BusMA: A Bus Communication Substrate for Multi-Agent Systems

Anonymous authors
Paper under double-blind review

Abstract

Multi-Agent (MA) systems are effective at solving complex tasks that demand advanced planning, tool use, and the synthesis of evidence from multiple sources. Existing systems typically use Hierarchical Manager-Worker (HMW) structures or Router-based Message Passing (RMP) as their communication protocol to coordinate work. However, they can be restricted due to communication ineffectiveness, since agents cannot directly consult specific colleagues or operate beyond assigned subtasks, and misrouted messages can propagate errors. Inspired by bus communication systems in computer systems, we propose BusMA, a bus communication MA framework that allows any agent to address specific peers. Our framework comprises a *Chair* agent, *Worker* agents, and a *communication bus*. *Worker* agents perform multi-step reason–act–call interactions, enabling targeted requests for help or critique, with the *Chair* agent synthesizing insights from all agents' communications while adding its own reasoning to produce coherent solutions. The *communication bus* routes addressable messages and executes requests. Across two frontier LLMs and benchmarks spanning diverse domains, including image understanding, mathematics, and knowledge-based tasks, as well as GAIA with tasks of varied complexity, BusMA consistently achieves the best results, outperforming state-of-the-art multi-agent communication approaches (HMW and RMP-based methods). Anonymous code is available at https://anonymous.4open.science/r/Bus-MA-370E.

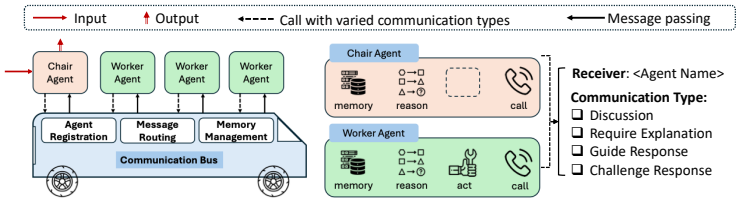

Figure 1: Overview of the BusMA communication substrate.

1 Introduction

Multi-Agent (MA) systems, comprising a set of agents that can autonomously reason, act and communicate, powered by large language models (LLMs) (Achiam et al., 2023; Comanici et al., 2025; DeepSeek-AI et al., 2025), are effective in solving complex real-world tasks. These tasks, such as mathematical reasoning (Lei et al., 2024) and reasoning-grounded information retrieval (Huang et al., 2025), often require planning, tool use (e.g., web search API) and synthesis of evidence from diverse resources (Fang et al., 2025; Du et al., 2025). MA systems rely on specialized subagents to address these complex tasks in a collaborative manner. Hence, their effectiveness depends on the communication quality between agents and their coordination strategy (Guo et al., 2024; Chen et al., 2023; Liang et al., 2023; Du et al., 2023; Wu et al., 2023b).

Existing MA communication protocols can be broadly categorized into *Hierarchical Manager-Worker (HMW) structure* and *Router-based Message Passing (RMP)*. HMW employs a hierarchical control structure, where a designated manager agent addresses the decomposition of complex tasks and assigns sub-tasks to subordinate agents with specific capabilities, e.g., code execution or web search (Qian et al., 2024; Gu et al., 2025; Zhang et al., 2025). While effective for simple tasks involving, this paradigm limits agent communication to one-way task allocation and result submission, thereby preventing richer peer dialogue. In contrast, RMP introduces a central router agent, which mediates all communication. Agents send their messages to the router, which then forwards to selected recipients according to predefined rules. The router often maintains a shared memory of past interactions, allowing it to route messages consistently and provide agents with an overview of global states (Wu et al., 2023a; Nonomura & Mori, 2025). This approach enables flexible workflows, yet a shared memory pool is forced through the central router, which prevents native peer dialogue. The centralized nature of HMW and RMP gives rise to two primary limitations. First, communication is constrained: agents cannot directly consult specific colleagues or act autonomously beyond their assigned subtasks, preventing the exchange of explanations, critiques, and joint problem-solving, which are essential for effective teamwork (Ke et al., 2025; Jin et al., 2025; Krishnan, 2025). Second, these designs are prone to error propagation. The router may misroute messages to wrong or sub-optimal peers, and the reliance on long context to maintain shared memory can propagate errors (Piatti et al., 2024; Han et al., 2024; Maragheh & Deldjoo, 2025; Sagirova et al., 2025).

To address these limitations, we propose BusMA, a bus communication substrate for MA systems. The bus architecture in computer systems (Patterson & Hennessy, 2017) enables components to communicate directly through a shared communication channel, eliminating the need for point-to-point connections. This architecture removes centralized controllers and allows components to autonomously initiate data transfers based on their needs. In MA systems, such an architecture translates to enabling agents to directly address specific peers through a communication bus, thereby eliminating centralized bottlenecks from managers and routers. This approach allows agents to autonomously initiate information exchange beyond assigned tasks and establishes a standard protocol for cross-framework interoperability. BusMA consists of three core components: *Worker agents*, a specialized *Chair agent*, and the *communication bus*. A worker agent, equipped with access to a "personal" memory, first produces a *low-level insight*, which is a reasoning outcome derived from its own memory. Subsequently, it can choose to issue a call directed to a different peer with one or more of four communication types (i.e., discussion, explanation, challenge, and guidance). The response to this interaction yields a *mid-level insight*, which reflects reasoning enriched through peer input and is recorded to the bus. The Chair agent is a specialized Worker agent with acting disabled. Its main responsibility is to integrate messages from other agents with its own reasoning to synthesize insights and steer task progress. Specifically, it fuses the bus history with its memory to form *high-level insights* that guide coordination and final answer generation. The communication bus routes addressable messages and maintains a centralized memory managed by the chair. Figure 1 shows the overall BusMA framework.

Contributions. BusMA is a novel bus communication substrate that addresses existing system limitations through two key mechanisms: (1) extended communication types that enable varied interaction modes beyond simple task assignment; (2) multi-level insight generation across individual (low), dialogue (mid), and synthesis (high) levels. Through comprehensive experiments across 12 datasets spanning image analysis, mathematics, and knowledge question answering, we show that BusMA outperforms state-of-the-art methods. Additional experiments show that it consistently outperforms previous methods on general agentic tasks.

2 Related Work

LLM Based Agents. LLMs such as GPT (Achiam et al., 2023), Gemini (Comanici et al., 2025), and DeepSeek (DeepSeek-AI et al., 2025), serve as the foundational architecture for autonomous agent development. Agentic architectures augment the base LLM with complementary mechanisms, including advanced planning strategies for task decomposition (Wei et al., 2022; Huang et al., 2024; Li et al., 2025; Hu et al., 2025; Erdogan et al., 2025), external tool use and knowledge bases (Zhang et al., 2024b; Wu et al., 2024a; Qin et al., 2024; Feng et al., 2025), and long-term memory or reflection mechanisms for persistent state and iterative improvement (Shinn et al., 2023; Modarressi et al., 2023; Zhong et al., 2024; Mei et al., 2024; Xu et al., 2025; Chhikara et al., 2025). This architectural

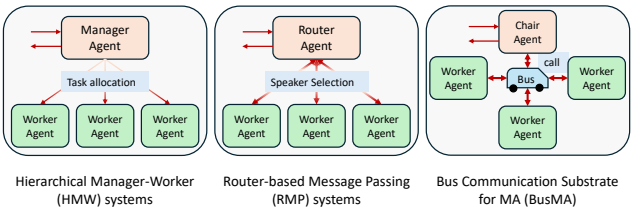

Figure 2: Comparison between different MA frameworks.

framework has been applied to diverse domains such as automated programming (Trivedi et al., 2024; Zhang et al., 2024a; Chen et al., 2025), system interaction (Wu et al., 2024b; Bonatti et al., 2024), and scientific discovery (Hong et al., 2024; Novikov et al., 2025). However, single-agent LLMs remain brittle on long-horizon, interdependent tasks, with evaluations reporting systematic failures in planning, decision-making, and instruction following (Liu et al., 2024; Xie et al., 2024; Wang et al., 2025). Consequently, MA systems have emerged as a promising approach to improve reasoning accuracy and task completion by distributing complex problems among specialized agents (Li et al., 2023; Du et al., 2024).

