# CausalMACE: Causality Empowered Multi-Agents in Minecraft Cooperative Tasks

**Qi Chai**[1], **Zheng Zhang**[1], **Junlong Ren**[1], **Deheng Ye**[2], **Zichuan Lin**[2], **Hao Wang**[1]*
[1]The Hong Kong University of Science and Technology (Guangzhou), [2]Tencent
{qchai315, zzhang302, jren686}@connect.hkust-gz.edu.cn
{dericye, zichuanlin}@tencent.com, haowang@hkust-gz.edu.cn

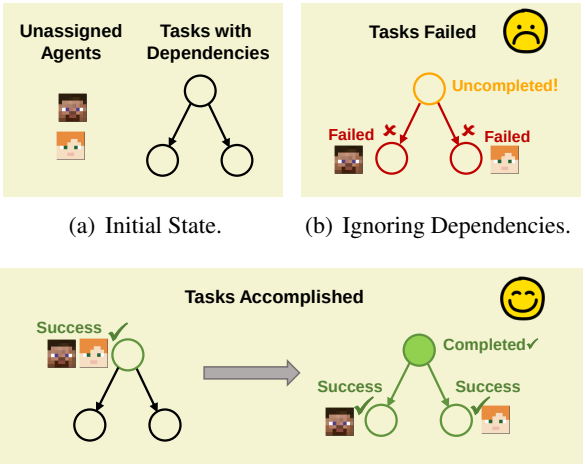
## Abstract

Minecraft, as an open-world virtual interactive environment, has become a prominent platform for research on agent decision-making and execution. Existing works primarily adopt a single Large Language Model (LLM) agent to complete various in-game tasks. However, for complex tasks requiring lengthy sequences of actions, single-agent approaches often face challenges related to inefficiency and limited fault tolerance. Despite these issues, research on multi-agent collaboration remains scarce. In this paper, we propose CausalMACE, a holistic causality planning framework designed to enhance multi-agent systems, in which we incorporate causality to manage dependencies among subtasks. Technically, our proposed framework introduces two modules: an overarching task graph for global task planning and a causality-based module for dependency management, where inherent rules are adopted to perform causal intervention. Experimental results demonstrate our approach achieves state-of-the-art performance in multi-agent cooperative tasks of Minecraft[1].

## 1 Introduction

In recent years, Large Language Models (LLMs) have shown significant abilities in various domains (Xu et al., 2023; Wang et al., 2023; Ćavar et al., 2024; Zubiaga, 2024; Fu et al., 2025). Beyond these fundamental domains, interest has increased in how to utilize LLMs' capabilities to make decisions in an open-world environment.

Minecraft, as a virtual interactive environment, offers a unique platform for such research. Various approaches (Wang et al., 2024b,a,c; Li et al., 2024b) have been developed within Minecraft to explore the decision-making capabilities of LLMs, achieving significant progress on in-game tasks.




(a) Initial State. (b) Ignoring Dependencies.

(c) Following Dependencies (Ours).

Figure 1: **The impact of dependencies among tasks.** (a) describes tasks with dependency (arrows). (b) shows the consequences of ignoring dependencies: agents failed to execute due to unmet prerequisites. (c) demonstrates the proper dependency processing: sequentially complete root node tasks before leaf nodes.

However, complex tasks that require lengthy sequences of action or collaboration remain a challenge, since single-agent approaches often face issues of inefficiency and limited fault tolerance.

To fully unlock the potential of LLMs for decision-making in open-world environments, it is essential to explore the usage of multi-agent systems. Research has shown that LLM agent teams can outperform a single agent in writing (Chan et al., 2024), reasoning (Xu et al., 2024) and code generation (Hong et al., 2024). However, existing methods often emphasize role assignment or communication to enhance performance, which does not fully exploit the parallelization potential of multi-agent systems, thus limiting their application in open-world decision-making tasks.

While existing methods (Chen et al., 2023; Dong et al., 2024) attempt to enhance the performance of multi-agent systems, they often encounter bottlenecks stemming from several unaddressed issues.

*Corresponding Author.
[1]https://github.com/qccq315/CausalMACE



*Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 14410–14426
November 4-9, 2025 

The first issue is that current multi-agent methods in Minecraft lack global task planning. While local task planning is relatively flexible, the absence of global planning may cause agents to deviate from the original plan after multiple iterations gradually. Another issue is that existing methods fail to consider the dependencies that exist within subtasks. This can result in assigned tasks being unachievable thereby reducing efficiency, as shown in Figure 1.

Given that the organization of subtasks in open-world tasks is inherently shaped by the environment's rules, we propose that causal relations should exist between these rules and the dependencies among subtasks. When these causal relations are clearly identified, subtasks can naturally be structured into a cohesive task graph. This assumption enables us to use causality (Yuan et al., 2023a; Wu et al., 2024) to guide tasks and to construct a coherent global task graph. Based on this premise, we propose CausalMACE (**Causal**ity Empowered **M**ulti-**A**gents in Minecraft **C**ooperativ**E** Tasks), a global causality planning framework. Specifically, we incorporate two new modules: one for maintaining an overarching task graph for global task planning, and another that utilizes causality to construct and manage the dependencies among subtasks. CausalMACE achieves an average performance improvement of 12% in multi-agent cooperative tasks and 7% in single-agent tasks, achieving state-of-the-art results.

Our contributions can be concluded as follows:

- We propose a novel framework for multi-agent cooperative tasks of Minecraft. An overarching task graph is proposed for global planning that facilitates comprehensive tasks.
- We propose to leverage causality to manage and build dependencies between subtasks. This ensures the task graph is aligned with the inherent rules of open-world environments.
- Experimental results demonstrate our framework surpasses existing baselines in multi-agent cooperative tasks and also achieves competitive results in single-agent tasks.

## 2 Related Work

### 2.1 Causal Discovery

The causal discovery method can be delineated into two groups, traditional approaches and large language models (LLMs) based methods. Traditional causal discovery approaches rely mainly on statistical techniques and algorithms to uncover causal structures in data. Several methods (Xiang and Kim, 2013; Chickering and Meek, 2015; Ramsey, 2015) calculate scores to investigate the entire range of possible edges, with the aim of finding a graph that fits the data the best. Some other researches (Spirtes et al., 2001, 2013) use conditional independence tests to infer causal structures, identifying dependencies among variables.

Recent advances in LLMs have offered the possibility of utilizing LLMs in causal discovery. Some studies attempt to use LLMs as an alternative tool for conditional independence tests in the traditional causal discovery process (Cohrs et al., 2024) or build causal graphs beyond the Markov equivalent class (Long et al., 2023). Despite these, there are also several data-free methods (Kıcıman et al., 2023; Zečević et al., 2023; Zhang et al., 2024), which show that LLMs can identify causal structures by interpreting metadata and natural language, similar to how human experts apply domain knowledge to construct causal models.

### 2.2 Multi-agent Systems

Due to the growing potential of multi-agent systems in addressing complex problems, research on this topic has been expanding rapidly. With the integration of LLMs, multi-agent collaboration has shown significant promise across various tasks, enhancing individual agent performance. Existing approaches can be categorized into two strategies: role assignment and communication management.

