolve (Zhang et al., 2025a) for

Findings of the Association for Computational Linguistics: ACL 2026, pages 25130–25143

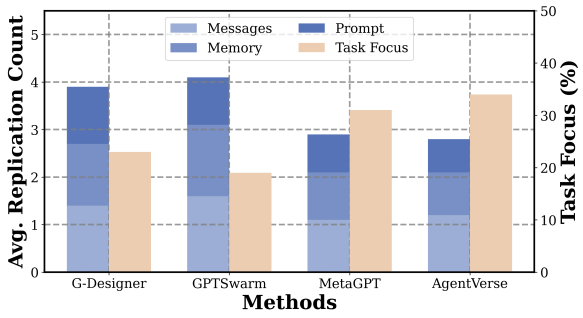


Figure 2: Information replication across four multi-agent systems on GSM8K. **Left axis (stacked bars):** average count of key fact occurrences across three components. **Right axis:** percentage of context window devoted to task-relevant content. Systems with denser topologies exhibit higher redundancy and lower task focus.

adaptive memory architectures. Despite notable progress, existing methods treat topology and memory as independent modules (as illustrated in Figure 1). We argue that unified management is essential, as separation forces information replication across modules and creates synchronization difficulties, hurting both efficiency and consistency.

To examine this effect, we analyze execution logs of four representative systems on GSM8K: G-Designer (Zhang et al., 2025c), GPTSwarm (Zhuge et al., 2024), MetaGPT (Hong et al., 2024) and AgentVerse (Chen et al., 2024b). As shown in Figure 2, significant information redundancy exists across all systems. The same critical information appears repeatedly in inter-agent messages, memory storage, prompt construction. On average, each key fact occurs 2.8 to 4.1 times across these components. G-Designer and GPTSwarm exhibit higher redundancy given their dense communication patterns. Such redundancy grows more severe in later collaboration stages. Notably, 67% of repetitions arise from agents on the same subtask. They repeatedly exchange and store identical knowledge through separate channels.

A seemingly intuitive solution is to apply deduplication or synchronization across modules. However, as long as topology and memory remain separate, each module must maintain its own copy of shared information. Deduplication targets storage redundancy, yet content continues flowing through multiple channels. Synchronization offers better consistency but incurs overhead proportional to copy count. In summary, practitioners find it challenging to *eliminate information redundancy without rethinking how collaboration itself is represented*. Against this backdrop, we argue that eliminating redundancy requires rethinking what the basic unit of collaboration should be. Such a unit must satisfy three properties: ♣ **Unified Representation:** Encoding both participants and shared knowledge in one place to avoid cross-module replication. ♦ **Dynamic Evolution:** Forming and dissolving as collaboration needs change. ♥ **Hierarchical Composition:** Merging into higher-order structures as shared understanding deepens. Hyperedges naturally provide these properties. One hyperedge links multiple agents and holds their shared state together. It can be created, modified, or merged as the task proceeds.

Based on this insight, we propose `EvoHyper`, a framework that treats each hyperedge as a unified collaboration unit carrying both structure and memory. The framework introduces three evolution operations. Update refines hyperedge memories based on collaboration feedback. Spawn creates new hyperedges when emerging coordination needs are detected. Merge consolidates related hyperedges into higher-order units with integrated memory. The Merge operation is particularly important. It enables collaboration structures to grow hierarchically as agents develop deeper shared understanding. To prevent memory explosion over long executions, we design a hierarchical memory structure with bounded capacity through periodic compression. Simple tasks maintain sparse and efficient structures. Complex tasks progressively develop richer higher-order collaborations. This unified paradigm allows `EvoHyper` to achieve **adaptive topology, bounded memory, and consistent shared state** throughout execution.

Our contributions can be summarized as follows:

❶ ***Paradigm Proposal.*** We conduct empirical analysis on four representative multi-agent systems and show that most information redundancy arises inside collaboration subtasks. Based on this finding, we propose a new paradigm that unifies structure and memory within hyperedges, treating each hyperedge as a self-contained collaboration unit.

❷ ***Practical Framework.*** We present `EvoHyper`, a unified framework that lets hypergraph topologies evolve through *Update*, *Spawn*, and *Merge* operations. A hierarchical memory mechanism with periodic compression prevents unbounded growth during long-horizon tasks.

❸ ***Experimental Validation.*** Extensive experiments across four benchmarks demonstrate that

`EvoHyper` is **(I) high-performing**, achieving $3.2\% \sim 7.8\%$ improvement over state-of-the-art methods; **(II) efficient**, reducing token consumption by up to 23.5%; and **(III) adaptive**, automatically adjusting topology complexity based on task requirements.

2 Related Work

2.1 LLM-based Multi-Agent Collaboration

Multi-agent collaboration integrates specialized capabilities of different agents through diverse mechanisms. Early work explored debate-based methods like ChatEval (Chan et al., 2023) and role-playing approaches like CAMEL (Li et al., 2023). Building on these foundations, structured workflows emerged with MetaGPT (Hong et al., 2024) encoding standardized operating procedures and ChatDev (Qian et al., 2024) combining natural and programming languages. To handle varying task demands, adaptive methods like ProAgent (Zhang et al., 2024) and AutoAgents (Chen et al., 2024a) dynamically adjust agent configurations. Recently, MacNet (Qian et al., 2025) organized over a thousand agents into directed acyclic graphs, revealing collaborative scaling laws. Yet across all these methods, topology and memory remain separate modules. Unifying them within a single structure motivates our hyperedge-based design.

2.2 Multi-Agents as Graphs

Graphs provide a natural abstraction for agent interactions. Early approaches relied on static structures including chains (Wei et al., 2022), trees (Yao et al., 2023a), and complete graphs (Qian et al., 2025). To overcome this rigidity, pruning-based methods like AgentPrune (Zhang et al., 2025b) and AgentDropout (Wang et al., 2025) learn sparse task-specific graphs. The field then progressed toward learning-based approaches with DyLAN (Liu et al., 2024) modeling dynamic agent networks and AgentVerse (Chen et al., 2024b) capturing emergent collaborative behaviors. More recent work like AFlow (Zhang et al., 2025d) constructs topologies autoregressively, while Wu et al. (Wu et al., 2024) demonstrate how graph learning enhances LLM-based planning. However, pairwise edges fundamentally limit representational capacity. A group of agents on one subtask must be decomposed into multiple separate edges. Hyperedges overcome this limitation by directly encoding multi-agent groups as unified collaboration units.

