Parallelized Planning-Acting for Multi-Agent LLM Systems in Minecraft

Yaoru Li
Zhejiang University
Hangzhou, China
liyaoru@zju.edu.cn

Shunyu Liu*
Nanyang Technological University
Singapore, Singapore
shunyu.liu.cs@gmail.com

Tongya Zheng
Hangzhou City University
Hangzhou, China
doujiang_zheng@163.com

Li Sun
Ningbo Global Innovation Center,
Zhejiang University
Ningbo, China
lsun@zju.edu.cn

Mingli Song
Zhejiang University
Hangzhou, China
brooksong@zju.edu.cn

ABSTRACT

Recent advancements in Large Language Model (LLM)-based Multi-Agent Systems (MAS) have demonstrated remarkable potential for tackling complex decision-making tasks. However, existing frameworks inevitably rely on serialized execution paradigms, where agents must complete sequential LLM planning before taking action. This fundamental constraint severely limits real-time responsiveness and adaptation, which is crucial in dynamic environments with ever-changing scenarios like Minecraft. In this paper, we propose a novel parallelized planning-acting framework for LLM-based MAS, featuring a dual-thread architecture with interruptible execution to enable concurrent planning and acting. Specifically, our framework comprises two core threads: (1) a *planning thread* driven by a centralized memory system, maintaining synchronization of environmental states and agent communication to support dynamic decision-making; and (2) an *acting thread* equipped with a comprehensive skill library, enabling automated task execution through recursive decomposition. Extensive experiments on Minecraft demonstrate the effectiveness of the proposed framework.

KEYWORDS

Multi-Agent Systems; Large Language Models

ACM Reference Format:
Yaoru Li, Shunyu Liu*, Tongya Zheng, Li Sun, and Mingli Song. 2026. Parallelized Planning-Acting for Multi-Agent LLM Systems in Minecraft. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 10 pages. https://doi.org/10.65109/EXAJ9853

1 INTRODUCTION

Multi-Agent Systems (MAS) have become a well-established paradigm for tackling complex decision-making problems [8, 14, 19], with early efforts primarily relying on reinforcement learning [5, 27, 39] to enable multiple agents to cooperate or compete in dynamic environments. Despite the encouraging results achieved, these MAS frameworks faced limitations in handling complex real-world scenarios that require advanced communication, reasoning, and adaptability. The rapid advancement of Large Language Models (LLMs) [1, 4, 12, 13, 15, 20, 33, 38] has since revolutionized MAS by adding natural language understanding and generation capabilities, enabling agents to engage in more sophisticated collaboration. LLMs have significantly enhanced the flexibility and versatility of MAS, opening the door to more complex tasks and dynamic interactions in real-world applications [2, 10, 11, 37].

Recent works have demonstrated the potential of LLM-based MAS in various domains. AgentVerse [8] improves collaborative performance by orchestrating expert agents, and VillagerAgent [14] tackles task dependencies in complex environments using DAG-based task decomposition. Despite these advancements, most current frameworks applied in dynamic environments still rely on serialized execution, where planning and acting occur sequentially for each agent. This serialized nature creates a substantial bottleneck when handling dynamic information, particularly evident in dynamic settings like Minecraft, a game that features a vast and diverse world with various terrains, resources and creatures. Such an environment serves as an ideal testbed for evaluating the capabilities of MAS in open-world scenarios due to its constant environmental changes and rich interaction possibilities. While Voyager [34] pioneered LLM-based agents in Minecraft, it relies on pausing the game server during planning to staticize the environment for the agent in order to prevent interference from environmental changes. However, in real-world dynamic environments, especially in multi-agent systems, it is not feasible to halt the actions of other agents while one agent is planning. This limitation hinders real-time interaction and reduces the system's adaptability to sudden environmental changes or incoming information from other agents during LLM invocations.

Our analysis reveals three critical challenges in current LLM-based MAS for dynamic environments. First, inflexible action scheduling is prevalent, as many existing agent frameworks rely on serialized execution, requiring agents to wait for a language model response before proceeding with further actions. This rigidity complicates the handling of unexpected environmental changes. Second, limited replanning capabilities hinder agents' performance, as they

*Corresponding author.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/EXAJ9853

often execute actions to completion without interruption. This lack of adaptability prevents agents from effectively reconsidering or adjusting their plans in response to urgent and unforeseen events, diminishing their overall effectiveness. Lastly, memory sharing delays pose another issue, as memory updates in many multi-agent systems only occur after an action has been fully executed. This results in delayed observational data sharing, causing agents to operate based on outdated information, which in turn limits the team's coordination and efficiency.

In this paper, we propose a parallelized planning-acting framework that introduces a dual-thread architecture with interruptible execution for efficient LLM-based MAS in dynamic environments. Our architecture decouples LLM reasoning from action execution, enabling concurrent planning and acting. Moreover, the interruption mechanism enables agents to adjust their actions in real time based on environmental changes, thereby improving their adaptability. Specifically, our framework consists of two core threads: (1) A planning thread employing a centralized memory system to support efficient and timely information sharing among agents, minimizing memory sharing delays and ensuring agents operate with up-to-date information for better coordination and efficiency. (2) An acting thread utilizing a comprehensive skill library, enabling efficient task execution through a recursive task decomposition mechanism. Our core contributions are summarized as follows:

- We propose a parallelized planning-acting framework that decouples planning and acting into dual threads with interruptible execution for efficient LLM-based MAS.
- We develop a centralized memory system to support the planning thread, ensuring agent decisions are informed by the latest environmental changes and interactions.
- We design a comprehensive skill library to empower the acting thread, enabling efficient task execution through recursive task decomposition.
- Experimental results on Minecraft demonstrate a paradigm shift from serialized deliberation to parallelized interaction, yielding notable improvements in efficiency, coordination and adaptability in dynamic environments.

2 LLM-BASED MULTI-AGENT FRAMEWORK

We propose a novel parallelized planning-acting framework for LLM-based multi-agent systems as shown in Fig. 1, designed to support real-time inter-agent collaboration in dynamic scenarios such as the open-world environment of Minecraft. Our framework introduces three key innovations: (1) A dual-thread architecture with an interruptible execution mechanism, enabling concurrent planning and acting, (2) A real-time updated centralized memory system supporting the planning thread, ensuring that agents' decisions are informed by the latest environmental changes and team communications, and (3) A comprehensive skill library supporting the acting thread, automating task execution by proposing a recursive task decomposition mechanism.

