S-AGENTS: SELF-ORGANIZING AGENTS IN OPEN-ENDED ENVIRONMENTS

Jiaqi Chen* **Yuxian Jiang*** **Jiachen Lu** **Li Zhang†**
Fudan University

https://github.com/fudan-zvg/S-Agents

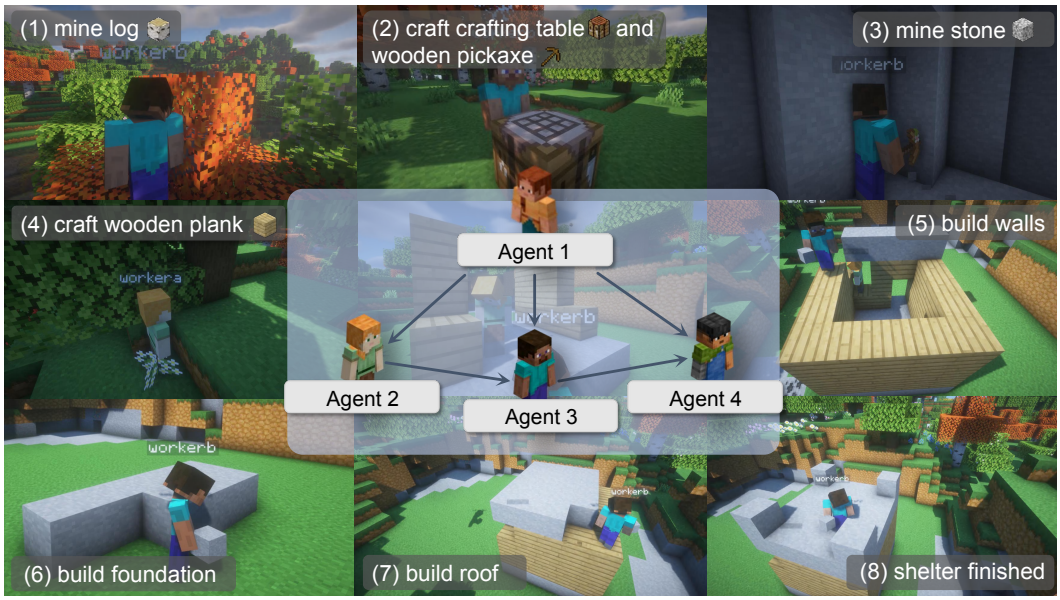


Figure 1: **Agent organization in open-ended environments.** Agent organization is a group of agents with a certain structure cooperating for shared goals. (1-3) depicts a group of agents collecting scattered rocks; (4-8) illustrates a group of agents building a shelter together. During their collaborative process, they autonomously orchestrated workflows without fixed steps by humans.

ABSTRACT

Leveraging large language models (LLMs), autonomous agents have significantly improved, gaining the ability to handle a variety of tasks. In open-ended settings, optimizing collaboration for efficiency and effectiveness demands flexible adjustments. Despite this, current research mainly emphasizes fixed, task-oriented workflows and overlooks agent-centric organizational structures. Drawing inspiration from human organizational behavior, we introduce a self-organizing agent system (S-Agents) with a "tree of agents" structure for dynamic workflow, an "hourglass agent architecture" for balancing information priorities, and a "non-obstructive collaboration" method to allow asynchronous task execution among agents. This structure can autonomously coordinate a group of agents, efficiently addressing the challenges of open and dynamic environments without human intervention. Our experiments demonstrate that S-Agents proficiently execute collaborative building tasks and resource collection in the Minecraft environment, validating their effectiveness.

*These authors contributed equally to this work.

†Li Zhang (lizhangfd@fudan.edu.cn) is the corresponding author with School of Data Science, Fudan University.

1 INTRODUCTION

The fundamental objective of artificial intelligence has long been the development of intelligent autonomous agents with the capacity to operate proficiently in open-ended environments (Weinbaum & Veitas, 2017; Fujita, 2009). Autonomous agents powered by Large Language Models (LLMs) (Kasneci et al., 2023; Touvron et al., 2023; Ji et al., 2023), especially GPT-4 (OpenAI, 2023b) have paved the way for innovative developments in this domain. These models showcase remarkable competencies in diverse domains, including instruction comprehension (Ouyang et al., 2022; Chung et al., 2022), decision-making (Shinn et al., 2023; Mandi et al., 2023; Zhang et al., 2023; Yao et al., 2023), tool usage Cai et al. (2023), code generation (Hong et al., 2023; Chen et al., 2023b), and diverse other domains (Tang et al., 2023). Moreover, LLM-powered agents have been employed in diverse tasks in open-ended environments, spanning from sandbox games, simulated environments, and robotics (Fan et al., 2022; Zeng et al., 2022; Brohan et al., 2023; Lu et al., 2023; Padmakumar et al., 2023). However, complex embodied tasks in open environments often necessitate collaboration among individual agents to achieve optimal outcomes (Woolley et al., 2010; Fehr & Gächter, 2000; Zhuge et al., 2023).

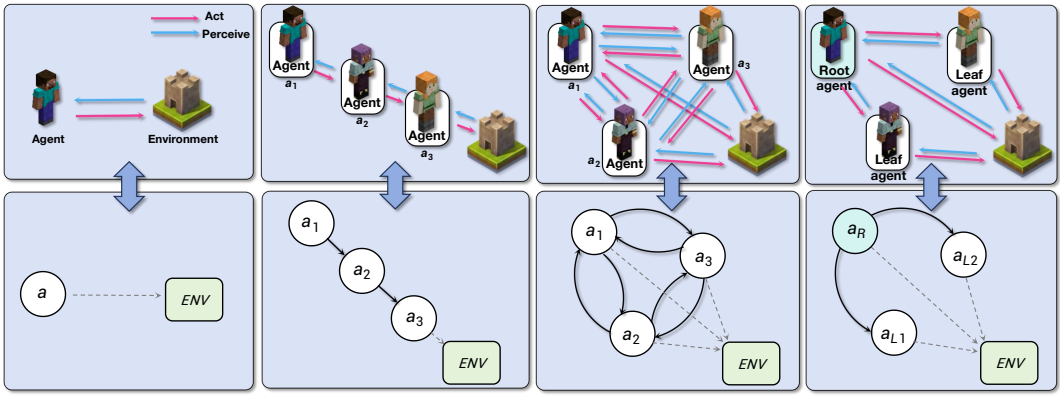
Unfortunately, the transition from single to multiple agents in an open-ended world introduces significant challenges in organizing a scalable group efficiently to tackle diverse tasks. The principal challenge in multi-agent organizations resides in structuring a multitude of agents. Previous research has predominantly concentrated on the creation of ***task-oriented, fixed workflows*** (Hong et al., 2023; Qian et al., 2023; Wu et al., 2023), neglecting to delve into the exploration of an organizational framework characterized by its ***agent-centric approach and flexibility in workflow***. Identifying optimal connections among individuals and empowering agents to autonomously define a collective workflow present a novel challenge (Grossi et al., 2006; Jensen et al., 2017; Chen et al., 2023c). Furthermore, agents in an organization must concurrently manage communication from both the surrounding environment and the organizational context.

In this study, we introduce S-Agents, a novel ***self-organizing*** multi-agent system that allows agents to flexibly arrange workflow autonomously, without the need for predefined human instructions, in open-ended environments. The system is specifically designed to operate within the open-world game Minecraft as its environment. It features: 1) A ***"tree of agents"*** organizational structure, comprising a root node (leadership agent) and multiple leaf nodes (executor agents), illustrated in Figure 2(d). The leadership agent autonomously arranges a flexible workflow without the need for human intervention. 2) An ***hourglass agent architecture*** that strives to balance priorities between the agent community and the physical environment, promoting coordinated actions. 3) A ***non-obstructive collaboration*** approach breaks away from the constraint of multiple intelligent agents sharing a fixed convergence beat, allowing agents to asynchronously execute collaborative tasks. This method is designed to alleviate delays induced by the slowest agent in each round, thereby enhancing overall efficiency.

2 RELATED WORK

LLM powered multi-agent collaboration Large Language Models (LLMs) have demonstrated phenomenal capabilities in various domains. Park et al. (2023); Akata et al. (2023); Xiang et al. (2023); Gong et al. (2023) drive multiple agents with LLM and simulate human-like conversations as well as some social behaviors; Recent attempts (Li et al., 2023; Dong et al., 2023; Qian et al., 2023) found that multi-agent collaboration could develop software following a fixed process, but could not coordinate autonomously; Multiple robotic arms (Mandi et al., 2023), and multiple agents (Zhang et al., 2023) working in collaboration all bring higher efficiency, but cannot be scaled up; While previous methodologies have demonstrated leading-edge performance on certain tasks, their collaboration mechanisms are often preordained and task-centric. However, the autonomous collaborative behavior of multi-agent embodied organizations in the open world remains an unclear topic in the current research. We aim to design an agent-centric organization that empowers agents to directly orchestrate workflows, inherently determining their collaboration framework. This approach should be versatile enough to tackle a wide range of embodied tasks.

