Divide-and-Conquer Is What LLM-Based Multi-Agent System Need

Anonymous ACL submission

Abstract

Large language model (LLM) based multi-agent systems offer promising capabilities in social simulation and complex task solving, yet face key challenges in system design, generalizability, and scalability. We introduce AGENTGROUPCHAT-V2 , a novel framework featuring: (1) a fully parallel divide-and-conquer architecture for efficient task decomposition and distributed processing; (2) an adaptive collaboration engine that dynamically selects heterogeneous LLMs and interaction strategies; (3) agent organization optimization for effective problem breakdown. Experiments show that AGENTGROUPCHAT-V2 achieves state-of-the-art results across several benchmarks, with substantial improvements on tasks such as GSM8K, AIME, and HumanEval, especially as task complexity increases. Our results demonstrate that AGENTGROUPCHAT-V2 enables the construction of robust and general-purpose LLM multi-agent systems, excelling in complex reasoning scenarios.

1 Introduction

Interest in multi-agent systems based on large language models (LLMs) has grown rapidly (Guo et al., 2024; Li et al., 2023; Xi et al., 2023; Liang et al., 2024), driven by their promise in domains such as social simulation (Gao et al., 2024; Park et al., 2023; Gu et al., 2024b) and complex task resolution (Hong et al., 2023; Wu et al., 2023). In social simulations, LLM agents provide new ways to model human-like interactions and study collective behavior (Gu et al., 2024b; Lee et al., 2023); for problem-solving, collaborative multi-agent networks enhance reasoning and planning capabilities through distributed intelligence (Huang et al., 2024; Wei et al., 2022; Du et al., 2023). Compared to single-agent systems, multi-agent architectures foster emergent strategies and superior collective intelligence (Ferber and Weiss, 1999; Dafoe et al., 2021; Gu et al., 2024b).

Despite their progress, current LLM multi-agent frameworks face major barriers regarding generalizability, scalability, and real-world utility (Ouyang et al., 2022; Cemri et al., 2025). Many popular systems target either social simulation (Gao et al., 2023; Gu et al., 2024b) or specific tasks such as software development (Hong et al., 2023; Wu et al., 2023), lacking methods to support diverse, complex problem types (Shoham and Leyton-Brown, 2008). Furthermore, they frequently adopt sequential execution patterns, or to say workflow (Sapkota et al., 2025; Zhang et al., 2025), resulting in inefficient computation and high resource overhead (Kumar, 2025). Notably, increased computational resources do not always yield better performance, and multi-agent approaches may sometimes underperform compared to single-agent baselines (Zhang et al., 2023; Cemri et al., 2025; Sapkota et al., 2025).

To overcome these limitations, it is important to design general and efficient collaboration mechanisms among agents. A promising direction is to rethink the organization of both tasks and agent interactions to fully exploit the benefits of multi-agent collaboration. In this context, we identify that the divide-and-conquer paradigm—which systematically decomposes complex tasks and distributes subtasks among specialized agents—is fundamental for improving both efficiency and adaptability in multi-agent systems (Shoham and Leyton-Brown, 2008; Ferber and Weiss, 1999). Through structured decomposition, responsibilities are clearly assigned, concurrency are maximized, and emergent behaviors are managed more effectively, enabling scalable and robust multi-agent intelligence.

To this end, we propose AGENTGROUPCHAT-V2 , centered on three innovations: (1) a fully parallel architecture with hierarchical manager modules supporting scalable, distributed operation; (2) dynamic task-level divide-and-conquer via hierarchical task decomposition and dependency management; and (3) execution-level specialization

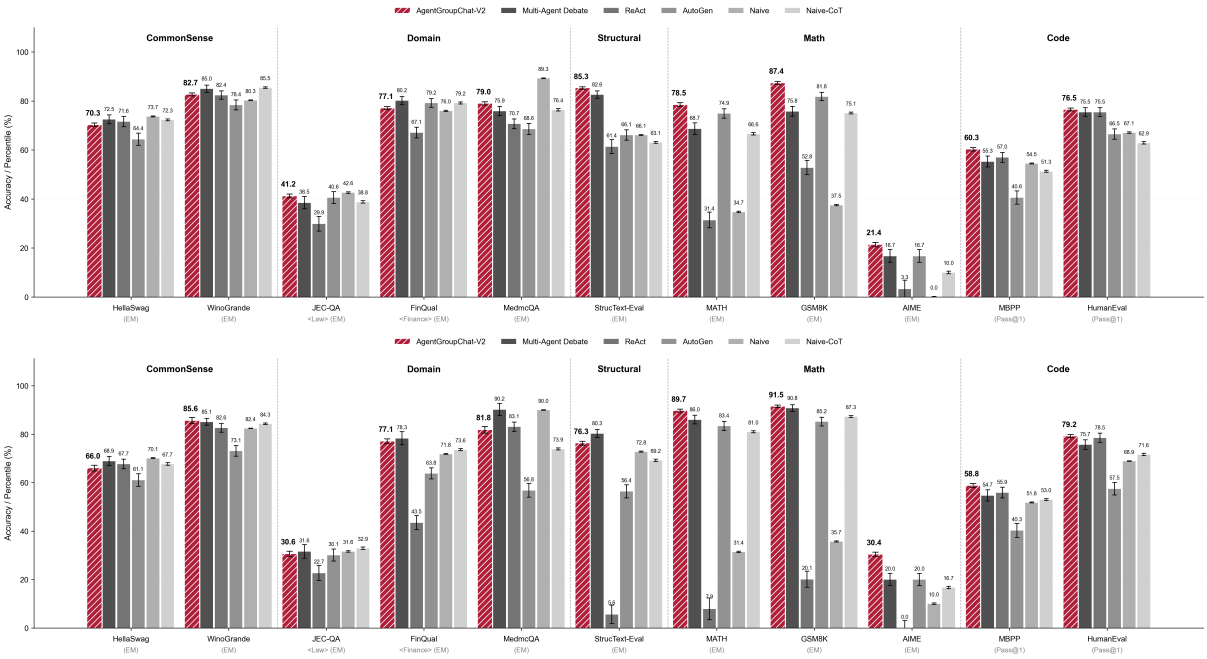

Figure 1: **Upper:** The performance of AGENTGROUPCHAT-V2 powered by Qwen2.5-72B. **Downer:** The performance of AGENTGROUPCHAT-V2 powered by Llama3.1-70B. Both models are evaluated across five diverse reasoning domains: commonsense reasoning, domain-specific knowledge, structural text understanding, mathematics, and code generation. AGENTGROUPCHAT-V2 consistently outperforms existing multi-agent approaches and baseline methods across both models and all benchmark categories, demonstrating the effectiveness, robustness and promising of multi-agent approach.

through adaptive role assignment to heterogeneous LLM agents. Our experiments confirm that AGENTGROUPCHAT-V2 significantly boosts performance and generalizability, especially on high-difficulty reasoning and generation tasks (see Figure 1).

2 Related Work

LLM-based multi-agent systems have demonstrated transformative potential in two key domains (Li et al., 2023; Wooldridge, 2009):

2.1 Social Dynamics Simulation

Recent advances integrate cognitive architectures with behavioral economics principles (Bates et al., 1994).Xie et al. (Xie et al., 2024a) and Han et al. (Han et al., 2023) develop game-theoretic frameworks capturing trust dynamics and market competition through iterative belief-updating mechanisms (Dafoe et al., 2021).Sociological simulations leverage event-driven architectures, with Park et al. (Park et al., 2023) modeling opinion evolution through social interaction cascades, while Zhang et al. (Zhang et al., 2024) employ hierarchical Bayesian networks for electoral behavior prediction (Lee et al., 2023). Emerging platforms like Gu's group chat simulator (Gu et al., 2024b) and Liu's rumor-control Twitter emulator (Liu et al., 2024) demonstrate practical applications in digital social dynamics (Gao et al., 2023).