For role assignment, CAMEL (Li et al., 2023) assigns roles to agents to optimize collaborative behaviors and mitigate hallucinations. MetaGPT (Hong et al., 2024) leverages role specialization to coordinate multiple LLMs in simulating real-world software development workflows. Triad (Zong et al., 2024) addresses knowledge base question answering tasks by assigning three distinct roles, generalist, decision maker, and advisor, to agents based on LLMs. For communication management, ChatEval (Chan et al., 2024) organizes multiple agents to debate and express their opinions, effectively reducing biases in text evaluation. DyLAN (Liu et al., 2024) improves collaboration efficiency by dynamically managing the composition of agent teams. Magic (Xu et al., 2024) employs probabilistic graphical models to enhance reasoning and facilitate interaction within agent teams. However, these methods predominantly operate in static temporal frameworks, neglecting the dynamic temporal

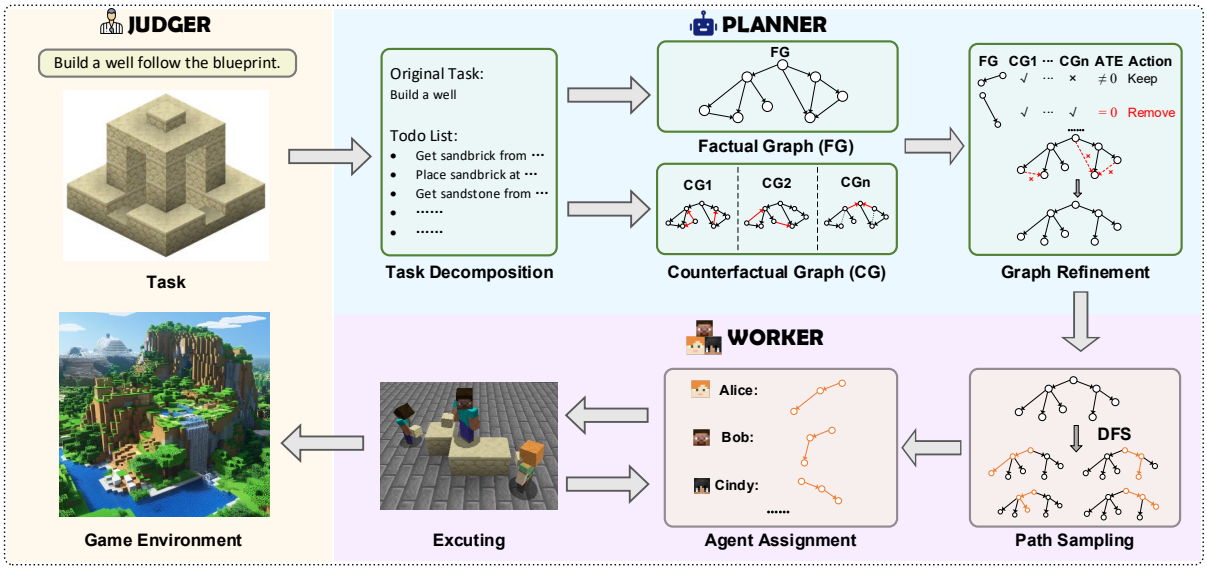



Figure 2: **Overview of the framework.** Our framework contains Judger, Planner and Worker. The **Judger** defines the objective and gives the environment feedback. The **Planner** decomposes the task into subtasks and constructs a dependency graph via game rules. The **Worker** assigns subtasks to agents to process. *ATE* denotes the average treatment effect, which is introduced in Section 3.4.

coordination crucial for fully exploiting the parallelization potential inherent in multi-agent systems.

### 2.3 Agents in Minecraft

Minecraft, which offers a rich environment for agent research, has emerged as a versatile platform for the testing of intelligent agents. Several researches (Zhou et al., 2024; Yuan et al., 2024; Li et al., 2024a) focus on low-level control policies with reinforcement learning (RL). With the development of LLMs, researchers attempt to enhance agents' capabilities by utilizing LLMs to process complex environmental feedback. DEPS (Wang et al., 2024b) pioneers this transition by implementing the first LLM-powered agent framework in Minecraft. Voyager (Wang et al., 2024a) adopts skill libraries to boost task execution efficiency. Additionally, techniques like multi-modal memories and knowledge graphs have been employed to enhance agents' adaptability and task performance (Wang et al., 2024c; Li et al., 2024b).

Building on the progress of single-agent studies, some researchers have begun exploring the potential of multi-agent collaboration within Minecraft. Multi-agent systems offer significant advantages in task allocation and resource management, enabling more effective handling of complex scenarios in the game. AgentVerse (Chen et al., 2023), which is the initial study, has started to address challenges related to communication and coordination. It divides its framework into four components, each providing specific guidance to multiple LLM agents, effectively organizing agent groups, and outperforming single-agent systems. VillagerAgents (Dong et al., 2024) attempts to break down the task into subtasks that can be processed in parallel. However, these approaches often overlook the causal relations between the game rules and dependencies among subtasks during task decomposition, limiting cooperative efficiency in complex tasks.

## 3 Method

### 3.1 Preliminary

**Causality.** It represents the causal-and-effect relationships between variables. Causality seeks to understand how changes in one variable $X$ (the cause) *directly affect* another variable $Y$ (the effect), i.e., $X \rightarrow Y$. The Structural Causal Model (SCM) (Pearl, 2009) provides a formal framework for identifying and leveraging causality. It employs directed acyclic graphs (DAGs) to represent causal structures, where nodes represent distinct variables and edges denote causal relationships between them.

**Mediator.** $M$ is the variable between $X$ and $Y$, where $X \rightarrow M$ and $M \rightarrow Y$. In this situation, the causal effect of $X$ can reach $Y$ by $X \rightarrow M \rightarrow Y$. Such a chain can be called a **causal path** so long as the causal effect can be transmitted.

**Confounder.** $C$ is a variable that affects both $X$ and $Y$, i.e., $C \rightarrow X$ and $C \rightarrow Y$. It may create a spurious association between $X$ and $Y$.

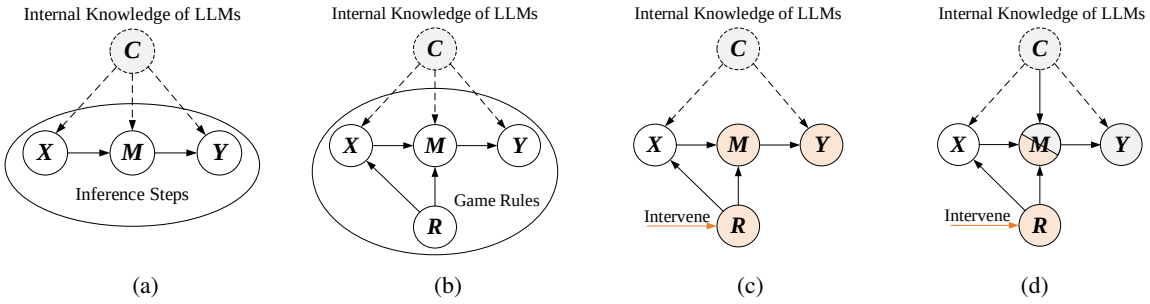



Figure 3: **Procedures for generating the dependencies.** $X$ and $Y$ denote the input and output, $M$ is the inference steps of LLMs and $C$ represents the internal knowledge of LLMs. (a) and (b) indicate the procedures with and without game rules $R$, respectively, where the dotted arrow denotes the uncertain causal edges. (c) shows the successful intervention procedure, (d) reveals the failure cases where the effect of $R$ on $M$ is blocked by $C$.