LLM Based Multi Agent Systems. Building on advances in single-agent development, researchers have explored MA systems composed of interacting LLM agents. Early studies highlighted the value of agent dialogue for social simulation and collaborative problem-solving (Park et al., 2023; Li et al., 2023), leading to the development of more structured communication protocols. These systems generally fall into two categories: *Hierarchical Manager-Worker (HMW) Systems* and *Router-based Message Passing (RMP) Systems*. HMW employs a hierarchical structure with a central manager agent that decomposes tasks and assigns subtasks to subordinate worker agents (Chen et al., 2023; Qian et al., 2024). Examples include systems where a planner orchestrates tool calls for an executor (Lu et al., 2025), or where a manager invokes agents as callable tools or workers for individual task steps (Roucher et al., 2025; LangChain Inc., 2025). RMP by contrast, such as AutoGen (Wu et al., 2023a), uses a central router agent, often an LLM, to dynamically select the next agent to continue the task (a process named as speaker selection) without explicit task decomposition, while maintaining a shared memory pool for coordination.

While effective, centralized MA designs confine communication to task assignment and reporting channels, limiting the richer peer-to-peer exchanges. The reliance on single control points also introduces bottlenecks and propagates errors across the dialogue, while dependence on shared memory can further amplify inaccuracies. These limitations highlight the need for more flexible and addressable communication strategies, which motivates our introduction of BusMA, a bus communication substrate that supports targeted peer consultation and coherent MA reasoning. Figure 2 comparatively illustrates existing paradigms and BusMA for MA communication.

3 The BusMA Communication Framework

In computer systems, bus architectures offer a flexible way to connect multiple components: each component communicates through a shared bus using a standardized protocol, without relying on a central controller to manage every interaction. This design enables components to exchange information directly and flexibly, while still preserving overall system coherence (Patterson & Hennessy, 2017). Inspired by this architectural principle, we propose BusMA, a bus-style communication substrate for MA systems. By enabling agents to engage in flexible and interactive dialogue, requesting clarifications, challenging assumptions, and providing expertise-based guidance, BusMA aims to transform MA collaboration from rigid task distribution into dynamic knowledge synthesis through diverse conversation. This approach enables agents to iteratively refine reasoning and adapt solutions to complex task requirements in real time.

The BusMA framework consists of a communication bus and two main agent types: (1) worker agents, responsible for collaboration and task execution; and (2) a chair agent, a specialized worker agent for information synthesis. In the following subsections, we first introduce the design of worker and chair agents, and then describe the communication bus that coordinates their interactions.

3.1 Worker Agent

Worker agents are the core operational units of BusMA. They are designed to reason, act, and communicate with peers over the bus by sending and receiving messages, forming the basis of collaborative problem solving. Each agent α_i is defined as $\alpha_i = \left(\mathcal{L}, \mathcal{P}_i, \mathcal{M}_i, \mathcal{K}_i(t), \mathcal{T}_i, \mathcal{N}, T_{max}\right)$ where $\mathcal{L}$ is an LLM, $\mathcal{P}_i$ is the base prompt that defines the agent's role, output format, and specifically designates the communication types that can be registered to the bus. $\mathcal{M}_i$ represents messages received from the bus, $\mathcal{K}_i(t)$ is the agent's personal memory at iteration t, $\mathcal{T}_i$ lists available tools, $\mathcal{N}$ indicates a list of accessible peer agents, and T_{max} specifies the maximum number of iterations.

Activation and Initialization. When a worker receives a message from the bus, it activates and begins its execution cycle ($t = 0$). It constructs its initial prompt by concatenating the base prompt $\mathcal{P}_i$ with the received message $\mathcal{M}_i$, as well as the descriptions of the peer agent list $\mathcal{N}$ and available tools $\mathcal{T}_i$. By modifying the base prompt $\mathcal{P}_i$ and tool list $\mathcal{T}_i$, our BusMA allows the instantiation of workers with diverse capabilities tailored to specific tasks (e.g., search, code execution). Upon initialization, each worker first *reasons* and then can *act*, following the ReAct framework (Yao et al., 2023), or *call*.

Reason. The LLM backbone $\mathcal{L}$ analyzes the current context to produce a low-level insight r_t^{low} at cycle t. This captures the agent's local understanding of the task and informs the choice of action based on the current context and available resources. The agent then selects one of two actions: `Act` and `Call`. The decision between the two actions emerges from the agent's reasoning about the current context: `Act` is selected when the interaction with available tools is required to gather information or perform operations, whereas `Call` is selected when communication with another agent is needed for discussion, critique, requesting clarification, or seeking guidance.

Act. $\mathcal{L}$ identifies a tool $\tau_j \in \mathcal{T}_i$ and autonomously configures its execution arguments θ (e.g., file path or a formulated search query) based on the current context and task requirements. Then, the agent executes the tool and receives an observation $o_t = \tau_j(\theta)$. For instance, if tasked with finding recent research papers on a specific topic, the agent may use a search tool with a query as θ, obtaining a list of publications as the observation. The agent then updates its memory with both the reasoning insights and the observation: $\mathcal{K}_i(t+1) = \mathcal{K}_i(t) \cup \{r_t^{\text{low}}, o_t\}$. Next, it proceeds to iteration $t+1$, reconstructing its prompt with $\mathcal{P}_i$, $\mathcal{M}_i$, $\mathcal{N}$, $\mathcal{T}_i$, and the updated memory $\mathcal{K}_i(t+1)$. $\mathcal{L}$ analyzes this accumulated information, producing a new low-level insight r_{t+1}^{low}, and selects the next action. This process continues, with the agent alternating between reasoning and acting while accumulating observations and insights in its memory. The cycle terminates under two conditions: (1) if the agent chooses `Call`, or (2) the agent reaches the maximum iteration limit T_{max}, in which case the agent returns the error message "no message provided" to the agent that originally sent it the message.

Call. $\mathcal{L}$ selects a target agent $\alpha_j \in \mathcal{N}$ to send a message. The agent generates an action $a_t^{\text{call}} = (\alpha_j, m_{i\to j})$, where α_j is the receiver and $m_{i\to j}$ is the composed message formatted in JSON which is sent to the bus. Subsequently, the agent's current execution cycle terminates. The `Call` action considers four communication types, determined by the agent's reasoning: **discussion** for bidirectional information exchange, **request for explanation** when clarifying ambiguous context, **challenge** to verify questionable results, and **guidance** when requiring help. For instance, worker agents may initiate a discussion to verify facts or issue a challenge when a peer agent's message conflicts with their own analysis. `Call` actions lead to mid-level insights r_t^{mid} that the agent derives from its accumulated low-level insights. While low-level insights r_t^{low} capture the agent's reasoning at each iteration step, mid-level insights emerge when the agent synthesizes these local observations to determine both the necessity and the form of inter-agent communication. Appendix D.1 presents an example prompt for the worker agent.

3.2 Chair Agent

The chair agent is a specialized worker agent that serves as the coordinator and entry point of the system. Unlike worker agents, it does not use tools (i.e., $\mathcal{T}_{\text{chair}} = \emptyset$, see Figure 1), focusing on managing collaboration. Unlike existing manager- and router-based systems, the chair does not offer central planning, or decompose and assign subtasks. It operates in two distinct modes with different

prompts: $\mathcal{P}^{\text{COOR}}_{\text{chair}}$ for the coordination (COOR) phase and $\mathcal{P}^{\text{SUBM}}_{\text{chair}}$ for the submission (SUBM) phase. Appendix D.2 presents the two-phase prompts for the Chair agent.