2.2 Collaborative Problem-Solving

Cutting-edge systems employ structured debate protocols and knowledge fusion mechanisms (Du et al., 2023).Xiong's FORD framework (Xiong et al., 2023) enhances reasoning through tri-phase argumentation processes, complemented by Du's knowledge graph-based consensus formation (Du et al., 2023).Software engineering innovations like Qian's dialogue-driven development (Qian et al., 2024) and Hong's documentation-centric workflow (Hong et al., 2023) establish new paradigms for AI-assisted programming (Wang et al., 2023).Domain-specific implementations showcase methodological cross-pollination, from Sun's legal argumentation system with adversarial validation (Sun et al., 2024) to Yu's cognitive conflict-driven MOOC platform (Yu et al., 2024), demonstrating versatile problem-solving architectures (Ferber and Weiss, 1999).

Although impressive progress has been made in both social simulation and collaborative problem-solving, previous LLM-based multi-agent systems still lack solutions that are truly effective and broadly applicable for general-purpose tasks. In

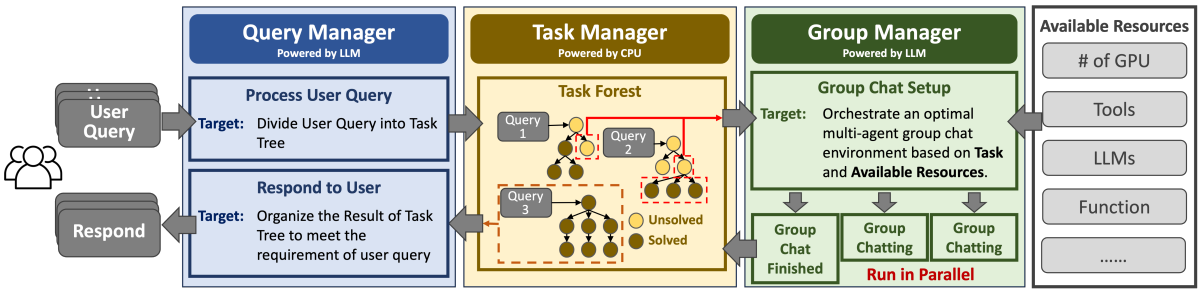

Figure 2: Illustration of AgentGroupChat-V2 framework, which composes of three main components: Query Manager, Task Manager and Group Manager. The framework illustrates the complete workflow from user query processing through task decomposition and management to multi-agent group chat execution, with arrows indicating data flow between components. Task Forest visualization demonstrates how queries are transformed into hierarchical task structures with solved (brown) and unsolved (yellow) nodes, while parallel group chats are carrying out in Group Manager.

contrast, the core insight of AgentGroupChat-V2 is to allow agents to autonomously adopt a divide-and-conquer paradigm—dynamically decomposing tasks and organizing cooperation according to the problem structure. This enables a flexible and scalable multi-agent system that can generalize across domains, addressing a key gap left by prior approaches.

3 Framework of AgentGroupChat-V2

This work introduces AgentGroupChat-V2 , featuring a modular architecture composed of three main components: *Query Manager*, *Task Manager*, and *Group Manager* (see Figure 2). This design enables flexible data exchange and function invocation through standardized interfaces, allowing each module to scale independently with computational demand and supporting horizontal system expansion.

3.1 Query Manager

The Query Manager serves as the system's frontend, handling user interaction and integrating LLMs as inference engines. It receives user queries, performs semantic analysis to decompose them into task tree structures, and forwards these to the Task Manager. After task processing is complete, the Query Manager collects and standardizes results, ensuring that responses accurately and appropriately address user requirements.

3.2 Task Manager

The Task Manager acts as the central coordination hub, managing overall task execution. Typically implemented as a single instance, it maintains a task forest representing multiple task trees derived from user input. The Task Manager tracks bidirectional relationships between tasks, forming a comprehensive dependency graph. It allocates and schedules tasks based on their structure and system resource availability, enabling parallel execution where possible. For hierarchical dependencies, it ensures information flow between child and parent tasks, and upon completion of all sub-tasks, returns consolidated results to the Query Manager.

3.3 Group Manager

The Group Manager is responsible for orchestrating multi-agent collaboration to execute assigned tasks. It manages the preparation and deployment of agent groups, including the selection of suitable LLMs, resource allocation, and scheduling. Multiple Group Manager instances can operate in parallel, each supervising independent agent groups, which allows the system to efficiently scale its computational resources according to demand.

4 Group Chat Design

This section details the organizational and implementation mechanisms of group chat in AgentGroupChat-V2 , the core of multi-agent collaboration and problem-solving. As illustrated in Figure 3, the system adopts a structured approach to both task management and agent group orchestration.

4.1 Task

In AgentGroupChat-V2 , each *Task* is distinctly defined by an identifier, description, parent and child relationships, and processing result. Root tasks are parentless, and leaf tasks have empty child sets. Tasks progress through several states: initialization (structure created), waiting (awaiting prerequisites), execution (delegated to Group Man-

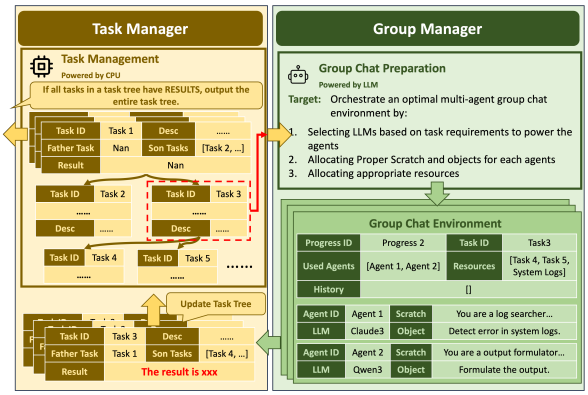

Figure 3: Detailed implementation of Task Manager and Group Manager modules showing: (1) Task Management structure with hierarchical organization of tasks, including Task ID, descriptions, parent-child relationships, and result tracking; (2) Group Chat Preparation process outlining agent selection criteria and resource allocation strategies; and (3) Group Chat Environment configuration displaying the agent infrastructure with Claude3 and Qwen3 LLMs assigned specific roles, workspace allocations, and specialized functions for systematic problem-solving through collaborative interactions.

ager), completion (result produced) or failure (error encountered). This structure forms hierarchical task trees that support flexible decomposition and dependency tracking.

4.2 Group

A *Group* represents a collaborative work unit managed by the Group Manager. Each group contains a progress ID, a linked task, assigned resources, and a set of agents. Agents within the group are specified by identity, LLM engine (e.g., Claude3, Qwen3), workspace (Scratch), assigned object (Object), and History. Scratch is for agent-internal computation, Object documents the role for interpretability, and History preserves conversational context for persistent and multi-turn collaboration. Groups are assembled with diverse agent roles, and their lifecycle includes preparation (configuration), activity (discussion), result integration, and termination.

4.3 Group Environment Configuration

Group environment configuration determines the initial state and capabilities of each group. This includes progress and task IDs, the list and order of agents, and shared resources. Group Manager orchestrates configuration, assigning agents and resources by analyzing the task needs and available LLMs. Each agent receives an appropriate identity, Scratch space, task Object, and an initialized History.