**Instrumental variable.** Given a causal path $X \rightarrow Y$, an instrumental variable is a variable that satisfies the following conditions: 1) It directly affects $X$, 2) No path exists from the instrumental variable to any confounder $C$ that affects both $X$ and $Y$, 3) It has no direct effect on $Y$.

### 3.2 Overview

Our proposed framework is shown in Figure 2, which contains three main components: Judger, Planner, and Worker. The Judger initiates tasks within the game. It defines the objectives the agents need to achieve and provides feedback and evaluations of their performance.

Upon receiving a task from the Judger, the Planner decomposes it into a series of subtasks. It identifies the dependencies between these subtasks by leveraging the game rules and refines the resulting task graph using causal inference. This refined graph, which represents the structured dependencies and execution orders, is then passed on to the Worker for further action.

Technically, the Worker processes the refined task graph provided by the Planner. It employs a depth-first search (DFS) algorithm to explore all possible execution paths within the graph and then assigns these tasks to the agents for execution. The agents operate autonomously with reflection. Their actions are processed by the Judger, which also assesses the task execution.

### 3.3 Judger Interface

The Judger serves as an interface component that mediates interactions between agents and the game environment. It functions as both an action validator and a performance evaluator within the framework. When agents attempt to execute actions, the Judger processes these actions through the game environment and returns updated state information. Additionally, it incorporates evaluation functions that compute various performance metrics based on agent behaviors and outcomes. The Judger maintains a clear separation between the game environment and the agents' actions while providing standardized interfaces for both action processing and performance assessment.

### 3.4 Planner with Casual Intervention

The Planner aims to transform a given task into a structured set of subtasks, which is represented by a causal dependency graph. The entire procedure involves three steps: task decomposition, initial graph construction, and graph refinement.

We enhance the global task planning capability. Upon receiving an input task from the Judger, the Planner employs LLMs to directly decompose it into a set of subtasks, $S = \{s_1, s_2, \ldots, s_n\}$. These subtasks preserve global goal consistency with the input task. Then, we use the set of game rules $R = \{r_1, r_2, \ldots, r_k\}$ as explicit guidelines to identify direct dependencies between subtasks. These rules give an initial version of the global task dependency graph $G_{\text{init}}$ and ensure that it is structured logically. However, it is observed $G_{\text{init}}$ may contain extra or incorrect edges due to hallucinations and biases stemming from the LLMs' internal knowledge (Yuan et al., 2023b; Wu et al., 2024).

To resolve this issue, we aim to find out the actual dependencies between subtasks and refine $G_{\text{init}}$. Specifically, we define two subtasks, $s_p$ and $s_q$, as *input* $X$, and their dependencies given by LLMs as *output* $Y$. The *mediator* that maps $X$ to $Y$ is denoted as $M$, which refers to the LLM inference process. Since LLMs are used as the resolver, their internal knowledge may have a potential influence on $X$, $Y$ and $M$. Therefore, we define the internal

knowledge of LLMs as the *confounder* $C$, as illustrated in Figure 3(a). Then, we introduce the set of game rules $R$ as external knowledge. As the dependencies between subtasks should be constrained by the rules, $R$ has a causal effect on both *input* $X$ and *mediator* $M$. This process is shown in Figure 3(b).

Next, we aim to ensure that the final *output* $Y$ adheres to the set of game rules $R$. Since $M$ is the only controllable variable that directly affects $Y$, it is necessary to discern whether the primary effect of the inference steps $M$ comes from $R$ or internal knowledge $C$. As tracking the causal relationship between $C$ and $M$ is difficult, an alternative method is to leverage the variable $R$ as an instrumental variable (Yuan et al., 2023a) instead, which can reach $Y$ through the causal path $R \rightarrow M \rightarrow Y$.

By intervening on $R$, we can observe how it affects $Y$. This effect is the average treatment effect (ATE), which allows us to estimate the causal effect on $Y$. Specifically, for each game rule $r_i$, we query LLM for a counterfactual rule $r_i^*$, which is contradictory to $r_i$. For example, the counterfactual rule for "*You must have a block before you place it.*" can be expressed as: "*You can place a block even if you do not have it*". This procedure generates a modified set of rules $R_i^* = \{r_1, r_2, ..., r_i^*, ..., r_k\}$ where only $r_i$ is replaced while other rules remain unchanged. Following the modified set of rules as explicit guidelines, while taking $s_p$ and $s_q$ as *input* $X$, we observe whether the *output* $Y$ changes. Therefore, for the given $X$ and its inference steps $M$, the ATE of the certain rule $r_i$ is defined as:

$$\begin{aligned} ATE(r_i, X) = \mathbb{E}(Y|X, M, do(R)) - \\ \mathbb{E}(Y|X, M, do(R_i^*)), \end{aligned} \tag{1}$$

where $do(\cdot)$ denotes the causal intervention operator (Pearl, 2009), representing active manipulation of the variable, forcibly setting it to a given state.

This equation aligns with intuition, as if $r_i$ does not affect the output $Y$ (i.e., $ATE(r_i, X) = 0$), which means if $r_i$ is replaced with $r_i^*$, the output $Y$ will remain consistent with the previous.

To aggregate the causal effects across all individual rules into a unified impact, we compute the expectation over all $ATE(r_i, X)$ by averaging the individual effects across all rules:

$$\begin{aligned} ATE(R, X) &= \mathbb{E}_i(ATE(r_i, X)) \\ &= \frac{1}{k} \Sigma_{i=0}^{k} ATE(r_i, X). \end{aligned} \tag{2}$$

With the calculated ATE, we proceed to refine the initial graph $G_{init}$. For each edge in $G_{init}$, if it is correctly generated by the game rule set $R$ with given input node pair $X_i$, then $ATE(R, X_i) \neq 0$. This indicates that the set of game rules has a measurable influence on the outcome, as illustrated in Figure 3(c). When $ATE(R, X_i) = 0$, it implies that the edge is independent of the entire set of game rules, as demonstrated in Figure 3(d). Therefore, such edges should be removed to ensure that all edges adhere to the set of game rules.

### 3.5 Worker with Agent Assignment

The Worker is responsible for translating the dependency graph of subtasks into concrete action plans for each agent, ensuring that the subtasks are executed efficiently and in alignment with the dependencies. The Worker takes the dependency graph given by the Planner as input and each agent will output the specific actions. The process begins with analyzing the dependency graph with the DFS algorithm. This step identifies all the possible paths comprising the required subtasks while respecting their dependencies. Each path represents a sequence of subtasks that need to be executed. To accomplish the whole task, agents need to finish all the paths. Once an agent finishes its assigned path, it will be assigned to another.