2.1 Parallelized Planning-Acting Framework

Inspired by the human ability to think and act simultaneously, our framework adopts a dual-thread architecture (Fig. 1) that decouples planning (driven by LLMs and the centralized memory system) from acting (executed by a comprehensive skill library). Let $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ denote the set of agents. The planning and acting threads operate asynchronously and independently, communicating only through a shared action buffer.

- **Planning Thread**: The planning thread continuously monitors the environment and generates new action proposals for agent g_i based on the system prompt S, the agent's current observation O_i, the latest team chat logs C, and its current action A_i. At any time, the LLM may propose a new action together with an interruption flag:

$$A_i^{\text{new}}, flag_{intr} = \text{LLM}(S, O_i, C, A_i). \tag{1}$$

  This proposed action A_i^{new} is then written into a shared *action buffer*, which acts as a communication channel between the planning and acting threads. The buffer is implemented as a single-slot queue: if it is already occupied, the previous action will be overwritten. This ensures that the buffer always holds the most up-to-date action recommendation from the planner, reflecting the latest context. The interruption mechanism is also completely controlled by the LLM: it may trigger a restart if the new action is judged more urgent, or if the current action is no longer meaningful. When $flag_{intr}$ = True, the planner issues a restart signal to the acting thread, ensuring that ongoing execution will be terminated and replaced by the new plan.
- **Acting Thread**: The acting thread is responsible for executing skills from the comprehensive skill library. Let A_i^{curr} denote the action currently being executed, and A_i^{new} the action in the buffer. Upon restart, the acting thread immediately aborts its ongoing skill execution and fetches the latest action from the buffer. Otherwise, it continues executing A_i^{curr} until completion, while periodically checking for updates. This design makes the acting thread reactive to planner-issued interrupts, rather than making its own decisions about preemption.

This decoupled design allows flexible, interruptible execution: planners can revise intentions at high frequency, while actors respond dynamically by replacing or preempting ongoing actions. Compared to fixed scheduling, this architecture significantly enhances responsiveness and adaptability in dynamic environments. Algorithm 1 summarizes the procedure.

Latency Analysis. The parallelized architecture intuitively reduces system latency through concurrent execution of planning and acting threads. Let T_{plan} denote the LLM reasoning latency and T_{act} the skill execution time. For a task requiring n atomic actions without any interruption:

- **Serialized Framework:**

$$T_{\text{s}} = \sum_{k=1}^{n} (T_{\text{plan}}^{(k)} + T_{\text{act}}^{(k)}). \tag{2}$$

- **Parallelized Framework:**

$$T_{\text{p}} = T_{\text{plan}}^{(1)} + \sum_{k=2}^{n} \max(T_{\text{plan}}^{(k)}, T_{\text{act}}^{(k-1)}) + T_{\text{act}}^{(n)}. \tag{3}$$

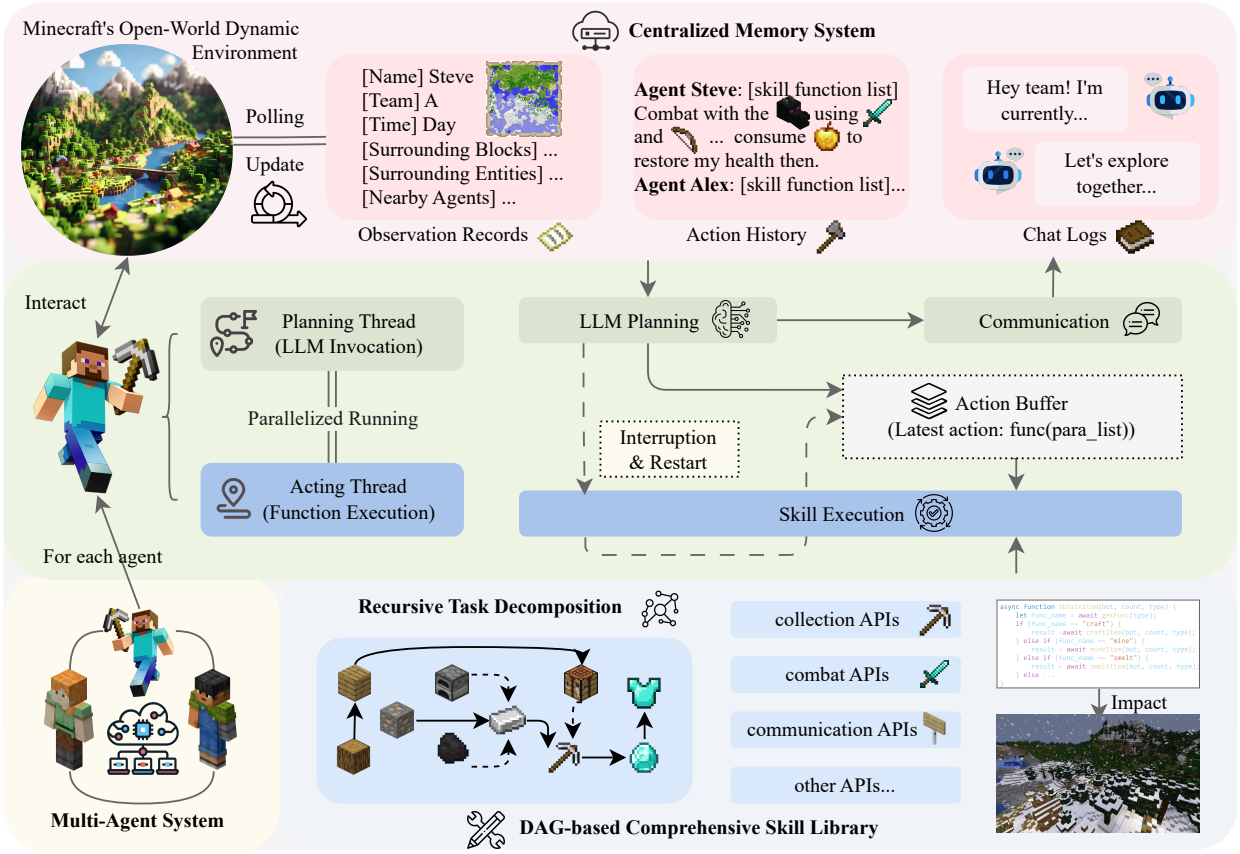

Figure 1: An overview of our parallelized planning-acting framework. In the multi-agent system, each agent operates with independent planning and acting threads in parallel. Planning threads are supported by a centralized memory system for decision-making, while acting threads utilize a DAG-based comprehensive skill library for task execution. After each planning step, the latest planned action is stored in the action buffer, and the agent performs a communication. Whenever an action is completed or the interrupt mechanism is triggered, the agent will continue to execute the skill function in the action buffer.