Algorithm 1 StartGroupChat (GroupManager)

Input: max_action_turn, agent_ids, initial_env
Output: final_env, task_result
env ← initial_env
for turn = 1 to max_action_turn **do**
  for current_agent in agent_ids **do**
    env ← current_agent.perceive(env)
    (message, target_agent) ← current_agent.decide_action(env)
    dialogue_history ← EXECUTEACTION(current_agent, env, message, target_agent)
    env ← UpdateEnvironment(dialogue_history, env)
  end for
  discussion_summary ← SummarizeGroupMessages(env)
  env ← UpdateEnvironment(discussion_summary, env)
  is_complete ← TaskManager.CheckTaskCompletion(env)
  if is_complete **then**
    return env, ExtractTaskResult(env)
  end if
end for
return env, ExtractTaskResult(env)

4.4 Group Chat Orchestration

Group chat proceeds according to Algorithm 1, coordinated by Group Manager over a series of action turns. At each turn, agents sequentially perceive the environment, generate messages, interact with others, and update the environment and dialogue history. After each round, the system summarizes the discussion and checks if the task has been completed; results can be output early upon successful completion.

4.5 Agent Interaction

Agent interactions (Algorithm 2) support both broadcast and pairwise communication. Upon receiving an action, an agent may broadcast to all members or engage in an alternating dialogue loop with a specific target agent until a stopping criterion is met.

4.6 Chat Results Processing

To ensure effective utilization of group discussions, the Group Manager summarizes dialogues, extracts key conclusions, and stores them in agents' memories to inform future turns. It also validates the quality and completeness of outcomes, and formats results for downstream consumption by the Task Manager.

Baseline	LLM	GSM8K	MATH					AIME
			Level1	Level2	Level3	Level4	Level5	(2024)
Naive	Qwen2.5-72B	37.52±1.2	60.17±1.8	49.41±1.5	37.16±1.3	26.49±1.1	21.78±0.9	0.0±0.0
	Llama-3.1-70B	35.70±1.4	50.44±1.6	44.11±1.7	30.97±1.2	28.20±1.3	19.84±0.8	10.0±2.8
Naive-CoT	Qwen2.5-72B	75.13±2.8	85.84±2.1	77.65±2.5	75.22±2.3	63.68±2.4	48.25±2.9	6.7±2.3
	Llama-3.1-70B	87.33±3.2	91.15±1.9	88.82±2.7	87.17±2.6	80.77±3.1	67.32±3.4	16.7±3.6
ReAct	Qwen2.5-72B	52.76±7.8	53.10±6.9	35.88±8.2	36.73±7.5	28.21±6.4	19.46±5.8	3.3±1.9
	Llama-3.1-70B	20.09±8.8	28.32±7.7	11.76±6.3	7.52±5.2	4.27±4.1	2.33±2.8	0.0±0.0
AutoGen	Qwen2.5-72B	81.80±6.7	89.38±5.8	85.29±6.2	**88.05±7.1**	**71.79±6.9**	54.86±8.2	16.7±4.2
	Llama-3.1-70B	85.21±7.8	96.46±5.2	92.94±6.6	91.15±6.8	82.48±7.3	66.93±8.7	20.0±4.8
Multi-Agent Debate	Qwen2.5-72B	75.81±4.9	85.84±4.2	79.41±5.1	75.66±4.7	66.24±5.3	52.14±5.8	13.3±3.5
	Llama-3.1-70B	90.82±5.2	97.35±3.8	92.94±4.6	**92.48±4.9**	**85.90±5.5**	71.98±5.7	20.0±4.1
Open Deep Research	Qwen2.5-72B	72.34±3.2	82.48±3.6	76.47±3.8	72.12±3.5	63.85±3.9	49.73±4.1	10.0±2.8
	Llama-3.1-70B	88.67±3.8	94.69±3.1	89.71±4.0	89.38±3.7	83.24±4.2	68.95±4.4	16.7±3.2
AgentGroupChat-V2	Qwen2.5-72B	**87.41±3.6**	**92.92±3.3**	**90.00±3.2**	84.07±4.1	**71.79±3.8**	**59.10±4.3**	**23.3±3.8**
	Llama-3.1-70B	**91.50±4.2**	**98.23±3.1**	**94.12±3.7**	88.94±4.8	81.20±4.1	**83.54±3.9**	**26.7±4.2**

Table 1: Experiment result on Math Problem.

Baseline	LLM	MBPP			HumanEval		
		pass@1	pass@3	pass@5	pass@1	pass@3	pass@5
Naive	Qwen2.5-72B	54.4±1.4	60.0±1.6	62.0±1.5	67.0±1.7	78.3±1.3	81.6±1.2
	Llama-3.1-70B	51.8±1.5	60.8±1.4	63.6±1.6	68.9±1.8	80.9±1.1	83.5±0.9
Naive-CoT	Qwen2.5-72B	51.3±2.8	61.5±2.4	65.0±2.6	62.8±2.9	78.6±2.2	82.4±2.1
	Llama-3.1-70B	53.0±3.1	60.5±2.7	62.7±2.8	71.5±3.2	81.2±2.5	83.5±1.9
ReAct	Qwen2.5-72B	57.0±6.8	**65.3±5.9**	**68.5±6.2**	75.5±5.7	**84.3±4.8**	**86.8±5.1**
	Llama-3.1-70B	55.8±7.2	62.3±6.4	64.8±6.7	78.5±6.1	**85.7±5.3**	**87.6±4.9**
AutoGen	Qwen2.5-72B	40.6±8.4	54.3±7.1	59.2±6.8	66.4±6.5	80.8±5.9	84.2±6.2
	Llama-3.1-70B	40.2±8.7	55.0±7.3	60.0±6.9	57.5±7.8	75.2±6.8	80.2±6.4
Multi-Agent Debate	Qwen2.5-72B	55.2±5.8	63.7±5.2	66.6±5.5	75.5±4.9	83.1±5.1	85.4±5.4
	Llama-3.1-70B	54.6±6.1	**63.0±5.6**	**65.7±5.8**	75.6±5.7	84.8±4.8	87.6±5.2
Open Deep Research	Qwen2.5-72B	53.1±4.2	61.4±3.8	64.2±4.1	73.8±3.6	81.4±3.9	83.7±4.0
	Llama-3.1-70B	52.9±4.4	60.8±4.1	63.4±4.3	74.2±4.2	82.9±3.7	85.8±3.8
AgentGroupChat-V2	Qwen2.5-72B	**60.3±3.8**	63.7±3.6	64.4±4.1	**76.4±3.9**	82.3±4.2	84.1±3.7
	Llama-3.1-70B	**58.8±4.2**	60.3±4.5	60.8±4.3	**79.2±3.6**	80.3±4.7	80.9±4.4

Table 2: Experiment result on Code Generation.

5 Experiment Setup

5.1 Task & Benchmark

Mathematical Reasoning: We evaluate on **GSM8K** (Cobbe et al., 2021), **MATH** (Hendrycks et al., 2021), and **AIME** (Art of Problem Solving, 2024). Accuracy on all math benchmarks is verified through symbolic answer equivalence (see Appendix A for detailed evaluation criteria). **Code Generation**: Benchmarks include **MBPP** (Austin et al., 2021) and **HumanEval** (Chen et al., 2021). Pass rate calculation and details on pass@k methodology are presented in Appendix A. **Domain-Specific Tasks**: We test financial reasoning (**FinQual** (Xie et al., 2024b)), legal QA (**JEC-QA**), and medical QA (**MedmcQA** (Pal et al., 2022)). Full descriptions of dataset construction are provided in Appendix B. **Structural Text Understanding**: **StrucText-Eval** (Gu et al., 2024a) assesses structured text processing (details in Appendix B). **Commonsense Reasoning**: **HellaSwag** (Zellers et al., 2019) and **WinoGrande** (Sakaguchi et al., 2021); see Appendix B for dataset details.