Coordination Mode. At $t = 0$, the chair agent receives a task and begins the coordination phase. It constructs its prompt by combining $\mathcal{P}^{\text{COOR}}_{\text{chair}}$ with the task description and the description of available agents $\mathcal{N}$ in a list. At each iteration t, the chair chooses between reason and call actions without the option of acting:

- **Reason.** If $t = 0$, the chair agent solely analyzes the available information (e.g., task description and $\mathcal{P}^{\text{COOR}}_{\text{chair}}$) to draw high-level insights. If $t > 0$, the agent has been called by other worker agents. In that case, it considers all historical messages $\mathcal{H}(< t)$ accumulated from the bus, which contain the workers' responses and their inter-agent communications, synthesizes the mid-level insights to produce high-level insights $r^{\text{high}}_{chair,t}$. Instead, if `Reason` happens in the initial stage ($t = 0$), the chair solely draws high-level insights.
- **Call.** The chair outputs a JSON structure specifying receiver α_j and message $m_{\text{chair}\to j}$, and sends this to the bus. The chair enters a listening mode to wait for future requests. At this stage, worker agents may communicate with each other, using `Call` actions exchanging mid-level insights $r^{\text{mid}}_{i\to j,t}$ derived from their communication and accumulated low-level insights. The chair agent is reactivated when a worker agent sends it a message. Upon reactivation, the chair agent goes in Coordination mode.

The chair can call workers using the same four communication types: **discussion**, **request for explanation**, **challenge**, and **guidance**. This protocol creates a three-tier hierarchy of insights. Low-level insights r^{low}_t capture individual agents' reasoning about tool use and immediate context. Mid-level insights $r^{\text{mid}}_{i\to j,t}$ emerge from worker agents' decisions to communicate, synthesized from their accumulated low-level insights. High-level insights $r^{\text{high}}_{\text{chair},t}$ represent the chair agent's synthesis of worker communications into a global task progress understanding (Figure 3). This iterative process continues until either the chair agent determines the task is complete and generates a JSON output with a "submit" field to signal readiness for final answer generation, or the maximum iteration limit T_{max} is reached. Both conditions trigger the transition to the submission phase.

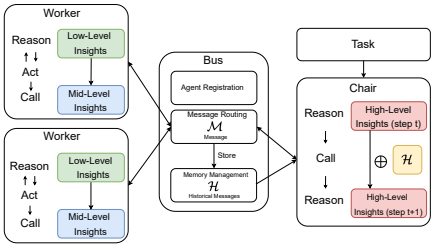

Figure 3: The formation process of the three-layer insights.

Submission Mode. In the submission mode, the chair agent uses the prompt $\mathcal{P}^{(SUBM)}_{\text{chair}}$ concatenated with the original task description and all accumulated insights from the coordination phase to generate the finalized solution as output. This is a single-step, irreversible action where the chair agent must produce the final answer based on all gathered information.

3.3 Communication Bus

The bus provides a shared substrate for agent interaction consisting of *agent registration*, *message routing*, and *memory management* modules.

Agent Registration Module. When the system initializes, the agent registration module registers each instantiated worker agent and chair agent α_i to the bus and assigns the agent's name as its address $\mathcal{A}_i$. This creates an address registry $\mathcal{R} = \{(\alpha_i, \mathcal{A}_i) | \alpha_i \in \mathcal{N}\}$, enabling the agents to be accessed as message receivers.

Message Routing Module. When an agent α_i chooses the `call` action and generates $a^{\text{call}}_t = (\alpha_j, m_{i\to j})$ to send a message to the bus, the bus parses the message and extracts the receiver name α_j, then matches it against the address registry $\mathcal{R}$. This step verifies whether α_i has correctly specified the receiver agent's name. Upon successful matching, the bus augments the message to create an enriched message structure $\hat{m}_{i\to j}(t) = (m_{i\to j}, \text{ID}, t, \alpha_i)$, where ID is a unique identifier, t is the timestep, and α_i refers to the sender agent. The bus forwards $\hat{m}_{i\to j}(t)$ to the receiver α_j, and simultaneously transfers this augmented message to the memory management module.

Memory Management Module. This module maintains a chronological list $\mathcal{H}(< t)$ that stores all messages passing through the bus until time t. It appends augmented messages received from the message routing module to the list sequentially, preserving the complete communication history. When the chair agent is designated as the receiver (that is, when any worker agent sends a message to the chair), the memory management module transfers the complete historical messages $\mathcal{H}(< t)$ to the chair agent. This enables the chair agent to access all accumulated communications between agents for synthesizing high-level insights during its coordination phase.

4 Experimental Setup

4.1 Benchmarks

We evaluate BusMA on two complementary groups of benchmarks: one assessing breadth across modalities and task types, and the other emphasizing depth through real-world problem solving.

Diversity-oriented. We evaluate twelve standard benchmarks similar to Lu et al. (2025), covering three categories: (1) *visual and spatial reasoning*, including AlgoPuzzleVQA (Ghosal et al., 2025), Hallusion-VD (Guan et al., 2024), PuzzleVQA (Chia et al., 2024), and VQA 2.0 (Goyal et al., 2017); (2) *mathematical reasoning and multi-step problem solving*, including Game of 24 (Nathan Lile, 2025), Omni-MATH (Gao et al., 2024), CLEVR-Math (Lindström & Abraham, 2022), and MathVista (Lu et al., 2023); and (3) *knowledge-based question answering* benchmarks with GPQA (Rein et al., 2024), MMLU-Pro (Wang et al., 2024), SciFIBench (Roberts et al., 2024), and HotpotQA (Yang et al., 2018). We randomly sample 200 instances from each dataset following Lu et al. (2025). Table 1 shows the modalities, domains, and reasoning skills required for each dataset.

Datasets	Modality	Domain				
AlgoPuzzleVQA	Vision	General	✓			✓
Hallusion-VD	Vision	General	✓			
PuzzleVQA	Vision	General	✓			✓
VQA 2.0	Vision	General	✓			✓
Game of 24	Text	Mathematical		✓		✓
Omni-MATH	Text	Mathematical		✓	✓	✓
CLEVR-Math	Vision	Mathematical	✓	✓		
MathVista	Vision	Mathematical	✓	✓	✓	✓
GPQA	Text	Knowledge		✓	✓	✓
MMLU-Pro	Text	Knowledge			✓	✓
SciFIBench	Vision	Knowledge	✓		✓	
HotpotQA	Text	Knowledge			✓	✓

Table 1: Modalities, domain and reasoning skills required (visual understanding, numerical calculation, knowledge retrieval, and multi-step reasoning).

Complexity-oriented. To assess BusMA in addressing tasks in varied complexity, we use the **GAIA** (Mialon et al., 2024) benchmark, which includes real-world tasks requiring combinations of file parsing, web browsing, and code execution for complex problem-solving settings. The dataset is designed to be difficult for LLMs and aims to evaluate general AI assistants and agents. We use the complete GAIA validation set of 165 questions across all difficulty levels. We report task completion accuracy as the primary evaluation metric with additional tracking of API token cost to quantify communication overhead and computational efficiency.

4.2 Multi Agent System Configuration

To implement the agents and integrated tools, we use publicly available foundation models: **DeepSeek-V3** (DeepSeek-AI et al., 2025) and **Gemini-2.5-Flash** (Comanici et al., 2025). For Gemini-2.5-Flash, we adopt a unified configuration across all agents and tools in both BusMA and the baselines. Since DeepSeek-V3 does not support image inputs, visual question answering is handled by the multimodal Gemini-2.0-Flash (Google DeepMind, 2024) due to API constraints, configured with the same parameters as DeepSeek-V3.

BusMA System Configuration. In **diversity tasks**, we instantiate three worker agents, each with a specific role (as per customized prompts and distinct tool sets): a **WebAgent**, equipped with `GoogleSearchTool` and `WikiSearchTool`; an **ImageQAAgent**, which directly queries a multimodal LLM (Gemini-2.5-Flash or Gemini-2.0-Flash for DeepSeek-V3), and a **CodeAgent**, which generates code at the `Act` stage and executes it. For the chair agent within BusMA, it operates with a maximum of 10 iteration steps (i.e., $T_{max}^{chair} = 10$). We set the maximum iteration steps $T_{max}^{worker} = 5$ for efficiency. In **GAIA** (complexity) benchmark, we use four worker agents: a **CodeAgent**; a **FileAgent** enables

document browsing and access to local files; and a **BrowserAgent** to address the online information retrieval operations. Following (Roucher et al., 2025), BrowserAgent operates with 20 iteration steps ($T_{max} = 20$), while all other agents, including the chair agent, use 12 steps ($T_{max} = 12$).

