

S-Agents: Self-organizing Agents in Open-ended Environments

Jiaqi Chen*, Yuxian Jiang*, Jiachen Lu and Li Zhang[†]

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-obtrusive 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.

I. 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 [1], [2]. Autonomous agents powered by Large Language Models (LLMs) [3]–[5], especially GPT-4 [6] have paved the way for innovative developments in this domain. These models showcase remarkable competencies in diverse domains, including instruction comprehension [7], [8], decision-making [9], [10], tool usage [11], code generation [12], and diverse other domains [13]. Moreover, LLM-powered agents have been employed in diverse tasks in open-ended environments, spanning from sandbox games, simulated environments, and robotics [14]–[17]. However, complex embodied tasks in open environments often necessitate collaboration among individual agents to achieve optimal outcomes [18], [19].

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** [12], [20], 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 [21]–[23]. Furthermore,

All authors are with Fudan University. *Equal contribution. [†]Li Zhang is the corresponding author with School of Data Science, Fudan University (lizhangfd@fudan.edu.cn).



Fig. 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.

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-obtrusive 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.

II. RELATED WORK

a) LLM powered multi-agent collaboration: Large Language Models (LLMs) have demonstrated phenomenal capabilities in various domains. Generative agent [26] drives multiple agents with LLM and simulate human-like conversations as well as some social behaviors; Recent attempts [12] found that multi-agent embodied collaboration could develop software following a fixed process, but could not coordinate autonomously; Multiple robotic arms [27],

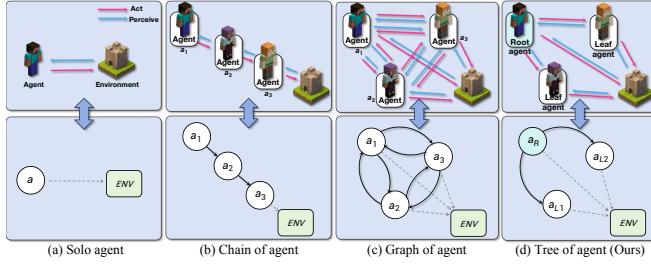


Fig. 2. **Schematic organizational structure comparison.** (a) Solo agent [24]: Direct interaction with the physical environment; (b) Chain of agents [12], [25]: Specialized agents sequentially perform their designated tasks and command the actions of the next agent; (c) Graph of agents [26]: Decentralized 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.

and multiple agents [28] 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.

b) 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. [14], [29] are established benchmarks for evaluating single-agent algorithms. While [30], [31] focus on designing specific structures based on human instructions. On the other hand, Malmo [32] 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 [33], [34]. [35] perform large-scale pre-training on game-playing videos. [36], [37] proposes a closed-loop feedback framework for a single agent, allowing the agent to achieve its goals vis interact with the environment. [24] uses LLMs to automatically generate the next task based on the environment, continuously enriching the skill library and exploring the world. 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.

III. 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.

A. Organizational structure of agents

1) Agents as a graph: We design an agent-centric organization, specifically, we do not predefine the specific roles and functions of agents [12]. 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 $\mathcal{A} = \{a_1, \dots, a_n\}$, and the environment, represented as p ,

$$V = \{a_1, \dots, a_n, p\}, n > 1. \quad (1)$$

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.

a) Graph of agents: The (fully connected) graph of agents [26] is an organizational structure where all agents are interconnected, allowing mutual command and feedback (shown in Figure 2(c)), represented as

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

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. IV-C) 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.

b) Chain of agents: To address the issue of command cycles and capitalize on the expertise of specialized agents, a straightforward approach [12] is the Chain of agents (CoA) (as in Figure 2(b)), defined as follows:

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

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.