3 Preliminary

3.1 Multi-Agent System as Graph

We model an LLM-based multi-agent system as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{v_1, \ldots, v_N\}$ represents N agents and $\mathcal{E}$ denotes edges governing information flow. Each agent $v_i \in \mathcal{V}$ is defined as

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

where $\texttt{Base}_i$ specifies the language model, $\texttt{Role}_i$ defines expertise, $\texttt{State}_i$ maintains history, and $\texttt{Plugin}_i$ provides external tools. Each agent receives prompt $\mathcal{P}$ and generates response $\mathcal{R}_i = v_i(\mathcal{P}_{\text{sys}}, \mathcal{P}_{\text{usr}})$ where $\mathcal{P}_{\text{sys}} = \{\texttt{Role}_i, \texttt{State}_i\}$ and $\mathcal{P}_{\text{usr}}$ contains the task and messages from other agents. Given task $\mathcal{Q}$, agents interact for K rounds. At round t, agent v_i generates

$$\mathcal{R}_i^{(t)} = v_i\left(\mathcal{P}_{\text{sys}}^{(t)}, \left\{\mathcal{Q}, \bigcup_{v_j \in \mathcal{N}_{\text{in}}(v_i)} \mathcal{R}_j^{(t)}\right\}\right) \quad (2)$$

where $\mathcal{N}_{\text{in}}(v_i)$ denotes the in-neighborhood of v_i.

3.2 Hypergraph Representation

Graphs model only pairwise interactions. When multiple agents jointly handle one subtask, graph models must decompose this into multiple edges, leading to multi-hop message passing. A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ extends graphs by allowing each hyperedge $e_k \subseteq \mathcal{V}$ to connect arbitrary numbers of vertices. The structure is encoded by incidence matrix $\mathbf{H} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{E}|}$ where $h_{v,e} = 1$ if $v \in e$ and 0 otherwise.

Given node feature matrix $\mathbf{X}^{(l)} \in \mathbb{R}^{|\mathcal{V}| \times D}$ at layer l, the hypergraph convolutional layer computes

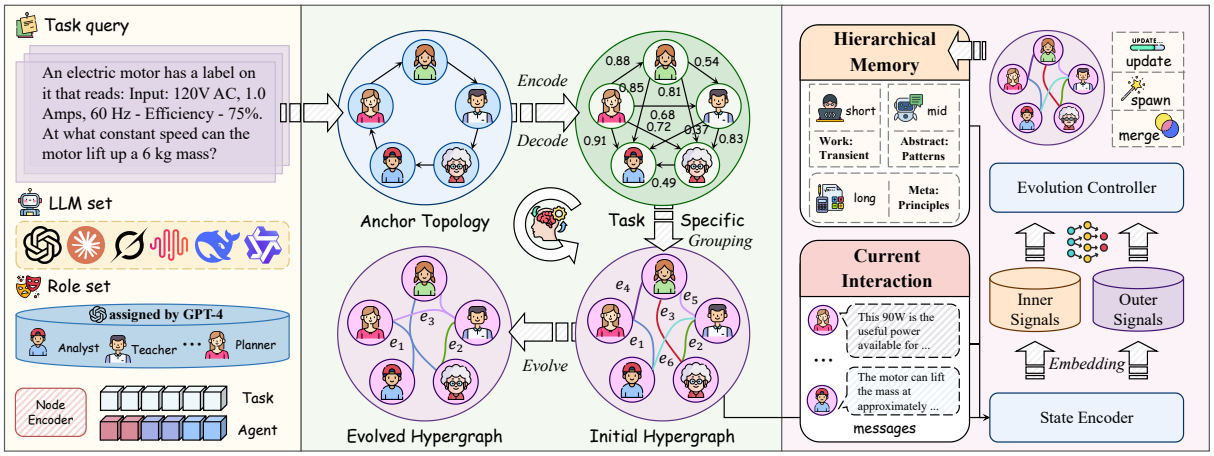
$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{D}^{-1/2}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\top}\mathbf{D}^{-1/2}\mathbf{X}^{(l)}\mathbf{\Theta}^{(l)}\right) \quad (3)$$

where $\mathbf{D}$ and $\mathbf{B}$ are diagonal matrices of vertex and hyperedge degrees respectively, $\mathbf{W}$ assigns weights to hyperedges, $\mathbf{\Theta}^{(l)}$ is a learnable transformation, and $\sigma(\cdot)$ is an activation function. This node-edge-node transformation enables single-step synchronization within collaboration groups.

4 Methodology

4.1 Collaboration Unit with Shared Memory

We maintain a dynamic hypergraph $\mathcal{H}^{(t)} = (\mathcal{V}, \mathcal{E}^{(t)}, \mathcal{M}^{(t)})$ at round t, where $\mathcal{V}$ is the agent



(a) MAS Construct (b) Propagation & Optimization (c) Memory & Evolution

Figure 3: Overview of `EvoHyper`. **(a)** Agents and a task-specific virtual node are encoded into embeddings to enable task-aware topology generation. **(b)** Starting from an anchor topology, a variational encoder-decoder generates a task-adaptive hypergraph where each hyperedge binds collaborating agents with shared memory. The topology evolves through Update, Spawn, and Merge operations as tasks unfold. **(c)** Hierarchical memory and current interactions are encoded into inner signals (for memory refinement) and outer signals (for structural adaptation). The evolution controller uses both to orchestrate dual-loop evolution: adapting memory within tasks and optimizing policy across tasks.

set, $\mathcal{E}^{(t)}$ is the hyperedge set, and $\mathcal{M}^{(t)}$ collects memory states. Each hyperedge serves as a self-contained collaboration unit that binds participating agents with shared memory. Formally, a hyperedge $e \in \mathcal{E}^{(t)}$ is defined as

$$e = (\mathcal{A}_e, \mathbf{m}_e) \tag{4}$$

where $\mathcal{A}_e \subseteq \mathcal{V}$ denotes participating agents and $\mathbf{m}_e$ denotes shared memory. Unlike prior methods where each agent maintains private copies, $\mathbf{m}_e$ provides a single source of truth for the collaboration unit.

When agent v_i takes action at round t, its context is constructed as

$$c_i^{(t)} = \text{Concat}\left(\mathcal{P}_{\text{sys}},\ \mathcal{Q},\ \bigoplus_{e: v_i \in \mathcal{A}_e} \mathbf{m}_e^{(t)}\right) \tag{5}$$

where $\bigoplus$ aggregates memories from all hyperedges containing v_i.

4.2 Initial Topology Generation

We generate the initial topology using a variational autoencoder with sparsity control.

Encoding. We construct node features by encoding each agent's profile:

$$\mathbf{x}_i = \text{NodeEncoder}(\texttt{Role}_i, \texttt{Base}_i, \texttt{Plugin}_i) \tag{6}$$

We also introduce a task-specific virtual node v_{task} with embedding $\mathbf{x}_{\text{task}} = \text{NodeEncoder}(\mathcal{Q})$ that connects to all agents, facilitating global information flow. Let $\tilde{\mathbf{X}}$ denote the concatenation of agent embeddings and task embedding. Given a simple anchor topology $\mathbf{H}_{\text{anchor}}$ as prior structure, the encoder maps features to latent distributions through two hypergraph convolutional layers:

$$\boldsymbol{\mu} = \text{HGCN}_{\mu}(\tilde{\mathbf{X}}, \mathbf{H}_{\text{anchor}}) \tag{7}$$

$$\log \boldsymbol{\sigma} = \text{HGCN}_{\sigma}(\tilde{\mathbf{X}}, \mathbf{H}_{\text{anchor}}) \tag{8}$$

Latent representations are then sampled as $\mathbf{h}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \text{diag}(\boldsymbol{\sigma}_i^2))$.