5.2 Baseline Methods

We consider five representative baselines: **Naive**, **Naive-CoT** (Wei et al., 2022), **ReAct** (Yao et al., 2022), **AutoGen** (Wu et al., 2023), **Multi-Agent Debate** (Liang et al., 2023) and **Open Deep Research** (Mavroudis, 2025). Implementation and protocol details can be found in Appendix C.

5.3 Large Language Models

All experiments use **Qwen2.5-72B-Instruct** and **Llama-3.1-70B-Instruct-Turbo**, with full model

Algorithm 2 EXECUTEACTION{current_agent, env, message, target_agent}

Input: current_agent, env, message, target_agent, max_chat_turn
Output: dialogue_history
turn_count ← 1
dialogue_history ← [ ]
if target_agent is AllGroupMembers **then**
  message ← GenerateMessage(current_agent, env, message, target_agent)
  Append (current_agent, target_agent, message) to dialogue_history
  return dialogue_history
end if
sender ← current_agent
receiver ← target_agent
while turn_count ¡ 2 · max_chat_turn **do**
  response ← GenerateResponse(receiver, dialogue_history, env)
  if not response **then**
    return dialogue_history
  end if
  Append (receiver, sender, response) to dialogue_history
  temp ← sender
  sender ← receiver
  receiver ← temp
  turn_count ← turn_count + 1
end while
return dialogue_history

usage and inference details available in Appendix D.

6 Experiment

6.1 Overall Performance

6.1.1 Mathematical Reasoning Analysis

Finding 1

AGENTGROUPCHAT-V2 demonstrates strong mathematical reasoning by leveraging effective task decomposition and agent specialization, particularly excelling on high-complexity problems.

Table 1 reports results across GSM8K, MATH, and AIME, where AGENTGROUPCHAT-V2 not only achieves the highest overall accuracy, but its advantages notably expand with problem complexity. For example, on the demanding AIME(2024), AGENTGROUPCHAT-V2 attains 30.4% accuracy (Llama-3.1-70B), nearly double that of the best baseline. The gap is less pronounced on simpler datasets (e.g., GSM8K), but becomes striking on the more layered MATH and in the progression from lower to higher levels within each benchmark. The marked improvement from Naive to Naive-CoT highlights the benefit of explicit stepwise reasoning, but only AGENTGROUPCHAT-V2 further leverages dedicated agent roles for effective divide-and-conquer. Importantly, as the diversity and interdependence of solution steps grow, single-agent and even loosely-structured frameworks stall, while specialized agents in AGENTGROUPCHAT-V2 coordinate to solve highly non-trivial subtasks in parallel.

6.1.2 Code Generation Analysis

Finding 2

Collaborative multi-agent analysis promotes rapid convergence to high-quality initial solutions in code generation, but performance improvements diminish for broader sampling, highlighting the need for task-specific exploration strategies.

As shown in Table 2, AGENTGROUPCHAT-V2 consistently delivers the best performance in pass@1 accuracy, achieving 79.20% on HumanEval and 76.46% on MBPP. These results suggest that multi-agent collaborative analysis quickly converges to high-quality first solutions, outperforming both naive and chain-of-thought approaches. For larger values of k (e.g., pass@5), ReAct and similar iterative frameworks gain ground, as their exploration strategy generates more diverse candidate programs. This confirms that while divide-and-conquer collaboration is ideal for tasks emphasizing correctness and logic in minimal samples, a more exploratory approach may be beneficial when breadth and coverage are prioritized.

6.1.3 Structural Text Understanding Analysis

Finding 3

AGENTGROUPCHAT-V2 robustly handles increasing structural complexity, maintaining higher accuracy as task depth and breadth grow.

Performance on StrucText-Eval (Table 3) further demonstrates AGENTGROUPCHAT-V2 ' s resilience to context and structure complexity. When both the width and depth of structured text expand, most methods—including sophisticated single-agent and multi-agent baselines—exhibit sharp performance degradation (sometimes ¿30% absolute drop). In contrast, AGENTGROUPCHAT-

Method	LLM	Width=1			Width=2			Width=3		
		Depth=1	Depth=2	Depth=3	Depth=1	Depth=2	Depth=3	Depth=1	Depth=2	Depth=3
Naive	Llama3.1-70B	78.5±1.2	77.2±1.4	72.8±1.6	69.3±1.5	63.5±1.3	54.2±1.1	66.8±1.4	53.3±1.2	42.7±0.9
	Qwen2.5-72B	81.9±1.1	80.7±1.3	77.9±1.5	72.5±1.4	68.0±1.2	55.4±1.0	70.3±1.3	56.8±1.1	45.0±0.8
Naive-CoT	Llama3.1-70B	**84.1±2.1**	**83.8±2.3**	76.2±2.8	**75.5±2.6**	**70.3±2.4**	53.7±2.9	**73.2±2.7**	57.1±3.1	43.5±2.2
	Qwen2.5-72B	**86.7±1.9**	**85.3±2.2**	**78.3±2.5**	**77.8±2.4**	**71.4±2.1**	54.8±3.2	**75.5±2.6**	55.2±2.8	42.3±2.0
ReAct	Llama3.1-70B	5.2±3.1	3.8±2.8	2.1±1.9	4.1±2.7	2.9±2.3	1.3±1.5	3.5±2.4	1.8±1.7	0.5±0.8
	Qwen2.5-72B	8.1±4.2	6.7±3.8	4.9±3.2	6.3±3.5	4.2±2.9	2.8±2.1	5.1±3.1	3.1±2.4	1.2±1.3
AutoGen	Llama3.1-70B	28.4±7.8	24.7±6.9	19.8±5.8	22.1±6.2	17.3±5.4	12.6±4.7	18.9±5.6	13.2±4.9	8.7±3.8
	Qwen2.5-72B	34.2±8.1	31.1±7.4	26.5±6.8	27.8±6.9	22.9±6.1	17.4±5.2	24.3±6.3	18.1±5.5	12.8±4.3
Multi-Agent Debate	Llama3.1-70B	83.3±5.2	82.2±5.6	74.8±6.1	74.1±5.8	68.9±5.4	52.2±6.7	72.4±5.9	54.7±6.3	40.3±5.1
	Qwen2.5-72B	82.1±4.9	81.5±5.3	75.2±5.7	76.3±5.5	67.6±5.8	53.8±6.2	71.2±5.6	55.4±6.1	41.7±4.8
Open Deep Research	Llama3.1-70B	81.8±3.6	80.4±3.8	73.2±4.1	72.9±3.9	67.2±4.0	50.8±3.7	70.6±4.2	52.9±3.5	38.9±3.8
	Qwen2.5-72B	80.7±3.4	79.8±3.7	73.8±4.0	74.8±3.8	66.1±3.9	52.4±3.6	69.5±4.1	53.7±3.3	40.2±3.5
AgentGroupChat-V2	Llama3.1-70B	81.7±3.8	80.8±4.1	**79.1±3.6**	73.6±4.2	69.2±3.9	**58.4±4.3**	72.8±4.0	**62.9±3.7**	**48.7±3.4**
	Qwen2.5-72B	83.5±3.6	82.9±3.9	77.3±4.0	76.7±3.8	70.8±4.1	**59.2±3.5**	73.6±3.7	**64.7±3.3**	**52.1±3.1**

Table 3: Exact Match (EM) Accuracy for StrucText-Eval.

Role Type	Role Setting
General Role	Agent-001 is a helpful assistant.
Specialized Role	Agent-001 is a error detection specialist focused on identifying calculation mistakes, logical inconsistencies, and reasoning flaws in mathematical solutions.
	Agent-002 is a logical reasoning specialist specialized in mathematical proof construction, step-by-step deduction, and logical argument validation.
	Agent-003 is a context comprehension specialist responsible for understanding problem statements, extracting key information, and summarizing lengthy mathematical contexts.
	Agent-004 is a computational specialist dedicated to performing accurate calculations, numerical analysis, and algebraic manipulations.
	Agent-005 is a solution verification specialist focused on checking final answers, validating solution paths, and ensuring mathematical correctness.