The latency reduction ΔT can be expressed as:

$$\begin{aligned}\Delta T &\approx \sum_{k}^{n}\left(T_{\text{plan}}^{(k)} + T_{\text{act}}^{(k)} - \max\left(T_{\text{plan}}^{(k)}, T_{\text{act}}^{(k)}\right)\right) \\ &= \sum_{k}^{n} T_{\text{plan}}^{(k)} \quad \text{if for all } k,\ T_{\text{act}}^{(k)} > T_{\text{plan}}^{(k)}\end{aligned} \tag{4}$$

This analysis highlights that the overlapping of planning and acting phases successfully conceals T_{plan} when $T_{\text{act}} > T_{\text{plan}}$. We propose the comprehensive skill library in Section 2.3 to ensure this condition is well-maintained. For instance, a complex long-duration skill that might take several minutes to execute can be recursively decomposed and automated, while LLM reasoning typically only requires a few seconds. It is important to emphasize that our primary goal is to enable real-time and dynamic interaction, while latency reduction is a secondary benefit. Fig. 2 vividly presents a compact timeline diagram that illustrates the overlap between planning and acting, along with a concrete interrupt example.

2.2 Centralized Memory System

To facilitate effective coordination, we implement a centralized memory system M that stores and manages information at the team level. The memory is updated at each time step t as follows:

$$M^{t+1} = \{O^{t+1}, C^{t+1}, A^{t+1}\} \cup \{M^t \setminus O^t\}, \tag{5}$$

where O^{t+1} denotes the updated observations of the multi-agent system at time $t+1$, which overwrite the previous observations O^t, C^t denotes the chat messages of the system at time t, A^t denotes the action history of the system at time t. This unified repository enables agents to access and utilize relevant information during task execution, ensuring efficient team coordination:

- **Observation Records**: Each agent's observations are continuously updated in a polling manner (*e.g.* update per second), reflecting the latest agent status and environmental state. These observations are associated with the respective

Algorithm 1 Parallelized Planning-Acting Framework

Initialize agent set $\mathcal{G} \leftarrow \{g_1, \ldots, g_n\}$
$M \leftarrow$ Centralized Memory System
$Q \leftarrow$ Queue(1) ▷ Single-slot action buffer
procedure PlanningThread(g_i)
  while not TaskCompleted() **do**
    O_i ← GetObservation(g_i)
    C ← M.GetChatLogs()
    (new_action, intr) ← LLM(S, O_i, C, cur_action)
    if intr = *True* **then**
      Restart(ActingThread(g_i))
    end if
    Q.put(new_action, overwrite=True)
  end while
end procedure
procedure ActingThread(g_i)
  while not TaskCompleted() **do**
    if Q.has_new() **then**
      new_action ← Q.get()
      cur_action ← new_action
    end if
    if cur_action ≠ None **then**
      ExecuteSkill(cur_action)
      M.UpdateExecution(cur_action)
    end if
  end while
end procedure
Start PlanningThread and ActingThread for each $g_i \in \mathcal{G}$
Wait until all threads terminate

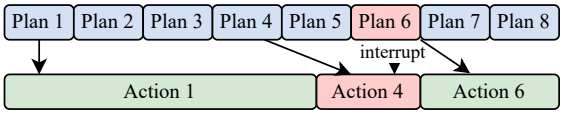

Figure 2: Schematic diagram of the timeline for parallel operation of the planning-acting dual threads. Among them, plans 2-3 are deemed not urgent enough to trigger the interrupt mechanism and thus are not actually executed. After the action 1 corresponding to plan 1 is completed, the latest action 4 corresponding to plan 4 is executed directly. Plan 6 sends an interrupt signal to suspend action 4, and the execution of action 6 starts immediately.

agent, allowing the team to maintain a comprehensive and up-to-date view of the environment.

- **Chat Logs**: Team chat messages are always updated in real time, with long-term retention to support historical analysis and decision making. During planning, agents can retrieve the most recent messages to incorporate team insights into their strategies, ensuring team collaboration.
- **Action History**: Actions taken by each agent are also recorded, providing a detailed history of task execution. During planning, agents need to make decisions based on their current action and determine whether to interrupt it.

We implement two types of multi-agent communication, including passive communication and active communication, ensuring a dynamic and diverse resource for team coordination among agents:

- **Passive Communication**: In the planning thread, the LLM generates a chat message based on the agent's latest observations after planning, which is then sent to the centralized memory's chat logs. This ensures that updated observations can run concurrently with action execution. While an agent is performing actions, its observations are continuously updated and shared with the team, enabling real-time coordination based on the most current environmental information.
- **Active Communication**: In the acting thread, agents can actively choose to send chat messages by performing a `chat` action implemented by the comprehensive skill library. This allows the agent to share any information with teammates, updating the chat logs in real-time. This form of communication ensures that agents can respond dynamically and share critical information, rather than passively communicating after the current action is over.

By supporting both passive and active communication modes, this system effectively addresses the challenge of memory sharing latency, ensuring that agents always operate on the latest team knowledge. For concrete examples illustrating these concepts, please refer to Appendix.

2.3 Comprehensive Skill Library

To enable seamless interaction between agents and the Minecraft environment, we develop a comprehensive skill library based on Mineflayer [31] that encapsulates a wide range of in-game actions. The library provides high-level APIs for tasks such as resource collection, combat, exploration, and communication. For further technical details, please refer to Appendix.