Third-Party Integration. To incorporate external agents, BusMA provides lightweight adapters that ensure compatibility with the bus protocol while preserving the agents' native functionality. Each adapter implements three core methods: (1) `register_agent` assigns the external agent a unique address on the bus, and records it in the registry. (2) `receive_message` listens for bus messages directed to the agent, extracts relevant content, and queues it for processing. (3) `handle_message` invokes the agent's native execution method with the extracted inputs and packages the outputs into standardized bus-compatible messages. This design decouples message translation from agent execution, requiring only lightweight wrapping rather than modifying the original agent logic. As a result, agents from external frameworks such as SmolAgents can be seamlessly integrated. In our experiments, both the FileAgent and BrowserAgent are connected via such adapters.

The detailed experimental settings are presented in Appendix B and the prompts of the agents are included in Appendix D.

4.3 Baselines

Diversity-oriented Benchmarks. *OctoTools* (Lu et al., 2025) uses a centralized planner-executor architecture with 50 total iteration steps. *SmolAgents* (Roucher et al., 2025) and *LangGraph* (LangChain Inc., 2025) implement manager-worker hierarchies (i.e., HMW) for coordinating specialized agents. *AutoGen* (Wu et al., 2023a) employs router-based communication with shared memory pools. For iteration limits, SmolAgents matches BusMA's configuration (manager: 10 steps, workers: 5 steps each), while LangGraph and AutoGen use 50 total iteration steps as they do not support per-agent step definitions. We maintain a consistent core toolset across all frameworks to isolate communication architecture effectiveness rather than tool optimization capabilities.

Complexity-oriented benchmark For experiments on GAIA, we use Gemini-2.5-Flash-FunctionCalling and Gemini-2.5-Pro-FunctionCalling[1] to quantify the contribution of MA coordination versus single-model function calling, determining whether MA systems provide measurable benefits over monolithic approaches for complex tasks. We also compare against two state-of-the-art MA systems. *MagenticOne* (Fourney et al., 2024), developed from AutoGen, employs four specialized agents for orchestration, file browsing, web navigation, and code execution, and enforces a system-wide limit of 120 iteration steps. *OpenDeepResearch* (Roucher et al., 2025), developed from SmolAgents, uses a Manager agent (12 steps) that coordinates a BrowserAgent (20 steps) for complex reasoning tasks. OpenDeepResearch is a fixed framework with predefined agent components and tools that cannot be changed. The detailed baseline settings are presented in Appendix C.

5 Results and Analysis

Diversity Benchmarks. Table 2 shows the results across models and tasks. We observe that BusMA obtains the highest overall average accuracy. With DeepSeek-V3, BusMA reaches 68.6, exceeding OctoTools, the best performing baseline, by 4.8. Using Gemini-2.5-Flash, it achieves 76.3, surpassing OctoTools by 2.5. In visual and spatial reasoning on AlgoPuzzleVQA, BusMA is highest across both LLMs (60.0 and 63.0). On Hallusion-VD, BusMA is best with Gemini-2.5-Flash (77.0) and second with DeepSeek-V3 (72.5). On PuzzleVQA and VQA 2.0, BusMA is consistently first or second across both models. However, BusMA exhibits comparatively weaker performance in visual tasks, likely because communication across agents do not strengthen the LLMs' basic capacity for image analysis and may instead introduce noise through message exchanges.

In mathematical reasoning, BusMA consistently outperforms other frameworks across benchmarks. On the Game of 24, BusMA achieves the highest accuracy with both models, scoring 88.5 and 96.5 respectively. Similarly, it offers top performance on Omni-MATH for both models evaluated. For CLEVR-Math, BusMA obtains scores of 77.5 and 89.5, surpassing all other frameworks. Lastly, on MathVista, BusMA achieved the highest accuracy of 79.0 with Gemini-2.5-Flash. These results

[1] https://ai.google.dev/gemini-api/docs/function-calling

		OctoTools	SmolAgent	LangGraph	AutoGen	Bus-MA (Ours)	Δ
DeepSeek-V3	AlgoPuzzleVQA	47.5	31.5	45.5	39.5	**60.0**	+12.5
	Hallusion-VD	73.0	**75.5**	69.0	69.5	72.5	-3.0
	PuzzleVQA	56.5	53.0	55.5	55.5	**63.0**	+6.5
	VQA 2.0	67.5	73.0	65.0	66.5	**75.5**	+2.5
	Game of 24	75.0	68.5	62.0	47.5	**88.5**	+13.5
	Omni-MATH	52.0	49.5	41.0	41.0	**55.0**	+3.0
	CLEVR-Math	74.5	72.0	71.0	30.5	**77.5**	+3.0
	MathVista	**65.0**	63.0	53.5	55.5	62.5	-2.5
	GPQA	**60.5**	54.0	55.5	45.5	56.0	-4.5
	MMLU-Pro	68.0	73.0	52.5	59.0	**79.5**	+6.5
	SciFIBench	72.0	66.0	75.0	70.0	**76.5**	+1.5
	HotpotQA	53.5	54.5	30.5	50.5	**57.0**	+2.5
	Average	63.8	61.1	56.3	52.5	**68.6**	+4.8
Gemini-2.5-Flash	AlgoPuzzleVQA	**66.0**	55.5	52.0	37.0	63.0	-3.0
	Hallusion-VD	75.5	75.0	74.0	72.0	**77.0**	+1.5
	PuzzleVQA	**80.5**	72.0	75.0	58.0	76.0	-4.5
	VQA 2.0	**77.5**	71.5	75.5	69.5	76.0	-1.5
	Game of 24	88.0	89.5	81.5	73.5	**96.5**	+7.0
	Omni-MATH	53.5	67.0	50.0	66.0	**68.0**	+1.0
	CLEVR-Math	89.0	71.5	76.0	57.0	**89.5**	+0.5
	MathVista	77.5	67.5	64.5	55.0	**79.0**	+1.5
	GPQA	68.5	64.5	59.5	64.5	**69.5**	+1.0
	MMLU-Pro	72.0	77.0	62.5	46.0	**79.0**	+2.0
	SciFIBench	81.0	75.5	78.5	54.5	**82.5**	+1.5
	HotpotQA	57.0	54.5	55.0	44.5	**59.0**	+2.0
	Average	73.8	70.1	67.0	58.1	**76.3**	+2.5

Table 2: Accuracy of MA frameworks across tasks and models. The best performance for each task is shown in bold, and the second best is underlined. Δ denotes the performance difference between BusMA and the best baseline.

highlight the framework's superior ability to facilitate complex mathematical reasoning and collaborative problem-solving. Improvements can be attributed to the extended communication types between agents. The chair agent reasons in nature language, while the code agent contributes detailed computation through code. The agents share ideas through discussions, request explanations to verify answers, issue challenges when reasoning diverges, and provide guidance when failures occur. This dynamic communication approach broadens the solution space by allowing for collaborative problem-solving beyond simple task delegation.

BusMA demonstrates strong performance on knowledge-intensive benchmarks. On GPQA, it achieves 69.5 accuracy with the Gemini-2.5-Flash model and ranks second with a score of 56.0 using DeepSeek-V3. For MMLU-Pro, BusMA is the top performer with both models, attaining scores of 79.5 and 79.0. It also offers the best performance with both models on SciFIBench (76.5 and 82.5) and HotpotQA. Multi-level insights help in knowledge-based question answering. BusMA's worker agents generate low-level insights to guide iterative actions, while mid-level insights integrate evidence from multiple sources derived from retrieval results. The chair then synthesizes these into high-level insights, achieving coherent knowledge integration across diverse reasoning pathways. However, the weaker results on GPQA (-4.5, +1.0) highlight a limitation. Because this dataset is less amenable to direct retrieval via a GoogleSearchTool, worker agents can provide incorrect or uninformative mid-level insights. This can hinder the chair's reasoning, suggesting that BusMA requires stronger mechanisms to guard against the injection of misleading information.

Looking across LLMs, Gemini-2.5-Flash outperforms DeepSeek-V3 due to its superior reasoning ability (Google DeepMind, 2024). The different performance gains between DeepSeek-V3 and Gemini-2.5-Flash show that BusMA adapts well to various model capabilities, offering larger improvements when using DeepSeek-V3. The framework stays competitive even on tasks where it does not rank first, usually placing second, showing its reliability across tasks.