To balance work distribution, the Worker needs to track how "busy" each task path is. Therefore, we introduce a new variable for each path: the busy rate $br$. This variable simultaneously captures two factors, assigned agent numbers in each path and agent density near the path entries. Consider a scenario where $k$ agents are executing path $p$, the busy rate $br$ of the path $p$ is computed as:

$$br_p = \Sigma_{i=0}^{k} \frac{1}{d_i}, \tag{3}$$

where $d_i$ represents the number of subtasks between the current task of agent $i$ and the path entrance. The busy rate of a path $p$ dynamically increases with the number of agents newly assigned to it. It reflects the involvement of the current path and serves as an indicator of potential resource contention. While assigning an agent to a new path, the Worker prioritizes the path $p^*$ with the minimum busy rate, i.e.,

$$p^* = \arg\min_{p} (br_p). \tag{4}$$

Once an agent receives its designated path, it begins executing each subtask in the path sequentially.

| Method | Agent Number | Construction Avg. Score | | | Escape Avg. Score | | Cooking Avg. Score | | | All Avg.* Score |
|---|---|---|---|---|---|---|---|---|---|---|
| | | CR (%) | VHR (%) | Efficiency (%/min) | CR (%) | Efficiency (%/min) | CR (%) | ACR (%) | Efficiency (%/min) | CR (%) |
| AgentVerse (ICLR'23) | 2 | - | - | - | - | - | 29.75 | 48.64 | 3.54 | 29.75 |
| VillagerAgent (ACL'24) | 2 | 36.45 | 49.05 | 3.88 | 73.29 | 149.4 | 73.75 | 58.11 | 6.98 | 56.90 |
| | 3 | 52.17 | 61.02 | 6.26 | 69.78 | 227.4 | 85.26 | 55.60 | **21.90** | 68.82 |
| CausalMACE (Ours) | 2 | 56.59 | 63.17 | 8.94 | **77.08** | 246.71 | 79.10 | 65.53 | 7.17 | 68.76 |
| | 3 | 65.45 | 69.30 | **9.58** | 72.28 | **276.67** | 86.00 | 70.12 | 13.92 | 75.38 |
| | 6 | **76.04** | **78.99** | 8.20 | 69.25 | 169.59 | **88.75** | **72.30** | 16.31 | **81.09** |

Table 1: **Comparison on cooperative tasks.** *CR* denotes completion rate, *VHR* denotes view hit rate, *ACR* denotes agent contribution rate and *Avg.** denotes the weighted average based on the number of tasks. With the same number of agents, our method shows improvements across all settings.

During execution, the agent interacts with the environment, completing actions as specified by the subtasks. Following the ReAct framework (Yao et al., 2023), the agent iteratively generates actions and observes the environment until it makes a successful movement in the environment. The result of movement and environmental feedback will be recorded and the agent will first decide the statement of its current subtask and then update its strategy through a self-reflection process (Ji et al., 2023). This procedure will last until all paths are finished or the Judger returns an end signal.

## 4 Experiment

### 4.1 Experiment Setup

#### 4.1.1 Environment

We set up Minecraft Java Edition Server version 1.19.2 and host it on Ubuntu 20.04. Following Wang et al. (2024a); Dong et al. (2024), Mineflayer (PrismarineJS, 2013), a JavaScript mod is installed to establish a platform for agents to get information from the environment and take action. Details are provided in Appendix A.

#### 4.1.2 Benchmarks and Metrics

**Multi-agent cooperative tasks.** We use VillagerBench (Dong et al., 2024), which includes three types of multi-agent tasks: construction cooperation, farm-to-table cooking and escape room challenge. Construction cooperation requires agents to understand the specified blueprint and complete the construction. Farm-to-table cooking requires the agent to perform hunting or harvesting according to the recipe to obtain ingredients and then gather all the ingredients together to craft the task object. The escape room challenge requires agents to individually activate the correct mechanisms to collectively solve the puzzle.

There are three different evaluation metrics to assess the effectiveness of the method for all these three tasks, i.e., completion rate (**CR**), efficiency (**E**) and balanced agent utilization score (**BS**). Specifically, the completion rate represents task progress, with multiple indicators in each task for calculation. Assume the number of accessed indicators and total indicators amount are $I_a$ and $I_t$, completion rate can be defined as Equation (5):

$$\mathbf{CR} = \frac{I_a}{I_t}. \tag{5}$$

Efficiency describes the ratio of the completion rate and the time agents take to execute the action. It measures the extent to which actions chosen by the agents contribute to the task completion. In an $N$ agents scenario, this metric can be calculated by the following equation, where $ET$ denotes the agents' execution time set, $t_j$ denotes the execution time of agent $j \in \{1, 2, \ldots, N\}$:

$$\mathbf{E} = \frac{\mathbf{CR}}{\Sigma_j t_j}, t_j \in ET. \tag{6}$$

BS aims to evaluate the distribution of workload. A higher BS should indicate better equilibrium, which means each agent's active running time becomes more similar. Specifically:

$$\mathbf{BS} = 1 - \frac{1}{N}\Sigma_j(std(\frac{|t_j - \min(ET)|}{T_{max} - \min(ET)})), \tag{7}$$

where $T_{max}$ indicates the maximum action time for each task, actions exceeding this time limit will be considered as timed out. These evaluation metrics simultaneously assess the method's task completion ability and its capability to utilize multiple agents. Despite the three primary metrics **CR**, **E** and **BS**, some tasks incorporate auxiliary evaluation metrics such as view hit rate (**VHR**) and agent contribution rate (**ACR**). Detailed information on tasks and corresponding metric formulations are provided in Appendix C.

| Method | No. | Construction (%) | Escape (%) | Cooking (%) | Avg.* (%) |
|---|---|---|---|---|---|
| AgentVerse | 2 | - | - | 91.41 | 91.41 |
| VillagerAgent | 2 | 91.09 | 92.15 | **95.45** | 92.00 |
| | 3 | 87.51 | 75.99 | 63.36 | 79.93 |
| CausalMACE (Ours) | 2 | **91.17** | 94.08 | 94.62 | 92.82 |
| | 3 | 90.50 | **96.34** | 92.41 | **93.32** |
| | 6 | 86.07 | 91.38 | 90.81 | 88.93 |

Table 2: **BS on each task.** *No.* denotes the agent number and *Avg.** is the weighted average based on the number of tasks. A higher BS represents a more balanced workload.

| Method | Wood | Cobblestone | Iron | Gold | Diamond |
|---|---|---|---|---|---|
| **RL-based Agents** | | | | | |
| STG-T (NeurIPS'24) | 6.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| VPT-bc (NeurIPS'22) | 20.0 | 18.0 | 0.0 | 0.0 | 0.0 |
| PTGM (ICLR'24) | 63.0 | 59.0 | 0.0 | 0.0 | 0.0 |
| VPT-rl (NeurIPS'22) | 99.0 | 99.0 | 60.0 | 0.0 | 15.0 |
| **LLM-based Agents** | | | | | |
| ReAct (ICLR'23) | 26.7 | 20.0 | 0.0 | 0.0 | 0.0 |
| Inner Mono (PMLR'23) | 36.7 | 66.7 | 0.0 | 0.0 | 0.0 |
| DEPS (NeurIPS'24) | 77.0 | 48.5 | 16.3 | 0.0 | 0.6 |
| Jarvis-1 (TPAMI'24) | 91.6 | 94.2 | 33.8 | 14.5 | 9.2 |
| Optimus-1 (NeurIPS'24) | 98.6 | 92.4 | 46.7 | 8.5 | 11.6 |
| Voyager* (TMLR'24) | **100.0** | **100.0** | 76.5 | 52.9 | 29.4 |
| **CausalMACE (Ours)** | **100.0** | **100.0** | **82.3** | **70.6** | **41.1** |

Table 3: **Success rate on single-agent tasks.** * denotes that we re-conduct the method for adapting our settings.