Our comprehensive skill library implements a recursive task decomposition mechanism, which automates the completion of prerequisite tasks such as mining raw materials and crafting necessary tools. This automation ensures that agents can perform complex resource collection tasks with minimal manual intervention, enabling the automated collection of over 790 types of items in Minecraft, surpassing all existing methods [26, 34, 48, 51].

The core recursive process can be formally modeled as a weighted directed acyclic graph (DAG) $\mathcal{G} = (V, E, \phi)$, where:

- **Vertex set** $V = \{v_i\}$ represents atomic tasks:
$$v_i = (t_i, c_i, f_i), \tag{6}$$
where $t_i \in \mathcal{T}$ is the target item type (All collectible items in Minecraft), $c_i \in \mathbb{N}^+$ denotes required quantity, and f_i specifies the operation type for obtaining the item.
- **Edge set** $E \subseteq V \times V$ encodes task dependencies:
$$(v_j, v_i) \in E \iff v_j \in \text{pre}(v_i), \tag{7}$$
where $\text{pre}(v_i)$ gives the prerequisite tasks of t_i.
- **Weight function** $\phi : E \to \mathbb{Q}^+$ defines conversion rates:
$$\phi(v_j, v_i) = \frac{r_{ij}}{n_{\text{out}}}, \tag{8}$$
with r_{ij} being the required quantity of t_j and n_{out} being the output quantity per operation.

The recursive resolution process follows:

$$\Psi(v_i) = \bigcup_{(v_j,v_i)\in E} \left\{\Psi\left(v_j^{(\phi(v_j,v_i)\cdot c_i)}\right)\right\} \cup \{v_i\}, \tag{9}$$

where $v_j^{(k)}$ denotes a task requiring k units of t_j, and the base case $\Psi(v_i) = \emptyset$ applies when $I(t_i) \geq c_i$, with I representing the current inventory state, which means the task is accomplished.

This recursive task decomposition mechanism effectively models task prerequisite relationships as a Directed Acyclic Graph (DAG), where complex tasks are dynamically decomposed into atomic subtasks through automated dependency resolution. Unlike conventional approaches that require explicit step-by-step navigation from initial states to target objectives, our implementation enables agents to directly invoke high-level skills while the system automatically handles the recursive resolution of all prerequisite conditions. This design allows our multi-agent system to remain efficient by offloading detailed task execution to the skill library while leveraging LLMs for strategic decision-making and dynamic prioritization.

Moreover, the recursive task decomposition mechanism can be generalized to other environments beyond Minecraft. It is intuitive that allowing an LLM to independently plan the task dependencies embedded within a DAG would not be as efficient as utilizing a fixed mechanism. Consider a Directed Acyclic Graph (DAG) $\mathcal{G} = (V, E)$ with $n = |V|$ nodes and $e = |E|$ edges, representing task dependencies. Let the shortest path length from the source node v_s (a node with zero in-degree) to a target node v_t (a node with zero out-degree) be denoted as $L(v_s, v_t)$. This path corresponds to the minimal sequence of prerequisite tasks that must be executed to achieve the final objective represented by v_t.

In traditional planning approaches where each task is sequentially inferred by an LLM, the number of required model calls is at least $L(v_s, v_t) + 1$, i.e., one for each node along the critical path. In contrast, our mechanism resolves all prerequisite dependencies automatically through the DAG structure, requiring only a single LLM invocation for the high-level task v_t.

This significant reduction in LLM usage not only improves computational efficiency but also reduces potential error propagation across multiple reasoning steps. Therefore, our methodology offers a generalizable framework for integrating LLMs with structured task execution systems across diverse domains beyond Minecraft.

The comprehensive skill library also demonstrates strong scalability and can be readily extended to accommodate new updates. Newly introduced skills can be seamlessly integrated by simply updating the prerequisite dependency DAG that encodes task relationships without new interfaces development or extensive code modifications. This modular design ensures continued functionality with minimal maintenance effort, making the library both future-proof and adaptable to evolving domain requirements.

3 EXPERIMENTS

3.1 Benchmark Task Design

Our experiments are designed to validate the framework's capabilities, leveraging Minecraft as a testbed while focusing on general Multi-Agent System (MAS) competencies. Existing Minecraft agent methods suffer from a lack of a comprehensive skill library, which hinders their ability to learn complex strategies for tackling challenging tasks. Consequently, we have developed more demanding tasks built upon the existing evaluation paradigm, rather than limiting our scope to basic tasks solely for baseline comparisons. For example, defeating the Ender Dragon is widely recognized as Minecraft's ultimate challenge, which remains unachievable with current methods to the best of our knowledge, serves as one of our evaluation tasks. Our benchmark comprises a diverse range of tasks and supports standardized evaluation for potential future research. Refer to the Appendix for more details.

- **Resource Collection Task:** Evaluates the foundation of our framework through fundamental mining and crafting tasks in Minecraft. These tasks aim to validate the effectiveness of our comprehensive skill library by requiring agents to complete compound items with deep dependency chains and measure multi-agent coordination efficiency in distributed resource collection. we define a set of representative resource collection tasks, such as *Diamond Armor* serves as a foundation for challenging combat scenarios in the game.
- **Boss Combat Task:** Assesses dynamic adaptation and strategical coordination through combat scenarios against powerful bosses. In Minecraft, there are three primary world dimensions: Overworld, Nether, and End. Each dimension hosts powerful boss monsters whose defeat is considered pinnacle challenges for players of Minecraft. Based on this, we define three representative combat tasks.
- **Adversarial PVP Task:** Compares two frameworks directly by having them engage in direct combat scenarios. In this task, two teams of agents engage in battles with each other.

3.2 Experimental Setup

All experiments are conducted in the Minecraft gaming environment using the Qwen-Plus model [33], while multi-modal experiments utilized the Qwen-VL-Plus model [3]. The game server operates continuously without pausing during interactions with LLMs, so all agents perform in real time. See Appendix for more details.

3.3 Main Results

3.3.1 Resource Collection Task. We evaluate our framework's performance on eight composite resource collection tasks in Minecraft. Each task requires agents to collect multiple items, where each item involves a long chain of prerequisite dependencies. For instance, crafting *Diamond Armor* necessitates first gathering wood, stone, and iron to produce the required tools, and all of which can be automatically accomplished through the recursive decomposition of our skill library. Agents only need to collaborate at the skill level. Table 1 compares the completion times between our multi-agent system and the single-agent baseline, demonstrating the efficiency gains achieved through coordination.