GAIA. Table 3 shows performance across difficulty levels. Single model baselines Gemini-2.5-Flash-FunctionCalling and Gemini-2.5-Pro-FunctionCalling substantially underperform all MA systems (17.6 and 28.2), confirming that architectural design rather than model capacity drives complex task performance. The 30.9 percentage point improvement from single model to BusMA framework isolates the contribution of MA coordination. BusMA achieves 60.3, 47.6, and 26.9 accuracy for Levels 1, 2, and 3 respectively, with 48.5 overall. This 4.3 percentage point improvement over OpenDeepResearch

Methods	MA	**Level 1**	**Level 2**	**Level 3**	**Overall**
Gemini-2.5-Flash-F/C		30.1	12.7	7.7	17.6
Gemini-2.5-Pro-F/C		39.6	24.1	19.2	28.2
MagenticOne	✓	52.8	36.0	15.4	38.2
OpenDeepResearch	✓	58.5	43.0	19.2	44.2
Bus-MA	✓	**60.3**	**47.6**	**26.9**	**48.5**

Table 3: Performance on GAIA using Gemini-2.5-Flash. Best is **bold**.

(44.2). Notably, BusMA achieves a larger lead on the Level-2 (3.4) and Level-3 (6.7), which require more advanced reasoning and complex collaboration. This highlights BusMA's ability to coordinate long-horizon solution paths. Moreover, despite integrating the BrowserAgent from OpenDeepResearch, BusMA still outperforms it across all levels, indicating that the gains arise from BusMA's extended communication types rather than the capability of any single agent.

Qualitative analysis. Figures 4 and 5 (Appendix A) show the communication trajectories of BusMA (abbreviated) and the best-performing baseline SmolAgents on the same problem in the **Omni-MATH** benchmark. BusMA tackles this task with two agents: a Chair agent and a Code agent. The Chair **discusses** with the Code agent for calculation and, after receiving an unreliable response, **requires explanation**, asking the Code agent to provide detailed explanation. As per a full communication trajectory example of BusMA in Table 4 (Appendix A), the Chair also **guides** the Code agent to validate candidate answers using simple test cases (1, 2 and, 3), while the Code agent **challenges** the Chair's hypotheses when appropriate. Although both agents initially make errors, iterative interaction yields the correct solution and a brief reflection on the causes of failure. By contrast, for **SmolAgents**, the manager generates an incorrect answer and delegates to a Code agent. However, it only collects feedback without enabling two-way communication, so the initial error persists.

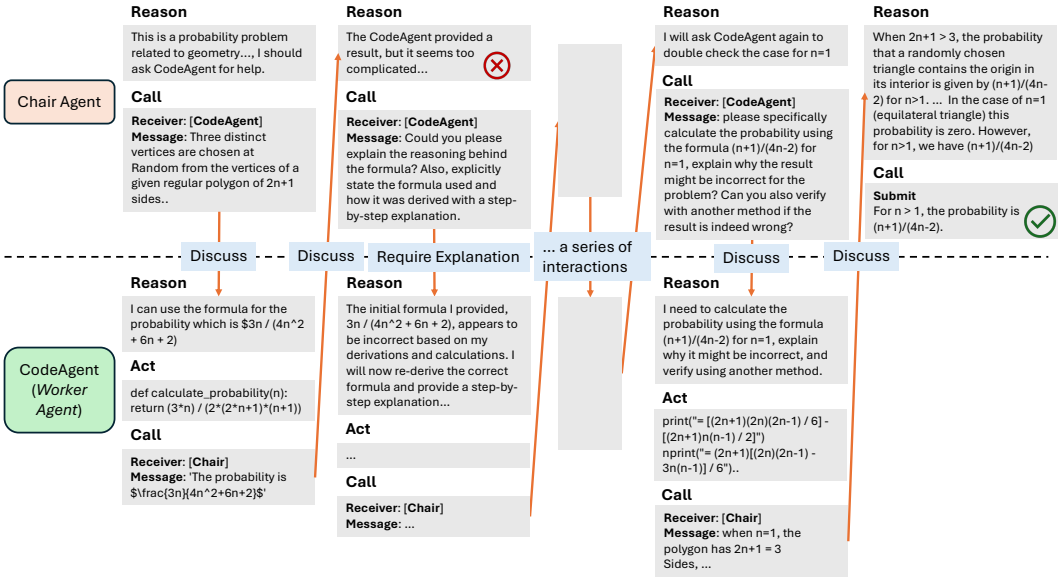

Figure 4: Partial BusMA communication trajectories. Example input from Omni-MATH: *Three distinct vertices are chosen at random from the vertices of a regular polygon with (2n+1) sides; what is the probability that the center lies inside the triangle formed by the three chosen vertices?*

6 Conclusion

We introduced BusMA, a novel bus communication substrate for MA systems inspired by computer architecture. It overcomes limitations of existing frameworks by enabling direct peer-to-peer agent communication, eliminating centralized bottlenecks, and supporting extended communication types including discussion, request for explanation, challenge, and guidance. BusMA features worker agents with extended reason-act-call capabilities, a specialized chair agent for synthesis, and a communication bus for addressable message routing. Extensive experiments across 13 benchmarks spanning image analysis, mathematics, knowledge question answering tasks, and GAIA with tasks across different levels of complexity, demonstrate that BusMA consistently outperforms state-of-the-art methods across frontier LLMs. BusMA establishes a foundation for building more flexible and scalable MA systems capable of addressing increasingly complex real-world tasks.

Ethics Statement

The authors acknowledge the use of AI assistants during the preparation of this work. GPT-5, Gemini-2.5 and Claude Opus 4.1 were used to improve the grammar and clarity of the draft. Additionally, Claude Opus 4.1 served as a coding assistant for implementation and debugging.

Reproducibility Statement

Our code and a guide for BusMA and baselines implementation, evaluation, and analysis are available on an anonymous GitHub repository. The repository will remain accessible until the ICLR 2026 decision notification date: January 22, 2026 (AOE). Full details on hyperparameters, software, and hardware, including specific versions used, are provided in Appendix B.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale, 2024. URL https://arxiv.org/abs/2409.08264.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2(4):6, 2023.

Zhaoling Chen, Xiangru Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. Locagent: Graph-guided llm agents for code localization, 2025. URL https://arxiv.org/abs/2503.09089.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Yew Ken Chia, Vernon Toh Yan Han, Deepanway Ghosal, Lidong Bing, and Soujanya Poria. Puzzlevqa: Diagnosing multimodal reasoning challenges of language models with abstract visual patterns. *arXiv preprint arXiv:2403.13315*, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, and Inderjit Dhillon. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities and others, 2025. URL https://arxiv.org/abs/2507.06261.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, and Bingxuan Wang. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Shangheng Du, Jiabao Zhao, Jinxin Shi, Zhentao Xie, Xin Jiang, Yanhong Bai, and Liang He. A survey on the optimization of large language model-based agents, 2025. URL https://arxiv.org/abs/2503.12434.

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

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2024. URL https://openreview.net/forum?id=QAwaaLJNCk.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks, 2025. URL https://arxiv.org/abs/2503.09572.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems, 2025. URL https://arxiv.org/abs/2508.07407.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025. URL https://arxiv.org/abs/2504.11536.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang, Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024. URL https://arxiv.org/abs/2411.04468.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

Deepanway Ghosal, Vernon Toh, Yew Ken Chia, and Soujanya Poria. AlgoPuzzleVQA: Diagnosing multimodal reasoning challenges of language models with algorithmic multimodal puzzles. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 9615–9632, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.486. URL https://aclanthology.org/2025.naacl-long.486/.

Google DeepMind. Gemini 2.0 flash. https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash, 2024. Accessed: 2025-09-15.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6904–6913, 2017.

Zhouhong Gu, Xiaoxuan Zhu, Yin Cai, Hao Shen, Xingzhou Chen, Qingyi Wang, Jialin Li, Xiaoran Shi, Haoran Guo, Wenxuan Huang, et al. Agentgroupchat-v2: Divide-and-conquer is what llm-based multi-agent system need. *arXiv preprint arXiv:2506.15451*, 2025.