**Single-agent tasks.** Following Wang et al. (2024a,c); Li et al. (2024b), we choose five representative items in Minecraft gameplay, i.e., Wood, Cobblestone, Iron Ingot, Gold Ingot and Diamond. The tasks require an agent to obtain each item from scratch with an empty inventory respectively, which is strongly relative to the agent's planning capability and long-term exploration ability. To evaluate methods on these tasks, we directly utilize success rate (**SR**) as the metric.

## 4.2 Comparison with State-of-the-Arts

### 4.2.1 Main Results

In this section, we compare our method with previous works. For multi-agent collaborative tasks, we compare with AgentVerse (Chen et al., 2023) and VillagerAgent (Dong et al., 2024). For single-agent tasks, we compare with traditional reinforcement learning methods such as VPT (Baker et al., 2022), STG-T (Zhou et al., 2024), PTGM (Yuan et al., 2024)), and LLM-based methods such as vanilla ReAct (Yao et al., 2023), Inner Mono (Huang et al., 2023), DEPS (Wang et al., 2024b), Jarvis-1 (Wang et al., 2024c), Optimus-1 (Li et al., 2024b) and Voyager (Wang et al., 2024a). We use GPT-4o with default hyper-parameters for LLM-based methods.

The results on multi-agent collaborative tasks are listed in Table 1 and 2. Our method achieves superior performance over baseline approaches across most of the settings in both completion rate and efficiency, with particularly notable gains in construction cooperation. This result reveals the effectiveness of our method in multi-agent scenarios. While maintaining an excellent task completion rate and higher average BS, we observe a slight decrease in BS in several settings. These variations primarily stem from that some subtasks are naturally harder than others. When agents are assigned to a path with difficult subtasks, they take longer to finish. Such variations highlight broader challenges in workload balancing by estimating task durations, which are independent of our core method and need further research in the future.

The results on single-agent collaborative tasks are listed in Table 3. Our approach achieves perfect completion in basic single-agent scenarios while maintaining competitive performance as task complexity increases. This effectiveness is likely because, with our causal intervention, the agent can more reasonably schedule and execute subtasks. These results show that our method is also suitable for single-agent tasks.

### 4.2.2 Dynamic Agent Numbers

Here we discuss how agent quantity influences task performance. Existing study (Dong et al., 2024) suggests that while scaling agent numbers initially improves outcomes, excessive agents may trigger resource competition and reduce effectiveness. However, we argue that for fixed-task scenarios, increasing agents could lower individual efficiency but still elevate collective performance, provided that coordination mechanisms and resource allocation are properly optimized.

As demonstrated in Table 1, the completion rate keeps increasing with the additional agents attending the task, except for the escape room challenge, where the difficulty of the task increases with the number of agents. These results indirectly reflect the adaptability of our framework in managing multi-agent collaboration under varying team sizes.

## 4.3 Ablation Study

We conduct ablation studies on construction tasks to validate our framework's key components. We specifically choose construction tasks due to their comprehensive requirements, including spatial planning, material collection, and real-time pathfinding in dynamic environments. The results


Figure 4: **Visualization of construction cooperation.** VillagerAgent exhibits incoherent structural features, while CausalMACE maintains structural rationality and high consistency to ground truth with only minor local difference.

| Row | Setting | | | Construction | | |
|---|---|---|---|---|---|---|
| | Busy Rate | Causal | Graph | CR (%) | VHR (%) | Efficiency (%/min) |
| 1 | ✘ | ✘ | ✘ | 52.68 | 55.99 | 3.93 |
| 2 | ✓ | ✘ | ✘ | 57.53 | 56.70 | 5.31 |
| 3 | ✘ | ✘ | ✓ | 57.92 | 61.08 | 5.12 |
| 4 | ✓ | ✘ | ✓ | 60.60 | 64.61 | 7.73 |
| 5 | ✘ | ✓ | ✓ | 72.49 | 75.20 | 6.98 |
| 6 | ✓ | ✓ | ✓ | **76.04** | **78.99** | **8.20** |

Table 4: **Ablation study on construction cooperation.** We adopt 6 agents for each setting.

| Method | Pillager Tents  | Ruined Portal  |
|---|---|---|
| Luban | 172.7 | 311.2 |
| VillagerAgent | 101.1 | 175.7 |
| **CausalMACE** | **81.0** | **146.4** |

Table 5: **Average time consumption (seconds).** The image denotes the goal of construction. We adopt 6 agents for each method.

are listed in Table 4.

**Row 6** in Table 4 corresponds to the complete framework with all components included, which achieves the best completion rate and efficiency.

The removal of **Busy Rate** means to randomly assign paths to agents instead of workload-based allocation. **Row 5** shows it has few effects on the completion rate but has a significant impact on efficiency, as it may cause repetitive actions and result in an unbalanced workload among agents, increasing execution time.

The removal of **Causal Intervention** indicates to utilize the initial graph $G_{init}$ without causal refinement. This leads to a significant decrease in the completion rate as shown in **Row 4**, representing the importance of causality to maintain the correct dependencies among subtasks.

The removal of **Graph** means to construct no graph and directly assigns subtasks to agents. Note that without **graph**, **causal intervention** is disabled automatically. **Row 2** demonstrates that the graph removal further influences the completion rate and efficiency simultaneously, indicating the foundational importance of the graph.

**Row 1** and **Row 3** represent the joint ablation of Busy Rate with Causal Intervention and Graph, respectively. The results indicate that, regardless of whether Causal Intervention or Graph is present, the absence of Busy Rate leads to a degradation in efficiency.

### 4.4 Case Study

Figure 4 compares the results of construction tasks between our method and VillagerAgent. While VillagerAgent produces almost no discernible structure, our method closely resembles the ground truth, with only a few blocks being incorrect or missing.

We also compare CausalMACE with Luban (Guo et al., 2024), which specializes in construction tasks. While Luban has demonstrated superior performance in pure construction, it does not address a complete task pipeline that includes resource acquisition and logistics management. Therefore, we omit these phases for Luban in our comparison. Even with these advantages, Luban ultimately completes tasks with significantly higher time consumption. Table 5 shows the time cost of different methods. Luban far exceeds the time consumption of other methods, highlighting CausalMACE's operational efficiency.

## 5 Conclusion

We present a novel global causality planning framework that significantly enhances the capabilities of multi-agent systems in open-world environments. By leveraging causality to manage and construct dependencies among subtasks from a global view, our approach ensures more efficient task arrangements. This addresses the limitations of existing multi-agent methods and enhances the overall effectiveness of task execution. Additionally, our findings suggest that incorporating causality into task planning can provide a more structured and efficient approach to managing complex tasks, leading to the possibility for further exploration.

## Limitations

While our framework demonstrates significant performance improvements across various Minecraft tasks, constraints still exist. The current causal intervention highly depends on the reasoning capacity of LLMs, making smaller models less viable options. This suggests that future improvements may leverage causality during model pretraining or fine-tuning for better adaptability.

## Acknowledgments

This research is supported by the National Natural Science Foundation of China (No. 62406267), Tencent Rhino-Bird Focused Research Program and the Guangzhou Municipal Science and Technology Project (No. 2025A04J4070).

## References

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. 2022. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654.

Damir Ćavar, Zoran Tiganj, Ludovic Veta Mompelat, and Billy Dickson. 2024. Computing ellipsis constructions: Comparing classical nlp and llm approaches. In *Proceedings of the Society for Computation in Linguistics 2024*, pages 217–226.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better llm-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. 2023. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

David Maxwell Chickering and Christopher Meek. 2015. Selective greedy equivalence search: finding optimal bayesian networks using a polynomial number of score evaluations. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 211–219.