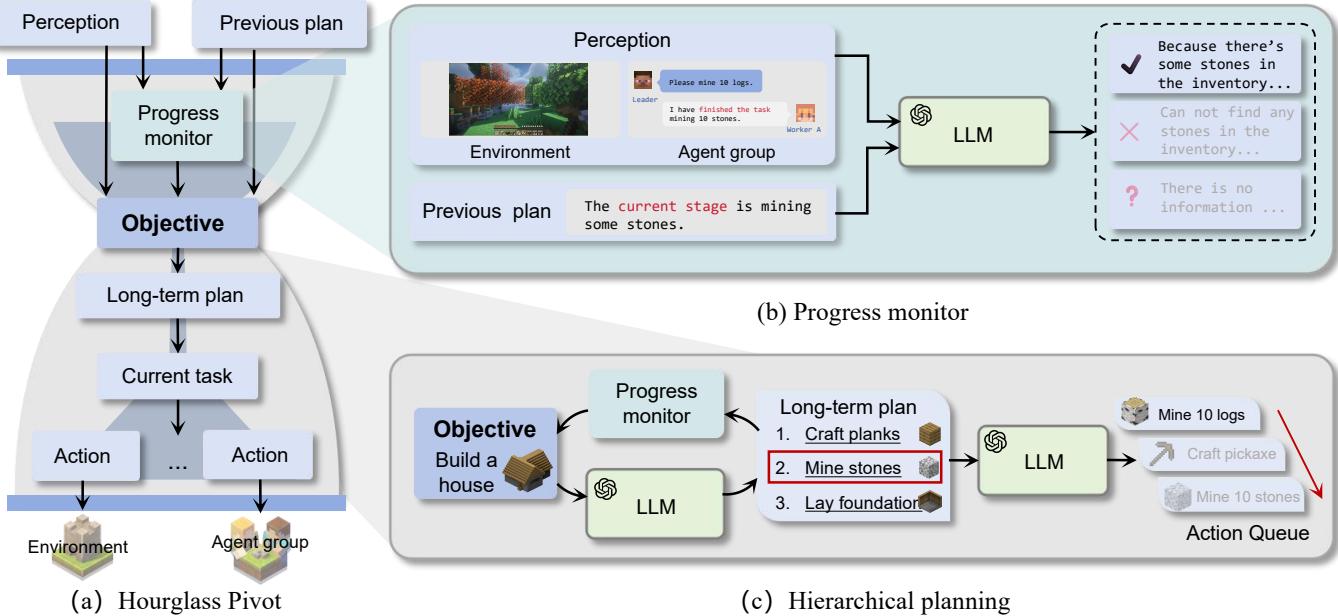


Fig. 3. An illustration of hourglass agent architecture. (a) **Hourglass agent framework:** The upper segment: Processes inputs like perception and the previous plan. These inputs undergo a series of operations, converging towards a unified and consistent objective (the bottleneck of the hourglass). 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.

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

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

In this structure, the root agent a_r serves as the central command authority, directing tasks to the leaf agents, $\mathcal{E}_{\text{root}} = \{(a_r, a_i) \mid a_i \in \{a_{l1}, a_{l2}, \dots, a_{ln}\}\}$. 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.

B. 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.

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 [24]. 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.

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

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

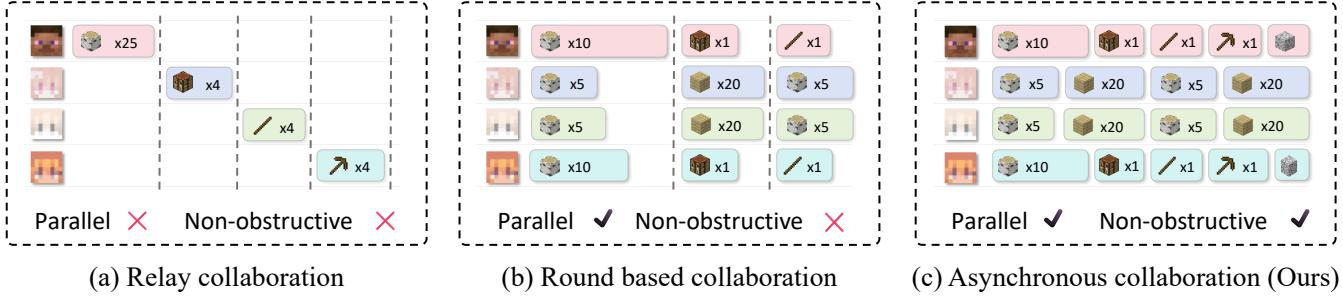


Fig. 4. **Comparison of collaboration strategies.** (a) involves one agent sequentially executing tasks after another, with no parallelization; (b) is round-based, executing round by round, while (c) is asynchronous. Colored regions indicate tasks being performed, and white regions denote agent idleness.