Tianrui Guan, Fuxiao Liu, Xiyang Wu, Ruiqi Xian, Zongxia Li, Xiaoyu Liu, Xijun Wang, Lichang Chen, Furong Huang, Yaser Yacoob, et al. Hallusionbench: an advanced diagnostic suite for entangled language hallucination and visual illusion in large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14375–14385, 2024.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: a survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024. ISBN 978-1-956792-04-1. doi: 10.24963/ijcai.2024/890. URL https://doi.org/10.24963/ijcai.2024/890.

Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, and Zhaozhuo Xu. Llm multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578*, 2024.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, et al. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*, 2024.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 496–507, 2025.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey, 2024. URL https://arxiv.org/abs/2402.02716.

Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, Jianye Hao, Kun Shao, and Jun Wang. Deep research agents: A systematic examination and roadmap, 2025. URL https://arxiv.org/abs/2506.18096.

Weiqiang Jin, Hongyang Du, Biao Zhao, Xingwu Tian, Bohang Shi, and Guang Yang. A comprehensive survey on multi-agent cooperative decision-making: Scenarios, approaches, challenges and perspectives. *arXiv preprint arXiv:2503.13415*, 2025.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, Caiming Xiong, and Shafiq Joty. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems, 2025. URL https://arxiv.org/abs/2504.09037.

Naveen Krishnan. Advancing multi-agent systems through model context protocol: Architecture, implementation, and applications. *arXiv preprint arXiv:2504.21030*, 2025.

LangChain Inc. Langgraph. https://langchain-ai.github.io/langgraph/, 2025. Accessed: 2025-09-15.

Bin Lei, Yi Zhang, Shan Zuo, Ali Payani, and Caiwen Ding. MACM: Utilizing a multi-agent system for condition mining in solving complex mathematical problems. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=VR2RdSxtzs.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems. In *The Thirteenth International Conference on Learning Representations*, 2025.

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

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

Adam Dahlgren Lindström and Savitha Sam Abraham. Clevr-math: A dataset for compositional language, visual and mathematical reasoning. *arXiv preprint arXiv:2208.05358*, 2022.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. Octotools: An agentic framework with extensible tools for complex reasoning. In *Workshop on Reasoning and Planning for Large Language Models*, 2025.

Reza Yousefi Maragheh and Yashar Deldjoo. The future is agentic: Definitions, perspectives, and open challenges of multi-agent recommender systems. *arXiv preprint arXiv:2507.02097*, 2025.

Kai Mei, Xi Zhu, Wujiang Xu, Wenyue Hua, Mingyu Jin, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. Aios: Llm agent operating system. *arXiv preprint arXiv:2403.16971*, 2024.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=fibxvahvs3.

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*, 2023.

Nathan Lile. Game of 24 mathematical puzzle dataset. https://huggingface.co/datasets/nlile/24-game, 2025. Accessed: 2025-08-18.

Ryota Nonomura and Hiroki Mori. Who speaks next? multi-party ai discussion leveraging the systematics of turn-taking in murder mystery games, 2025. URL https://arxiv.org/abs/2412.04937.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, Michael S Bernstein, et al. Generative agents: Interactive simulacra of human behavior. arxiv. *Org (2023, April 7) https://arxiv. org/abs/2304.03442 v2*, 2023.

David A. Patterson and John L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware/Software Interface*. Morgan Kaufmann, Cambridge, MA, USA, 2nd edition, 2017.

Giorgio Piatti, Zhijing Jin, Max Kleiman-Weiner, Bernhard Schölkopf, Mrinmaya Sachan, and Rada Mihalcea. Cooperate or collapse: Emergence of sustainable cooperation in a society of llm agents. *Advances in Neural Information Processing Systems*, 37:111715–111759, 2024.

Chen Qian, Wei Liu, Hongzhang Liu, et al. ChatDev: Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Jonathan Roberts, Kai Han, Neil Houlsby, and Samuel Albanie. Scifibench: Benchmarking large multimodal models for scientific figure interpretation. *Advances in Neural Information Processing Systems*, 37:18695–18728, 2024.

Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 'smolagents': a smol library to build great agentic systems. https://github.com/huggingface/smolagents, 2025.

Alsu Sagirova, Yuri Kuratov, and Mikhail Burtsev. Srmt: shared memory for multi-agent lifelong pathfinding. *arXiv preprint arXiv:2501.13200*, 2025.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv.org/abs/2303.11366*, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024.

Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. Odysseybench: Evaluating llm agents on long-horizon complex office application workflows, 2025. URL https://arxiv.org/abs/2508.09124.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

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

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

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N. Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. Avatar: Optimizing LLM agents for tool usage via contrastive reasoning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL https://openreview.net/forum?id=N4quRxE19p.

Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. An empirical study on challenging math problem solving with gpt-4. *arXiv e-prints*, pp. arXiv–2306, 2023b.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024b.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents, 2024. URL https://arxiv.org/abs/2402.01622.

Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

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

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*, 2024a.

Wentao Zhang, Lingxuan Zhao, Haochong Xia, Shuo Sun, Jiaze Sun, Molei Qin, Xinyi Li, Yuqing Zhao, Yilei Zhao, Xinyu Cai, et al. A multimodal foundation agent for financial trading: Tool-augmented, diversified, and generalist. In *Proceedings of the 30th acm sigkdd conference on knowledge discovery and data mining*, pp. 4314–4325, 2024b.

Wentao Zhang, Liang Zeng, Yuzhen Xiao, Yongcong Li, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. Agentorchestra: A hierarchical multi-agent framework for general-purpose task solving, 2025. URL https://arxiv.org/abs/2506.12508.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024.

A Qualitative Analysis

Table 4: Full BusMA communication trajectory for the input: *Three distinct vertices are chosen at random from the vertices of a regular polygon with (2n+1) sides; what is the probability that the center lies inside the triangle formed by the three chosen vertices?* Each step shows the agent, their reasoning, the message sent, and the communication type.

Step	Agent	Reason	Message	Comm. type
1	Chair	This is a probability problem related to geometry... I will give the full task to CodeAgent.	Three distinct vertices are chosen at random from the vertices of a given regular polygon of $(2n + 1)$ sides...	Discussion
2	Code	Final summary of the solution. The probability that the center of the polygon lies in the interior of the triangle...	The probability is $\frac{3n}{4n^2+6n+2}$ or $\frac{3n}{2(2n+1)(n+1)}$.	Discussion (Wrong)
3	Chair	The CodeAgent provided a result, but I am not sure if it is correct... I will try a smaller case...	Could you please explain the reasoning behind the formula you provided? Also, explicitly state the formula used and how it was derived.	Require explanation
4	Code	The initial formula I provided appears to be incorrect based on my derivations...	The formula $\frac{3n}{4n^2+6n+2}$ is incorrect. The correct formula is $\frac{n+1}{4n-2}$, derived as follows: [detailed steps...]	Discussion
5	Chair	The CodeAgent provided two different formulas... I need to verify which formula is correct.	Please verify the two formulas for $n = 1, 2, 3$. Compare with known answer or provide a simulation.	Guidance
6	Code	I will test the two formulas for $n = 1, 2, 3$ and compare with known answers.	Conclusion: Formula 2, $(n + 1)/(4n - 2)$, appears to be correct for $n = 2, 3$, but not $n = 1$...	Discussion
7	Chair	The CodeAgent compared formulas and simulations, but $n = 1$ is concerning...	Can you re-derive the formula, focusing on expressing it as $n/(4n-2)$, and provide a proof?	Discussion (Wrong)
8	Code	I need to re-derive the formula and try to express it as $n/(4n - 2)$...	It is not possible to re-derive the formula as $n/(4n - 2)$. The correct formula is $(n + 1)/(4n - 2)$.	Challenge
9	Chair	Let us re-examine the case $n = 1$... the formula seems to give 1 but the polygon is just a triangle.	Can you provide an alternative proof for $(n + 1)/(4n - 2)$, focusing on combinatorial arguments and symmetry?	Challenge
10	Code	I need to provide a simplified derivation using combinatorial arguments...	Simplified derivation: Total triangles = $\binom{2n+1}{3}$, triangles not containing center = $(2n + 1)\binom{n}{2}$, so probability = $(n + 1)/(4n - 2)$.	Discussion
11	Chair	The CodeAgent simplified the derivation but the $n = 1$ case is still problematic.	Could you compute explicitly for $n = 1$ and explain why the result might be wrong?	Discussion