Embodied agents in Minecraft Minecraft is an open-ended, three-dimensional world that is a free experimental environment for building numerous benchmarks and agent methods. Fan et al.



(a) Solo agent (b) Chain of agent (c) Graph of agent (d) Tree of agent (Ours)

Figure 2: **Schematic organizational structure comparison.** **(a)** Solo agent Wang et al. (2023a): Direct interaction with the physical environment; **(b)** Chain of agents Qian et al. (2023); Hong et al. (2023): Specialized agents sequentially perform their designated tasks and command the actions of the next agent; **(c)** Graph of agents Park et al. (2023): Decentralized structure allowing all agents to command each other; **(d)** Tree of agents: Centralized structure retaining one agent as a leadership agent (root agent a_r), with other executor agents (leaf agent a_{l1}, a_{l2}) executing commands.

(2022); Johnson et al. (2016); Kanervisto et al. (2022) are established benchmarks for evaluating single-agent algorithms. While Mohanty et al. (2022); Kiseleva et al. (2022a;b); Mohanty et al. (2023) focus on designing specific structures based on human instructions. On the other hand, Malmo (Perez-Liebana et al., 2019) offers an artificially designed game environment for multi-agent cooperation but lacks the necessary openness and diversity in tasks and environments. Building upon these benchmarks, several advanced works explore different approaches to realizing embodied agents. Many prior works utilize reinforcement learning to learn human game behavior (Kanitscheider et al., 2021; Lin et al., 2021; Mao et al., 2022). Fan et al. (2022); Baker et al. (2022) perform large-scale pre-training on game-playing videos. Wang et al. (2023c); Zhu et al. (2023) proposes a closed-loop feedback framework for a single agent, allowing the agent to achieve its goals vis interact with the environment. Wang et al. (2023a) uses LLMs to automatically generate the next task based on the environment, continuously enriching the skill library and exploring the world. Zhao et al. (2023) introduced visual perception capabilities to the LLM agent in Minecraft. However, there has been limited exploration into the design of organizational structures for multiple LLM agents and the architecture of embodied agents that can be organized.

3 METHODOLOGY

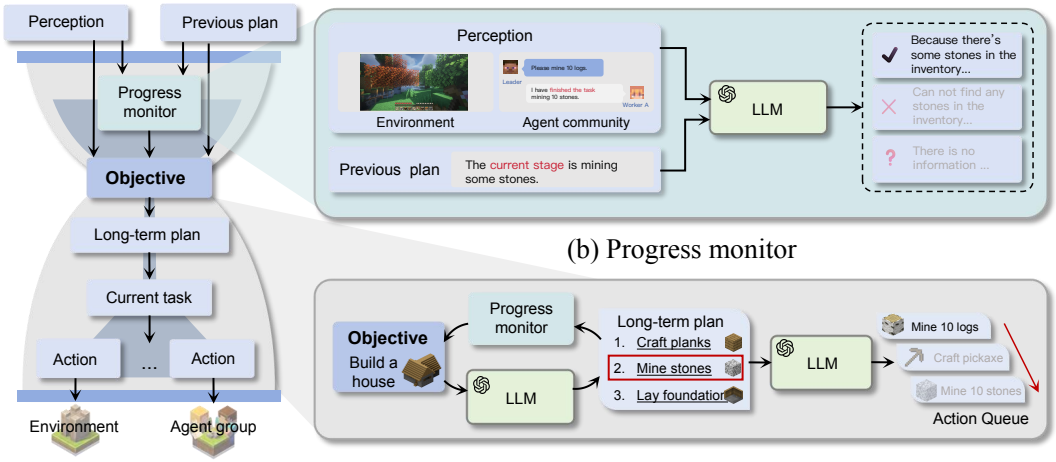
In this section, we introduce LLM-based self-organizing agents (S-Agents), including (i) an efficient directed tree of agents as an organizational structure, (ii) an hourglass agent framework for unified goal management and dynamic planning, and (iii) a non-obstructive collaboration paradigm allowing non-blocking parallelization.

3.1 ORGANIZATIONAL STRUCTURE OF AGENTS

3.1.1 AGENTS AS A GRAPH

We design an agent-centric organization, specifically, we do not predefine the specific roles and functions of agents (Hong et al., 2023; Qian et al., 2023; Wu et al., 2023). Instead, we place them in relationships, allowing them to autonomously allocate tasks and coordinate workflows based on circumstances and needs. This is the essence of ***self-organizing***. Self-organizing agents collaborate to coordinate multiple sub-tasks ***without the need for human intervention***, working towards a shared goal. This requires agents to have a certain organizational structure that allows tasks to be effectively transferred among agents. To model the agent organizational structure, we formulate a graphical representation known as the ***agent graph*** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. This graph is constructed based on the sets of agents, denoted as $\mathcal{A} = \{a_1, ..., a_n\}$, and the environment, represented as p,

$$V = \{a_1, ..., a_n, p\}, n > 1. \tag{1}$$



(a) Hourglass framework (b) Progress monitor (c) Hierarchical planning

Figure 3: **An illustration of hourglass agent architecture. (a)** ***Hourglass agent framework:*** *The upper segment:* Processes inputs like perception and the previous plan. These inputs undergo a series of operations, converging towards a unified and consistent objective (the bottleneck of the hourglass). *The lower segment:* Involves the decomposition of an objective through hierarchical planning. **(b)** ***Progress monitor:*** Utilizes LLM to assess the current progress status of the ongoing task. **(c)** ***Hierarchical planning:*** Comprises two stages: Task planner and action planner. See Appendix A.1 and B for example of planning and full prompt, respectively

We subsequently define the edges defined as $\mathcal{E} = \{(e_{ij})\}$, where directed edge $e_{ij} = (v_i, v_j)$ exists for $v_i, v_j \in \mathcal{V}$, $i \neq j$ if v_i actively acts (takes actions on the environment p or issues commands to other agents $\mathcal{V} - \{v_i, p\}$ and passively perceives feedback from v_j. The organizational structure of agents involves the design of the edge set $\mathcal{E}$ among agents. Existing research has primarily categorized these structures into the following two types.

Graph of agents The (fully connected) graph of agents (Park et al., 2023) is an organizational structure where all agents are interconnected, allowing mutual command and feedback (shown in Figure 2(c)), represented as

$$\mathcal{E}_{GoA} = \{(v_i, v_j) \mid v_i, v_j \in \mathcal{A}, i \neq j\}. \tag{2}$$

Theoretically, this structure promotes extensive communication and facilitates the flow of tasks within agent group $\mathcal{A}$, contributing to the emergence of self-organizing properties. However, interconnected agents (Sec. 4.3) exhibit bidirectional connections among agent pairs, *i.e.*, $(a_i, a_j), (a_j, a_i) \in \mathcal{E}$, forming command cycles that allow for mutual command issuance. These cycles introduce the potential for chaos, characterized by unpredictable and conflicting behaviors.

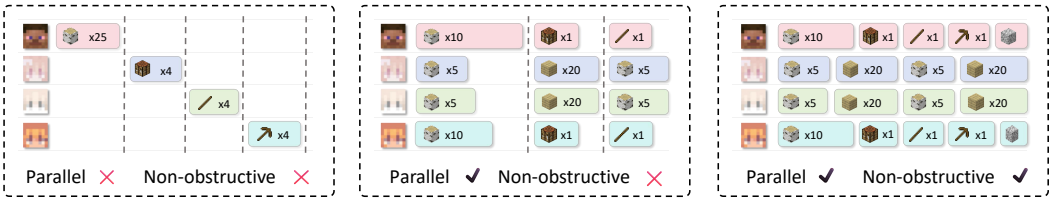
Chain of agents To address the issue of command cycles and capitalize on the expertise of specialized agents, a straightforward approach (Hong et al., 2023) is the Chain of agents (CoA) (as in Figure 2(b)), defined as follows:

$$\mathcal{E}_{CoA} = \{(v_i, v_{i+1}) | \ v_i \in \mathcal{A}\}. \tag{3}$$

In this fixed structure, instructional information unidirectionally flows from the first agent to the last, culminating in the completion of the final product. For instance, in a task requiring stone excavation, the a_1 undertakes lumbering, passing the baton to the a_2 for crafting a wooden pickaxe, subsequently transmitting the task to a_3 for stone excavation. While such a structure successfully avoids command cycles, it cannot flexibly adjust workflows, resulting in the potential for self-organization being compromised.