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 [38], 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 12 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 (highlighted in green in Figure 12 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 [24], 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. III-B.2.

C. Non-obstructive collaboration

The previous approaches can be categorized into two distinct types. In relay collaboration, exemplified by Chain of agents [12] 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 [39], 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.

IV. EXPERIMENTS

A. 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 [24]. In contrast to completing the exploration of a single item [24], [37], [40], 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.

a) 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.

b) Implement details: For large language models, we leverage OpenAI’s gpt-4 [6] for planning of root agent and task evaluation. Additionally, we utilize the gpt-3.5-turbo-16k APIs [41] 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 [24], our simulation environment is constructed on the MineDojo framework [14],

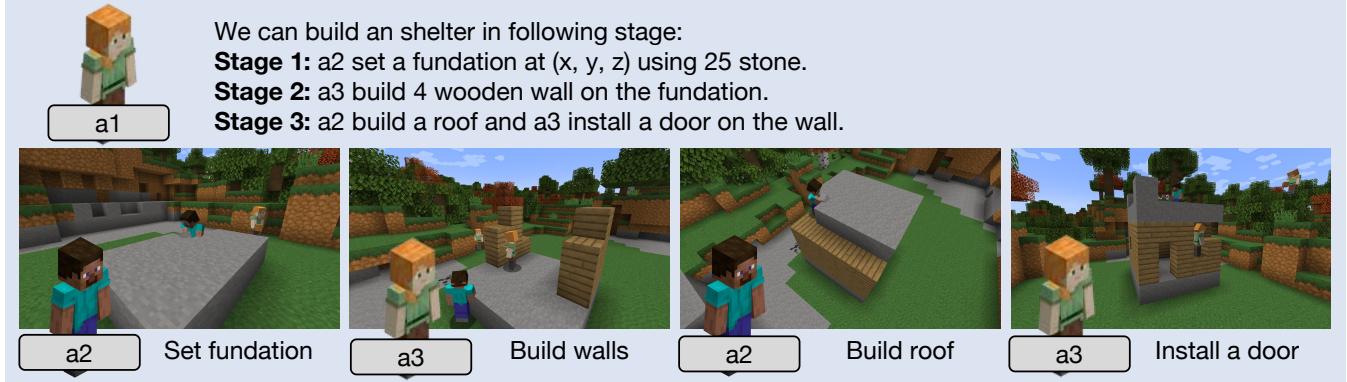


Fig. 5. **Collective shelter construction.** The root agent (a1) systematically arranges the tasks and schedules the leaf agents (a2 & a3) for phased execution.



Fig. 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.

and we make use of the Mineflayer JavaScript APIs [42] for motor controls. We initialize all agents using the skill library pre-trained by Voyager [24].

B. Evaluation Results

a) *Impact of organizational structure:* To comprehend the influence of distinct organizational structures on efficiency, we conducted “mine 50 logs” tasks, evaluating their mean prompt times and time costs. According to Figure 7(a), the tree of agents (ToA) outperforms the chain of agents (CoA) and graph of agents (GoA), requiring only 7.5 minutes and 3.8 mPT.

b) *Multi-agent organization vs. Solo agent:* To comprehend the disparity in outcomes between a multi-agent organization and a solo agent, we conducted experiments on four distinct difficulty levels of collection tasks (mining 50 logs / 100 logs / 50 iron / 100 iron). We measured the time costs associated with each task.