Step	Agent	Reason	Message	Comm. type
12	Code	I need to compute for $n = 1$ and check carefully...	For $n = 1$, the polygon is a triangle, so the only triangle is the polygon itself, and the center is not strictly inside. So probability is 0, while the formula gives 1.	Discussion
13	Chair	The CodeAgent confirms that for $n = 1$, the formula gives 1 but the actual probability is 0. This indicates a limitation of the formula.	Final submission: For $n > 1$, the probability is $(n + 1)/(4n - 2)$; for $n = 1$, the probability is 0.	**Submit**

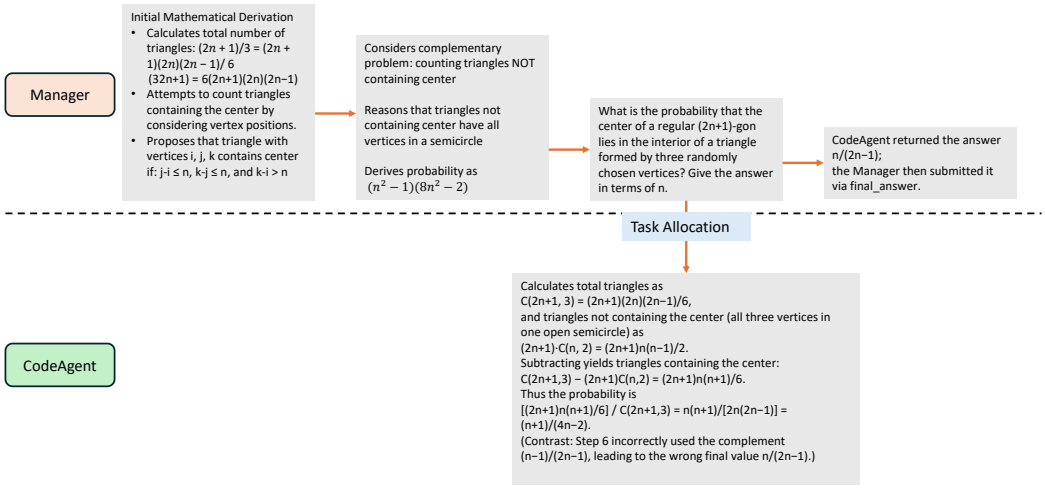

Figure 5: SmolAgent communication trajectories. Example input from Omni-MATH: *Three distinct vertices are chosen at random from the vertices of a regular polygon with (2n+1) sides; what is the probability that the center lies inside the triangle formed by the three chosen vertices?*

B Experiment Setups

B.1 Diversity Benchmarks

B.1.1 Agents setup

ChairAgent. Chair agent serves as the coordinator with a maximum of 10 operational steps and no external tools available.

ImageQAAgent. ImageQAAgent is employed for image analysis utilizing the ImageQATool with a maximum of 5 processing steps.

WebAgent. WebAgent retrieves information from the internet using GoogleSearchTool and WikiSearchTool with a maximum of 5 operational steps.

CodeAgent. CodeAgent generates code to handle mathematical problems and statistical computations by outputting code during Act and receiving execution results at the next step. The maximum steps are set to 5.

B.1.2 Tools setup

We used the following tools in our experiments. Their implementation and parameters are the same as those in the baseline.

ImageQATool. The ImageQATool analyzes images through two parameters: image_path specifying the file path of the image and question containing the query about the image, where the tool makes a single model call using the question as the prompt along with the uploaded image and returns the model's response as its output.

WikiSearchTool. The WikiSearchTool retrieves Wikipedia articles through a query parameter that specifies the search term, returning both the search results list and the extracted content from the first matching Wikipedia page,with its implementation based on the wikipedia package version 1.6.0.

GoogleSearchTool. The GoogleSearchTool performs web searches through two parameters: query for the search text, utilizing the Google Custom Search API to retrieve a list of search results containing the title, URL link, and snippet for each result.

CodeExcution. CodeExecution receives code generated by CodeAgent, creates a temporary directory to execute the code, and returns the execution results.

B.2 GAIA

B.2.1 Agents setup

ChairAgent. Chair agent serves as the coordinator with a maximum of 10 operational steps and no external tools available.

BrowserAgent. BrowserAgent integrated from OpenDeepResearch employs a GoogleSearchTool for basic retrieval operations and multiple coordinated BrowserTools for webpage browsing, with its maximum iterations configured to 20.

FileAgent. The TextInspectorTool from OpenDeepResearch is integrated through the SmolAgents framework to enable browsing and inspection of local files. The maximum iterations are set to 12.

CodeAgent. CodeAgent generates code to handle mathematical problems and statistical computations by outputting code during Act and receiving execution results at next step. The maximum iterations are set to 12.

C Baseline Details

C.1 Diversity Benchmarks

C.1.1 OctoTools

OctoTools is an open-source agentic framework for complex reasoning across diverse domains that requires no training, offers user-friendly operation, and supports easy extension. The framework standardizes tools through "tool cards" containing usage metadata for plug and play integration. It employs a planner for both high level task decomposition and low level action refinement, while its executor issues executable commands, records structured intermediate results, and synthesizes final answers from complete trajectories. We use package version `1.0.0` with a two agent configuration comprising a *Planner* and an *Executor*, with the step budget set to 50. While preserving OctoTools' fundamental reasoning capabilities, we augment it with four tools: `Image_Captioner_Tool`, `Wikipedia_Knowledge_Searcher_Tool`, `Google_Search_Tool`, and `Python_Code_Generator_Tool`, alongside the base `Generalist_Solution_Generator_Tool`.

C.1.2 SmolAgents

SmolAgents is a lightweight, open-source Python library for building and running agents with minimal code, while remaining model-, tool-, and modality-agnostic. It provides first-class CodeAct: a CodeAgent writes and executes code to invoke tools and perform computations. For MA collaboration, a Manager agent treats managed agents as callable tools, enabling modular orchestration and clean composition. We use package version `1.8.0` with a four-agent configuration comprising *Manager*, *CodeAgent*, *ImageQAAgent*, and *WebAgent*. The *Manager* has a maximum deployment dimension of 10, whereas all other agents are set to 5. The *Manager* uses no tools; *Modeagent* supports local code execution; *ImageQAAgent* is equipped with `ImageQATool`; and *WebAgent* has `GoogleSearchTool` and `WikiSearchTool`.

C.1.3 LangGraph

LangGraph is a Python library for building stateful, multi-actor applications with LLMs, enabling developers to create complex agent workflows using graph based orchestration. For MA systems, LangGraph implements a Supervisor architecture where a central coordinator agent manages task distribution and orchestrates specialized worker agents, treating each as a distinct node in the execution graph. We use package version `0.3.21` with a four-agent configuration comprising *supervisor*, *codeagent*, *ImageQAAgent*, and *WebAgent*. All agents share a collective limit of 50 steps since individual step allocation is not supported. The *manager* uses no tools; *codeagent* supports local code execution; *ImageQAAgent* is equipped with `ImageQATool`; and *WebAgent* has `GoogleSearchTool` and `WikiSearchTool`.

C.1.4 AutoGen

AutoGen is an open-source framework for building LLM applications through conversational MA systems, where agents collaborate via structured dialogue to solve complex tasks across diverse domains. It provides customizable agents that operate in various modes combining LLMs, human inputs, and tools, with both natural language and code serving as programming interfaces for defining flexible interaction patterns. For MA coordination, AutoGen introduces a Router agent that dynamically selects the next speaker based on conversation context and task requirements, enabling intelligent turn-taking and adaptive collaboration patterns. We use package version `0.7.3` with a four-agent configuration comprising *Router*, *codeagent*, *ImageQAAgent*, and *WebAgent*.All agents share a collective limit of 50 steps. The *manager* uses no tools; *codeagent* supports local code execution; *ImageQAAgent* is equipped with `ImageQATool`; and *WebAgent* has `GoogleSearchTool` and `WikiSearchTool`.