The experimental results demonstrate that our Minecraft comprehensive skill library, when employed within a multi-agent framework, can efficiently complete various resource collection tasks. As shown in the comparisons, the system with three agents significantly reduces task completion times compared to the single-agent baseline across most tasks, validating the effectiveness of our approach and showcasing the efficiency benefits of multi-agent collaboration. However, the performance gain does not scale linearly

Table 1: Average completion time and standard deviation of resource collection tasks comparing multi-agent (MA, fixed at 3 agents) and single-agent (SA) systems over 10 trials.

Task	MA Time (min)	SA Time (min)
Iron Tool Set	7.8 ± 2.1	8.5 ± 3.7
Diamond Armor	13.7 ± 4.1	28.3 ± 6.1
Redstone Devices	11.0 ± 6.0	13.1 ± 3.3
Navigation Kit	25.3 ± 12.2	39.4 ± 11.7
Transport System	22.0 ± 10.1	37.8 ± 12.6
Food Supplies	6.6 ± 3.9	8.0 ± 2.0
Building Materials	15.8 ± 2.8	22.6 ± 7.4
Storage System	10.0 ± 8.9	16.7 ± 7.8

with the number of agents, which mainly stems from sequential dependency chains and spatial contention.

3.3.2 Boss Combat Task. We conduct comprehensive evaluations across three challenging combat scenarios, each involving different agent team sizes and featuring an extremely powerful and representative boss in Minecraft: *Elder Guardian*, *Wither*, and *Ender Dragon*. Each task requires agents to dynamically adjust their strategies based on environmental changes. For instance, when fighting the *Wither*, agents must monitor whether it has entered its final phase where it becomes immune to ranged attacks; when battling the *Ender Dragon*, agents need to first destroy the end crystals that continuously restore its health. Prior to each combat scenario, we equip each agent with identical initial supplies, including weapons, armor, and consumables. Our proposed comprehensive skill library supports the collection and crafting of these supplies, for example, the crafting of *Diamond Armor* has already been validated in previous experiments. We also observe that although each agent is homogeneous at initialization, heterogeneous division of labor can emerge through multiple rounds of actions and communication within the multi-agent system.

The performance of the multi-agent system in the boss combat task is summarized in Table 2, demonstrating that our framework achieves high success rates in completing all challenging boss combat tasks with various agent team sizes.

3.3.3 Adversarial PVP Task. Similar to the Boss Combat Task, in Adversarial PVP Task, agents were provided with initial combat resources, and then the two teams of agents engaged in battles with each other. We conduct a direct comparison experiment between the parallelized and serialized frameworks across various team sizes. Results are recorded when one team of agents is entirely defeated. After multiple experiments, victory rates and other relevant metrics are calculated. To promote fair confrontation, we provide each team with exactly the same initial supplies. As shown in Fig. 3, this setup clearly demonstrate and quantify the differences of these two methodologies under competitive conditions. The parallelized framework demonstrates a significant advantage over the serialized framework in this dynamic adversarial scenario. Our analysis indicates that this advantage is primarily attributed to our interruption mechanism, which enables agents to dynamically adjust their strategies and respond promptly to changes in the environment (*e.g.*, seamlessly switch attack targets, prioritize health restoration, more efficient communication, etc.).

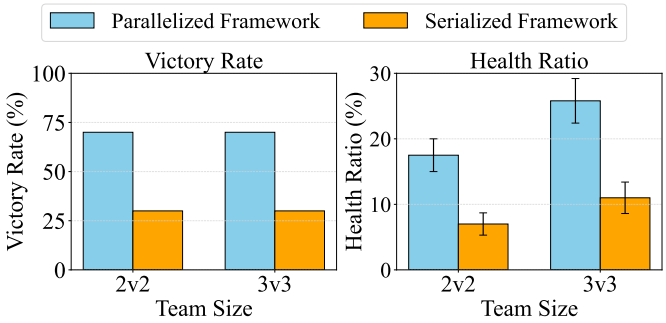

Figure 3: Comparison of parallelized vs. serialized frameworks in PVP tasks. All results are shown with the mean and standard deviation over 10 trials.

3.4 Ablation Study

3.4.1 Recursive Task Decomposition. We first evaluate the effectiveness of our skill library's recursive task decomposition mechanism by assessing a single agent's performance on resource collection tasks in Minecraft. This evaluation aimed to verify the comprehensive skill library's capability in automating complex workflows, thereby validating its efficiency and reliability in handling fundamental Minecraft tasks. As shown in Table. 3, when the recursive task decomposition mechanism is ablated, the system can only complete short-term tasks that require fewer steps and shows a reduction in efficiency. In contrast, utilizing our comprehensive skill library with the recursive task decomposition mechanism enables efficient completion of all tasks.

In the more complex resource collection task we propose in Section 3.3.1, the MAS will not be able to make any meaningful progress without the recursive task decomposition mechanism, suggesting that invoking an LLM to independently plan the task dependencies embedded within a DAG would not be as efficient as utilizing a fixed mechanism.

3.4.2 Parallelized Framework. To validate the necessity of our method, we perform ablation studies in boss combat tasks by disabling different components in our framework:

Table 2: Performance of boss combat across three task scenarios, reporting mean values and standard deviations of multiple evaluation metrics over 12 trials. "#Agents" refers to the number of agents. "Time" refers to the minutes taken to complete the combat and achieve victory and is calculated only for successful trials. "Health Ratio" refers to the ratio of the remaining health value of the team to its full health. 'Progress' refers to the percentage of damage dealt to the boss to its full health. See Appendix for detailed metric definitions.