1) *Parallelization to save time:* In Figure 7(a), CoA involves one agent sequentially executing 1/3 collection tasks after another, with no parallelization. GoA avoids command cycle issues due to the upgrade to the more powerful gpt-4. ToA achieves the shortest completion time for identical tasks. As shown in Figure 7(c), for simpler tasks like mining 50/100 logs, employing multiple agents significantly reduces the time by 5.1 and 7.2 minutes, respectively. Comparing the Solo agent with ToA(1) in Figure 7(b) reveals that

merely adding a leadership agent (ToA(1)) for one-on-one guidance does not bring a significant improvement. However, increasing the number to three leaf agents (ToA(3)) has a chance of enhancing efficiency.

2) *Mutual redundancy enhances robustness:* For more challenging tasks, solo agent attempts were largely ineffective (indicated as NaN in Figure 7(c)). A solo agent faces heightened probabilities of encountering various unforeseen challenges in an open-world setting, such as getting lost, obstructions by surrounding 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.

C. Agent behaviors within organization

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 based on the effective allocation of human resources and strategic management practices. This phenomenon is also observed in agent organizations. As illustrated in Figure 8(a), the root agent, functioning as the leader of the organization, is tasked not only with solving individual issues but with equitably decomposing the workload. This property is achieved by instructing

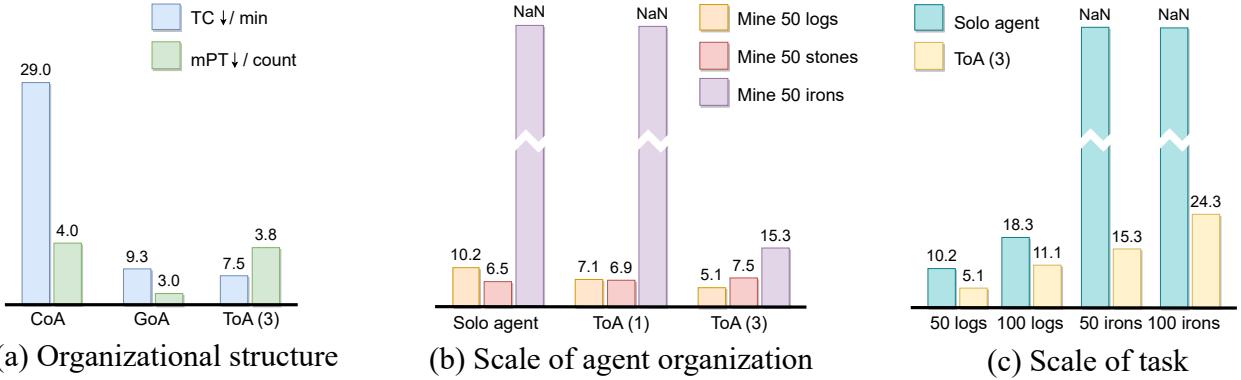


Fig. 7. **Collaborative collection:** (a) Efficiency of three agents mining 50 stones under different organizational structures; CoA and GoA each consist of 3 agents, where ToA(n) represents an agent organization with one root and n leaves. Due to the presence of command cycle issues in GoA, the planning module, which utilized gpt-3.5-turbo-16k, was upgraded to gpt-4 for experimentation. (b) Time cost (/min) of 1, 2, and 3 agents on various tasks, NaN denotes no progress for over 40 minutes. (c) Efficiency comparison (time cost/min) between multi-agent and solo agent across different task difficulties.