3.1.2 Tree of Agents

To avoid the command cycles in GoA and maintain the potential for self-organization, we propose a directed ***tree of agents*** (ToA), which introduces a leadership agent as the root node of



(a) Relay collaboration (b) Round based collaboration (c) Asynchronous collaboration (Ours)

Figure 4: **Comparison of collaboration strategies.** (a) involves one agent sequentially executing tasks after another, with no parallelization; (b) is round-based, executing round by round, while (c) is asynchronous. `Colored regions` indicate tasks being performed, and `white regions` denote agent idleness.

the agent tree, with other agents serving as leaf nodes, as illustrated in Figure 2(d). Let $\mathcal{V} = \{a_r, a_{l1}, a_{l2}, ..., a_{ln}, p\}$, where $n > 2$, a_r denotes root agent, and a_l denotes i-th leaf agent. Define the edges $\mathcal{E}$ as:

$$\mathcal{E}_{ToA} = \mathcal{E}_{\text{root}} \cup \bigcup_{i=1}^{n-1} \mathcal{E}_{li}. \tag{4}$$

In this structure, the root agent a_r serves as the central command authority, directing tasks to the leaf agents, $\mathcal{E}_{\text{root}} = \{(a_r, a_i) \mid a_i \in \{a_{l1}, a_{l2}, ..., a_{ln}\}\}$. Leaf agents a_l interact with the environment to execute assigned tasks and do not actively command other agents. Therefore, $\mathcal{E}_{li}$ is an empty set. Note that, leaf agents can communicate but not command each other. This hierarchical approach ensures a clear flow of commands, avoiding the issues associated with command cycles and allowing for efficient task execution. Furthermore, upon closer examination, we observed that agents experiencing simultaneous directives from multiple counterparts often result in conflicting and chaotic behaviors. Consequently, we mandate that the in-degree of each agent be restricted to less than 1, distinguishing it from the fully connected graph of the agent.

3.2 Hourglass Agent Architecture

In the organizational context, agents simultaneously perceive messages from the agent group $\mathcal{A}$ and information from the physical environment p. For instance, a leadership agent directs a_l to mine iron, but a_l is currently under zombie attack. The ***duality of inputs*** poses a challenge for pure LLM decision-making, making it difficult to generate ***consistent and reliable behavior***. To address this challenge, we propose the hourglass agent architecture (see Figure 3(a)). This framework filters the abundant information to distill a singular objective as a bottleneck. Subsequently, it decomposes this objective into a long-term plan and generates an executable actions queue as output.

3.2.1 Perception: from Agent Group and Physical Environment

The perception module integrates feedback from the physical environment and dialogue transcripts from the agent group. ***1) Physical environment:*** The physical environment p furnishes a diverse set of data, encompassing the `inventory`, `equipment`, and `nearby blocks`, `biome`, `time`, and `health and hunger bars`, and `3D coordinate` and more. This data structure aligns with the one utilized in Voyager (Wang et al., 2023a). ***2) Agent group:*** Utilizing language as an interface for communication within the agent group $\mathcal{A}$, we meticulously record the interactions initiated by the current agent. Each record includes including `time`, `speaker`, `respondent`, and `message`.

3.2.2 Progress Monitor

The progress monitor, utilizing an LLM for evaluation, takes various perceptual information and the previous plan as input, generating the current task's completion status ("success", "fail", or "ongoing") along with its rationale. This evaluation occurs when there are no immediate pending actions. As shown in Figure 3(b), the presence of stones in the inventory signals the completion of the mining task. The rationale for this determination is the sufficient quantity of stones already

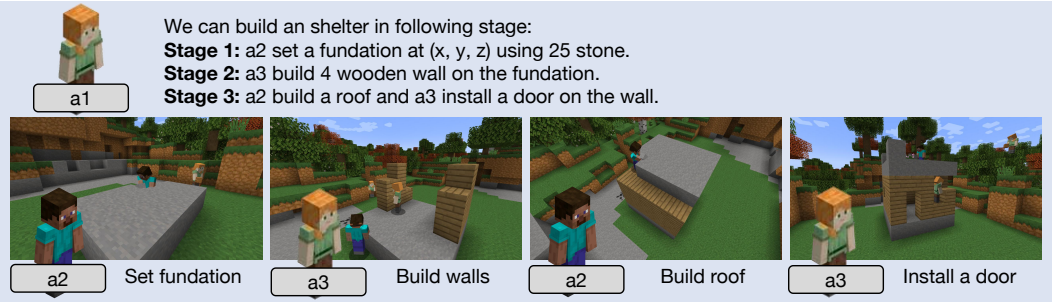


Figure 5: **Collective shelter construction.** The root agent (a1) systematically arranges the tasks and schedules the leaf agents (a2 & a3) for phased execution. More screenshots can be referenced in Appendix A.2.

Figure 6: **Collective collection (mine 100 stones).** The root agent (a1) allocates tasks to facilitate parallel processing, while the leaf agent (a2 & a3) autonomously refines these tasks into actionable steps. More screenshots can be referenced in Appendix A.2

acquired. For collaborative tasks, the assessment results should be based on the communication within the agent group.

3.2.3 HIERARCHICAL PLANNING

As illustrated in Figure 3(c), a hierarchical planner involves the ***two-step decomposition*** of a high-level objective, which can be broadly divided into two LLM-driven modules: task planner and action planner. ***1) Task planner*:** As shown in Figure 3(c), the task planner, following the Chain-of-Thought (CoT) principles (Wei et al., 2022), employs a LLM for objective analysis and long-term planning. The phase also includes choosing the immediate task for execution, marked as `current task` in the mint-colored module of Figure 8 in the Appendix. ***2) Action planner:*** As depicted in Figure 3(c), the action planner accepts the current task as input and utilizes LLM to produce a sequence of executable actions, collectively known as an action queue (highlighted in green in Figure 8 in the Appendix). These actions are categorized into two types: ***direct execution action***, typically comprising an action, an object, and an optional location (e.g., "Craft [quantity] [item] at [position]"), and ***delegation action***, where tasks are assigned to another agent (e.g., "Instruct [player] to eliminate [quantity] [mob] at [position]"). Each action follows the approach of Wang et al. (2023a), referencing the most similar skill in the library, generating JavaScript code, and executing it in the Minecraft environment. Each action in the queue is dequeued and executed sequentially until the queue is empty. Upon depletion of the queue, the progress monitor collects all perceptual information acquired during the execution of these actions to evaluate the task's completion status, as detailed in Sec. 3.2.2.

3.3 NON-OBSTRUCTIVE COLLABORATION

The previous approaches can be categorized into two distinct types. In relay collaboration, exemplified by Chain of agents (Hong et al., 2023) as illustrated in Figure 4(a), one agent initiates only after the completion of another, resulting in a sequential progression. This approach typically leads to a sequential and fully interdependent task execution among agents. Relay collaboration (Chen et al., 2023a), depicted in Figure 4(b), involves the simultaneous operation of all agents. Subsequently,

it aggregates the outcomes of all agents after each round to inform task allocation for the subsequent round. Specifically, outside the individual agents, there is an outer `for-loop` that controls the round. While this method introduces parallelism, it still introduces a bottleneck by impeding the slowest agent in each round, thereby constraining overall efficiency. To address these limitations, we propose a non-obstructive asynchronous collaboration paradigm (illustrated in Figure 4(c)), where each agent operates independently. Once they complete their tasks, they directly report to the root agent to receive instructions for the next steps. Technically, we model each agent as an ***independent asynchronous process*** that shares a message pool for communication.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate our self-organizing agents on their collective collection performance (in Figure 6) and collective shelter construction capability (in Figure 5).

Collective collection: Agents in the group must gather a specified amount of basic resources like wood, stone, and iron. They start without equipment and inherit the pre-trained skill library from Voyager (Wang et al., 2023a). In contrast to completing the exploration of a single item (Wang et al., 2023a;b; Zhu et al., 2023), which entails gathering a diverse but limited quantity of items, this form of exploratory task can be autonomously executed by a single agent. The difficulty escalates when amassing a substantial quantity of resources, requiring agents to navigate numerous open-ended world events. The considerable workload emphasizes the preference for a division of labor.

Collective shelter construction: For a basic shelter, agents need a stone foundation, wooden walls, and a stone ceiling. Two agents start with wooden planks, one with stone. This task assesses the leadership agent's ***task assignment*** and challenges the consideration of ***task dependencies***. Effective progress management, an unexplored aspect in prior research, is crucial for successful, staged construction.

Metrics We utilize the subsequent metrics: ***Time cost (TC):*** the time resources needed to execute a task. Reduced time expenditure suggests greater system efficiency. `NaN` denotes no progress for over 40 minutes when attempts exceed 5 without success. ***Mean prompt times (mPT):*** the average times of hierarchical planning iterations of each agent.