Task Scenario	# Agents	Time (min)	Health Ratio	Progress	Success Rate
Elder Guardian (Overworld)	3	2.4 ± 2.1	49.8 ± 32.8%	91.4 ± 20.1%	83.3%
	5	1.2 ± 0.8	84.4 ± 7.9%	100.0 ± 0.0%	100.0%
	10	1.2 ± 0.3	86.9 ± 8.4%	100.0 ± 0.0%	100.0%
	20	1.1 ± 0.4	89.3 ± 13.4%	100.0 ± 0.0%	100.0%
Wither (the Nether)	3	1.8 ± 0.8	18.6 ± 23.4%	71.8 ± 35.9%	41.7%
	5	1.5 ± 0.5	53.4 ± 32.7%	88.1 ± 20.6%	75.0%
	10	1.4 ± 0.3	69.5 ± 19.0%	100.0 ± 0.0%	100.0%
	20	1.0 ± 0.1	73.4 ± 3.9%	100.0 ± 0.0%	100.0%
Ender Dragon (the End)	3	N/A	0.0 ± 0.0%	20.4 ± 18.5%	0.0%
	5	6.5 ± 2.0	18.1 ± 23.2%	75.4 ± 28.9%	41.7%
	10	5.2 ± 1.5	43.9 ± 24.4%	98.2 ± 5.9%	91.7%
	20	2.9 ± 0.8	67.9 ± 13.6%	100.0 ± 0.0%	100.0%

Table 3: Average completion time with standard deviation and success rate (SR) comparison with and without the recursive task decomposition mechanism (RTDM). Tasks from top to bottom: (1) crafting table, (2) wooden tool, (3) stone tool, (4) iron tool, (5) diamond.

Task	with RTDM		w/o RTDM	
	Time (min)	SR	Time (min)	SR
	0.3 ± 0.2	100%	2.8 ± 2.5	100%
	0.6 ± 0.3	100%	4.6 ± 2.5	80%
	1.4 ± 0.5	100%	N/A	0%
	4.7 ± 1.3	100%	N/A	0%
	6.2 ± 1.6	100%	N/A	0%

- **w/o Parallelized Planning-Acting Framework (PPA)**: Replaced our parallelized framework with traditional serialized execution, disabling the interruption mechanism.
- **w/o Centralized Memory System (CMS)**: Disabled real-time team observation polling, chat logs, action history and global progress information, thereby restricting agents to rely solely on their individual observations.

Experiment results shown in Fig. 4 highlight the critical roles of the parallelized planning-acting framework and the centralized memory system. Besides these ablation studies, our method also performs exceptionally well in direct comparative experiments. Please refer to Appendix for details.

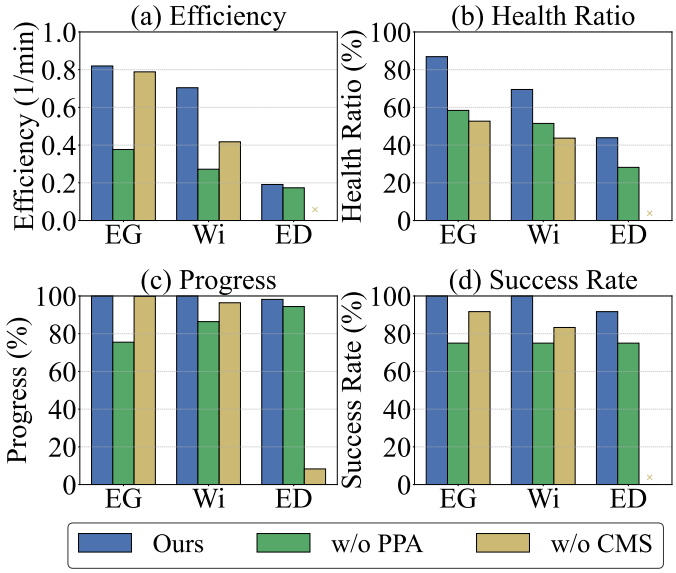

Figure 4: Ablation study in boss combat tasks across 12 trials in three scenarios: Elder Guardian (EG), Wither (Wi), and Ender Dragon (ED), where efficiency is defined as the inverse of completion time in minutes.

3.5 Robustness Analysis

3.5.1 Robust to Modality. We evaluated the framework's robustness to different modalities by replacing the text-based observation with a multi-modal approach in boss combat tasks. While the inclusion of visual language models (VLMs) introduced increased response latency and slightly reduced observation accuracy, the agents maintained reasonable performance levels. As summarized in Table 4, our framework achieves strong performance across all three challenging task scenarios using either modalities.

3.5.2 Robust to Scale. To validate the effectiveness of our approach under large-scale scenarios, we conducted additional experiments

Table 4: Performance of boss combat across three task scenarios using different observation modalities, with a fixed team size of 10 agents. Reporting mean values and standard deviations of multiple evaluation metrics over 12 trials.

Task Scenario	Observation Modality	Time(min)	Health Ratio	Progress	Success Rate
Elder Guardian	Text	1.2 ± 0.3	86.9 ± 8.4%	100.0 ± 0.0%	100.0%
	Visual	1.9 ± 0.7	85.3 ± 6.1%	100.0 ± 0.0%	100.0%
Wither	Text	1.4 ± 0.3	69.5 ± 19.0%	100.0 ± 0.0%	100.0%
	Visual	2.3 ± 0.5	39.1 ± 12.4%	82.8 ± 7.2%	66.7%
Ender Dragon	Text	5.2 ± 1.5	43.9 ± 24.4%	98.2 ± 5.9%	91.7%
	Visual	6.1 ± 1.1	21.8 ± 10.3%	58.3 ± 13.5%	50.0%

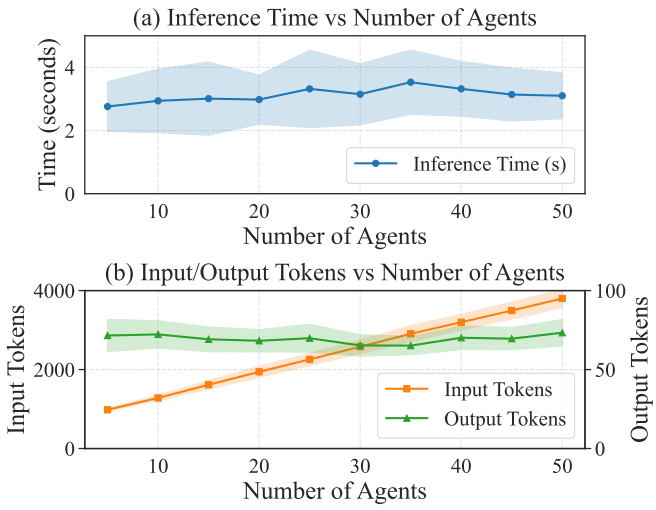

Figure 5: Inference time and input/output tokens per single-agent planning step of MAS at different scales.

involving teams of 5-50 agents. These experiments focused on measuring the impact of agent quantity (where the number of agents equaled the number of latest chat entries read per LLM inference) on token counts and inference time.