Table 4: Role configuration for multi-agent mathematical problem-solving collaboration.

V2 's divide-and-conquer strategy achieves the highest accuracy in the most challenging (width=3, depth=3) scenarios, showing only moderate loss with increased context. This underscores the benefits of explicit recursive decomposition and hierarchical collaboration for tasks involving layered, interdependent information.

Finding 4

Specialized agent roles combined with moderate-scale collaboration significantly outperform homogeneous teams, scaling positively with agent diversity and demonstrating that targeted division of labor avoids redundancy and amplifies collective reasoning.

6.2 Ablation Study

To elucidate the impact of team design and interaction strategy, we conduct systematic ablation experiments on MATH-100, varying agent count and dialogue rounds, and distinguishing between *general* and *specialized* role assignments, see Table 4 for definitions.

In AgentGroupChat-V2 with specialized roles (Fig. 5), accuracy grows robustly with team size: with 5 specialized agents and 3 dialogue rounds, accuracy peaks at 58%, a striking improvement of 64.6% over minimal settings. Notably, each additional agent consistently brings non-trivial gains, with collaboration between distinct expert perspectives enabling broader coverage and more rigorous validation of solution steps.

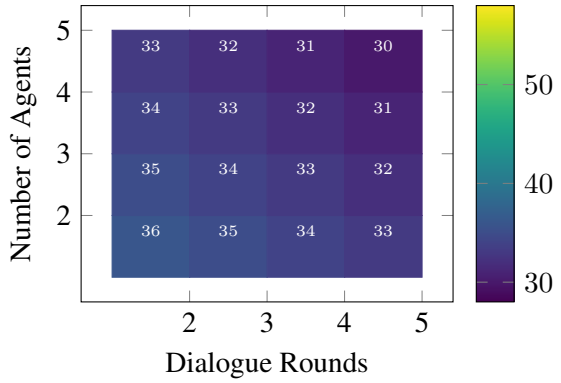

Figure 4: AGENTGROUPCHAT-V2 w/ General Role performance on MATH-100

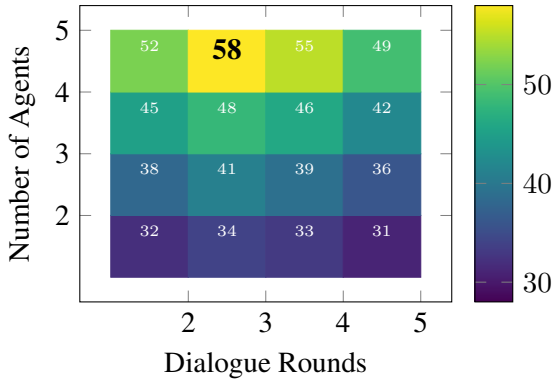

Figure 5: AGENTGROUPCHAT-V2 w/ Specified Role performance on MATH-100

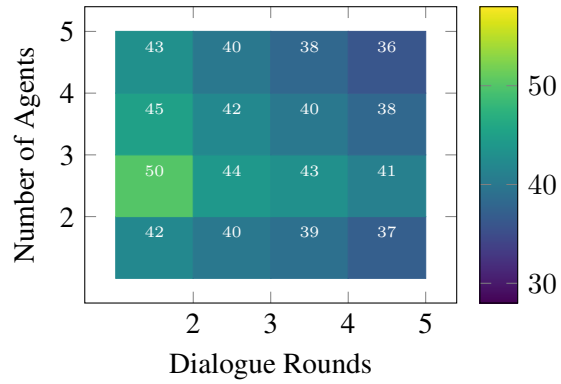

Figure 6: AutoGen performance on MATH-100

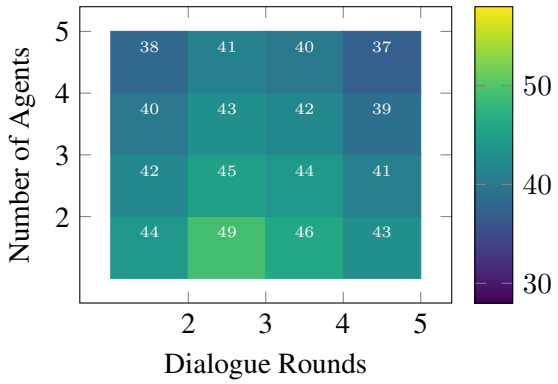

Figure 7: Multi-Agent Debate performance on MATH-100

Finding 5

Homogeneous (general) agent groups show stagnant or declining performance as either agent count or rounds increase, due to cumulative information redundancy and lack of complementary reasoning.

General-role configurations (Fig. 4) see accuracy plateau or degrade as more agents are added, with negligible improvement from increasing dialogue rounds. This effect highlights that, absent specialized functions, collaboration fails to create new insights and mostly produces redundant or even conflicting reasoning.

Finding 6

Traditional multi-agent frameworks (AutoGen, Debate) suffer diminishing returns or negative scaling at larger team sizes, illustrating the necessity of proactive role differentiation and dialogue orchestration to unlock benefits in large-scale collaborative problem solving.

AutoGen and Multi-Agent Debate (Figs. 6, 7) serve as external controls—both show early improvements with more agents or dialogue depth, but soon flatten or reverse, likely due to coordination overhead without sufficient specialization.

Further results. Experiments on **domain-specific QA** (Finance, Law, Medical), as well as commonsense reasoning (HellaSwag, WinoGrande), are provided in Appendix E. These results reveal that for tasks with low logical complexity or heavy reliance on external knowledge, the divide-and-conquer scheme may introduce unnecessary deliberation and overhead, and simpler strategies can be preferable. Rich qualitative **examples and discussion** of task decomposition can be found in Appendix F.

7 Conclusion

We introduces AGENTGROUPCHAT-V2 in this paper, a framework for LLM-based multi-agent systems featuring hierarchical task division and adaptive collaboration. Experiments show strong gains in reasoning and code generation, setting a new standard for complex multi-agent applications.

Limitations

AGENTGROUPCHAT-V2 , as a multi-agent system framework based on large language models (LLMs), has achieved notable progress in broad compatibility and efficient task collaboration. Nevertheless, our evaluations reveal two main limitations. First, experimental results indicate that for tasks heavily reliant on extensive external knowledge or with relatively low logical complexity (such as commonsense reasoning or domain-specific question answering), introducing multi-agent division of labor and parallel discussion may lead to unnecessary inference redundancy and increased computational overhead. In such cases, a single model invocation or a simple reasoning chain may outperform the multi-agent approach. Second, the system's performance is highly contingent upon the quality of task decomposition and the appropriateness of agent role assignment. For problems that are difficult to structurally decompose or involve highly coupled steps, the divide-and-conquer strategy does not necessarily yield solutions superior to those provided by a single agent.

Ethical Concerns

This study centers on algorithmic and system architectural design, with all evaluation tasks selected from publicly available and widely-used standard datasets. No sensitive personal information, simulated societal opinion, or real-world decision-making applications are involved. The multi-agent interactions are solely employed to enhance reasoning and problem-solving abilities, explicitly excluding the generation of misleading conclusions, group manipulation, or the deployment of autonomous systems with real-world operational authority. During the research process, neither the codebase nor experimental logs store any user privacy information; only aggregated evaluation data are released externally. Potential ethical risks, such as information bias or uncontrollable behavior emerging from automated collaboration, are primarily determined by the underlying LLM and task setup, rather than AGENTGROUPCHAT-V2 itself. All experiments are conducted within closed, secure environments. Should the framework be applied in scenarios involving human users or open environments in the future, further assessments and governance will strictly adhere to prevailing ethical guidelines in academia and industry.