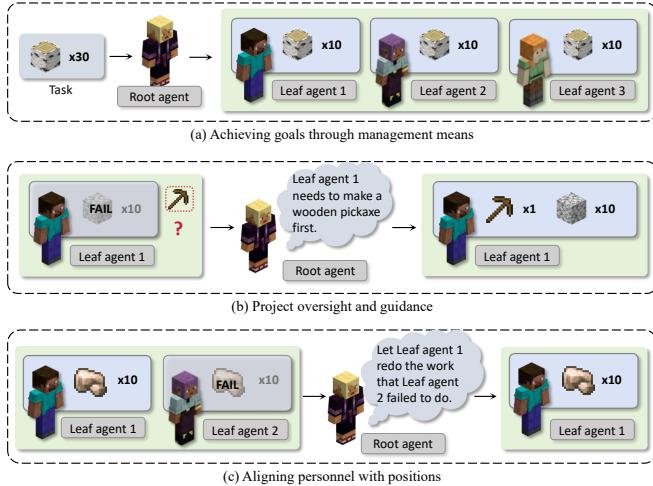


Fig. 8. **Human-like leadership behaviors in self-organizing agent group.**

the task planning prompt of leadership agent to assign tasks to leaf agents rather than itself. The action planner generates communication tasks to notify the corresponding agents.

Project oversight and guidance: As depicted in Figure 8(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. Starting from foundational tasks, the leader systematically guided the team toward the goal, showcasing the potential of the LLM agent as a genuine leader.

Aligning personnel with positions: “*Aligning personnel with positions*” refers to aligning suitable personnel with corresponding positions, a fundamental principle in human society where leaders engage in human resources management. In the context of an agent organization, as depicted in Figure 8(c), we observe a proactive leadership agent (root agent a_r) leveraging a nuanced understanding of past experiences and skills of team members to strategically allocate tasks.

2) Behaving like human employees: During collaborative processes in agent organization, we’ve observed leaf agents displaying behaviors that mimic human employees.

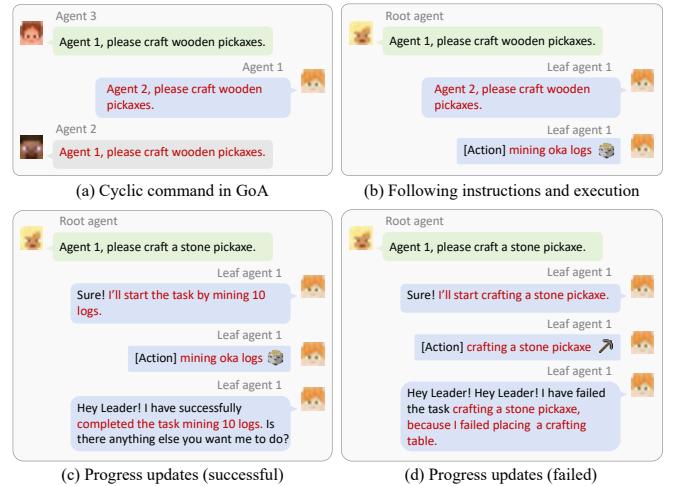


Fig. 9. **Agent behaviors that mimic human employees.**

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 9(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. In contrast, as shown in Figure 9(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 9(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.

V. CONCLUSION

In conclusion, this study has introduced self-organizing agents (S-Agents), an 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 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. As artificial general intelligence advances, understanding the organizational dynamics of scalable multi-agent systems becomes crucial, with S-Agents initiating explorations in this field, poised for adaptable enhancement in diverse embodied tasks.