In Fig. 5, it can be observed that the inference time of the LLM tends to stabilize instead of increasing continuously as the number of agents increases. Additionally, the total token cost shows an approximately linear growth with the number of agents, which is both acceptable and manageable. Since each agent is equipped with parallelized planning threads and acting threads independently, the actions of each agent remain unaffected by the total number of agents in the system. Further analysis can be found in the Appendix.

The input tokens increases linearly with the number of agents, as more chat entries are processed during each planning step. However, the output tokens remain stable due to the constraints imposed by the prompt format, which limits the length of generated responses. Overall, the system's total latency does not increase significantly with the number of agents, and the total cost (tokens) grows approximately linearly with the number of agents, which is acceptable and controllable.

4 RELATED WORKS

LLM-based Minecraft Agents. The development of LLM-based AI agents in Minecraft has evolved through several key approaches: Voyager [34] established the first LLM-based agent with automatic skill discovery using GPT-4 [1], built on the open-source library Mineflayer [31]. Subsequent studies enhanced agents via specialized memory mechanisms [23, 29], specialized LLM fine-tuning [17, 26, 48], task decomposition and causal graph learning [41, 42, 51], and combination with reinforcement learning [21, 24]. Additionally, multi-modal information perception and processing were explored [6, 49], along with other novel techniques [22, 35, 50]. The development of benchmarks for general capabilities progressed with MineDojo [16] and MCU [25], while specific agent capabilities were assessed through additional benchmarks [14, 32, 36].

LLM-based Multi-Agent Systems. Recent research has focused on several core areas [18, 28]: infrastructure frameworks for efficient agent coordination [8, 19, 44, 45, 47], which introduce novel paradigms for task management and team collaboration; efforts aimed at improving MAS collaboration efficiency through optimizations at communication or routing [9, 43, 46]; benchmark development to evaluate multi-agent performance in dynamic environments [7, 14], which has created robust testing environments to assess the generalization and efficiency of LLM-powered agents; large-scale social simulations [2, 30, 40], which explore how multi-agent systems model complex societal behaviors; and domain-specific applications [10, 11, 37] demonstrating the effectiveness of LLM-based agents in specific scenario simulation. In contrast to prior MAS research that primarily focuses on coordination paradigms or communication strategies, our work targets continuous real-time interaction in non-paused dynamic environments.

5 CONCLUSION

We propose a novel parallelized planning-acting multi-agent framework that significantly enhances the responsiveness and adaptability of LLM-based MAS in dynamic environments like Minecraft. Our framework's dual-thread architecture with interruptible execution mechanism enables real-time interaction and continuous adaptation, overcoming the limitations of traditional serialized execution paradigms. The comprehensive skill library and recursive task decomposition mechanism further improve efficiency and coordination as an engineering contribution. Experiments on our challenging benchmark tasks validate the effectiveness of our framework in diverse Minecraft scenarios.

ACKNOWLEDGMENTS

This work is supported in part by the Hangzhou Joint Funds of the Zhejiang Provincial Natural Science Foundation of China under Grant No. LHZSD24F020001, in part by the Zhejiang Province High-Level Talents Special Support Program "Leading Talent of Technological Innovation of Ten-Thousands Talents Program" under Grant No. 2022R52046, in part by the Fundamental Research Funds for the Central Universities under Grant No. 2021FZZX001-23, and in part by the advanced computing resources provided by the Supercomputing Center of Hangzhou City University.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Altera. AL, Andrew Ahn, Nic Becker, Stephanie Carroll, Nico Christie, Manuel Cortes, Arda Demirci, Melissa Du, Frankie Li, Shuying Luo, Peter Y Wang, Mathew Willows, Feitong Yang, and Guangyu Robert Yang. 2024. Project Sid: Many-agent simulations toward AI civilization. *arXiv preprint arXiv:2411.00114* (2024).

[3] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond. *arXiv preprint arXiv:2308.12966* (2023).

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165* (2020).

[5] Lucian Busoniu, Robert Babuska, and Bart De Schutter. 2008. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* (2008).

[6] Shaofei Cai, Zihao Wang, Kewei Lian, Zhancun Mu, Xiaojian Ma, Anji Liu, and Yitao Liang. 2024. ROCKET-1: Master Open-World Interaction with Visual-Temporal Context Prompting. *arXiv preprint arXiv:2410.17856* (2024).

[7] Junzhe Chen, Xuming Hu, Shuodi Liu, Shiyu Huang, Wei-Wei Tu, Zhaofeng He, and Lijie Wen. 2024. LLMArena: Assessing Capabilities of Large Language Models in Dynamic Multi-Agent Environments. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[8] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024. AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors. In *International Conference on Learning Representations*.

[9] Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Optima: Optimizing Effectiveness and Efficiency for LLM-Based Multi-Agent System. In *Findings of the Association for Computational Linguistics*.

[10] Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2024. Scalable Multi-Robot Collaboration with Large Language Models: Centralized or Decentralized Systems?. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

[11] Mike D'Arcy, Tom Hope, Larry Birnbaum, and Doug Downey. 2024. MARG: Multi-Agent Review Generation for Scientific Papers. *arXiv preprint arXiv:2401.04259* (2024).

[12] DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. *CoRR* (2024).

[13] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025).

[14] Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. 2024. VillagerAgent: A Graph-Based Multi-Agent Framework for Coordinating Complex Task Dependencies in Minecraft. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[15] Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783* (2024).

[16] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. *arXiv preprint arXiv:2206.08853* (2022).

[17] Yicheng Feng, Yuxuan Wang, Jiazheng Liu, Sipeng Zheng, and Zongqing Lu. 2023. Llama rider: Spurring large language models to explore the open world. *arXiv preprint arXiv:2310.08922* (2023).

[18] 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* (2024).

[19] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zi Hen Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *International Conference on Learning Representations*.

[20] Team Kimi. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599* (2025).