Decoding with Sparsity Control. The decoder transforms latent representations into hypergraph structure through two phases. First, we construct a sketched affinity matrix $\mathbf{S} \in [0,1]^{N \times N}$. For each pair of agents, we compute

$$\mathbf{S}_{ij} = g(\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{\text{task}}) \tag{9}$$

where $g(\cdot)$ is parameterized by a feed-forward network with Gumbel-Softmax sampling. Specifically, let $\varpi_{ij} = \text{FFN}([\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_{\text{task}}])$, then

$$g(\cdot) = \text{Sigmoid}\left(\frac{\log \epsilon - \log(1-\epsilon) + \varpi_{ij}}{\tau}\right) \tag{10}$$

where $\epsilon \sim \text{Uniform}(0,1)$ and temperature τ controls discreteness.

The sketched matrix $\mathbf{S}$ is typically dense. To achieve task-adaptive sparsity, we refine it through

low-rank regularization. Let $\mathbf{Z} \in \mathbb{R}^{N \times r}$ contain the top-r left singular vectors of $\mathbf{S}$. We optimize weight matrix $\mathbf{W} \in \mathbb{R}^{r \times r}$ to minimize

$$\begin{aligned}\mathcal{L}_{\text{refine}} =& \frac{1}{2}\|\mathbf{S} - \mathbf{Z}\mathbf{W}\mathbf{Z}^\top\|_F^2 + \zeta\|\mathbf{W}\|_* \\ &+ \frac{1}{2}\|\mathbf{A}_{\text{anchor}} - \mathbf{Z}\mathbf{W}\mathbf{Z}^\top\|_F^2 \end{aligned} \tag{11}$$

where $\|\mathbf{W}\|_* = \sum_i \lambda_i$ is the nuclear norm encouraging sparsity, and $\mathbf{A}_{\text{anchor}}$ is the pairwise adjacency derived from anchor topology. The coefficient ζ controls sparsification strength. The refined matrix is obtained as

$$\tilde{\mathbf{S}} = \mathbf{Z}\mathbf{W}\mathbf{Z}^\top \tag{12}$$

This formulation enables automatic complexity adjustment: simple tasks yield sparse topologies while complex tasks produce denser structures. Finally, we convert $\tilde{\mathbf{S}}$ to hyperedges by grouping strongly connected agents. For each agent i, we form a hyperedge connecting i with its top-k neighbors in $\tilde{\mathbf{S}}$, producing initial topology $\mathcal{H}^{(0)}$.

4.3 Hypergraph Evolution Operations

The hypergraph evolves through three operations during execution.

Update. The Update operation refines memory of an existing hyperedge based on new agent outputs. Given hyperedge e with memory $\mathbf{m}_e^{(t)}$ and outputs $\{o_i^{(t)}\}_{v_i \in \mathcal{A}_e}$:

$$\mathbf{m}_e^{(t+1)} = f_{\text{update}}\left(\mathbf{m}_e^{(t)},\ \{o_i^{(t)}\}_{v_i \in \mathcal{A}_e}\right) \tag{13}$$

where f_{update} extracts task-relevant content and integrates it with existing memory. This operation accumulates knowledge while preserving hyperedge structure.

Spawn. The Spawn operation creates new hyperedges when coordination needs arise among agents not yet bound together. Let $\mathcal{A}_{\text{new}} \subseteq \mathcal{V}$ be agents requiring joint collaboration:

$$\mathcal{E}^{(t+1)} = \mathcal{E}^{(t)} \cup \{e_{\text{new}}\} \tag{14}$$

where $e_{\text{new}} = (\mathcal{A}_{\text{new}}, \mathbf{m}_{\text{init}})$ and $\mathbf{m}_{\text{init}}$ is derived from task context and relevant outputs. This enables topology expansion as new subtasks emerge.

Merge. The Merge operation consolidates related hyperedges into higher-order units, central to hierarchical structure formation. Given semantically related hyperedges $e_i = (\mathcal{A}_{e_i}, \mathbf{m}_{e_i})$ and $e_j = (\mathcal{A}_{e_j}, \mathbf{m}_{e_j})$:

$$e_{\text{merged}} = \left(\mathcal{A}_{e_i} \cup \mathcal{A}_{e_j},\ f_{\text{fuse}}(\mathbf{m}_{e_i}, \mathbf{m}_{e_j}, \mathcal{Q})\right) \tag{15}$$

where f_{fuse} synthesizes both memories into unified representation conditioned on task $\mathcal{Q}$. Original hyperedges are removed after merging. Crucially, f_{fuse} performs lossy compression ensuring $|f_{\text{fuse}}(\cdot)| \leq C_{\max}$, allowing structures to become increasingly abstract without unbounded growth.

4.4 Hierarchical Memory Management

To prevent memory explosion over long executions, we organize each hyperedge memory into three levels:

$$\mathbf{m}_e = (\mathbf{m}_e^{\text{work}}, \mathbf{m}_e^{\text{abs}}, \mathbf{m}_e^{\text{meta}}) \tag{16}$$

The working level $\mathbf{m}_e^{\text{work}}$ holds transient information with capacity C_w. The abstract level $\mathbf{m}_e^{\text{abs}}$ stores distilled patterns with capacity C_a. The meta level $\mathbf{m}_e^{\text{meta}}$ maintains compressed principles with capacity C_m. Total size is bounded by $C_w + C_a + C_m$.

When working level approaches capacity, compression promotes salient content upward:

$$\mathbf{m}_e^{\text{abs}} \leftarrow f_{\text{compress}}(\mathbf{m}_e^{\text{abs}}, \mathbf{m}_e^{\text{work}}) \tag{17}$$

where f_{compress} identifies patterns and integrates them into abstract level. Working level is then cleared. Similar promotion occurs from abstract to meta when needed. This ensures valuable knowledge is preserved at higher levels while transient details are consolidated.

4.5 Learnable Evolution Controller

A learnable policy network governs the evolution operations.

State Encoding. At round t, we encode the current system state into a vector representation. The state encoder aggregates hypergraph structure, memory contents, agent outputs, and task information:

$$\mathbf{s}^{(t)} = f_{\text{state}}\left(\mathcal{H}^{(t)}, \{\mathbf{m}_e^{(t)}\}_{e \in \mathcal{E}^{(t)}}, \{o_i^{(t)}\}_{i=1}^N, \mathcal{Q}\right) \tag{18}$$

where f_{state} is implemented as a hypergraph neural network that captures both local memory states and global topological patterns.