References

Art of Problem Solving. 2024. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 2024-06-09.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Joseph Bates and 1 others. 1994. The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, and 1 others. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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*.

Allan Dafoe, Yoram Bachrach, Gillian Hadfield, Eric Horvitz, Kate Larson, and Thore Graepel. 2021. Cooperative ai: machines must learn to find common ground.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Jacques Ferber and Gerhard Weiss. 1999. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-wesley Reading.

Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. 2024. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *Humanities and Social Sciences Communications*, 11(1):1–24.

Chen Gao, Xiaochong Lan, Zhihong Lu, Jinzhu Mao, Jinghua Piao, Huandong Wang, Depeng Jin, and Yong Li. 2023. S3: Social-network simulation system with large language model-empowered agents. *arXiv preprint arXiv:2307.14984*.

Zhouhong Gu, Haoning Ye, Xingzhou Chen, Zeyang Zhou, Hongwei Feng, and Yanghua Xiao. 2024a. Structext-eval: Evaluating large language model's reasoning ability in structure-rich text. *arXiv preprint arXiv:2406.10621*.

Zhouhong Gu, Xiaoxuan Zhu, Haoran Guo, Lin Zhang, Yin Cai, Hao Shen, Jiangjie Chen, Zheyu Ye, Yifei Dai, Yan Gao, and 1 others. 2024b. Agentgroupchat: An interactive group chat simulacra for better eliciting emergent behavior. *arXiv preprint arXiv:2403.13433*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Xu Han, Zengqing Wu, and Chuan Xiao. 2023. "guinea pig trials" utilizing gpt: A novel smart agent-based modeling approach for studying firm competition and collusion. *arXiv preprint arXiv:2308.10974*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *Preprint*, arXiv:2402.02716.

Abhishek Kumar. 2025. Large language model based multi-agent system augmented complex event processing pipeline for internet of multimedia things. *arXiv preprint arXiv:2501.00906*.

Sanguk Lee, Tai-Quan Peng, Matthew H Goldberg, Seth A Rosenthal, John E Kotcher, Edward W Maibach, and Anthony Leiserowitz. 2023. Can large language models capture public opinion about global warming? an empirical assessment of algorithmic fidelity and bias. *arXiv preprint arXiv:2311.00217*.

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

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, and 1 others. 2024. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *Intelligent Computing*, 3:0063.

Yuhan Liu, Xiuying Chen, Xiaoqing Zhang, Xing Gao, Ji Zhang, and Rui Yan. 2024. From skepticism to acceptance: Simulating the attitude dynamics toward fake news. *arXiv preprint arXiv:2403.09498*.

Vasilios Mavroudis. 2025. Opendeepresearch.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. 2022. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, and 1 others. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Ranjan Sapkota, Konstantinos I Roumeliotis, and Manoj Karkee. 2025. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *arXiv preprint arXiv:2505.10468*.

Yoav Shoham and Kevin Leyton-Brown. 2008. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

Jingyun Sun, Chengxiao Dai, Zhongze Luo, Yangbo Chang, and Yang Li. 2024. Lawluo: A chinese law firm co-run by llm agents. *arXiv preprint arXiv:2407.16252*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

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. *Advances in neural information processing systems*, 35:24824–24837.

Michael Wooldridge. 2009. *An introduction to multiagent systems*. John wiley & sons.

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

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Chengxing Xie, Canyu Chen, Feiran Jia, Ziyu Ye, Kai Shu, Adel Bibi, Ziniu Hu, Philip Torr, Bernard Ghanem, and Guohao Li. 2024a. Can large language model agents simulate human trust behaviors? *arXiv preprint arXiv:2402.04559*.

Qianqian Xie, Weiguang Han, Zhengyu Chen, Ruoyu Xiang, Xiao Zhang, Yueru He, Mengxi Xiao, Dong Li, Yongfu Dai, Duanyu Feng, and 1 others. 2024b. Finben: A holistic financial benchmark for large language models. *Advances in Neural Information Processing Systems*, 37:95716–95743.

Kai Xiong, Xiao Ding, Yixin Cao, Ting Liu, and Bing Qin. 2023. Examining inter-consistency of large language models collaboration: An in-depth analysis via debate. *arXiv preprint arXiv:2305.11595*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Jifan Yu, Zheyuan Zhang, Daniel Zhang-li, Shangqing Tu, Zhanxin Hao, Rui Miao Li, Haoxuan Li, Yuanchun Wang, Hanming Li, Linlu Gong, and 1 others. 2024. From mooc to maic: Reshaping online teaching and learning through llm-driven agents. *arXiv preprint arXiv:2409.03512*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. 2025. Evoflow: Evolving diverse agentic workflows on the fly. *arXiv preprint arXiv:2502.07373*.

Xinnong Zhang, Jiayu Lin, Libo Sun, Weihong Qi, Yihang Yang, Yue Chen, Hanjia Lyu, Xinyi Mou, Siming Chen, Jiebo Luo, and 1 others. 2024. Electionsim: Massive population election simulation powered by large language model driven agents. *arXiv preprint arXiv:2410.20746*.

Yifan Zhang, Jingqin Wang, Jianye Yu, and Joey Tianyi Wen. 2023. Multi-agent reinforcement learning: A comprehensive survey. *arXiv preprint arXiv:2312.10256*.

A Details of Experiment Setup

A.1 Mathematical Reasoning Evaluation

For GSM8K, MATH, and AIME, we use symbolic equivalence to verify result accuracy. For AIME, exact match evaluation is employed on numerical answers.

A.2 Code Generation Evaluation Criteria

MBPP is evaluated by the unit-test pass rate: a solution is deemed correct if it passes all provided test cases.

HumanEval is assessed using the pass@k metric[1]:

$$\text{Pass@}k = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \tag{1}$$

where n is the total number of generated solutions and c is the number of correct solutions.

B Datasets

B.1 Mathematical Reasoning

GSM8K (Cobbe et al., 2021): 1,300 grade school math problems with step-by-step reasoning. **MATH** (Hendrycks et al., 2021): 12,000 problems covering algebra, geometry, combinatorics, etc., with five difficulty levels. **AIME** (Art of Problem Solving, 2024): Problems from the American Invitational Mathematics Examination; assesses complex mathematical reasoning.

B.2 Code Generation

MBPP (Austin et al., 2021): 500 Python problems, primarily algorithmic, each with multiple test cases. **HumanEval** (Chen et al., 2021): 164 Python function synthesis tasks, evaluated using random solution sampling (see Appendix A).

B.3 Domain-Specific Tasks

FinQual (Xie et al., 2024b): 1,000 finance-related questions from CFA and FinQA, including both MCQs and numerical tasks. **JEC-QA**: 26,365 legal exam items for the Chinese Bar exam, focusing

[1] As in (Chen et al., 2021), we use $n = 10$ samples per problem and report pass@1, pass@3, and pass@5.

on legal comprehension and scenario judgment. **MedmcQA** (Pal et al., 2022): 194,000 medical MCQs from Indian medical entrance exams.

B.4 Structural Text Understanding

StrucText-Eval (Gu et al., 2024a): Evaluates comprehension and manipulation of structured, hierarchical textual data.

B.5 Commonsense Reasoning

HellaSwag (Zellers et al., 2019): 70,000 multiple-choice questions; each instance offers four options for next-sentence prediction. **WinoGrande** (Sakaguchi et al., 2021): 44,000 pronoun disambiguation tasks to test contextual commonsense reasoning.

C Baselines

Naive: Each task is directly assigned to a single LLM, without explicit intermediate reasoning or decomposition.