Kai-Hendrik Cohrs, Gherardo Varando, Emiliano Diaz, Vasileios Sitokonstantinou, and Gustau Camps-Valls. 2024. Large language models for constrained-based causal discovery. *arXiv preprint arXiv:2406.07378*.

Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. 2024. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft. *arXiv preprint arXiv:2406.05720*.

Honghao Fu, Junlong Ren, Qi Chai, Deheng Ye, Yujun Cai, and Hao Wang. 2025. Vistawise: Building cost-effective agent with cross-modal knowledge graph for minecraft. *Preprint*, arXiv:2508.18722.

Yuxuan Guo, Shaohui Peng, Jiaming Guo, Di Huang, Xishan Zhang, Rui Zhang, Yifan Hao, Ling Li, Zikang Tian, Mingju Gao, et al. 2024. Luban: Building open-ended creative agents via autonomous embodied verification. *arXiv preprint arXiv:2405.15414*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2023. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, pages 1769–1782. PMLR.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating llm hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843.

Emre Kıcıman, Robert Ness, Amit Sharma, and Chenhao Tan. 2023. Causal reasoning and large language models: Opening a new frontier for causality. *arXiv preprint arXiv:2305.00050*.

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

Hao Li, Xue Yang, Zhaokai Wang, Xizhou Zhu, Jie Zhou, Yu Qiao, Xiaogang Wang, Hongsheng Li, Lewei Lu, and Jifeng Dai. 2024a. Auto mc-reward: Automated dense reward design with large language models for minecraft. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16426–16435.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. 2024b. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2024. A dynamic llm-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*.

Stephanie Long, Alexandre Piché, Valentina Zantedeschi, Tibor Schuster, and Alexandre Drouin. 2023. Causal discovery with language models as imperfect experts. In *ICML 2023 Workshop on Structured Probabilistic Inference & Generative Modeling*.

Judea Pearl. 2009. Causal inference in statistics: An overview.

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

Joseph D Ramsey. 2015. Scaling up greedy causal search for continuous variables. *arXiv preprint arXiv:1507.07749*.

Peter Spirtes, Clark Glymour, and Richard Scheines. 2001. *Causation, prediction, and search*. MIT press.

Peter L Spirtes, Christopher Meek, and Thomas S Richardson. 2013. Causal inference in the presence of latent variables and selection bias. *arXiv preprint arXiv:1302.4983*.

Boshi Wang, Xiang Yue, and Huan Sun. 2023. Can chatgpt defend its belief in truth? evaluating llm reasoning via debate. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11865–11881.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Shawn Ma, and Yitao Liang. 2024b. Describe, explain, plan and select: interactive planning with llms enables open-world multi-task agents. *Advances in Neural Information Processing Systems*, 36.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. 2024c. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Junda Wu, Tong Yu, Xiang Chen, Haoliang Wang, Ryan Rossi, Sungchul Kim, Anup Rao, and Julian McAuley. 2024. Decot: Debiasing chain-of-thought for knowledge-intensive tasks in large language models via causal intervention. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14073–14087.

Jing Xiang and Seyoung Kim. 2013. A* lasso for learning a sparse bayesian network structure for continuous variables. *Advances in neural information processing systems*, 26.

Fangzhi Xu, Zhiyong Wu, Qiushi Sun, Siyu Ren, Fei Yuan, Shuai Yuan, Qika Lin, Yu Qiao, and Jun Liu. 2023. Symbol-llm: Towards foundational symbol-centric interface for large language models. *arXiv preprint arXiv:2311.09278*.

Lin Xu, Zhiyuan Hu, Daquan Zhou, Hongyu Ren, Zhen Dong, Kurt Keutzer, See Kiong Ng, and Jiashi Feng. 2024. Magic: Investigation of large language model powered multi-agent in cognition, adaptability, rationality and collaboration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7315–7332.

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

Haoqi Yuan, Zhancun Mu, Feiyang Xie, and Zongqing Lu. 2024. Pre-training goal-based models for sample-efficient reinforcement learning. In *The Twelfth International Conference on Learning Representations*.

Junkun Yuan, Xu Ma, Ruoxuan Xiong, Mingming Gong, Xiangyu Liu, Fei Wu, Lanfen Lin, and Kun Kuang. 2023a. Instrumental variable-driven domain generalization with unobserved confounders. *ACM Transactions on Knowledge Discovery from Data*, 17(8):1–21.

Lifan Yuan, Yangyi Chen, Ganqu Cui, Hongcheng Gao, Fangyuan Zou, Xingyi Cheng, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023b. Revisiting out-of-distribution robustness in nlp: Benchmarks, analysis, and llms evaluations. *Advances in Neural Information Processing Systems*, 36:58478–58507.

Matej Zečević, Moritz Willig, Devendra Singh Dhami, and Kristian Kersting. 2023. Causal parrots: Large language models may talk causality but are not causal. *Transactions on Machine Learning Research*.

Yuzhe Zhang, Yipeng Zhang, Yidong Gan, Lina Yao, and Chen Wang. 2024. Causal graph discovery with retrieval-augmented generation based large language models. *arXiv preprint arXiv:2402.15301*.

Bohan Zhou, Ke Li, Jiechuan Jiang, and Zongqing Lu. 2024. Learning from visual observation via offline pretrained state-to-go transformer. *Advances in Neural Information Processing Systems*, 36.

Chang Zong, Yuchen Yan, Weiming Lu, Jian Shao, Eliot Huang, Heng Chang, and Yueting Zhuang. 2024. Triad: A framework leveraging a multi-role llm-based agent to solve knowledge base question answering. *arXiv preprint arXiv:2402.14320*.

Arkaitz Zubiaga. 2024. Natural language processing in the era of large language models.

## A Action Functions

We provide the action functions that are available for agents here.

**NavigateTo**(*<pos>*). Move to the given position. The function returns True if the movement succeeds and False if the position is invalid or no valid path exists to the destination.

**CheckContainer**(*container*). Access the specified container. The function returns information about the contents of the container.

**WithdrawItem**(*container*, *item*, *amount*). Attempt to retrieve *amount* of *item* from the *container*. The function returns the success status of the action.

**ScanEntities**(*item*, *distance*). Attempt to locate *item* within *distance*. The function returns the position of the item. If there is no *item* within *distance*, the function returns *None*.

**Equip**(*item*). Put *item* in the main hand slot for usage or placement. The function returns True if it succeeds and False if the agent has no *item* in inventory.

**PlaceBlock**(*item*, *<pos>*). Attempt to place *item* at *<pos>*. The function returns True if the placement succeeds, and False with the reason if the placement fails. The reasons include: the agent has not equipped the *item*, the agent is too far from *<pos>*, the *<pos>* is in the air without support blocks, the *<pos>* is taken by the other block, etc.

**Handover**(*item*, *agent*). Give *item* to *agent*. The function returns the success status of the action.

**Craft**(*item*, *amount*). Attempt to craft *amount* of *item*. The function returns True if the crafting succeeds and False with the recipe and reason. Possible reasons include: the agent does not have enough ingredients in the inventory, the agent is too far from the crafting table, etc.