Method	Mul.	Ada.	Rob.	MMLU	GSM8K	MultiArith	SVAMP	AQuA	HumanEval	Avg.
Single-Agent Methods										
Vanilla	✗	✗	✗	82.14	85.40	93.15	87.18	70.34	71.68	81.65
CoT	✗	✗	✗	82.65$_{\uparrow 0.51}$	87.17$_{\uparrow 1.77}$	94.79$_{\uparrow 1.64}$	88.32$_{\uparrow 1.14}$	73.91$_{\uparrow 3.57}$	75.52$_{\uparrow 3.84}$	83.73
ComplexCoT	✗	✗	✗	83.78$_{\uparrow 1.64}$	87.62$_{\uparrow 2.22}$	95.86$_{\uparrow 2.71}$	90.17$_{\uparrow 2.99}$	77.58$_{\uparrow 7.24}$	74.94$_{\uparrow 3.26}$	84.99
SC (CoT)	✗	✗	✗	82.66$_{\uparrow 0.52}$	87.93$_{\uparrow 2.53}$	96.88$_{\uparrow 3.73}$	88.69$_{\uparrow 1.51}$	75.08$_{\uparrow 4.74}$	77.30$_{\uparrow 5.62}$	84.75
SC (ComplexCoT)	✗	✗	✗	83.65$_{\uparrow 1.51}$	86.14$_{\downarrow 0.74}$	96.94$_{\uparrow 3.79}$	89.72$_{\uparrow 2.54}$	77.69$_{\uparrow 7.35}$	77.94$_{\uparrow 6.26}$	85.35
AutoGPT	✗	✗	✗	83.42$_{\uparrow 1.28}$	87.85$_{\uparrow 2.45}$	96.12$_{\uparrow 2.97}$	89.34$_{\uparrow 2.16}$	76.53$_{\uparrow 6.19}$	79.18$_{\uparrow 7.50}$	85.41
PHP	✓	✗	✗	83.45$_{\uparrow 1.31}$	95.50$_{\uparrow 10.1}$	98.10$_{\uparrow 2.84}$	90.02$_{\uparrow 3.44}$	79.00$_{\uparrow 8.66}$	82.96$_{\uparrow 11.36}$	88.17
ReAct	✗	✗	✗	83.12$_{\uparrow 0.98}$	88.24$_{\uparrow 2.84}$	95.37$_{\uparrow 2.22}$	89.15$_{\uparrow 1.97}$	76.42$_{\uparrow 6.08}$	78.35$_{\uparrow 6.67}$	85.11
ToT	✗	✗	✗	83.89$_{\uparrow 1.75}$	89.06$_{\uparrow 3.66}$	96.52$_{\uparrow 3.37}$	90.24$_{\uparrow 3.06}$	77.95$_{\uparrow 7.61}$	80.12$_{\uparrow 8.44}$	86.30
GoT	✗	✗	✗	84.01$_{\uparrow 1.87}$	89.47$_{\uparrow 4.07}$	96.73$_{\uparrow 3.58}$	90.38$_{\uparrow 3.20}$	78.24$_{\uparrow 7.90}$	81.26$_{\uparrow 9.58}$	86.68
Static Multi-Agent Topologies										
Chain	✓	✗	✗	82.35$_{\uparrow 0.21}$	85.57$_{\uparrow 0.17}$	94.38$_{\uparrow 1.23}$	83.41$_{\downarrow 3.77}$	70.94$_{\uparrow 0.60}$	80.88$_{\uparrow 9.20}$	82.92
Star	✓	✗	✗	80.79$_{\downarrow 1.35}$	85.55$_{\uparrow 0.15}$	93.79$_{\downarrow 0.64}$	88.09$_{\uparrow 0.91}$	68.57$_{\downarrow 1.77}$	75.65$_{\uparrow 3.97}$	82.07
Tree	✓	✗	✗	81.89$_{\downarrow 0.25}$	84.56$_{\downarrow 0.84}$	94.60$_{\uparrow 1.45}$	89.25$_{\uparrow 2.07}$	72.84$_{\uparrow 2.50}$	77.38$_{\uparrow 5.70}$	83.42
Complete Graph	✓	✗	✗	83.15$_{\uparrow 1.01}$	86.49$_{\uparrow 1.09}$	97.20$_{\uparrow 4.05}$	89.48$_{\uparrow 2.30}$	79.21$_{\uparrow 8.87}$	83.75$_{\uparrow 12.07}$	86.55
Random Graph	✓	✗	✗	83.76$_{\uparrow 1.62}$	86.14$_{\uparrow 0.74}$	95.46$_{\uparrow 2.31}$	85.41$_{\downarrow 1.77}$	74.07$_{\uparrow 3.73}$	82.66$_{\uparrow 10.98}$	84.58
Adaptive Multi-Agent Frameworks										
AutoGen	✓	✗	✗	82.13$_{\downarrow 0.01}$	90.06$_{\uparrow 7.92}$	93.80$_{\uparrow 0.65}$	88.44$_{\downarrow 1.26}$	73.65$_{\uparrow 3.31}$	85.41$_{\uparrow 13.73}$	85.58
MetaGPT	✓	✗	✗	83.24$_{\uparrow 1.10}$	89.84$_{\uparrow 4.44}$	95.12$_{\uparrow 1.97}$	89.56$_{\uparrow 2.38}$	76.18$_{\uparrow 5.84}$	85.90$_{\uparrow 14.22}$	86.64
LLM-Blender	✓	✗	✗	81.22$_{\downarrow 0.92}$	89.17$_{\uparrow 3.77}$	94.27$_{\uparrow 1.12}$	88.77$_{\uparrow 1.59}$	77.05$_{\uparrow 6.71}$	84.52$_{\uparrow 12.84}$	85.83
LLM-Debate	✓	✗	✓	83.69$_{\uparrow 1.55}$	90.23$_{\uparrow 4.83}$	96.27$_{\uparrow 3.12}$	90.56$_{\uparrow 3.38}$	77.52$_{\uparrow 7.18}$	83.79$_{\uparrow 12.11}$	87.01
DyLAN	✓	✓	✓	80.16$_{\downarrow 1.98}$	88.16$_{\uparrow 2.76}$	94.27$_{\uparrow 1.12}$	87.40$_{\uparrow 0.22}$	74.16$_{\uparrow 3.82}$	89.70$_{\uparrow 18.02}$	85.64
GPTSwarm	✓	✓	✓	83.98$_{\uparrow 1.84}$	89.74$_{\uparrow 4.34}$	97.84$_{\uparrow 4.69}$	86.42$_{\downarrow 0.76}$	78.16$_{\uparrow 7.82}$	88.49$_{\uparrow 16.81}$	87.44
AgentVerse	✓	✓	✗	83.52$_{\uparrow 1.38}$	90.12$_{\uparrow 4.72}$	96.45$_{\uparrow 3.30}$	89.87$_{\uparrow 2.69}$	77.83$_{\uparrow 7.49}$	86.24$_{\uparrow 14.56}$	87.34
COPPER	✓	✓	✗	83.76$_{\uparrow 1.62}$	91.35$_{\uparrow 5.95}$	96.82$_{\uparrow 3.67}$	90.18$_{\uparrow 3.00}$	78.42$_{\uparrow 8.08}$	87.53$_{\uparrow 15.85}$	88.01
AutoAgents	✓	✓	✗	83.45$_{\uparrow 1.31}$	90.58$_{\uparrow 5.18}$	96.15$_{\uparrow 3.00}$	89.64$_{\uparrow 2.46}$	77.29$_{\uparrow 6.95}$	86.87$_{\uparrow 15.19}$	87.33
G-Designer	✓	✓	✓	84.25$_{\uparrow 2.11}$	92.18$_{\uparrow 6.78}$	97.56$_{\uparrow 4.41}$	91.02$_{\uparrow 3.84}$	78.94$_{\uparrow 8.60}$	88.72$_{\uparrow 17.04}$	88.78
AgentPrune	✓	✓	✓	84.15$_{\uparrow 2.01}$	91.86$_{\uparrow 6.46}$	97.38$_{\uparrow 4.23}$	90.73$_{\uparrow 3.55}$	78.65$_{\uparrow 8.31}$	88.15$_{\uparrow 16.47}$	88.49
AgentDropout	✓	✓	✓	84.08$_{\uparrow 1.94}$	91.52$_{\uparrow 6.12}$	97.21$_{\uparrow 4.06}$	90.45$_{\uparrow 3.27}$	78.51$_{\uparrow 8.17}$	87.68$_{\uparrow 15.00}$	88.24
EvoHyper (Ours)	✓	✓	✓	**86.42**$_{\uparrow 4.28}$	**96.49**$_{\uparrow 11.09}$	**99.18**$_{\uparrow 6.03}$	**93.71**$_{\uparrow 6.53}$	**81.89**$_{\uparrow 11.55}$	**92.28**$_{\uparrow 20.60}$	**91.66**