Naive-CoT (Wei et al., 2022): Tasks are completed by a single LLM with chain-of-thought prompts, eliciting reasoning chains.

ReAct (Yao et al., 2022): Combines reasoning, action, and observation in a single-agent loop to iteratively decompose and solve tasks.

AutoGen (Wu et al., 2023): Multi-agent conversation framework combining AssistantAgent and UserProxyAgent, facilitating modular dialogue-based task solving.

Multi-Agent Debate (Liang et al., 2023): Multiple agents collaboratively analyze and debate a problem, iterating until consensus.

D Large Language Models

Our experiments employ **Qwen2.5-72B-Instruct** and **Llama-3.1-70B-Instruct-Turbo**, two leading open-source LLMs. Qwen is selected for its high performance on math and reasoning tasks, Llama-3.1 for its balance of capabilities and generation speed. All LLMs are run in inference mode with default decoding settings unless otherwise stated.

E Additional Experiments

E.1 Commonsense Reasoning

Table 5 presents experimental results on commonsense reasoning tasks (HellaSwag, WinoGrande), where direct model prompting or naive approaches often outperform collaborative frameworks due to the straightforward nature of the problems.

Baselines	LLM	HellaSwag	WinoGrande
Naive	Qwen2.5-72B	**73.7**	80.3
	Llama-3.1-70B	**70.1**	82.4
Naive-CoT	Qwen2.5-72B	72.3	**85.5**
	Llama-3.1-70B	67.7	84.3
ReAct	Qwen2.5-72B	71.6	82.4
	Llama-3.1-70B	67.7	82.6
AutoGen	Qwen2.5-72B	64.4	78.4
	Llama-3.1-70B	61.1	73.1
Multi-Agent Debate	Qwen2.5-72B	72.5	85.0
	Llama-3.1-70B	68.9	85.1
AGENTGROUPCHAT-V2	Qwen2.5-72B	70.3	82.7
	Llama-3.1-70B	66.0	**85.6**

Table 5: Experiment result on Commonsense Reasoning.

E.2 Domain-Specific Tasks

Table 6 reports results for financial, legal, and medical professional QA (FinQual, JEC-QA, MedmcQA). AGENTGROUPCHAT-V2 shows domain-adaptive effectiveness but is sometimes matched or surpassed by simpler competitive baselines on knowledge retrievalintensive tasks.

E.3 Task Manager Detailed Analysis

The task decomposition process is conducted autonomously by the LLM. Through our research on the Task Manager, we have found that the granularity of task decomposition has a significant impact on performance. Our empirical analysis reveals:

1. For complex tasks such as code generation and mathematical reasoning: we ask LLM to decompose the task into 5-7 subtasks. 2. For relatively simple tasks such as commonsense reasoning and knowledge retrieval: we ask LLM to decompose the task into 3-5 subtasks.

F Case Study and Qualitative Analysis

We provide illustrative cases on task tree decomposition and inter-agent group chat. For example, Figure 8 and 9 show typical hierarchical breakdowns in software engineering and document writing scenarios, with agent assignments detailed in each phase. Our group chat example demonstrates how specialized agents contribute distinct perspectives, requirement analysis, code, and review, progressively refining solutions within structured dialogue

Method	Model	JEC-QA (Law) EM Accuracy	FinQual (Finance) EM Accuracy	MedmcQA (Medical) EM Accuracy
Naive	Llama-3.1-70B	31.58	71.83	73.90
	Qwen2.5-72B	42.56	76.00	89.30
Naive-CoT	Llama-3.1-70B	32.93	73.60	89.97
	Qwen2.5-72B	38.80	79.20	76.40
ReAct	Llama-3.1-70B	22.71	43.50	83.10
	Qwen2.5-72B	29.92	67.10	70.70
AutoGen	Llama-3.1-70B	30.08	63.76	56.79
	Qwen2.5-72B	40.60	79.16	68.60
Multi-Agent Debate	Llama-3.1-70B	31.58	78.28	90.20
	Qwen2.5-72B	38.50	80.20	75.90
AGENTGROUPCHAT-V2	Llama-3.1-70B	30.62	77.09	81.82
	Qwen2.5-72B	41.20	77.11	79.00

Table 6: Experiment results on Law, Finance, and Medical multiple-choice QA tasks.

rounds. These findings highlight the effective orchestration and benefit of specialization in complex tasks.

And we provide prompt for Multi-Agent Discussion Environment Generation in Tab. 10 and Tab. 11.

Table 7: Agent Assignment for Interactive Data Visualization Tool Development Tasks

Task Name	Participating Agents
Module Interface Design	Requirement Analyst Agent, Code Implementation Agent, Code Review Agent
File Parsing Implementation	Requirement Analyst Agent, Code Implementation Agent, Code Review Agent
Data Processing Implementation	Requirement Analyst Agent, Code Implementation Agent, Code Review Agent
Data Display Implementation	Requirement Analyst Agent, Code Implementation Agent, Code Review Agent
Interactive Function Implementation	Requirement Analyst Agent, Code Implementation Agent, Code Review Agent
Testing and Verification	Test Planning Agent, Test Execution Agent, Quality Assurance Agent

Table 8: Agent Assignment for Blockchain Technology Analysis Article Writing Tasks

Task Name	Participating Agents
Technology Survey	Research Planning Agent, Research Execution Agent, Content Review Agent
Case Collection	Research Planning Agent, Research Execution Agent, Content Review Agent
Market Analysis	Research Planning Agent, Research Execution Agent, Content Review Agent
Writing Technology Analysis Chapters	Writing Planning Agent, Writing Execution Agent, Content Review Agent
Writing Market Analysis Chapters	Writing Planning Agent, Writing Execution Agent, Content Review Agent
Overall Optimization	Integration Planning Agent, Integration Execution Agent, Quality Assurance Agent

Table 9: Two-Round Agent Dialogue for File Parsing Task

<table>
<tr><th>Round</th><th>Agent</th><th>Dialogue Content</th></tr>
<tr><td rowspan="3">Round 1</td><td>Requirement Analyst Agent</td><td>Let me analyze this file parsing task... We need to implement a parsing function supporting CSV and Excel files. Core requirements include: 1) File format validation; 2) Automatic character encoding detection; 3) Data parsing with JSON output; 4) Large file handling capability... I suggest implementing the basic framework first.</td></tr>
<tr><td>Code Implementation Agent</td><td>

```
def parse_file(file_path):
    try:
        if file_path.endswith('.csv'):
            df = pd.read_csv(file_path)
        elif file_path.endswith('.xlsx'):
            df = pd.read_excel(file_path)
        else:
            return {"error": "Unsupported format"}
        # ... data processing logic
        return df.to_dict('records')
    except Exception as e:
        return {"error": str(e)}
```