**Smelt**(*item*, *amount*, *fuel*). Attempt to use the furnace to smelt *item* with *fuel*. The function returns the same information with **Craft**.

**MineBlock**(*<pos>*). Attempt to dig the block at *<pos>*. The function returns True if the mining succeeds and False with the reason. Possible reasons include: *<pos>* is air, the agent is too far from *<pos>*, the agent does not have proper tools, etc.

**Toggle**(*<pos>*). Attempt to operate an interactive object at *<pos>* (e.g., doors, pressure plates, buttons). The function returns the previous and current status of the object if the operation succeeds.

**UseOn**(*target*). Attempt to use the tool in the main hand slot to *target*. The function returns True if the action is valid and succeeds. Valid actions include: use shears on sheep, use bucket on cow, etc.

**Attack**(*target*). Attack the target with the tool in the main hand slot. The function returns True if this attack succeeds.

## B Reference Prompts

We have shared the reference prompts we use for different modules below. Table 6 shows the prompt structure for task decomposition, while Table 7 outlines the prompts for dependency prediction. Environmental perception and historical action summarization prompts are detailed in Table 8. The execution logic for agents is guided by Table 9, and the prompts for reflection are shown in Table 10. Our demo video can be found in the attached file.

## C Detailed Experimental Setups and Metrics

This section will describe our experimental setups and metrics in more detail. For multi-agent cooperative tasks, we conduct experiments on three different scenarios, including construction cooperation, farm-to-table cooking and escape room challenge. All the settings (e.g., task amounts, metrics) follow VillagerBench (Dong et al., 2024). For single-agent tasks, we conduct experiments on item gathering following (Wang et al., 2024c; Li et al., 2024b). We will introduce the settings respectively.

### C.1 Construction Cooperation

Construction cooperation requires agents to construct a structure or a building following a specified blueprint which includes various blocks. Since the process of obtaining blocks can be complex, considering the inherent task difficulty, the requirement for agents to mine blocks is omitted, which keeps the setting consistent with VillagerBench (Dong et al., 2024). This task evaluates the understanding of spatial dependencies and task requirements in multi-agent collaboration.

**Completion Rate in Construction Cooperation**: The completion rate in this task represents the block hit rate, which is the ratio of correctly placed blocks to the total number of blocks. Only blocks with the correct orientation, type and position are considered correctly placed. The formal

## Task Decomposition:

Your current mission is to lead all the players and execute a set of specified tasks within the Minecraft environment.

--- Background Information ---

Our system manages the task as a Graph. In this turn, you need to decompose the tasks into steps, each step can be seen as a node. Next, we will construct the nodes into a graph. You can use the function to present each step, these functions should follow the rules below:

- Functions take some arguments, you need to decide how to design the function.

- Functions should be composed by atom steps, Atom steps are as follows: $\{available\ actions\}$.

- Functions should be executable, you need to make sure no matter which state the agent is in, it can achieve the goal by calling the function.

The function has the following JSON component:

{

"name": string, function name.

"call": list of string, the argument list you input if you want to call the function.

"description": description of the function.

"function body": how does the function work.

}

A node has the following JSON component:

{

"id": int, id of the step start from 1.

"description": string, description of the step, more detail than a name, for example, place block needs position and facing, craft or collect items needs the number of items.

"step": the function you choose and the arguments you need to input.

}

*** Important Notice ***

- Task Decomposition: These atom steps should be small, specific, and executable with MineFlayer code, as you will be using MineFlayer to play Minecraft. Each atom step should contribute to the completion of the overall task. When necessary, the sub-tasks can be identical for faster task accomplishment. Be specific for the atom steps, for example, make sure to specify the materials needed.

- After all the steps are done, you need to make sure the whole task is done.

- In Minecraft, an item can be put in the agent's inventory, chest, or on the ground. You can use the item in the agent's inventory or chest, but you can not use the item on the ground.

- Integration and Finalization: In some tasks, you will need to integrate your individual efforts. For example, when crafting complicated stuff that requires various materials, after collecting them, you need to consolidate all the materials with one of the players.

Here is the query:

The environment information around: $\{env\}$.

The high-level task: $\{task\}$.

Your response should exclusively include:

- All Functions;

- Complete List of Nodes.

Formatted as: "functions": [], "nodes": [].

Your Response should contain ALL definitions of the functions and a COMPLETE list of nodes structed as ONE JSON.

Table 6: Prompts for construction task decomposing.

## Subtask Dependencies Prediction

Your current mission is to lead all the players and execute a set of specified tasks within the Minecraft environment.

– Background Information –

Our system manages the task as a Hierarchical Causal Graph. In this turn, I will give you a list of nodes, and you need to estimate whether certain nodes have a causal effect on other nodes.

You should also check if other nodes affect the node. If node a have causal effect on node x and node y, which means if a is not completed, x and y can not start, you should add new jsons to the edge list {"chosen_node": a, "target_node": x}, {"chosen_node": a, "target_node": y}.

If node z has a causal effect on the certain node a, which means if z is not completed, a can not start, you should add a new JSON to the edge list {"chosen_node": z, "target_node": a}.

*** Important Notice ***

The causal effect should follow some basic Minecraft rules.

- You can not place/use something unless you get it from container first;

- You can not place something if you have not equipped it;

- The block at the lower place should be placed first, and the block at the higher place should be placed later.

Here is the query: $\{nodes\}$.

The node you need to deal with now: $\{node\,id\}$.

Your response should exclusively include:

- ALL Causal Effect of the chosen node.

Formatted as: {"causal effect": []}.

Think step by step. Your Response should formatted as ONE JSON.

Table 7: Prompts for construction dependencies prediction.

## Environmental Data Summarization:

You are a helpful assistant in Minecraft. Based on the environment info and the task, extract the key information and summarize the environment info in a concise and informative way. You should focus on the entities, blocks and creatures in the environment, and provide a summary of the environment info.

The environment info: $\{info\}$

The task: $\{task\}$.

Return with Entity, Blocks, Creatures, Interactive-Items and give all these positions of these blocks and entities like chest, crafting table, furnace, animals and plants.

## History Action Summarization:

You are a helpful assistant in Minecraft. Your name is $\{name\}$. Your task is to create a concise running summary of actions and information results in the provided text, focusing on key and potentially important information to remember.

You will receive the current summary and your latest actions. Combine them, adding relevant key information from the latest development in 1st person's past tense and keeping the summary concise. The subject of the sentence should be $\{name\}$.

Summary So Far: $\{summary\}$.

Latest Actions: $\{actions\}$.

Return with your summary.

Table 8: Prompts for data and history summarizing.

## Agent Execution

The relevant data of task: $\{task\}$.

The state of the agent: $\{state\}$.

The agent's actions in the last time segment partially: $\{history\}$.

The environment info: $\{env\}$.

The agent's inventory status: $\{inventory\}$.

The Minecraft knowledge card:

- In minecraft world x,z is the horizontal coordinate, y is the vertical coordinate.

- You can place the block to the world.

- You can find the item in the chest. Item in the chest can not directly be seen or used, take it out and use it or equip it.

- If there is no item in the chest, maybe you can find the item at the other chest, get it from the other agent, dig it up or craft it.

- One bucket can hold one item, if you want to get more items, you need to get more buckets at first.

- Do not change the blocks other agents placed without permission.

- Your height is two block, if you need to move to somewhere and you find your target position is in the air, You can check a location that is one level below your target location.