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

Policy Network. The evolution controller is parameterized as a policy network $\pi(\cdot|\mathbf{s}^{(t)}; \Theta_\pi)$ that outputs a distribution over operations. For each candidate operation, the network computes:

$$p(\text{Update}_e|\mathbf{s}^{(t)}) = \sigma(\mathbf{w}_u^\top[\mathbf{s}^{(t)}, \mathbf{h}_e]) \quad (19)$$

$$p(\text{Spawn}_\mathcal{A}|\mathbf{s}^{(t)}) = \sigma(\mathbf{w}_s^\top[\mathbf{s}^{(t)}, \mathbf{h}_\mathcal{A}]) \quad (20)$$

$$p(\text{Merge}_{e_i,e_j}|\mathbf{s}^{(t)}) = \sigma(\mathbf{w}_m^\top[\mathbf{s}^{(t)}, \mathbf{h}_{e_i}, \mathbf{h}_{e_j}]) \quad (21)$$

where $\mathbf{h}_e$ denotes the embedding of hyperedge e, $\mathbf{h}_\mathcal{A}$ aggregates embeddings of agent subset $\mathcal{A}$, and σ is the sigmoid function. Operations exceeding a threshold τ_{op} are selected and executed sequentially to produce $\mathcal{H}^{(t+1)}$. We discuss the completeness of this operation set in Appendix D.

Dual-Loop Optimization. Our framework implements a dual-loop evolution mechanism inspired by meta-learning. The inner loop executes evolution operations within a single task, where the policy network determines operation sequences based on current states. The outer loop optimizes policy parameters Θ_π across tasks through gradient-based updates.

We define a composite reward combining task performance and efficiency:

$$R = R_{\text{task}} - \lambda_1 \cdot \frac{\text{TokenCost}}{T_{\text{budget}}} - \lambda_2 \cdot |\mathcal{E}^{(K)}| \quad (22)$$

where R_{task} measures answer correctness, the second term penalizes token consumption, and the third term encourages sparse final topologies. The

policy gradient is approximated as:

$$\nabla_{\Theta_\pi}\mathcal{L} \approx \frac{1}{M}\sum_{m=1}^{M} R_m \sum_{t=0}^{K} \nabla_{\Theta_\pi} \log \pi(op_m^{(t)}|\mathbf{s}_m^{(t)};\Theta_\pi) \tag{23}$$

Through this optimization, the system learns when to update memories, when to spawn new collaboration units, and when to merge related hyperedges, rather than relying on predefined heuristics.

5 Experiments

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

- (**RQ1**) Does `EvoHyper` achieve better task performance compared to existing multi-agent collaboration methods?
- (**RQ2**) Does the unified hypergraph representation reduce information redundancy and enable adaptive topology adjustment?
- (**RQ3**) How sensitive is `EvoHyper` to key hyperparameters such as memory capacity and evolution thresholds?
- (**RQ4**) How do the evolution operations and hierarchical memory contribute to overall performance?

5.1 Experiment Setup

Datasets and Metrics. We evaluate `EvoHyper` on three categories of benchmarks: **general reasoning** (MMLU (Hendrycks et al., 2021)), **mathematical reasoning** (GSM8K (Cobbe et al., 2021), MultiArith (Roy and Roth, 2016), SVAMP (Patel et al., 2021), AQuA (Ling et al., 2017)), and **code generation** (HumanEval (Chen et al., 2021)). Task performance is measured by accuracy for reasoning tasks and pass@1 for code generation. For efficiency, we report prompt token consumption. For structure analysis, we track average hyperedge size, evolution operations triggered, and memory compression ratio.

Baselines. We compare `EvoHyper` against three categories of baselines. ❶ **Single-agent methods** include CoT (Wei et al., 2022), ComplexCoT, Self-Consistency (Wang et al., 2023a), PHP, AutoGPT (Significant Gravitas, 2026), ReAct (Yao et al., 2023b), ToT (Yao et al., 2023a), and GoT (Besta et al., 2023). ❷ **Static multi-agent topologies** include Chain (Wei et al., 2022), Star (Wu et al., 2023), Tree (Yao et al., 2023a), Complete Graph (Qian et al., 2025), and Random Graph structures. ❸ **Adaptive multi-agent frameworks** include AutoGen (Wu et al., 2023), MetaGPT (Hong et al., 2024), LLM-Blender (Jiang et al., 2023), LLM-Debate (Du et al., 2024), DyLAN (Liu et al., 2024), GPTSwarm (Zhuge et al., 2024), AgentVerse (Chen et al., 2024b), COPPER, AgentPrune (Zhang et al., 2025b), AgentDropout (Wang et al., 2025), G-Designer (Zhang et al., 2025c). These baselines represent state-of-the-art approaches in topology design for multi-agent systems.

Implementation Details. We access GPT via the OpenAI API (`gpt-4-0613` and `gpt-3.5-turbo-0125`). Temperature is set to 0 for deterministic single-agent baselines (Vanilla, CoT, ComplexCoT, AutoGPT, ReAct, ToT, GoT). For Self-Consistency, we use temperature T=0.8 with 5 sampled reasoning paths following Wang et al. (2023b). For all multi-agent methods, temperature is set to 1 to enable diverse responses. The number of interaction rounds is K=3 across all experiments. The NodeEncoder uses `all-MiniLM-L6-v2` with embedding dimension D=384. The anchor hypergraph $\mathbf{H}_{\text{anchor}}$ is initialized as a simple chain structure. The hypergraph encoders HGCN_μ and HGCN_σ are two-layer networks with hidden dimension 64, rank r=16 for low-rank approximation, Gumbel-Softmax temperature τ=1×10^{-2}, and sparsity coefficient ζ=1×10^{-1}. For hierarchical memory, capacity bounds are set to C_w=512, C_a=256, C_m=128 tokens. Policy optimization uses Adam (lr=3×10^{-4}, M=10 sampled topologies). All experiments are conducted on 8× NVIDIA A100 80GB GPUs.