REFERENCES

- [1] D. Weinbaum and V. Veitas, “Open ended intelligence: the individuation of intelligent agents,” *JETAI*, 2017.
- [2] M. Fujita, “Intelligence dynamics: a concept and preliminary experiments for open-ended learning agents,” *AAMAS*, 2009.
- [3] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günemann, E. Hüllermeier, *et al.*, “Chatgpt for good? on opportunities and challenges of large language models for education,” *Learning and individual differences*, 2023.
- [4] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint*, 2023.
- [5] G.-P. Ji, M. Zhuge, D. Gao, D.-P. Fan, C. Sakaridis, and L. V. Gool, “Masked vision-language transformer in fashion,” *MIR*, 2023.
- [6] OpenAI, “Gpt-4 technical report,” 2023.
- [7] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, “Training language models to follow instructions with human feedback,” *NeurIPS*, 2022.
- [8] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, *et al.*, “Scaling instruction-finetuned language models,” *arXiv preprint*, 2022.
- [9] N. Shinn, B. Labash, and A. Gopinath, “Reflexion: an autonomous agent with dynamic memory and self-reflection,” *arXiv preprint*, 2023.
- [10] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, “Tree of thoughts: Deliberate problem solving with large language models,” *arXiv preprint*, 2023.
- [11] T. Cai, X. Wang, T. Ma, X. Chen, and D. Zhou, “Large language models as tool makers,” *arXiv preprint*, 2023.
- [12] S. Hong, X. Zheng, J. Chen, Y. Cheng, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, *et al.*, “Metagpt: Meta programming for multi-agent collaborative framework,” *arXiv preprint*, 2023.
- [13] X. Tang, A. Zou, Z. Zhang, Y. Zhao, X. Zhang, A. Cohan, and M. Gerstein, “Medagents: Large language models as collaborators for zero-shot medical reasoning,” *arXiv preprint*, 2023.
- [14] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar, “Minedojos: Building open-ended embodied agents with internet-scale knowledge,” *NeurIPS*, 2022.
- [15] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, *et al.*, “Rt-2: Vision-language-action models transfer web knowledge to robotic control,” *arXiv preprint*, 2023.
- [16] G. Lu, Z. Wang, C. Liu, J. Lu, and Y. Tang, “Thinkbot: Embodied instruction following with thought chain reasoning,” *arXiv preprint*, 2023.
- [17] A. Padmakumar, M. Inan, S. Gella, P. L. Lange, and D. Hakkani-Tur, “Multimodal embodied plan prediction augmented with synthetic embodied dialogue,” in *EMNLP*, 2023.
- [18] A. W. Woolley, C. F. Chabris, A. Pentland, N. Hashmi, and T. W. Malone, “Evidence for a collective intelligence factor in the performance of human groups,” *science*, 2010.
- [19] M. Zhuge, H. Liu, F. Faccio, D. R. Ashley, R. Csordás, A. Gopalakrishnan, A. Hamdi, H. A. A. K. Hammoud, V. Herrmann, K. Irie, *et al.*, “Mindstorms in natural language-based societies of mind,” *arXiv preprint*, 2023.
- [20] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang, “Autogen: Enabling next-gen llm applications via multi-agent conversation framework,” *arXiv preprint*, 2023.
- [21] D. Grossi, F. Dignum, V. Dignum, M. Dastani, and L. Royakkers, “Structural evaluation of agent organizations,” in *IJCAI*, 2006.
- [22] A. S. Jensen, V. Dignum, and J. Villadsen, “A framework for organization-aware agents,” *AAMAS*, 2017.
- [23] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan, “Scalable multi-robot collaboration with large language models: Centralized or decentralized systems?,” *arXiv preprint*, 2023.
- [24] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, “Voyager: An open-ended embodied agent with large language models,” *arXiv preprint*, 2023.
- [25] C. Qian, X. Cong, C. Yang, W. Chen, Y. Su, J. Xu, Z. Liu, and M. Sun, “Communicative agents for software development,” *arXiv preprint*, 2023.
- [26] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” 2023.
- [27] Z. Mandi, S. Jain, and S. Song, “Roco: Dialectic multi-robot collaboration with large language models,” *arXiv preprint*, 2023.
- [28] H. Zhang, W. Du, J. Shan, Q. Zhou, Y. Du, J. B. Tenenbaum, T. Shu, and C. Gan, “Building cooperative embodied agents modularly with large language models,” *arXiv preprint*, 2023.
- [29] A. Kanervisto, S. Milani, K. Ramanauskas, N. Topin, Z. Lin, J. Li, J. Shi, D. Ye, Q. Fu, W. Yang, *et al.*, “Minerl diamond 2021 competition: Overview, results, and lessons learned,” *NeurIPS 2021 Competitions and Demonstrations Track*, 2022.
- [30] S. Mohanty, N. Arabzadeh, M. Teruel, Y. Sun, A. Zholus, A. Skrynnik, M. Burtsev, K. Srinet, A. Panov, A. Szlam, *et al.*, “Collecting interactive multi-modal datasets for grounded language understanding,” *arXiv preprint*, 2022.
- [31] J. Gray, K. Srinet, Y. Jernite, H. Yu, Z. Chen, D. Guo, S. Goyal, C. L. Zitnick, and A. Szlam, “Craftassist: A framework for dialogue-enabled interactive agents,” *arXiv preprint*, 2019.
- [32] D. Perez-Liebana, K. Hofmann, S. P. Mohanty, N. Kuno, A. Kramer, S. Devlin, R. D. Gajina, and D. Ionita, “The multi-agent reinforcement learning in malmo” o (marl) o competition,” *arXiv preprint*, 2019.
- [33] I. Kanitscheider, J. Huizinga, D. Farhi, W. H. Guss, B. Houghton, R. Sampedro, P. Zhokov, B. Baker, A. Ecoffet, J. Tang, *et al.*, “Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft,” *arXiv preprint*, 2021.
- [34] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, “Mastering diverse domains through world models,” *arXiv preprint*, 2023.
- [35] B. Baker, I. Akkaya, P. Zhokov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune, “Video pretraining (vpt): Learning to act by watching unlabeled online videos,” *NeurIPS*, 2022.
- [36] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang, “Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents,” *arXiv preprint*, 2023.
- [37] X. Zhu, Y. Chen, H. Tian, C. Tao, W. Su, C. Yang, G. Huang, B. Li, L. Lu, X. Wang, *et al.*, “Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory,” *arXiv preprint*, 2023.
- [38] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *NeurIPS*, 2022.
- [39] W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C. Qian, C.-M. Chan, Y. Qin, Y. Lu, R. Xie, *et al.*, “Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents,” *arXiv preprint*, 2023.
- [40] Z. Wang, S. Cai, A. Liu, Y. Jin, J. Hou, B. Zhang, H. Lin, Z. He, Z. Zheng, Y. Yang, X. Ma, and Y. Liang, “Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models,” *arXiv preprint*, 2023.
- [41] OpenAI, “Introducing chatgpt,” 2023.
- [42] PrismarineJS, “Prismarinejs. prismarinejs/mineflayer: Create minecraft bots with a powerful, stable, and high level javascript api,” 2013.