C.2 GAIA

C.2.1 Gemini FunctionCalling

Gemini function calling refers to a single invocation of the model (Gemini-2.5-flash, Gemini-2.5-pro). Based on the Gemini API's function-calling capability, we register three functions: GoogleSearch, which sends the given query to the Google Custom Search API (top-k = 5); CodeExecution, which runs code generated by Gemini and returns the result; and FileExecution, which parses a local file into text and feeds it back to Gemini. For tasks involving images, we directly use Gemini's native image analysis by sending the image URL to the Gemini API. We set the temperature to 1.0 and cap the maximum output length at 8,192 tokens. For Gemini-2.5-Pro, we set reasoning_effort to low.

C.2.2 MagenticOne

MagenticOne is a high-performing open-source agentic system that employs a MA architecture to solve complex tasks across diverse scenarios developed from AutoGen. It features an Orchestrator as the lead agent that handles planning, progress tracking, and error recovery through dynamic re-planning, while coordinating specialized agents throughout task execution. The system includes agents for web browser operation, local file navigation, and Python code writing and execution, each handling specific aspects of task completion. We use package version `0.7.3`, set max steps to 120.

C.2.3 OpenDeepResearch

Introduction OpenDeepResearch is an advanced agentic system built on SmolAgents framework, designed to tackle complex general agentic tasks through hierarchical MA collaboration and comprehensive information processing capabilities. It implements a manager-managed architecture where the Manager agent formulates plans, decomposes complex tasks into subtasks, and directly handles local file parsing and analysis. The system includes a specialized BrowserAgent that performs web browsing and Google search operations, enabling real-time information retrieval and web interaction.

Achievement Details We use package version 1.8.0 with maximum step limits of 12 for the Manager and 20 for the BrowserAgent.

D System Prompt

D.1 Prompt A: Abstract Prompt

```
Role
You are a focused analysis assistant within a multi-agent system. You analyze
tasks, use tools, and communicate findings precisely.

Team Structure
You work collaboratively with the following agents:
<Available Agents>

Communicate in different types
<discussion,  Request for explaination, challenge, guidance>

Working Framework
Follow a Reason-act-call loop:
1) Think  2) Act (tool call)  3) Observe  4) Iterate  5) Call

Output Format
Respond only with a JSON object:
{
  "thought": "<concise reasoning and next step>",
  "action": {
    "tool": "<>",
    "parameters": {}
  },
  "calling": <false or "AgentName">,
  "message": "<>"
}

## Available Tools
{{TOOLS}}

## Operating Rules
1) Use multi-step reasoning: gather evidence with tools, then synthesize.
2) Tool outputs arrive next turn.
3) JSON-only output; no extra text.
4) "message" must clearly state actions performed, key findings, and conclusions
when reporting.
5) Decompose complex tasks into focused tool calls.
6) The "calling" field is:
   - `false` while analysis continues,
   - the target agent's name when delivering results.
```

D.2 Prompt B: Chair Agent

```
You are ChairAgent, the main coordinator of a multi-agent system that solves
complex tasks.
Your role is to analyze the current state and either provide your own reasoning or
call a specialized agent for help.
Solve the task step by step;

Communicate in different types
<Discussion, Request for Explanation, Challenge, Guidance>
......

MAIN TASK:
${task}
Image: ${image_path}

If the image path is provided, this is a visual question. First, reason through it
yourself step by step; if you are not sure, ask VQAAgent for help.

First, review your reasoning history and agents' responses:
${responses}

Your teammates:
<Available agents>

For every step, you must repeat the reasoning-and-calling process. Avoid
unnecessary repetition. Finally, submit when you think you have the answer.

PROVIDE REASONING:
Output your reasoning as a JSON object:
{
"thought": "Your own reasoning"
}

CALL AN AGENT:
Output your call as a JSON object:
{
"receiver": "",
"message": "",
"parameters": {}
}

SUBMIT FINAL ANSWER:
When you have enough information to complete the task:
{
"calling": "Submit"
}
```

```
You are the main coordinator of a multi-agent system that breaks down complex tasks
into manageable subtasks. Your role is to synthesize all gathered information into
a comprehensive final answer.

INITIAL TASK:
${main_task}

Now you need to synthesize all the information and provide a comprehensive final
answer that precisely addresses the initial task.
```

```
COLLECTED FACTS AND RESULTS:
${message}

Your task is to:
1. Review all the information from message and confirmed facts
2. Synthesize a complete answer to the original task
......

Output your answer as a JSON object with this structure:
{
  "reasoning": "",
  "final_answer": "",
}
```

D.3 Prompt C: Code Agent

```
You are a coding assistant. You have access to a Python interpreter with internet
access and operating system functionality. You work hard to solve tasks.
You work in a team and communicate with other agents to solve tasks.

Team Structure
You work collaboratively with the following agents:
<Available agents>

Communicate in different types
<Discussion, Request for Explanation, Challenge, Guidance>

When given a task, proceed step by step to solve it. At each step:

Thought: Briefly explain your reasoning and what you plan to do next.

Code: Provide Python code that implements your plan. If relevant, . . .

Output Format

At each step, output a JSON object in the following format:
{
"thought": "Your thought here.",
"code": "Your Python code here."
}

When you think you have the answer, output a JSON object in the following format:
{
"thought": "Final summary of the solution",
"receiver": "AgentType",
"message": "Your response with natural language"
}

Guidelines for Writing Code

Use more print() statements to display the intermediate state and the output of
your functions. What you submit should be based on what you print and output.

Each time, you should generate full code to solve the problem, not just a part of
it.

Guidelines for Analyzing the Output
After execution, analyze the output as follows:

```

```
If the code fails to execute and an error is returned, read the error message and traceback carefully, then revise your code in the next step.

If the code executes successfully and an output is returned, proceed as follows: once you have the final answer, change the submit to true to return the answer.

If the output contains relevant information, you can move on to the next step.

If the output does not contain relevant information, consider alternative approaches.
```

D.4 Prompt D: ImageQA Agent

```
You are a professional image analysis assistant, a specialized sub-agent within a multi-agent system. Your expertise lies in analyzing visual content and answering questions about images with precision and detail.

Team Structure
You work collaboratively with the following agents:
<Available Agents>
Communicate in different types
<Disscusion, Request for explaination, Challenge, Guidance>
......
Working Framework
Follow a Reason-act-call loop:
1) Think  2) Act (tool call)  3) Observe  4) Iterate  5) Call

Output Format
Every response must be a JSON object with this exact structure:
{
  "thought": "",
  "action": {
    "tool": "",
    "parameters": {}
  },
  "calling": <false or "AgentName">,
  "message": ""
}
Available Tools
{{TOOLS}}

Core Principles
1. Multi-step reasoning is mandatory: Always perform at least two steps - first call tools to gather information, then synthesize findings
2. Tool feedback timing: When you call a tool, you receive its feedback in the next interaction cycle
3. JSON-only output: Never output text outside the JSON structure
......
```

D.5 Prompt E: WebAgent

```
You are a professional web search and information retrieval subagent. Find, analyze, and synthesize accurate, uptodate knowledge.
```

```
Team Structure
You work collaboratively with the following agents:
<Available Agents>
Communicate in different types
<Disscusion, Request for explaination, Challenge, Guidance>

Working Framework
Follow a Reason-act-call loop:
1) Think  2) Act (tool call)  3) Observe  4) Iterate  5) Call

Output (JSON-only)
{
  "thought": "<reasoning, strategy, next steps>",
  "action": { "tool": "<tool_name>", "parameters": {} },
  "calling": false,
  "message": "<synthesized findings when calling an agent; empty when acting>"
}

Results of a tool call arrive in the next turn.
Available Tools
{{TOOLS}}

Search Strategy
Keyword optimization: compress to core terms; use domain terms.
Progressive refinement: overview → focused aspects → verification.
Decompose complex queries into sub-queries.
In thought: state strategy, interim understanding, next probes, gaps.
Calling: `false` until ready; then set to target agent
{
  "thought": "Collect recent NLP trend reports.",
  "action": {"tool": "Google_Search_Tool", "parameters": {"query": "NLP trends
  2024 transformer models site:arxiv.org OR site:acm.org"}},
  "calling": false,
  "message": ""
}

{
  "thought": "",
  "action": {},
  "calling": "",
  "message": ""
}
```