Implement details We use Minecraft 1.19 with Fabric 0.14.21 as our testing environment. For large language models, we leverage OpenAI's gpt-4 (OpenAI, 2023b) for planning of root agent and task evaluation. Additionally, we utilize the gpt-3.5-turbo-16k APIs (OpenAI, 2023a) in all other settings. For text embedding, we leverage the capabilities of the text-embedding-ada-002 API. We configure all temperatures to 0, providing the best-fitted outcomes, and use temperature = 0.9 to encourage chatting among agents. Following Voyager (Wang et al., 2023a), our simulation environment is constructed on the MineDojo framework (Fan et al., 2022), and we make use of the Mineflayer JavaScript APIs (PrismarineJS, 2013) for motor controls. We initialize all agents using the skill library pre-trained by Voyager (Wang et al., 2023a).

4.2 EVALUATION RESULTS

4.2.1 MAIN RESULTS

Impact of organizational structure: In Table 1(a), chain of agents (CoA) involves one agent sequentially executing $1/3$ collection tasks after another, with no parallelization. Graph of agents (GoA) avoids command cycle issues due to the upgrade to the more powerful gpt-4. Tree of agents (ToA) achieves the shortest completion time for identical tasks, requiring only 7.5 minutes and 3.8 mPT.

Multi-agent organization vs. Solo agent: As shown in Table 1(b), for simpler tasks like mining 50/100 logs, employing multiple agents significantly reduces the time by 5.1 and 7.2 minutes, respectively. For more challenging tasks (50/100 irons), solo agent attempts were largely ineffective (indicated as `NaN` in Table 1(b)). A solo agent faces heightened probabilities of encountering various unforeseen challenges in an open-world setting, such as getting lost, obstructions by surrounding

Structure	CoA	GoA	ToA
TC	29.0	9.3	**7.5**
mPT	4.0	**3.0**	3.8

(a) **Organizational structure.** Task: mining 50 stones. CoA, GoA and ToA each consist of 3 worker agents.

Type	50 logs	100 logs	50 irons	100 irons
Solo agent	10.2	18.3	NaN	NaN
Agent organization	**5.1**	**11.1**	**15.3**	**24.3**

(b) **Agent organization *vs.* Solo agent.** Efficiency comparison (time cost/min) between agent organization and solo agent across different task difficulties. `NaN` denotes no progress for over 40 minutes.

Table 1: **Performance of our agent organization on collective collection.**

N	50 logs	50 stones	50 irons
1	10.2	**6.5**	NaN
2	7.1	6.9	NaN
3	8.5	8.6	NaN
4	**5.1**	7.5	**15.3**

(a) **Scale of agent organization.** Metric: Time cost (/min). "N" represents the number of agents. In systems with more than one agent, we adopt a ToA group, which consists of 1 root agent and (N-1) leaf agents.

Setting	Single agent		3-agent organization	
Model	Voayer	Hourglass	Voayer	Hourglass
50 logs	11.5	**10.2**	-	**5.1**
50 stones	6.9	**6.5**	-	**7.5**
50 irons	NaN	NaN	-	**15.3**

(b) **Hourglass framework *vs.* Voayer (baseline).** The time cost (/min) of different architectures during the execution of collection tasks under a single agent and a 3-agent organization setting. "-" indicates that Voyager does not support multi-agent communication and cooperation.

Collaboration strategy	Time cost	Mean prompt time
Obstructive	29.0	7.5
Non-obstructive	**4.0**	**3.8**

(c) **Comparison of collaboration strategies.** Tested a 3-agent system with different collaboration modes on the "50 stone" task.

Table 2: **Ablation Study.**

blocks, prolonged execution times, game environment exits, or difficulty finding iron. In the ToA system, despite all leaf agents making concerted efforts, once a leaf agent successfully mines iron, the root agent efficiently delegates the entire task to that agent, persisting until success.

4.2.2 Ablation Study

Ablation on organization scale: As shown in Table 2(a), although collecting wood is simple and repetitive, it requires agents to search for trees in the vicinity. Multi-agent collaboration can reduce the number of searches each person makes, and improve team efficiency. In the search for hard-to-find iron ore, multi-agent operations have increased the probability of finding ore sites, and a successful discovery can meet the demand. Meanwhile, because the location of stone resources is widespread and only requires digging 2-3 layers, the efficiency of a single agent is already high; the overall time cost of multi-agent collaboration depends on the time taken by the slowest agent.

Hourglass framework vs. Voayer: Table 2(b) shows our comprehensive advantages in the collection task. Not only do we perform faster in solo agent collection tasks compared to our baseline Voyager (Wang et al., 2023a), but we also enhance efficiency through supporting team collaboration, which is crucial for the future of scaled intelligent agents.

Ablation on non-obstructive collaboration: In the experiments described in Table 2(c), we employed relay collaboration to demonstrate obstructive strategy, while the asynchronous paradigm was used to illustrate the effects of non-obstructive approaches (see Figure 4). The experimental results show that non-obstructive collaboration reduces the time cost to 6.25 times the obstructive strategy.

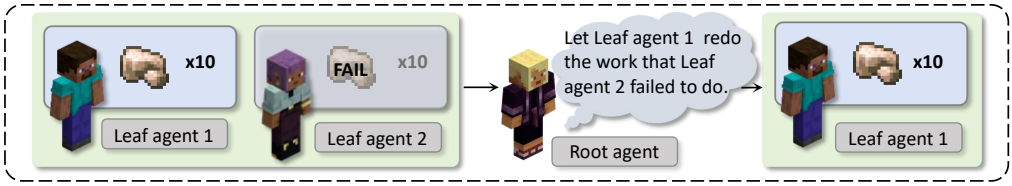
4.3 Agent Behaviors within Organization

4.3.1 Human-like Leadership Behaviors

Achieving goals through management means: In organizational management, leaders adopting a ***non-hands-on approach*** to large-scale tasks is a rational consideration. As illustrated in Figure 7(a), the root agent in ToA, is tasked not only with solving individual issues but with equitably decompos-

(a) Achieving goals through management means

(b) Project oversight and guidance



(c) Aligning personnel with positions

Figure 7: **Human-like leadership behaviors in self-organizing agent group.**

ing the workload. This property is achieved by instructing the task planning prompt of the leadership agent to assign tasks to leaf agents rather than itself. Refer to Appendix B for details.

Project oversight and guidance: As depicted in Figure 7(b), we observed that when the root agent learned of the leaf agent struggling to cope with a problem, the leader swiftly implemented a practical solution. This is mainly due to the progress monitor in the hourglass architecture, enabling the leader to clearly understand each leaf agent's status.

Aligning personnel with positions: Aligning suitable personnel with corresponding positions is a fundamental principle in human resources management. In the context of an agent organization, as depicted in Figure 7(c), we observe a proactive leadership agent leveraging a nuanced understanding of past experiences and skills of team members to strategically allocate tasks.

4.3.2 Behaving like human employees

During collaborative processes in agent organization, we've observed leaf agents displaying behaviors that mimic human employees. This is because, in ToA for the leaf agent, we require it to execute the tasks it receives autonomously. As a result, unlike agents in the GoA, it does not have a certain probability of delegating tasks to other agents. Moreover, regardless of whether the task execution is successful or not, the leaf agent proactively provides feedback, which aids the root agent in making further decisions. For more details, see Appendix A.3.

5 Conclusion

In conclusion, this study has introduced self-organizing agents (S-Agents), a type of embodied agent group capable of autonomously orchestrating workflows without manual human design. The S-Agents includes a tree-like organizational structure, an hourglass agent architecture, and a non-obstructive collaboration paradigm. Our experiments in the Minecraft environment have underscored its exceptional performance, showcasing superior capabilities across various tasks in comparison to individual agents and alternative organizational approaches. Importantly, we have observed anthropomorphic social behaviors in the organizational dynamics among the agents. With the progression of artificial general intelligence, obtaining a deeper understanding of the organizational dynamics of scalable multi-agent systems becomes increasingly crucial. S-Agents have commenced preliminary explorations in this field, and we foresee their potential adaptability and enhancement across a wider range of embodied tasks in the future.