Fairness of Comparison. All multi-agent baselines use identical settings: 5 `gpt-4-0613` agents, temperature 1, and the same evaluation scripts with identical test splits and answer extraction procedures. For single-agent methods, we use the same backbone with temperature 0 (except Self-Consistency). We retrieve the same average number of tokens per step across methods to ensure fair efficiency comparison.

5.2 Main Result (RQ1)

Table 1 reports performance across six benchmarks spanning general reasoning, mathematical problem solving, and code generation. `EvoHyper` consistently outperforms all baselines, achieving an aver-

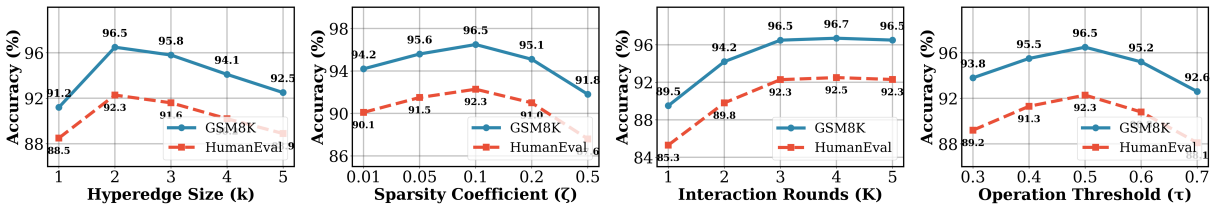


Figure 4: Hyperparameter sensitivity on GSM8K and HumanEval. **EvoHyper** achieves optimal performance at k=2, ζ=0.1, K=3, and τ_{op}=0.5.

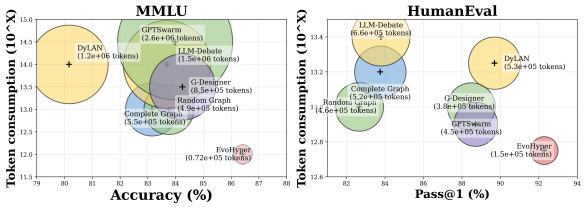


Figure 5: Performance and token consumption comparison on MMLU and HumanEval. Larger points indicate higher token cost.

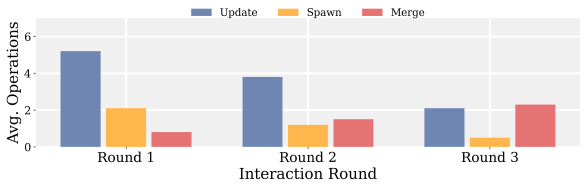


Figure 6: Evolution operation distribution across rounds. Update dominates early, Spawn peaks initially, and Merge increases later for hierarchical consolidation.

Method	MMLU↑	GSM8K↑	HumanEval↑	Avg.↑
EvoHyper (Full)	**86.42**	**96.49**	**92.28**	**91.73**
w/o Hypergraph	84.06	93.41	88.95	88.81
w/o Merge Op.	84.52	93.88	89.35	89.25
w/o Hierarchical Memory	85.67	95.54	91.02	90.74
w/o Evolution Controller	83.76	92.94	88.52	88.41
w/o VAE Framework	85.18	94.82	90.13	90.04

Table 2: Ablation study of key components in **EvoHyper** on representative benchmarks.

age accuracy of 91.66%, surpassing the strongest adaptive competitors G-Designer (88.78%) and AgentDropout (88.24%) by clear margins. The gains are most pronounced on complex coordination tasks: **EvoHyper** reaches 96.49% on GSM8K and 99.18% on MultiArith, and achieves 92.28% pass@1 on HumanEval. Notably, **EvoHyper** is the only method that simultaneously satisfies all three properties of multi-agent support, task-adaptive topology, and adversarial robustness. These results confirm that unifying collaboration structure and shared memory within hyperedges eliminates the cross-module redundancy that limits graph-based approaches, enabling both stronger coordination and lower token consumption.

5.3 Model Analysis (RQ2)

Figure 5 demonstrates **EvoHyper**'s efficiency advantage. Compared to baseline methods, **EvoHyper** achieves superior performance with significantly lower token consumption, as shown by the smaller bubble size. This efficiency stems from the unified hyperedge representation, which eliminates cross-module information duplication between separate topology and memory modules. A detailed breakdown of total token consumption, including both prompt and completion tokens generated by evolution operations, is provided in Appendix A, where we further show that prompt savings from redundancy elimination substantially outweigh the completion overhead of *Update* and *Merge*. Wall-clock time comparisons confirming that token reduction translates directly into faster LLM inference are reported in Appendix B.

Figure 6 reveals distinct evolution patterns across interaction rounds. *Update* operations dominate early rounds to establish shared memory within newly formed hyperedges, then decline as collaboration stabilizes. *Spawn* operations peak initially when emerging coordination needs are detected, then decrease as the topology matures. Conversely, *Merge* operations increase progressively across rounds, reflecting hierarchical consolidation as agents develop deeper shared understanding. To provide more intuitive insight into how these patterns manifest on concrete tasks, representative topology evolution traces for both simple arithmetic and complex code generation tasks are illustrated in Appendix C. Cross-LLM generalization results demonstrating that **EvoHyper**'s advantages hold consistently across heterogeneous and weaker agent configurations are reported in Appendix E. Training stability across random seeds is analysed in Appendix F.

5.4 Hyper-parameter Analysis (RQ3)

Figure 4 examines sensitivity to four key hyperparameters. For hyperedge size k, performance peaks at k=2 since k=1 degrades to pairwise edges while larger k yields overly sparse structures. Sparsity coefficient ζ=0.1 balances topology density optimally. Interaction rounds K show diminishing returns beyond K=3, where sufficient collaboration occurs without redundant communication. Operation threshold τ_{op}=0.5 achieves the best trade-off between evolution sensitivity and stability. These results validate our default configuration.

5.5 Ablation Study (RQ4)

Table 2 evaluates each component's contribution. Removing the evolution controller causes the largest drop (3.32%), confirming that learnable operation decisions are essential. The hypergraph structure ranks second (2.92%), validating that unified hyperedge representation outperforms pairwise edges. Removing Merge operations decreases accuracy by 2.48%, highlighting hierarchical consolidation's importance. The VAE framework contributes 1.69%, and hierarchical memory contributes 0.99% by preventing context explosion.