</td></tr>
<tr><td>Code Review Agent</td><td>Reviewing the implementation... The basic functionality exists, but several issues need attention: 1) No file size limitation, potential memory overflow; 2) Encoding detection relies entirely on pandas defaults, may cause garbled text; 3) Exception handling too simplistic, poor user experience; 4) Missing data type inference capability... Requires substantial improvements.</td></tr>
<tr><td rowspan="3">Round 2</td><td>Requirement Analyst Agent</td><td>Based on the code review feedback... Adding specific requirements: 1) 100MB file size limit; 2) Support UTF-8, GBK, ASCII encoding auto-detection; 3) Categorized error handling with user-friendly messages; 4) Intelligent data type recognition for dates, numbers, and text; 5) Chunked processing mechanism for large files... This should provide comprehensive coverage.</td></tr>
<tr><td>Code Implementation Agent</td><td>

```
import chardet
import os
from typing import Callable, Dict, Any

class FileParser:
    def __init__(self, chunk_size=10000):
        self.chunk_size = chunk_size

    def parse_file(self, file_path: str) -> Dict[str, Any]:
        # File size validation
        if os.path.getsize(file_path) > 100*1024*1024:
            return {"error": "File exceeds 100MB limit"}

        # Enhanced encoding detection
        with open(file_path, 'rb') as f:
            raw_data = f.read(50000)
            detected = chardet.detect(raw_data)
            encoding = detected['encoding'] if detected['confidence'] > 0.7 else 'utf-8'

        try:
            if file_path.endswith('.csv'):
                # ... chunked CSV processing
                return self._parse_csv_chunked(file_path, encoding)
            elif file_path.endswith(('.xlsx', '.xls')):
                # ... Excel processing with compatibility
                return self._parse_excel(file_path)
        except Exception as e:
            return {"error": f"Parse error: {str(e)}", "type": type(e).__name__}

    def _parse_csv_chunked(self, file_path, encoding):
        # ... streaming processing implementation
        chunks = []
        for chunk in pd.read_csv(file_path, encoding=encoding, chunksize=self.chunk_size):
            chunks.append(chunk)
        df = pd.concat(chunks, ignore_index=True)
        return {"data": df.to_dict('records'), "dtypes": df.dtypes.to_dict()}
```

</td></tr>
<tr><td>Code Review Agent</td><td>Final review assessment...</td></tr>
</table>

!<INPUT_1<! —— The description of the task that the agent group needs to address.
!<INPUT_2>! —— A JSON object or structured text includes the current task's ID, its parent task, and the results of any completed sibling or child tasks that provide necessary context.
!<INPUT_3>! —— A list of available Large Language Models (LLMs) and their potential specializations (e.g., '["Claude-3-Opus (complex reasoning)", "Qwen-1.5-72B-Chat (strong coding)", "GPT-4 (general purpose)"]').
!<INPUT_4>! —— A JSON object detailing available tools, where each key is a tool name and the value is its description (e.g., '{"code_interpreter": "Executes Python code in a sandboxed environment.", "web_search": "Searches the web for information."}').
!<INPUT_5>! —— A JSON object detailing available functions, where each key is a function name and the value is its description (e.g., '{"format_output": "Formats the final result into a specific JSON structure.", "log_analyzer": "Analyzes system logs to detect errors."}').

<<<HERE IS THE SPLIT – BELOW IS THE PROMPT>>>

You are a 'Group Manager', an expert AI orchestrator within the AgentGroupChat-V2 framework. Your primary objective is to design and configure an optimal multi-agent group chat environment to solve a specific task. You will analyze the task, consider the available resources, and define a team of specialized agents, including their roles, assigned models, tools, and resource allocations.

1. Context and Resources

Here is the context for the task you need to plan for, along with the resources you have at your disposal.

Task Information
* **Current Task to Solve:** !<INPUT_1>!
* **Full Task Hierarchy & Context:** !<INPUT_2>!

Available Resources
* **Available LLMs:** !<INPUT_3>!
* **Available Tools:** !<INPUT_4>!
* **Available Functions:** !<INPUT_5>!

2. Your Instructions

Based on the information above, perform the following steps:

1. **Analyze the Task:** Carefully examine the 'Current Task to Solve'. Pay close attention to the 'Full Task Hierarchy & Context'. The results from completed sub-tasks are critical inputs or constraints for the current task. Identify the key challenges, required steps, and potential outputs.

2. **Design the Agent Team:** Devise a team of at least two collaborative AI agents. For each agent, you must define a specific, non-overlapping role that contributes to solving the task. Consider creating roles such as:
* **Project_Manager:** Oversees the entire process, breaks down the task, and ensures the final output meets the requirements.
* **Code_Generator:** Writes code to solve parts of the task.
* **Code_Tester / QA_Engineer:** Tests the generated code, identifies bugs, and suggests fixes.
* **Data_Analyst:** Processes and analyzes data.
* **Critic / Arguer:** Challenges the approaches and solutions proposed by other agents to improve robustness.

Table 10: Prompt of multi-agent discussion environment generation (Part1)

* **Executor:** Runs code or tools and reports the results.
* **Formatter:** Formats the final output according to specific instructions.
* **Researcher:** Uses search tools to gather necessary information.

3. **Allocate Resources Strategically:** For each agent you design, you must assign resources from the available pools.
* **LLM Selection:** Assign the most suitable LLM from ‘Available LLMs‘ to each agent based on its role. For complex reasoning or planning (like a Project_Manager), assign a more powerful model. For simpler, repetitive tasks, a smaller model might suffice.
* **Tool/Function Assignment:** Assign relevant ‘Tools‘ and ‘Functions‘ to each agent. An agent should only have access to the resources necessary for its role to ensure efficiency and security. For example, only the ‘Researcher‘ gets the ‘web_search‘ tool.

3. Output Format

You **MUST** provide your response as a single, well-formed JSON object. Do not include any explanatory text or markdown formatting outside of the JSON block. The JSON object should be an array named ‘group_chat_environment‘, where each element is an object representing one agent.

Follow this exact schema:

```json
{
”group_chat_environment”: [
{
”agent_id”: ”Agent_1”,
”role_description”: ”You are a [ROLE_NAME]. Your responsibilities are to [DETAILED_RESPONSIBILITIES_AND_PERSONA]. You must collaborate with other agents to achieve the main goal.”,
”llm_model”: ”model_name_from_input_3”,
”assigned_tools”: [
”tool_name_from_input_5”
],
”assigned_functions”: [
”function_name_from_input_6”
]
},
{
”agent_id”: ”Agent_2”,
”role_description”: ”You are a [ROLE_NAME]. Your primary function is to [DETAILED_RESPONSIBILITIES_AND_PERSONA]. You will interact with other agents by...”,
”llm_model”: ”model_name_from_input_3”,
”assigned_tools”: [],
”assigned_functions”: [
”function_name_from_input_6”
]
}
]
}
```

Table 11: Prompt of multi-agent discussion environment generation (Part2)

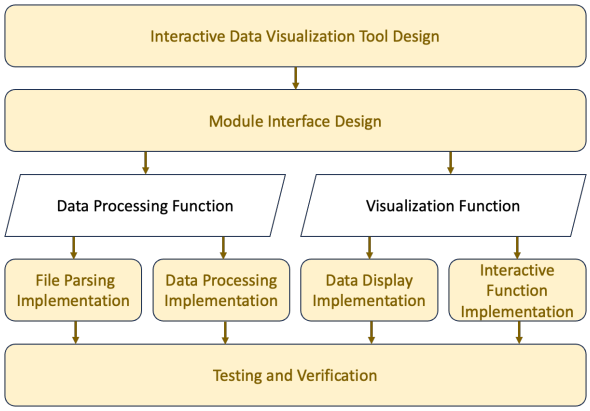

Figure 8: A schematic illustration of task decomposition for the "Interactive Data Visualization Tool Development" task by AGENTGROUPCHAT-V2 . Rectangular boxes represent specific executable tasks, while parallelograms denote annotation labels for task groupings. Arrow connections indicate dependency relationships between tasks, and all tasks not directly connected can be executed in parallel after the completion of their prerequisite tasks.

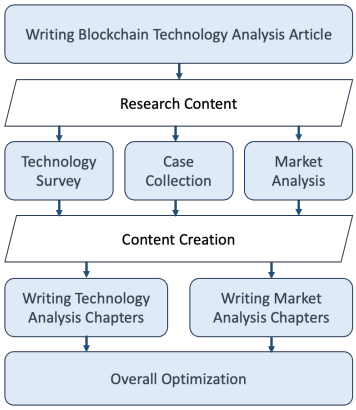

Figure 9: Another illustration of task decomposition for "Writing Blockchain Techonology Analysis Article" task by AGENTGROUPCHAT-V2