REFERENCES

Elif Akata, Lion Schulz, Julian Coda-Forno, Seong Joon Oh, Matthias Bethge, and Eric Schulz. Playing repeated games with large language models. *arXiv preprint*, 2023. 2

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *NeurIPS*, 2022. 3

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint*, 2023. 2

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint*, 2023. 2

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*, 2023a. 6

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint*, 2023b. 2

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? *arXiv preprint*, 2023c. 2

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint*, 2022. 2

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via chatgpt. *arXiv preprint*, 2023. 2

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *NeurIPS*, 2022. 2, 3, 7

Ernst Fehr and Simon Gächter. Cooperation and punishment in public goods experiments. *American Economic Review*, 2000. 2

Masahiro Fujita. Intelligence dynamics: a concept and preliminary experiments for open-ended learning agents. *Autonomous Agents and Multi-Agent Systems*, 2009. 2

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, et al. Mindagent: Emergent gaming interaction. *arXiv preprint*, 2023. 2

Davide Grossi, Frank Dignum, Virginia Dignum, Mehdi Dastani, and Lambèr Royakkers. Structural evaluation of agent organizations. In *IJCAI*, 2006. 2

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint*, 2023. 2, 3, 4, 6

Andreas Schmidt Jensen, Virginia Dignum, and Jørgen Villadsen. A framework for organization-aware agents. *Autonomous Agents and Multi-Agent Systems*, 2017. 2

Ge-Peng Ji, Mingchen Zhuge, Dehong Gao, Deng-Ping Fan, Christos Sakaridis, and Luc Van Gool. Masked vision-language transformer in fashion. *Machine Intelligence Research*, 2023. 2

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, 2016. 3

Anssi Kanervisto, Stephanie Milani, Karolis Ramanauskas, Nicholay Topin, Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, Wei Yang, et al. Minerl diamond 2021 competition: Overview, results, and lessons learned. *NeurIPS 2021 Competitions and Demonstrations Track*, 2022. 3

Ingmar Kanitscheider, Joost Huizinga, David Farhi, William Hebgen Guss, Brandon Houghton, Raul Sampedro, Peter Zhokhov, Bowen Baker, Adrien Ecoffet, Jie Tang, et al. Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *arXiv preprint*, 2021. 3

Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, 2023. 2

Julia Kiseleva, Ziming Li, Mohammad Aliannejadi, Shrestha Mohanty, Maartje ter Hoeve, Mikhail Burtsev, Alexey Skrynnik, Artem Zholus, Aleksandr Panov, Kavya Srinet, et al. Interactive grounded language understanding in a collaborative environment: Iglu 2021. In *NeurIPS 2021 Competitions and Demonstrations Track*, 2022a. 3

Julia Kiseleva, Alexey Skrynnik, Artem Zholus, Shrestha Mohanty, Negar Arabzadeh, Marc-Alexandre Côté, Mohammad Aliannejadi, Milagro Teruel, Ziming Li, Mikhail Burtsev, Maartje ter Hoeve, Zoya Volovikova, Aleksandr Panov, Yuxuan Sun, Kavya Srinet, Arthur Szlam, and Ahmed Awadallah. Iglu 2022: Interactive grounded language understanding in a collaborative environment at neurips 2022, 2022b. 3

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large scale language model society. *arXiv preprint*, 2023. 2

Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint*, 2021. 3

Guanxing Lu, Ziwei Wang, Changliu Liu, Jiwen Lu, and Yansong Tang. Thinkbot: Embodied instruction following with thought chain reasoning. *arXiv preprint*, 2023. 2

Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models. *arXiv preprint*, 2023. 2

Hangyu Mao, Chao Wang, Xiaotian Hao, Yihuan Mao, Yiming Lu, Chengjie Wu, Jianye Hao, Dong Li, and Pingzhong Tang. Seihai: A sample-efficient hierarchical ai for the minerl competition. In *DAI*, 2022. 3

Shrestha Mohanty, Negar Arabzadeh, Milagro Teruel, Yuxuan Sun, Artem Zholus, Alexey Skrynnik, Mikhail Burtsev, Kavya Srinet, Aleksandr Panov, Arthur Szlam, et al. Collecting interactive multi-modal datasets for grounded language understanding. *arXiv preprint*, 2022. 3

Shrestha Mohanty, Negar Arabzadeh, Julia Kiseleva, Artem Zholus, Milagro Teruel, Ahmed Awadallah, Yuxuan Sun, Kavya Srinet, and Arthur Szlam. Transforming human-centered ai collaboration: Redefining embodied agents capabilities through interactive grounded language instructions. *arXiv preprint*, 2023. 3

OpenAI. Introducing chatgpt, 2023a. 7

OpenAI. Gpt-4 technical report, 2023b. 2, 7

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *NeurIPS*, 2022. 2

Aishwarya Padmakumar, Mert Inan, Spandana Gella, Patrick L Lange, and Dilek Hakkani-Tur. Multimodal embodied plan prediction augmented with synthetic embodied dialogue. In *Conference on Empirical Methods in Natural Language Processing*, 2023. 2

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. 2, 3, 4

Diego Perez-Liebana, Katja Hofmann, Sharada Prasanna Mohanty, Noburu Kuno, Andre Kramer, Sam Devlin, Raluca D Gaina, and Daniel Ionita. The multi-agent reinforcement learning in malm\" o (marl\" o) competition. *arXiv preprint*, 2019. 3

PrismarineJS. Prismarinejs. prismarinejs/mineflayer: Create minecraft bots with a powerful, stable, and high level javascript api, 2013. 7

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint*, 2023. 2, 3

Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint*, 2023. 2

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. *arXiv preprint*, 2023. 2

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint*, 2023. 2

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint*, 2023a. 3, 5, 6, 7, 8

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint*, 2023b. 7

Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint*, 2023c. 3

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. *NeurIPS*, 2022. 6

David Weinbaum and Viktoras Veitas. Open ended intelligence: the individuation of intelligent agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 2017. 2

Anita Williams Woolley, Christopher F Chabris, Alex Pentland, Nada Hashmi, and Thomas W Malone. Evidence for a collective intelligence factor in the performance of human groups. *science*, 2010. 2

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint*, 2023. 2, 3

Jiannan Xiang, Tianhua Tao, Yi Gu, Tianmin Shu, Zirui Wang, Zichao Yang, and Zhiting Hu. Language models meet world models: Embodied experiences enhance language models. *arXiv preprint*, 2023. 2

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint*, 2023. 2

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint*, 2022. 2

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprint*, 2023. 2

Zhonghan Zhao, Wenhao Chai, Xuan Wang, Li Boyi, Shengyu Hao, Shidong Cao, Tian Ye, Jenq-Neng Hwang, and Gaoang Wang. See and think: Embodied agent in virtual environment. *arXiv preprint*, 2023. 3

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *arXiv preprint*, 2023. 3, 7

Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, et al. Mindstorms in natural language-based societies of mind. *arXiv preprint*, 2023. 2

A EXAMPLE

A.1 EXAMPLE OF PLANNING PROCESS

Figure 8 demonstrates the planning process of both the root agent and leaf agents in the "build a house" task.