6 Conclusion

We present **EvoHyper**, a framework that unifies communication topology and shared memory within an evolving hypergraph. Each hyperedge serves as a collaboration unit binding agents with shared memory, eliminating information redundancy from separate modules. The hypergraph evolves through Update, Spawn, and Merge operations governed by a learnable controller. Experiments on four benchmarks demonstrate substantial performance gains with reduced token consumption, confirming the effectiveness of our unified representation and adaptive evolution mechanism.

Limitations

Our work has limitations suggesting future directions. First, **EvoHyper** assumes all agents are reliable, while real-world scenarios may involve unreliable or adversarial agents. Extending our framework with robustness mechanisms is a promising direction. Second, the current evolution operations are predefined, and automatically discovering new operation types could further improve adaptability. Third, our experiments focus on reasoning and code generation tasks, and evaluation on broader domains such as embodied agents remains future work.

Acknowledgments

This work was supported by the Natural Science Foundation of Guangdong Province, China (Grant No. 2026A1515010176)

AI Usage Statement

AI assistance was used only for text polishing. All ideas, methods, experiments, and analyses are the authors' own work.

References

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. Technical Report.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2023. Graph of thoughts: Solving elaborate problems with large language models. *arXiv preprint arXiv:2308.09687*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. 2024a. Autoagents: A framework for automatic agent generation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pages 21–29.

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, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

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

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*.

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

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 158–167.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024. A dynamic llm-powered agent network for task-oriented agent collaboration. *Preprint*, arXiv:2310.02170.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, and 1 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Chatdev: Communicative agents for software development. *Preprint*, arXiv:2307.07924.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Scaling large language model-based multi-agent collaboration. *Preprint*, arXiv:2406.07155.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Significant Gravitas. 2026. Autogpt. `https://github.com/Significant-Gravitas/AutoGPT`. Software, accessed April 20, 2026.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023a. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. *Preprint*, arXiv:2203.11171.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, and Dongsheng Li. 2024. Can graph learning improve planning in llm-based agents? *Preprint*, arXiv:2405.19119.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. 2024. Proagent: Building proactive cooperative agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17591–17599.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. 2025a. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2025b. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *The Thirteenth International Conference on Learning Representations (ICLR)*.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Dawei Cheng, and Tianlong Chen. 2025c. G-designer: Architecting multi-agent communication topologies via graph neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xiao Cheng, Sirui Hong, Jinlin Wang, and 1 others. 2025d. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*. Oral Presentation.

Chulun Zhou, Chunkang Zhang, Guoxin Yu, Fandong Meng, Jie Zhou, Wai Lam, and Mo Yu. 2025. Improving multi-step rag with hypergraph-based memory for long-context complex relational modeling. *arXiv preprint arXiv:2512.23959*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, PMLR, pages 62743–62767.

A Total Token Consumption Analysis

In the main paper, efficiency is reported via prompt token consumption. Here we provide the complete breakdown including completion tokens, which are incurred by `EvoHyper`'s generative evolution operations (*Update* and *Merge*).

As shown in Table 3, `EvoHyper`'s evolution operations (*Update* and *Merge*) contribute approximately 2.4×10^4 additional completion tokens on GSM8K, accounting for roughly 25% of our total completion cost. However, the unified hyperedge design eliminates cross-module information duplication, saving approximately 1.3×10^5 prompt tokens across all agents and interaction rounds. This saving substantially outweighs the evolution overhead. As a result, `EvoHyper`'s total token consumption (9.6×10^5) remains 9.4% lower than G-Designer (10.6×10^5) and 49% lower than GPTSwarm (18.8×10^5). On HumanEval, `EvoHyper` achieves the lowest total consumption (3.8×10^5) among all adaptive methods.

Method	GSM8K			HumanEval		
	Prom.	Comp.	Total	Prom.	Comp.	Total
G-Designer	8.5	2.1	10.6	3.4	1.4	4.8
GPTSwarm	15.0	3.8	18.8	4.2	1.7	5.9
AgentDropout	7.2	1.8	9.0	3.0	1.2	4.2
`EvoHyper`	7.2	2.4	9.6	2.6	1.2	3.8

Table 3: Total token consumption (prompt + completion, in units of 10^5 tokens) on GSM8K and HumanEval. "Completion" for `EvoHyper` includes tokens generated by *Update* and *Merge* operations. Despite this overhead, `EvoHyper`'s total consumption remains competitive.

B Wall-Clock Time Analysis

Table 4 reports per-problem wall-clock time on GSM8K, averaged over 200 samples. We decompose total latency into LLM inference time and topology/overhead time to isolate the cost of dynamic evolution.

Method	LLM Infer. (s)	Overhead (s)	Total (s)
Complete Graph	45.2	0.0	45.2
G-Designer	38.6	3.2	41.8
AgentDropout	36.4	1.8	38.2
`EvoHyper`	**31.8**	4.5	**36.3**

Table 4: Per-problem wall-clock time on GSM8K (averaged over 200 samples). Topology overhead for `EvoHyper` includes HGCN encoding ($\sim$1.2 s), SVD sparsification ($\sim$0.8 s), and evolution operations across 3 rounds ($\sim$2.5 s). Despite the overhead, `EvoHyper` achieves the lowest total latency due to faster LLM inference enabled by token reduction.

`EvoHyper` introduces a topology overhead of 4.5s per problem. However, the reduced token consumption directly accelerates LLM inference from 45.2s (Complete Graph) to 31.8s, yielding the lowest total latency of 36.3s among all compared methods. The SVD-based sparsification operates on small $r \times r$ matrices (r=16), keeping

its cost at $\sim$0.8s and computationally negligible. These results confirm that the efficiency gains from eliminating information redundancy outweigh the overhead of dynamic topology evolution.

C Representative Topology Evolution Traces

We present two representative hypergraph evolution traces to illustrate how `EvoHyper` adapts its topology to task complexity. In both cases, the system begins from the same anchor topology and evolves through *Update*, *Spawn*, and *Merge* operations.

Simple Task: Arithmetic Reasoning (GSM8K). Given a grade-school math problem, the initial topology $\mathcal{H}^{(0)}$ contains 5 hyperedges, each binding 2 agents. During Round 1, three *Update* operations refine the working-level memory of active hyperedges based on intermediate computation results. No new coordination groups are needed. By Round 3, two *Merge* operations consolidate semantically overlapping hyperedges into unified units, yielding a final structure of 3 hyperedges with an average size of 2.1 agents:

$$\mathcal{H}^{(0)}\colon \{e_1(v_1,v_2),\, e_2(v_2,v_3),\, e_3(v_3,v_4),\, e_4(v_4,v_5),\, e_5(v_1,v_5)\}$$

$$\xrightarrow{\text{Update}\times 3,\ \text{Merge}\times 2} \quad \mathcal{H}^{(3)}\colon \{e'_1(v_1,v_2,v_3),\, e'_2(v_3,v_4),\, e'_3(v_4,v_5)\}$$

The topology naturally converges to a sparse structure, consistent with our reward formulation that penalizes unnecessary hyperedges ($\lambda_2|\mathcal{E}^{(K)}|$ in Equation (22) of the main paper).