- You should first try to take the action, only rethink if fail to do so.

- If you think you have something in your inventory, you should check your inventory instead of scanning around.

- Do not ask other players to give you something, because they need to do their work. Only require item if you fails to get it from env or chest.

- If you need to place a certain block at a certain place, and the place is already occupied by the kind of block you need to place, you should treat it as you have completed the task. Note: only do this if you can make sure that the block there is the same block as the block you need to place!

Think by steps and take action.

Table 9: Prompts for agent execution.

## Agent Reflection

You are in a Minecraft world. You are an agent. You need to use the action history compared with the task description to check whether the task is completed.

The check-structure:

{

"reasoning": str, the reasoning process.

"summary": str, the summary of the vital information of action history with detailed position number and other parameters, which are not included in the task description.

"status": bool, whether the task is completed.

}

Now you have tried to complete the task.

The task description is: $\{task\}$.

The action history is: $\{history\}$.

The state of the agent is: $\{state\}$.

Please check whether the task is completed and return a check-structure JSON.

Table 10: Prompts for agent reflection.

definition is as follows:

$$\textbf{CR} = \frac{\text{card}(CP)}{\text{card}(B)}, \tag{8}$$

where $B$ indicates the blocks set of the blueprint, $CP \subseteq B$ indicates the set of correctly placed blocks, $\text{card}(\cdot)$ indicates the cardinality of a set. A higher completion rate indicates a closer match to the blueprint, reflecting the agents' ability to accurately execute the construction plan.

**View Hit Rate in Construction Cooperation**: View hit rate (**VHR**) is a special metric for construction cooperation, it is used to assess the structural integrity and visual coherence of a construction from multiple viewpoints. By comparing the overlap between the constructed structure and the expected structure across a predefined set of viewpoints, a higher **VHR** score indicates a closer match between the actual and expected structures, suggesting better structural integrity and visual coherence. This metric is calculated by the Intersection over Union (IoU) of the constructed structure with the blueprint across various viewpoints. The formal definition is as follows:

$$\textbf{VHR} = \frac{1}{V}\sum_{i=1}^{V} IoU(\frac{P_i}{B_i}), \tag{9}$$

where $P_i$ indicates the constructed structure seen from viewpoint $i$, $B_i$ indicates what should the structure looks like from viewpoint $i$. $V$ is the number of viewpoints.

### C.2 Farm-to-Table Cooking

Farm-to-table cooking requires agents to prepare dishes according to the given recipe. In this task, agents need to gather information from the environment and adjust their strategies to collect ingredients from containers or sometimes to harvest crops and hunt animals in the wild. This task evaluates the agents' environmental perception and coordination capacities.

**Completion Rate in Farm-to-Table Cooking**: The completion rate in farm-to-table cooking represents the progress of the recipe. Assume that the recipe for preparing a certain dish needs $M$ ingredients and $N$ actions, the completion rate should be:

$$\textbf{CR} = \frac{\sum_{i=1}^{M} k_{mi} \times S_{mi} + \sum_{j=1}^{N} k_{aj} \times S_{aj}}{\sum_{i=1}^{M} S_{mi} + \sum_{j=1}^{N} S_{aj}}, \tag{10}$$

where $k$ represents the status of a certain ingredient or action. If the ingredient is gathered or the action is taken successfully, $k$ should be one, otherwise, it should be zero. $S_{mi}$ represents the score of $i$-th ingredient and $S_{aj}$ represents the score of $j$-th action. The higher completion rate represents closer to finishing preparing the dish.

**Agent Contribution Rate**: Agent contribution rate (**ACR**) measures if all agents contribute to the task. Assume the score contributed by agent $i$ is $C_i$, average score contributed by each agent is $C_{avg}$, we first calculate the standard deviation $\sigma(C)$ of contribution among all agents:

$$\sigma(C) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(C_i - C_{avg})^2}, \tag{11}$$

where $N$ refers to the number of agents. The deviation $\sigma(C)$ reflects the degree of dispersion of the contribution, which means a smaller $\sigma(C)$ leads to a more balanced distribution. We then calculate the agent contribution rate by normalizing $\sigma(C)$:

$$\textbf{ACR} = 1 - \frac{\sigma(C) - \sigma_{min}}{\sigma_{max} - \sigma_{min}}, \tag{12}$$

where $\sigma_{min}$ refers to the minimum possible deviation value, i.e., all agents contribute the same score, $\sigma_{max}$ refers to the maximum possible deviation value, i.e., one agent finishes all the process.

### C.3 Escape Room Challenge

The escape room challenge tests the agents' ability to work together for synchronization and sequential execution. In this task, agents will need to activate certain objects in the environment in order or simultaneously.

**Completion Rate in Escape Room Challenge** completion rate in the escape room challenge represents the finishing rate of room tasks. The formal definition is as follows:

$$\textbf{CR} = \frac{\sum_{i=1}^{N} \frac{c_i}{a_i} \times S_i}{\sum_{i=1}^{N} S_i}, \tag{13}$$

where $N$ denotes total number of the rooms, $i$ denotes $i$-th room, $c_i$ denotes the number of correct conditions, $a_i$ denotes all the conditions, $S_i$ denotes the assigned score of each task.

### C.4 Item Gathering

Item gathering challenges a single agent to collect specific items in the open-world environment from

scratch. The agent must navigate through different areas, identify the required items and retrieve them while managing potential obstacles. The task requires the agent to develop efficient strategies for exploration, item identification and collection. This task evaluates the agent's ability to autonomously explore the environment, make decisions based on available resources and execute the task while overcoming spatial and temporal challenges.

Specifically, the first task is to collect one log  . This is the most basic task in Minecraft because it serves as the foundational step for many other actions within the game. A log is one of the first resources that players encounter and collecting it unlocks a variety of crafting possibilities, such as turning it into planks, crafting tables, or other essential tools. The next task is to get a cobblestone  . This task is the most basic mining task. It tests whether the agent has the ability to craft and the necessary proficiency in chopping trees, as mining stone requires a wooden pickaxe as a tool, and this tool must be crafted by the agent itself.

If the agent is able to complete the two basic tasks mentioned above, it demonstrates the capability to tackle more advanced tasks. A typical advanced task is to obtain an iron ingot  , which requires mining and using a furnace to smelt ores. Acquiring iron is a fundamental step toward obtaining more advanced resources. The ability to obtain iron ingots indicates that the agent not only has proficient mining skills but is also capable of performing a series of operations such as searching for ores and smelting them.

The last two tasks are obtain a gold ingot  and a diamond  . The last two tasks are the most difficult in the item-gathering objectives. In Minecraft, gold and diamonds require an iron pickaxe to be obtained, which requires 3 iron ingots to be crafted. Therefore, if the agent is not skilled enough to complete all tasks up to acquiring iron ingots, they will not be able to complete these tasks. Even if the agent is skilled at obtaining iron and crafting an iron pickaxe, they will still need to explore underground sufficiently to ensure they find gold ore and diamond ore. Successfully completing these two tasks indicates that the agent is capable of performing high-level tasks in Minecraft.

For metrics on item gathering, we adopt the success rate. Assume that the agent makes $M$ attempts on a certain task, and $N$ of them succeed, the success rate (**SR**) should be:

$$\textbf{SR} = \frac{N}{M}. \tag{14}$$

A higher success rate indicates that the agent is more skilled at completing the task.