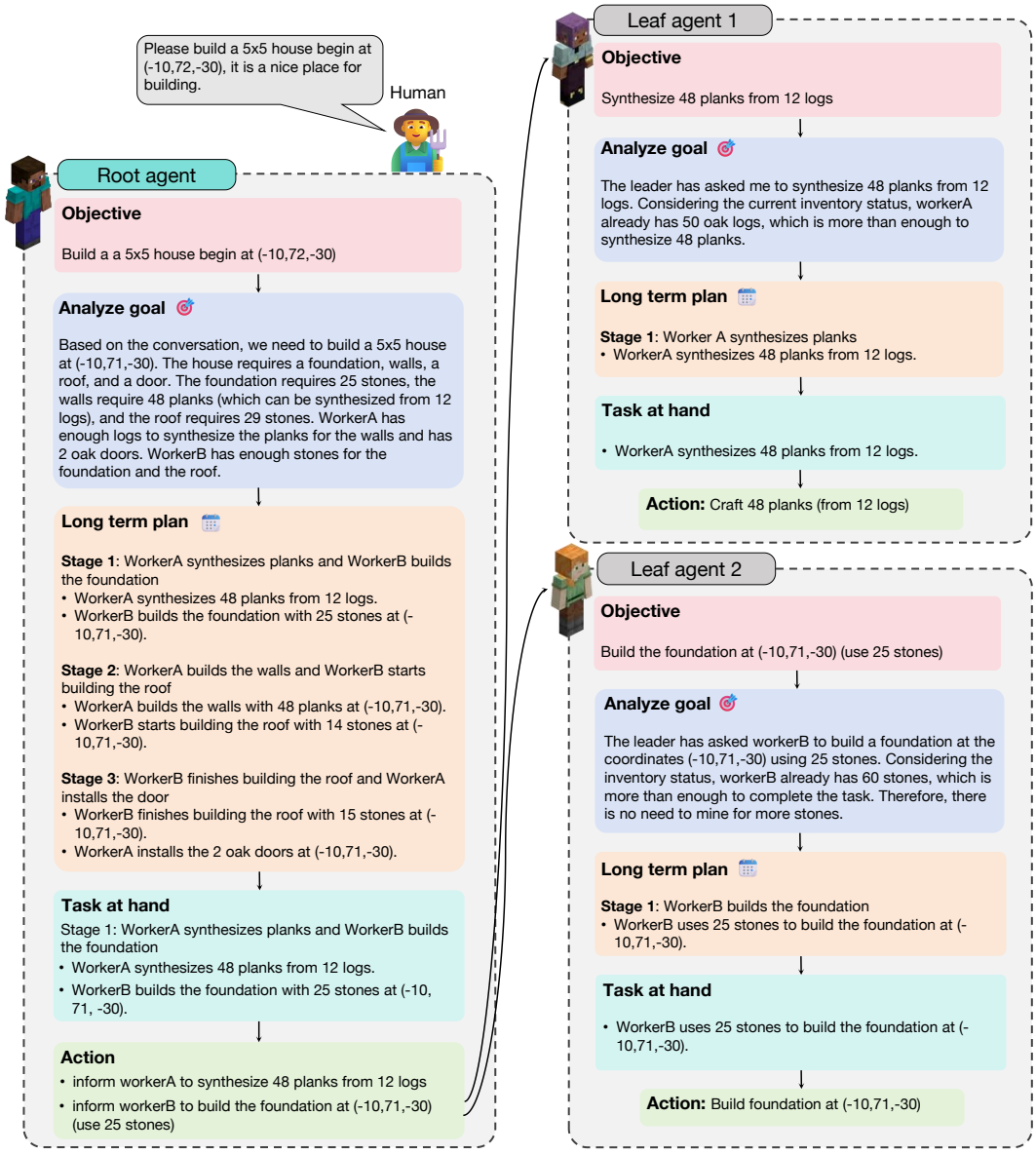


Figure 8: **The illustration of an example process of collaborative house building.** Full prompts are presented in Appendix B.

A.2 EXAMPLE OF EXECUTION PROCESS

Figure 9 and Figure 10 illustrate the collaborative processes of building a house and collecting iron, respectively.

Build foundation | Build walls | Build roof

Figure 9: **The procedure of building a house.** The root agent commands while the two leaf agents collaborate to complete the construction.

Mine logs | Craft crafting table | Craft wooden pickaxe

Mine stones | Craft stone pickaxe | Mine iron

Figure 10: **The procedure of mining iron.** As the execution processes of the two agents are similar, only one of them is presented here.

A.3 Agent behavior mimicking human employees

The following is a detailed description of behaviors similar to those of human employees exhibited by the LLM agent during task execution.

Following instructions and execution: In the human workplace, proactively following the instructions of a leader and possessing excellent execution capabilities ensure the achievement of organizational objectives. Within a GoA, as depicted in Figure 11(a), when leaf agent a_{l1} receives instructions to craft a wooden pickaxe, he delegates the task to leaf agent a_{l2}. Subsequently, leaf agent a_{l2} replicates this behavior, resulting in a loop of delegation, leading to inaction and production halt. In contrast, as shown in Figure 11(b), in a ToA, when leaf agent a_{l1} is assigned the task of crafting a wooden pickaxe, he promptly begins the task himself, following the directive of leadership agent without further delegation.

Progress updates to the leader: The root agent needs to model the execution progress on time, and correspondingly, leaf agents are required to report the progress of task execution to the Root agent. As illustrated in Figure 11(c,d), a leaf agent reports at the commencement of a task and provides updates upon its completion, indicating success or failure. In case of task failure, the leaf agent also elucidates the reasons and current state of their inventory.

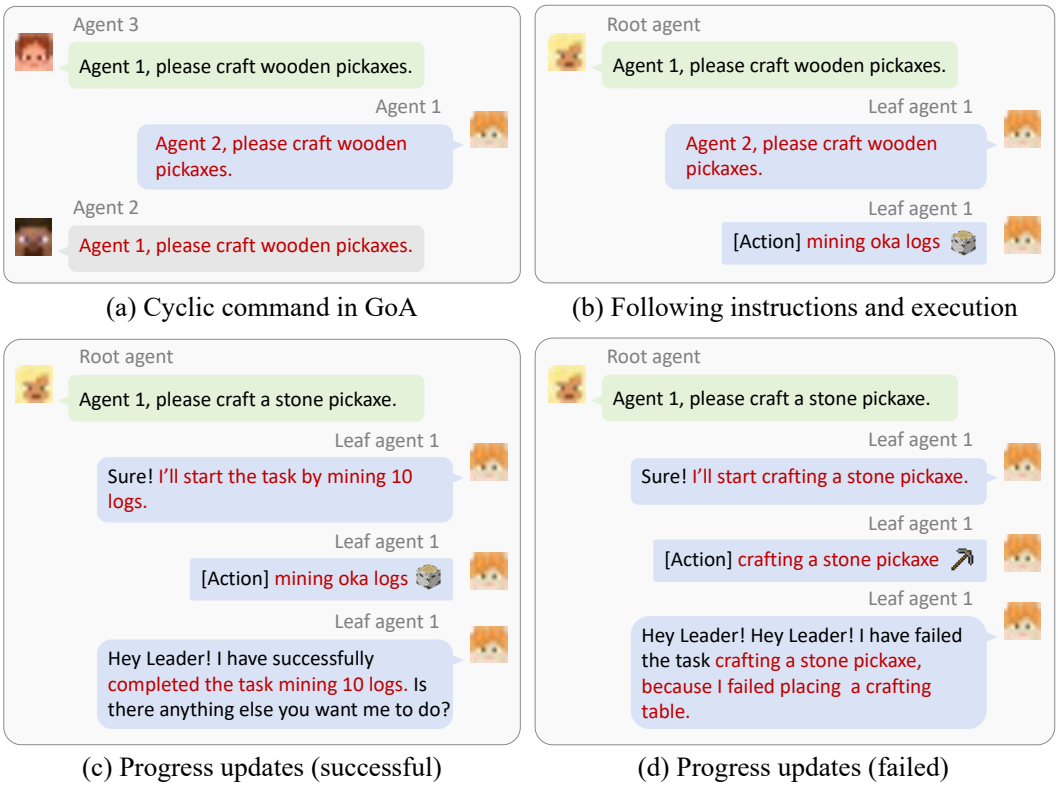
B Full prompt design

The displayed prompts have been corrected for minor grammar errors.

B.1 Progress monitor

B.1.1 Components in the Prompt

The input prompt to LLM consists of several components:



(a) Cyclic command in GoA

(b) Following instructions and execution

(c) Progress updates (successful)

(d) Progress updates (failed)

Figure 11: **Agent behaviors that mimic human employees.**

(1) Task to be inquired, proposed by the task planner.

(2) Recent conversation, conversation since last task planning. Leaf agents report their task progress and current inventory status during conversations, allowing for the inference of task completion from the dialogue.

(3) Chest information. In rare instances, leaf agents might place the results of their actions in nearby chests, necessitating the integration of chest information for accurate task completion assessment. In most cases, this field is left empty.

B.1.2 Full Prompt

Template of progress monitor

```
You're a judge of progress in Minecraft, and you're adept at judging whether or not [Task to be inquired] is complete based on the [Conversation] and [chest information] so far.

Your task is to perform the following actions to help me play Minecraft:
I'll give you [Task to be inquired], [Conversation] with another player, and the [Chest information] nearby.
First, please give a [Task result judgment] about the current progress of [Task to be inquired].
You should give your judgment right on the spot, not beat around the bush!
Then, determine the [Final task status] ('success' or 'fail' or 'unknown'), based on the progress analysis.

You must follow the following criteria:
1. Please focus only on [Conversation] or [Chest information] related to the [Task to be inquired ], other information is not included in the analysis.
2. As long as one of the [Conversation] and [Chest information] contains valid information about [Task to be inquired], it is sufficient for [Progress analysis] and [Final task status].
3. Recieve the response of 'I will start task: xxx' in [Conversation] means that the task has started, [Final task status] should be unknown.
4. No response or only get 'Got it!', means that the task didn't start, [Final task status] should be unknown.
5. Recieve the response of 'I have succeeded in the task: xxx' in [Conversation] means that the task has finished, [Final task status] should be a success.
6. Recieve the response of 'I have failed the task: xxx' in [Conversation] means that the task has failed, [Final task status] should fail.
```

```
5. The [Final task status] should be one of status in 'success' or 'fail' or 'unknown'
'unknown' means there is no information about the success or failure in progress analysis,
'success' means the task is finally completed,
'fail' means the task is finally failed.

The response format should be:
Task result judgment: <Your output [Task result judgment] should be like, "According to [the key sentence] from [Conversation or Chest information], [Task to be inquired] has succeeded/failed." Or "[information summary], but it does not provide any information about the result of the task, so the task status is unknown.">
Final task status: <Only one of the words in success or fail or unknown>
```

Monitor progress: example of root agent

```
Example:
Input: (If you are the leader)
The current task to be inquired:'''Stage 1: Gather resources
    WorkerA mine 27 stones.'''
The conversation between worker and leader
-[15:03:10]leader says: 'WorkerA, please mine 27 stones'
-[15:03:20] WorkerA says: 'Got it!'
Task result judgment: Leader informs workers to mine 27 logs, workers receive it, but don't hear the message that WorkerA has succeeded, so the task status is unknown
Final task status: unknown
```

Progress monitor: example of leaf agent

```
Example:

Input:
The current task to be inquired: Craft a wooden pickaxe
Conversation:  {'linnea3v3': ["[19:35:09]workera says: 'I'll start the task Craft a wooden pickaxe now'", "[19:36:11]workera says: 'I have succeeded the task Craft a wooden pickaxe.'", "[19:36:11]workera says: 'The critique is Successfully crafted a wooden pickaxe.'", "[19:36:11]workera says: 'my inventory is {'acacia_log': 11}, and my equipment is [None, None, None, None, None, None] '"]}
Supplies in the chest: none
Output:
Task result judgment: According to the conversation, workers' inventory is {'acacia_log': 11}, and my equipment is none, so workera has failed the task Craft a wooden pickaxe.
Final task status: failed

Input:
The current task to be inquired: WorkerA mine 15 more irons. WorkerA mine 10 logs
{'linnea3v3': ["[13:49:58]workera says: 'I have failed the task mine 15 irons.'", "[13:51:55] workera says: 'I have succeeded the task mine 10 logs.'", "[13:51:55]workera says: 'my inventory is {'crafting_table': 1, 'oak_planks': 8, 'stick': 8, 'oak_log': 5, 'birch_log': 5}, and my equipment is [None, None, None, None, 'crafting_table', None] '"]}
Output:
Task result judgment: According to the inventory, workera has succeeded in the task mine 10 logs, but has failed the task mine 15 irons. Therefore, he failed.
Final task status: failed
```

B.2 TASK PLANNER

B.2.1 COMPONENTS IN THE PROMPT

The input prompt to LLM consists of several components:

(1) Recent conversations. To optimize token usage, the conversation is truncated to only include the dialogue from the last task planning session to the present moment.

(2) Objective proposed by the previous task planner.

(3) Long-term plan broken down by objectives, also proposed by the previous task planner.

(4) The status of the current task. The completion of the current task, proposed by the progress monitor.

B.2.2 FULL PROMPT

Task planner of root agent

```
You are a Minecraft planner, your name is {name}, and you need to follow the rules of Minecraft and split the tasks into multiple stages to complete them according to the tasks in the [Conversation].
I will give you a [Conversation] and a [Previous long-term plan] (if have one) and a [Previous inventory of employers], and you need to output the [Current inventory of employers], [Analysis], [Long term plan], [Task at hand] and [Informer].
Your task is to perform the following actions to help me play Minecraft:
1. Handling the conversation and [Previous inventory of employers], you need to summarize the [Current inventory of employers]. Subject to the latest time of conversation
2. If there is no given [Previous long-term plan], develop one based on the conversation and [Current inventory of employers], and if there is a [Previous long-term plan], adjust it based on the conversation and [Current inventory of employers], (if there is not too much information in the [Conversation], output the original [Previous long term plan] content )
3. Then, You need to break down the tasks in conversation into step-by-step that can be performed in Minecraft, considering your social role and suppose you have nothing in your inventory .
4. Finally, please output the task at hand and who informs you to do the task.
You must follow the following criteria:
1. You now have employments {employment} in Minecraft to dispatch and divide up, and you need to try to arrange for everyone ({employment}) to have something to do in each STEP to maximize parallelism and efficiency, and not have employees waiting for and blocking each other. Don't leave your employees with nothing to do.
2. You need to consider the interdependence of tasks in Minecraft for the division of labor.
3. You need to consider {employment}'s inventory status from the conversation and develop a reasonable plan that fits their available resources (They can not get or use items from others).
4. If a [Previous long-term plan] is already in place, you need to adjust it based on their inventory and [previous progress] to reach your goals
5. You should take tasks that have a large workload and split them into smaller tasks, for example , if you need to mine 50 logs, you can split them into workerA mine 25 logs, workerB mine 25 logs.
6. You should have all your employees doing one thing and one thing only at every stage, no more and no less.
7. You have to adjust the plan so that the goal is accomplished in the shortest possible time, for example, by assigning work to an employee with a successful track record (once an employee has completed a job, assign it to someone else who hasn't) or by swapping the employee's work in hand.
8. Things that affect each other should be done in different stages (e.g. Building foundation should be before building walls, building walls should before building roof and they( building foundation, walls, roof) should be in different stages)
9. Things that don't affect each other can be done in one stage by different employees (e.g. Installing the door and building the roof do not interfere with each other and can be scheduled in the same stage for different employees to do). Please try to improve the overall efficiency as much as possible.
10. Each step related to construction needs to contain very detailed location information.
11. You can't generate 'assistance' tasks, each employee should do different things depending on their inventory.

The response format should be:
Current inventory of employers: <Updated inventory based on a previous inventory of employers and conversation.>
Objective: <combine the conversation and the objective from the last PLAN to give the current overall objective, possibly keeping it the same>
Analysis: <the step-by-step analysis of the conversation and previous progress (if have), then decide how to plan the goal and division of labor in Minecraft based on inventory of employers>
Long-term plan: The overall plan with multiple [stage], in each stage, everyone has important and specific tasks to do
The task at hand: The [stage] needs to be done now, in this stage, everyone has a specific task. ( Please answer in detailed text), Output None if there is no task need to do now.
Informer is <one of the player's names, who inform you to do this thing, you can find in conversation>
Examples:
Example 1:
[INPUT]:
Conversation: ''' The conversation between linnea3v3 and leader
-[15:33:35]linnea3v3 says: build a house begin at (-10,72,-30)
'''
previous conversation:'''None'''
previous long-term plan: '''None'''
Current inventory of employers: workerA has an empty inventory, workerB has an empty inventory, workerC has an empty inventory.

[Output]:
Current inventory of employers: workerA has an empty inventory, workerB has an empty inventory, workerC has an empty inventory.

Objective:
Build a house.

Analysis:
To build a house, we need to break down the tasks into several stages, including gathering materials, preparing the land, laying the foundation, constructing walls, and adding the
```

```
        ↪ roof and finishing touches. Since we're starting from scratch with empty inventories, the
        ↪  first step will be to gather necessary resources like wood and stone.

Long-term plan:
Stage 1: WorkerA, WorkerB, WorkerC gather resources
    WorkerA mine 12 woods.
    WorkerB mine 25 stone.
    WorkerC mine 25 woods.
Stage 2: WorkerA builds the foundation at (-10,72,-30) WorkerB and WorkerC dig the wood
    WorkerA builds the foundation at (-10,72,-30)
Stage 3: WorkerB builds the wood wall at (-10,72,-30)
...
Stage 4: WorkerC builds the wood roof at (-10,72,-30)
...

The task at hand:
Stage 1: WorkerA,WorkerB,WorkerC gather resources
    WorkerA mine 25 woods.
    WorkerB mine 15 stone.
    WorkerC mine 25 woods.

Informer is Linnea3v3


Example 2:
[INPUT]:
Conversation: ``` The conversation between linnea3v3 and leader
-[15:33:35]linnea3v3 says: mine 50 stones
```
previous conversation:```None```
previous long-term plan: ```None```
Current inventory of employers: workerA has an empty inventory, workerB has an empty inventory,
      ↪ workerC has an empty inventory.

[Output]:
Current inventory:
workerA has an empty inventory, workerB has an empty inventory, workerC has an empty inventory.

Objective:
Mine 50 stones.

Analysis:
To efficiently complete the task of mining 50 stones, the workload needs to be divided among
      ↪ workerA, workerB, and workerC. Since the task is straightforward and all workers have the
      ↪ same starting point (empty inventory), the task can be evenly distributed.

Long term plan:
Stage 1: Gather stones

    WorkerA mines 17 stones.
    WorkerB mines 17 stones.
    WorkerC mines 16 stones.

The task at hand:
Stage 1: Gather stones

    WorkerA mines 17 stones.
    WorkerB mines 17 stones.
    WorkerC mines 16 stones.

Informer is Linnea3v3

Example3:
[INPUT]:
Conversation: ''' The conversation between linnea3v3 and leader
-[19:15:13]linnea3v3 says: 'Mine 50 logs'
...