Complex Task: Multi-Function Code Generation (HumanEval). Given a programming task requiring multiple interdependent functions, the initial topology contains 6 hyperedges. Round 1 triggers 5 *Update* operations and 2 *Spawn* operations, creating new coordination units dedicated to testing and debugging roles. By Round 3, strategic *Merge* operations produce a hierarchical structure containing one high-order hyperedge of size 4 that coordinates the final integration step. The evolved topology has 4 hyperedges with an average size of 3.2 agents:

$$\mathcal{H}^{(0)}\colon 6 \text{ hyperedges (avg. size } 2.0)$$

$$\xrightarrow{\text{Update}\times 5,\ \text{Spawn}\times 2,\ \text{Round } 1\text{–}2} \xrightarrow{\text{Merge}\times 2,\ \text{Round } 3}$$

$$\mathcal{H}^{(3)}\colon 4 \text{ hyperedges (avg. size } 3.2)$$

including $e^*(v_1,v_2,v_3,v_4)$, a size-4 hyperedge whose shared memory integrates the outputs of all four core coding agents for final answer synthesis.

These two traces confirm the adaptive behaviour described in Section 3 of the main paper: simple tasks converge to sparse and efficient structures, while complex tasks organically develop richer higher-order collaborations as shared understanding deepens.

D Discussion on Additional Evolution Operations

We discuss why the three operations (*Update*, *Spawn*, *Merge*) form a sufficient and minimal set for topology evolution, and explain the design decision to exclude *Split* and *Delete*.

Split. A *Split* operation would decompose one hyperedge e_i into two disjoint sub-hyperedges e'_i and e''_i. This functionality is already implicitly covered by *Spawn*: the controller can spawn a new hyperedge e_{new} over a subset $\mathcal{A}' \subset \mathcal{V}_{e_i}$ of agents, while the *Update* operation simultaneously removes those agents' contributions from e_i's memory. The net effect is identical to *Split* but does not require a separate operation type.

Delete. A *Delete* operation would remove a hyperedge entirely. This is implicitly handled by the sparsity regularisation term $\lambda_2|\mathcal{E}^{(K)}|$ in the reward function (Equation (22) of the main paper), which continuously discourages unnecessary hyperedges. Redundant collaboration units receive weak *Update* signals over successive rounds and are eventually absorbed into neighbouring units through *Merge*, effectively disappearing from the active topology.

Ablation results. To empirically validate these design choices, we conducted ablation experiments adding *Split*, *Delete*, or both to the default operation set:

Adding either operation yields no improvement and slightly degrades performance (-0.08% to -0.64%), because the enlarged action space makes policy learning harder without providing new representational capacity. We therefore retain the minimal three-operation design.

Operation Set	GSM8K	HumanEval
Update+Spawn+Merge (default)	**96.49**	**92.28**
+ Split	96.32	92.15
+ Delete	96.41	91.87
+ Split + Delete	96.18	91.64

Table 5: Effect of adding Split and Delete operations on GSM8K and HumanEval. Adding these operations enlarges the action space without improving performance.

E Cross-LLM Generalization

We evaluate `EvoHyper` under heterogeneous and weaker LLM configurations to assess whether the performance advantages reported in Table 1 hold beyond the default 5×GPT-4 setting.

LLM Config.	Method	GSM8K	HumanEval
5×GPT-4 (def.)	G-Designer	92.18	88.72
	`EvoHyper`	**96.49** (+4.31)	**92.28** (+3.56)
5×GPT-3.5	G-Designer	78.42	72.15
	`EvoHyper`	**83.67** (+5.25)	**77.83** (+5.68)
2×GPT-4 + 3×GPT-3.5	G-Designer	85.34	80.46
	`EvoHyper`	**90.72** (+5.38)	**86.19** (+5.73)

Table 6: Performance under different LLM configurations on GSM8K and HumanEval. `EvoHyper` consistently outperforms G-Designer across all configurations, with larger gains observed for weaker or heterogeneous agent pools.

Three observations emerge from Table 6. First, `EvoHyper`'s advantage is consistent across all configurations, confirming that the gains do not depend on a specific LLM. Second, the improvement margins are larger for weaker or heterogeneous pools ($+5.25\%$ to $+5.73\%$) than for the homogeneous GPT-4 setting ($+3.56\%$ to $+4.31\%$). This is because the unified hyperedge memory becomes more valuable when individual agents have uneven capabilities: stronger agents' insights are stored directly in the shared hyperedge memory, allowing weaker agents within the same collaboration unit to benefit from them without additional communication rounds. Third, the relative ranking of methods is preserved across configurations, supporting the generalisability of conclusions drawn from the main experiments.

F Training Stability Analysis

We analyse training stability for the dual-loop optimisation to address concerns about convergence under joint multi-component training.

Empirical stability across seeds. We train `EvoHyper` with 5 independent random seeds on GSM8K and report final accuracy and loss statistics. As shown in Table 7, standard deviation in final accuracy is $\pm 0.34\%$, confirming stable and consistent convergence. The policy gradient loss decreases monotonically across all seeds without oscillation.

Method	Acc. (%) ↑	Loss ↓
G-Designer	92.18 ± 0.41	0.61 ± 0.04
AgentDropout	91.52 ± 0.38	0.67 ± 0.05
`EvoHyper`	$\mathbf{96.49 \pm 0.34}$	$\mathbf{0.24 \pm 0.02}$

Table 7: Training stability across 5 random seeds on GSM8K. Mean and standard deviation are reported for final accuracy and final training loss.

Design-level safeguards. Three architectural choices contribute to training stability. First, the hypergraph encoder is pretrained on anchor topologies before joint optimisation begins, providing a stable initialisation for the evolution controller. Second, the evolution controller updates its parameters only once per interaction round, while memory operations execute at each step, separating the timescales of the two nested loops and reducing gradient interference. Third, the hierarchical memory with bounded capacity ($C_w + C_a + C_m$ tokens per hyperedge) prevents unbounded state growth that could otherwise cause gradient explosion. Together, these safeguards ensure that joint training of the three components converges reliably.

Tier	Component	Drop(%)	Role
CORE	Hypergraph	−2.92	Foundation
	Evol. Ctrl.	−3.32	Adaptation
ENHANC.	Merge	−2.48	Consolidation
	VAE	−1.69	Init. quality
	Hier. Mem.	−0.99	Context ctrl.

Table 8: Component importance hierarchy (from Table 2). Average accuracy drop across all six benchmarks.

G Component Importance Hierarchy

Table 2 in the main paper reports accuracy drops for individual component ablations. Here we organise these results into an explicit importance hierarchy to clarify which components are essential to `EvoHyper`'s core functionality and which serve as performance enhancements.

A minimal version of **EvoHyper** retaining only the essential core (hypergraph structure with *Update* and *Spawn* operations, plus the evolution controller) achieves an average accuracy of 88.41%–88.81% across benchmarks, already surpassing the best baseline (G-Designer at 88.78%) on average. The remaining three components provide cumulative enhancements that bring the full system to 91.66%. Practitioners who require a lightweight deployment may therefore use the core-only variant at minimal performance cost.