APPENDIX

This section presents the process of multi-agent task execution and the specifics of planning.

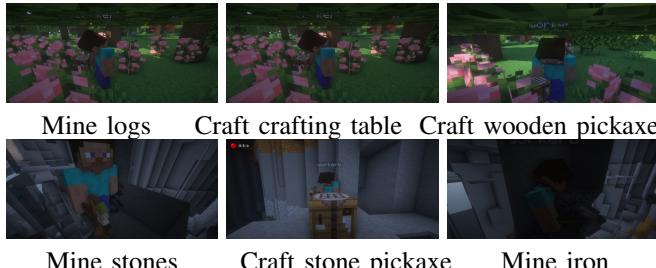


Build foundation

Build walls

Build roof

Fig. 10. **The procedure of building a house.** The root agent commands while the two leaf agents collaborate to complete the construction.



Mine logs
Mine stones

Craft crafting table
Craft stone pickaxe

Craft wooden pickaxe
Mine iron

Fig. 11. **The procedure of mining iron.** As the execution processes of the two agents are similar, only one of them is presented here.

A. Example of execution process

Figure 10 and Figure 11 illustrate the collaborative processes of building a house and collecting iron, respectively.

B. Example of planning process

Figure 12 demonstrates the planning process of both the root agent and leaf agents in the “build a house” task.

C. Full prompt design

1) *Progress monitor:* The input prompt to LLM consists of several components:

- (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.