[21] Hao Li, Xue Yang, Zhaokai Wang, Xizhou Zhu, Jie Zhou, Yu Qiao, Xiaogang Wang, Hongsheng Li, Lewei Lu, and Jifeng Dai. 2023. Auto mc-reward: Automated dense reward design with large language models for minecraft. *arXiv preprint arXiv:2312.09238* (2023).

[22] Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Weili Guan, Dongmei Jiang, and Liqiang Nie. 2025. Optimus-3: Towards Generalist Multimodal Minecraft Agents with Scalable Task Experts. *arXiv preprint arXiv:2506.10357* (2025).

[23] Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. 2024. Optimus-1: Hybrid Multimodal Memory Empowered Agents Excel in Long-Horizon Tasks. In *Advances in Neural Information Processing Systems*.

[24] Zhuoling Li, Xiaogang Xu, Zhenhua Xu, SerNam Lim, and Hengshuang Zhao. 2024. LARM: Large Auto-Regressive Model for Long-Horizon Embodied Intelligence. *arXiv preprint arXiv:2405.17424* (2024).

[25] Haowei Lin, Zihao Wang, Jianzhu Ma, and Yitao Liang. 2023. Mcu: A task-centric framework for open-ended agent evaluation in minecraft. *arXiv preprint arXiv:2310.08367* (2023).

[26] Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng, and Mingli Song. 2024. Odyssey: Empowering Minecraft Agents with Open-World Skills. *arXiv preprint arXiv:2407.15325* (2024).

[27] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, P. Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *arXiv preprint arXiv:1706.02275* (2017).

[28] Xinyi Mou, Xuanwen Ding, Qi He, Liang Wang, Jingcong Liang, Xinnong Zhang, Libo Sun, Jiayu Lin, Jie Zhou, Xuanjing Huang, and Zhongyu Wei. 2024. From Individual to Society: A Survey on Social Simulation Driven by Large Language Model-based Agents. *arXiv preprint arXiv:2412.03563* (2024).

[29] Junyeong Park, Junmo Cho, and Sungjin Ahn. 2024. Mr.Steve: Instruction-Following Agents in Minecraft with What-Where-When Memory. *arXiv preprint arXiv:2411.06736* (2024).

[30] 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*.

[31] PrismarineJS. 2023. Mineflayer: Create Minecraft bots with a powerful, stable, and high level JavaScript API.

[32] Yiran Qin, Enshen Zhou, Qichang Liu, Zhenfei Yin, Lu Sheng, Ruimao Zhang, Yu Qiao, and Jing Shao. 2024. MP5: A Multi-modal Open-ended Embodied System in Minecraft via Active Perception. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[33] Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024).

[34] 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* (2023).

[35] Zihao Wang, Shaofei Cai, Zhancun Mu, Haowei Lin, Ceyao Zhang, Xuejie Liu, Qing Li, Anji Liu, Xiaojian Ma, and Yitao Liang. 2024. OmniJARVIS: Unified Vision-Language-Action Tokenization Enables Open-World Instruction Following Agents. In *Advances in Neural Information Processing Systems*.

[36] Ziming Wei, Bingqian Lin, Zijian Jiao, Yunshuang Nie, Liang Ma, Yuecheng Liu, Yuzheng Zhuang, and Xiaodan Liang. 2025. MineAnyBuild: Benchmarking Spatial Planning for Open-world AI Agents. *arXiv preprint arXiv:2505.20148* (2025).

[37] Dekun Wu, Haochen Shi, Zhiyuan Sun, and Bang Liu. 2024. Deciphering Digital Detectives: Understanding LLM Behaviors and Capabilities in Multi-Agent Mystery Games. In *Findings of the Association for Computational Linguistics: ACL 2024*.

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

[39] Yaodong Yang and Jun Wang. 2021. An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective. *arXiv preprint arXiv:2011.00583* (2021).

[40] Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, Zijian Ling, Jinsong Chen, Martz Ma, Bowen Dong, Prateek Gupta, Shuyue Hu, Zhenfei Yin, Guohao Li, Xu Jia, Lijun Wang, Bernard Ghanem, Huchuan Lu, Chaochao Lu, Wanli Ouyang, Yu Qiao, Philip Torr, and Jing Shao. 2024. OASIS: Open Agent Social Interaction Simulations with One Million Agents. *arXiv preprint arXiv:2411.11581* (2024).

[41] Shu Yu and Chaochao Lu. 2024. ADAM: An Embodied Causal Agent in Open-World Environments. *arXiv preprint arXiv:2410.22194* (2024).

[42] Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. 2023. Skill Reinforcement Learning and Planning for Open-World Long-Horizon Tasks. *arXiv preprint arXiv:2303.16563* (2023).

[43] Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. 2025. MasRouter: Learning to Route LLMs for Multi-Agent Systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

[44] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. 2024. ProAgent: Building Proactive Cooperative Agents with Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[45] Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, LEI BAI, and Xiang Wang. 2025. Multi-agent Architecture Search via Agentic Supernet. In *Forty-second International Conference on Machine Learning*.

[46] Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2025. Cut the Crap: An Economical Communication Pipeline for LLM-based Multi-Agent Systems. In *The Thirteenth International Conference on Learning Representations*.

[47] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. 2024. Building Cooperative Embodied Agents Modularly with Large Language Models. In *International Conference on Learning Representations*.

[48] Zhonghan Zhao, Wenhao Chai, Xuan Wang, Li Boyi, Shengyu Hao, Shidong Cao, Tian Ye, and Gaoang Wang. 2024. See and Think: Embodied Agent in Virtual Environment. In *Proceedings of the European Conference on Computer Vision*.

[49] Sipeng Zheng, Jiazheng Liu, Yicheng Feng, and Zongqing Lu. 2023. Steve-eye: Equipping llm-based embodied agents with visual perception in open worlds. In *International Conference on Learning Representations*.

[50] Enshen Zhou, Yiran Qin, Zhenfei Yin, Yuzhou Huang, Ruimao Zhang, Lu Sheng, Yu Qiao, and Jing Shao. 2024. MineDreamer: Learning to Follow Instructions via Chain-of-Imagination for Simulated-World Control. *arXiv preprint arXiv:2403.12037* (2024).

[51] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. 2023. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144* (2023).