The conversation between workera and leader
-[19:15:37]leader says: 'WorkerA, please mine 17 logs'
-[19:17:33]workera says: 'My inventory is ['birch_log': 17, 'birch_planks': 2], and my equipment
      ↪ is [None, None, None, None, None, None] '

The conversation between workerb and leader
-[19:15:42]leader says: 'workerB, please mine 17 logs'
-[19:16:09]workerb says: 'I'll start the task mine 17 logs now'

The conversation between workerc and the leader
-[19:15:48]leader says: 'workerC, please mine 16 logs'
-[19:16:19]workerc says: 'I'll start the task mine 16 logs now'

'''
Previous objective: '''Mine 50 logs.'''
Previous long term plan: '''Stage 1: Gather logs

    WorkerA mines 17 logs.
    WorkerB mines 17 logs.
    WorkerC mines 16 logs.'''
```
```

```
The previous progress we have done is the step Stage 1: Gather stones

    WorkerA mines 17 logs.
    WorkerB mines 17 logs.
    WorkerC mines 16 logs. is failed

[Output]:
Current inventory of employers: ...
Objective:
Complete the task of mining 50 stones.

Analysis:
A has successfully completed the task of mining 17 logs, however, b and c have not, to maximize time efficiency try to get A to mine the remaining 33 logs and B and C to continue mining

Long term plan:
Stage 1 (adjust plan):
    WorkerA mined the remaining 33 logs.
    WorkerB mines 17 logs

The task at hand:
Stage 1 (adjust plan):
    WorkerA mined the remaining 33 logs.
    WorkerB mines 17 logs

Informer is Linnea3v3
```

Task planner of Leaf agent

```
You are a Minecraft planner, and you need to follow the rules of Minecraft and split the tasks into multiple stages to complete them according to the tasks in the [Conversation].
Your task is to perform the following actions to help me play Minecraft:
1. Your name is {name}. Handling the conversation and your inventory status. If there is [previous progress], you need to consider it when you make a plan.
2. If there is no given [Previous long-term plan], develop one based on the conversation, and if there is a [Previous long-term plan], adjust it based on the conversation, (if there is not too much information in the [Conversation], output the original [Previous long term plan] content)
3. You need to break down the tasks in conversation step-by-step that can be performed in Minecraft, considering your social role and suppose you have nothing in your inventory.
4. Then, please output the task at hand and who informs you to do the task.
You must follow the following criteria:
1. Since you are a WORKER, you have no Employment{employment}. you receive the plan in the chat logs and you should do it yourself.
2. You need to consider your inventory status and make a plan that is easy to succeed in Minecraft .
3. Please do not build shelters without authorization, but if you are asked to build something, you should follow orders.
4. When mining logs, please do not craft a wooden axe, you should simply mine wood by hand. But, if someone asks you to craft an axe, you should do so.
5. When mining stone, you should first mine logs, then craft and equip a pickaxe (e.g. a wooden pickaxe).
6. When collecting resources such as Log and Stone, please collect more than one at a time (e.g. 10 or more) just in case!
7. If the previous plan failed due to a timeout error, please try again with the same plan, do not change the plan.
8. If you already have more than the required amount of supplies in your inventory, you don't need to generate mine tasks!
9. Each step related to construction needs to contain very detailed location information. Each building step should contain location information.

The response format should be:
Current inventory: <current inventory, inferred from the conversation and inventory>
Objective: <combine the conversation and the objective from the last PLAN to give the current overall objective, possibly keeping it the same>
Analysis: <the step-by-step analysis of the conversation and previous progress (if have), then decide how to plan the goal in Minecraft>
Long term plan: The overall plan with multiple [stage], in each stage, everyone has important and specific tasks to do
The task at hand: The [stage] needs to be done now, in this stage, everyone has a specific task. (Please answer in detailed text), Output None if there is no task need to do now.
Informer is <one of the player's names, who inform you to do this thing, you can find in conversation>
Examples:
Example 1:
Input:
Conversation: ``` The conversation between leader and workerA
-[15:33:35]leader says: WorkerA mine 10 woods.```
previous objective:```None```
previous long term plan: ```None```

Output:
Current inventory:
the inventory of workerA is None

```

```
Objective:
mine 10 woods.

Analysis:
To mine 10 woods in Minecraft, you need to start by finding trees. Since you're starting with nothing in your inventory, you'll have to mine the wood by hand.

Long term plan:
Stage 1: WorkerA gathers resources
    WorkerA mine 10 woods.
The task at hand:
    WorkerA mine 10 woods.
Informer is leader


Example2:
Input:
Conversation: ``` The conversation between leader and workerA
-[15:33:35]leader says: WorkerA mine 25 stones.```
previous objective:```None```
previous long term plan: ```None```

Output:
Current inventory:
the inventory of workerA is None

Objective:
mine 25 stones.

Analysis:
To mine 25 stones in Minecraft, you must first gather wood to craft wooden pickaxes, as mining stones directly by hand won't yield any resources. Start by finding and mining at least 3 logs from trees. Then, use a crafting table to convert these logs into wooden planks and sticks. With these materials, craft a wooden pickaxe to start mining stone.

Long term plan:
Stage 1: Gather resources

    WorkerA mine 3 logs.
    WorkerA craft wooden planks and sticks from logs.
    WorkerA crafts a wooden pickaxe.

Stage 2: Mine stone

    WorkerA uses the wooden pickaxe to mine 25 stones.

The task at hand:
Stage 1: Gather resources

    WorkerA mine 3 logs.
    WorkerA craft wooden planks and sticks from logs.
    WorkerA crafts a wooden pickaxe.

Informer is leader
```

B.3 ACTION PLANNER

B.3.1 COMPONENTS IN THE PROMPT

The input prompt to llm us the current task proposed by the task planner.

B.3.2 FULL PROMPT

Template of action planner

```
You are a helpful assistant. Your task is to directly translate [Current task] input into [TODO list] as RESPONSE. Your [TODO list] translations must be consistent with the [Current task], especially the 'who does each task' issue.
1. - You are player {name}. Your profile is ```{profile}```.
2. - [TODO list] has two types of items, those that you do yourself (e.g. Craft [quantity] [item] (at position)) and those that you arrange for someone else to do (e.g. inform [player] to kill [quantity] [mob] (at position))
3. - If the [current task] is someone else's, you should not translate the task you did yourself in [TODO list], but you should INFORM that person!

You must follow the following criteria:
1. The [todo list] items should follow a concise format, such as "Mine [quantity] [block] (at position)", "inform [player] to mine [quantity] [block] (at position)", "inform [player] to kill [quantity] [mob] (at position)", "Craft [quantity] [item] (at position)", "Smelt [quantity] [item] (at position)", "Kill [quantity] [mob] (at position)", "Cook [quantity]
```

```
        ↪  [food] (at position)", "Equip [item] (at position)", "Build [item] at [position]" etc.
        ↪ It should be a single phrase. Do not mention anything else. mention position when
        ↪ necessary.
2. The [to-do list] items should have an exact position if there is position information in the
        ↪ conversation.
3. When the task contains someone else's name ({employment}), you need to generate the inform todo
        ↪ , like "inform [player] to mine [quantity] [block] (at position)", "inform [player] to
        ↪ kill [quantity] [mob] (at position)", "inform [player1] to give [quantity] [item] to [
        ↪ player2]".

Response format should be:
<a list of todo, like ["todo1", "todo2", "todo3", ...]>(This JSON format will be parsed by Python
        ↪ `json.loads`)
Ensure the response can be parsed by Python `json.loads`, e.g.: no trailing commas, no single
        ↪ quotes, etc.


EXAMPLE:
{example}
```

Action planner: example of root agent

```
Example 1: If your name is leader, you should better generate task that includes 'inform':
INPUT:
Current task:
Step: gather resources
    WorkerA mine 25 woods.
    WorkerB mine 15 stone.
RESPONSE:
["inform WorkerA to mine 25 woods", "inform workerB mine 15 stone"]

Example 2: If your name is leader:
INPUT:
Current task:
Step: WorkerA needs to build the foundation.
RESPONSE:
["inform workerA to build the foundation"]

Example 3:
INPUT:
Current task:
Stage 2: WorkerA builds the walls
    WorkerA uses 48 planks to build the walls at (-10,72,-30).
RESPONSE:
["inform workerA to build walls at (-10,72,-30) (use 48 planks)"]

Example 4:
INPUT:
Current task:
Stage 1: WorkerA crafts 4 planks from the log of WorkerB.

RESPONSE:
["inform workerB to give a log to workerA", "inform workerA to craft 4 planks"]
```

Action planner: example of leaf agent

```
Example 1: If your name is worker, you should not generate a task includes 'inform', you should
        ↪ only do the task yourself:
INPUT:
Current task:
Step: WorkerA mine 25 woods
RESPONSE:
["mine 25 woods"]
```