The specific template is as follows:

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

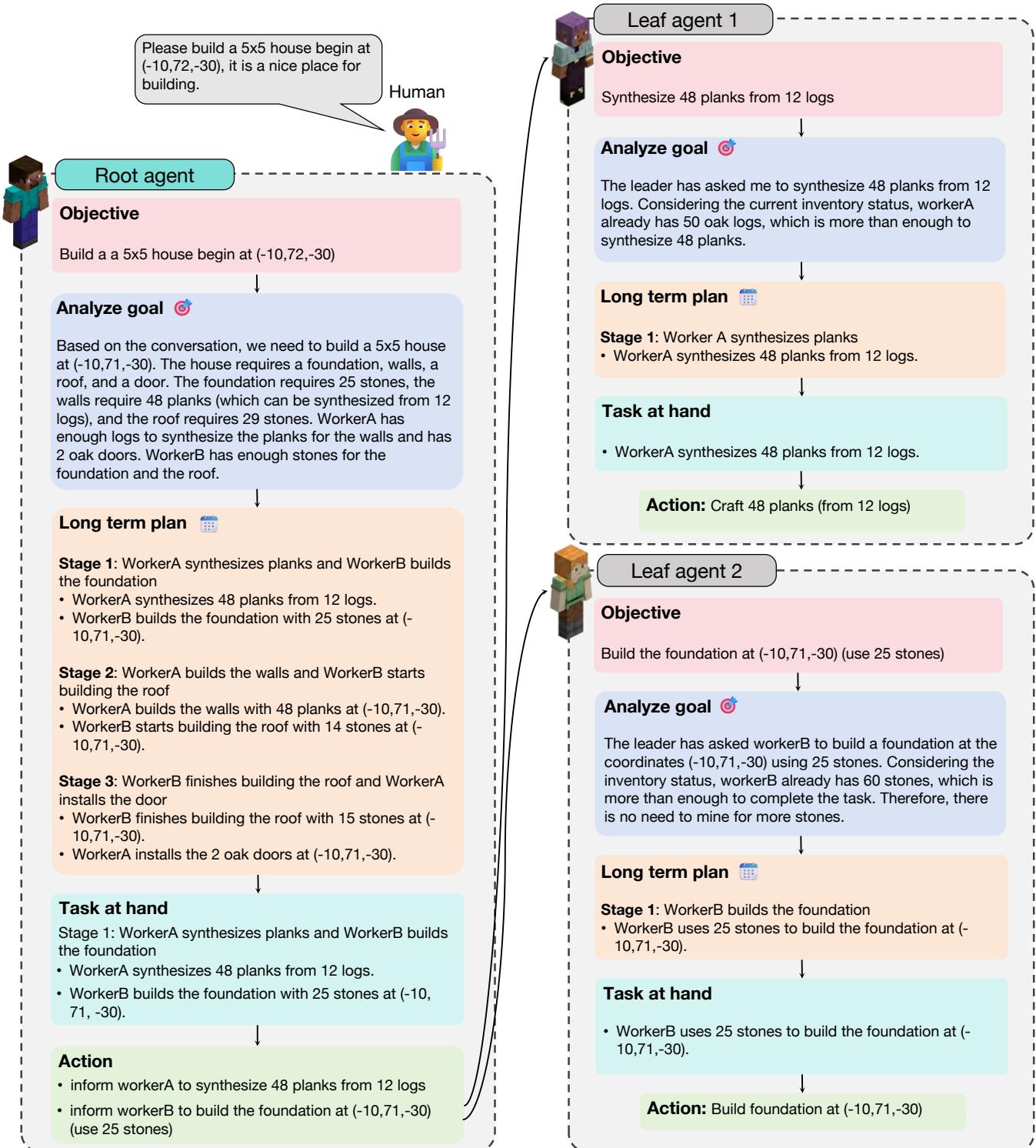


Fig. 12. The illustration of an example process of collaborative house building.

```

words in success or fail or unknown
>

```

Due to the differences in monitoring execution between root agents and leaf agents, we provide different examples to enhance their performance. Below is an example of a root agent.

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
```

Below is an example of a leaf agent.

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
```

2) *Task planner*: 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.

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] '

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

Below is the task planner prompt of the leaf agent:

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

3) *Action planner:* The input prompt to llm us the current task proposed by the task planner.

Below is the prompt of action planner:

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}

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

Below is the example of leaf agent in action planner:

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"]
```

Below is the example of root agent in action planner:

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: