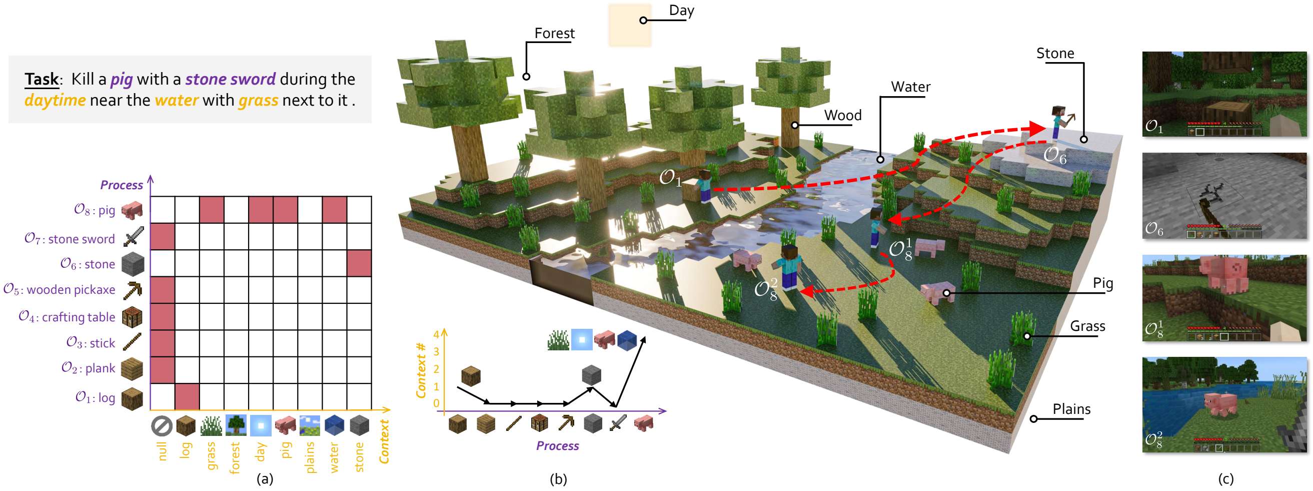
MP5: A Multi-modal Open-ended Embodied System in Minecraft via Active Perception

Yiran Qin[1,2*], Enshen Zhou[1,3*], Qichang Liu[1,4*],
Zhenfei Yin[1,5], Lu Sheng[3†], Ruimao Zhang[2†], Yu Qiao[1], Jing Shao[1‡]
[1]Shanghai Artificial Intelligence Laboratory
[2]The Chinese University of Hong Kong, Shenzhen (CUHK-Shenzhen)
[3]Beihang University [4]Tsinghua University [5]The University of Sydney
`yiranqin@linkcuhk.edu.cn` `zhouenshen@buaa.edu.cn` `liuqc21@mails.tsinghua.edu.cn`

https://iranqin.github.io/MP5.github.io/



Process	null	log	grass	forest	day	pig	plains	water	stone
$\mathcal{O}_8$: pig			■		■	■		■	
$\mathcal{O}_7$: stone sword	■								
$\mathcal{O}_6$: stone									■
$\mathcal{O}_5$: wooden pickaxe	■								
$\mathcal{O}_4$: crafting table	■								
$\mathcal{O}_3$: stick	■								
$\mathcal{O}_2$: plank	■								
$\mathcal{O}_1$: log		■							

Figure 1. The process of finishing the task *"kill a pig with a stone sward during the daytime near the water with grass next to it."*. **(a)** To achieve the final goal (*i.e.*, $\mathcal{O}_8$: "kill a pig 🐖"), a player should accomplish a list of sub-objectives $\{\mathcal{O}_i\}_{i=1}^{7}$ sequentially. During this process, the player should also be aware of some items in the environment, *e.g.*, "grass 🌿", "day ☀" and *etc.* **(b)** This diagram shows the number of these necessary items in the context that should be perceived for each sub-objective, during the task execution process. **(c)** Images marked by $\mathcal{O}_1$ and $\mathcal{O}_6$ show the observed ego-centric views in the process of achieving the corresponding sub-objectives. Images marked by $\mathcal{O}_8^1$ and $\mathcal{O}_8^2$ indicate how the player executes the action about the last sub-objective "kill a pig 🐖". This exemplar process tells that such long-horizon open-world embodied tasks in Minecraft should be solved both in the process-dependent and context-dependent way.

Abstract

It is a long-lasting goal to design an embodied system that can solve long-horizon open-world tasks in human-like ways. However, existing approaches usually struggle with compound difficulties caused by the logic-aware decomposition and context-aware execution of these tasks. To this end, we introduce MP5, an open-ended multimodal embodied system built upon the challenging Minecraft simulator, which can decompose feasible sub-objectives, design sophisticated situation-aware plans, and perform embodied action control, with frequent communication with a goal-conditioned active perception scheme. Specifically, MP5 is developed on top of recent advances in Multimodal Large Language Models (MLLMs), and the system is modulated into functional modules that can be scheduled and collaborated to ultimately solve pre-defined context- and process-dependent tasks. Extensive experiments prove that MP5 can achieve a 22% success rate on difficult process-dependent tasks and a

---

* Equal contribution † Corresponding author ‡ Project leader

91% success rate on tasks that heavily depend on the context. Moreover, MP5 exhibits a remarkable ability to address many open-ended tasks that are entirely novel.

1. Introduction

One of the core objectives of current embodied intelligence is to construct generalist agents that can solve long-horizon open-world embodied tasks, approaching the behavior patterns of human beings [1, 18, 21, 30]. However, the *process dependency* and *context dependency* in these tasks, such as those in Minecraft depicted in Fig. 1, hinder recent agents from achieving the aforementioned goal. To be specific, the former emphasizes the inherent dependency among the sub-objectives of one task or an action sequence to fulfill one sub-objective (such as “craft a stone sword” should be solved before “kill a pig”). The latter highlights that the execution of each sub-objective or even each action depends on the contextual information of the environment (such as “kill a pig” requires to find the target “pig” and its surrounding items “grass” and “water” during the “daytime” in the observed images, as shown in Fig. 1).

The recent success of Large Language Models (LLMs) has attempted to solve the process-dependent challenge, by using LLMs to break down a long-horizon process-dependent task into a sequence of feasible sub-objectives [33, 35, 41]. These methods [33, 41] simplify the context-dependent challenge by assuming the agents are all-seeing, *i.e.*, knowing everything about their state and the environment it locates in. However, to solve the context-dependent challenge, an embodied agent should additionally have: (1) the perception capability is open-ended, selective and give results tailored to diverse purposes (*e.g.*, for task planning or action execution), (2) the perception module can be compatibly scheduled along with the other modules (*e.g.*, planning and execution modules) by a unified interface, as an integrated system.

To this end, we introduce *MP5*, a novel embodied system developed within Minecraft, to meet the above expectations. Specifically, *MP5* comprises five interacting modules, *i.e.*, **Parser** decomposes a long-horizon task into a sequence of sub-objectives that should be completed one by one; **Percipient** answers various questions about the observed images, as the reference for the other modules; **Planner** schedules the action sequences of a sub-objective, as well as refines the following sub-objectives, given the current situation; **Performer** executes the actions along with frequent interaction with the environment; and **Patroller** checks the responses from the Percipient, Planner, and Performer, for the purpose of verifying current plans/actions, or feedback on potential better strategies. In our work, Percipient is a LoRA-enabled Multimodal LLM (MLLM). Among the pre-trained LLMs, Parser and Planner are augmented with external Memory, while Patroller is not.

Notably, *MP5* includes an *active perception* scheme by means of multi-round interaction between Percipient and Patroller, which is to actively perceive the contextual information in the observed images, with respect to various queries raised by Planner and Performer. It is the key enabler to solve context-dependent tasks. Patroller in this scheme relays compatible feedback to Planner and Performer accordingly, while eventually strengthening the planning skill in awareness of the situations and improving the action execution correctness in an embodied manner.

Extensive experiments prove that *MP5* can robustly complete tasks needed for long-horizon reasoning and complex context understanding. It achieved a 22% success rate on diamond-level tasks (*i.e.*, one of the hardest long-horizon tasks) and a 91% success rate on tasks requiring complex scene understanding (*i.e.*, need to perceive around $4 \sim 6$ key items in the observed images). Moreover, in Sec. 4.2.3, *MP5* can surprisingly address more open-end tasks both with heavy process dependency and context dependency.

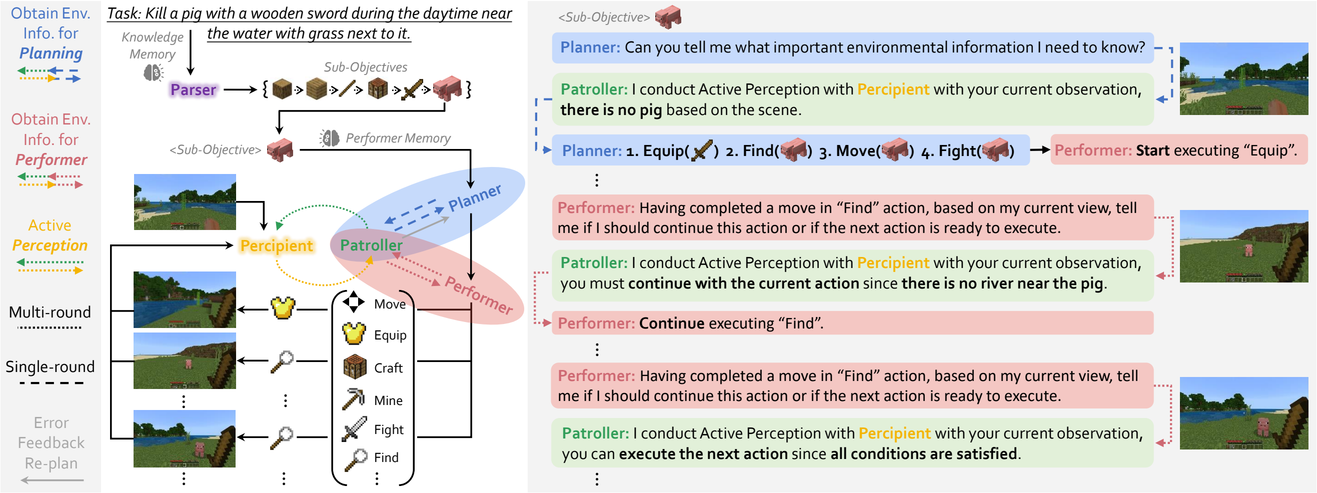
2. Preliminaries and Related Work

2.1. Multi-modal Large Language Models

With the development of Large Language Models (LLMs) like the GPT series [2, 26, 28], as well as open-source LLMs such as the LLaMA series [31, 32] and Vicuna [5], Multimodal Large Language Models (MLLMs) have emerged. Examples of such MLLMs include LLaVA [20], InstructBLIP [6], and LAMM [38], among others [4, 10, 27, 37, 40]. In this work, we introduce an MLLM called MineLLM, which is specifically designed and trained for Minecraft, and leverage its perception, interaction, and analysis capabilities to build up Percipient for *MP5*, and further enable an objective-conditioned active perception scheme.

2.2. Agents in Minecraft

Previous works[3, 7, 9, 11, 18, 21, 39] attempt to use approaches such as hierarchical RL, goal-based RL, and reward shaping to train an agent in Minecraft. MineCLIP [9] enables the resolution of various open-ended tasks specified in free language, even without any manually designed dense rewards. DreamerV3 [12] succeeds in training agents in Minecraft with a learned world model. VPT [1] builds a foundation model for Minecraft by learning from massive videos. Based on VPT, Steve-1 [17] also explores bringing in MineCLIP [9] to get an instruction following policy with high performance. The development of recent large language model-related work Voyager [33], DEPS [35], GITM [41] further promote the advancement of agents in long-horizon tasks. These works use pre-trained large language models as the zero-shot planners[14] for agents, leveraging the powerful reasoning capabilities of large language models to obtain



Planner: Can you tell me what important environmental information I need to know?

Patroller: I conduct Active Perception with **Percipient** with your current observation, **there is no pig** based on the scene.

Planner: 1. Equip() 2. Find() 3. Move() 4. Fight()

Performer: **Start** executing "Equip".

Performer: Having completed a move in "Find" action, based on my current view, tell me if I should continue this action or if the next action is ready to execute.

Patroller: I conduct Active Perception with **Percipient** with your current observation, you must **continue with the current action** since **there is no river near the pig**.

Performer: **Continue** executing "Find".

Performer: Having completed a move in "Find" action, based on my current view, tell me if I should continue this action or if the next action is ready to execute.

Patroller: I conduct Active Perception with **Percipient** with your current observation, you can **execute the next action** since **all conditions are satisfied**.

Figure 2. Overview of module interaction in *MP5*. After receiving the task instruction, *MP5* first utilizes Parser to generate a sub-objective list. Once a sub-objective is passed to the Planner, the Planner Obtaining Env. Info. for Perception-aware Planning. The performer takes frequently Perception-aware Execution to interact with the environment by interacting with the Patroller. Both Perception-aware Planning and Execution rely on the Active Perception between the Percipient and the Patroller. Once there are execution failures, the Planner will re-schedule the action sequence of the current sub-objective. Mechanisms for collaboration and inspection of multiple modules guarantee the correctness and robustness when *MP5* is solving an open-ended embodied task.

continuous operation instructions or executable policy lists.

We take advantage of the reasoning capability of LLM to build up our own agent. Existing LLM agents [41, 41] in Minecraft feed scene data from simulation platforms [9, 11] into large language models for task planning. However, for embodied agents in real scenes, it is clearly unrealistic to use accurate scene data directly. Therefore, agents need to be robust to make decision corrections despite inaccurate or erroneous perception information. Moreover, open-ended tasks need hierarchical reasoning [21] and complex open-ended context understanding [1, 9], classical perception networks can only output fixed perception results and cannot provide corresponding perception information according to the task, making it impossible to understand open-ended scenarios. Therefore, we design *MP5*, an embodied agent with open-ended capabilities that can solve the problem of open-ended tasks.

3. Method

In this section, we first give an overview of our proposed *MP5*, for solving context-dependent and process-dependent tasks in an open-world and embodied environment, such as Minecraft (Sec. 3.1). Next, we elaborate on how to implement an active perception scheme (Sec. 3.2). This scheme plays a vital role in *MP5* to solve context-dependent tasks, since it reliably grounds the visual content according to different kinds of objectives, and thus strengthens the planning skill and execution correctness with respect to context-dependent tasks. Then, we show how to plan and update action sequences in awareness of the situations, and how to reliably execute these actions in an embodied environment (Sec. 3.3). Finally, we give necessary implementation details about *MP5* in Sec. 3.4.

3.1. Overview

As demonstrated in Fig. 2, our *MP5* includes five major modules, *i.e*., Parser, Percipient, Planner, Performer, and Patroller. To be specific, Percipient is a parameter-efficiently fine-tuned Multimodal Large Language Model (MLLM) that is specified to the Minecraft environment. The Parser, Planner, and Patroller are pre-trained Large-language Models (LLMs). We also include retrieval-augmented generation (RAG) to enhance the quality of responses generated by Parser and Planner. Performer is an interface that explains each action from the action sequence into executable commands that directly control the game character.

Why can *MP5* solve context-dependent and process-dependent tasks? *MP5* includes an *active perception* scheme by means of multi-round interactions between Percipient and Patroller, which is to actively perceive the environmental information in the observed images, with respect to various objectives raised by Planner or Performer. With the help of this scheme, Planner can schedule or update action sequences in awareness of the observed images, inventory status and *etc*., resulting in a *situation-aware planning*; Performer can execute actions that are adapted to the embodied environment, resulting in a *embodied action execution*. Patroller in this scheme can also feedback on better choices

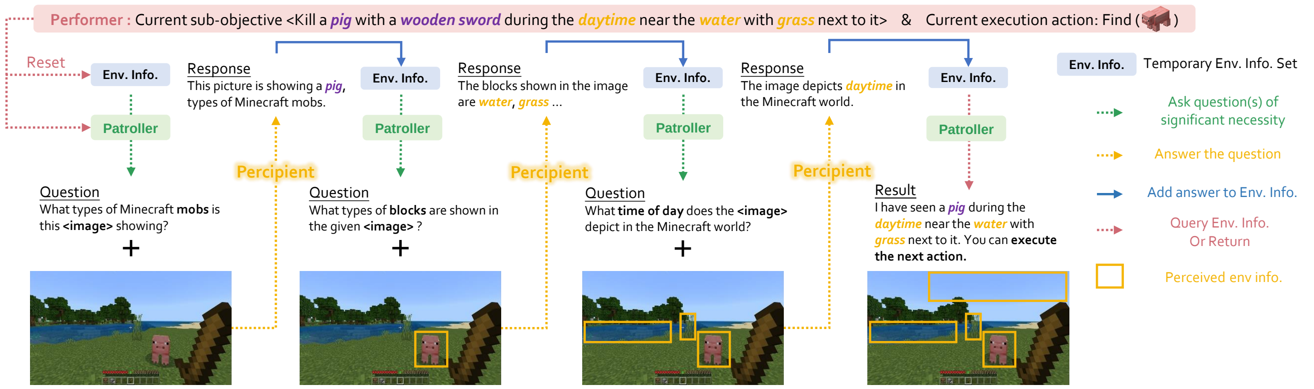

Performer : Current sub-objective <Kill a pig with a wooden sword during the daytime near the water with grass next to it> & Current execution action: Find ( )
Reset
Env. Info.
Patroller
Question
What types of Minecraft mobs is this <image> showing?
+
Response
This picture is showing a pig, types of Minecraft mobs.
Percipient
Env. Info.
Patroller
Question
What types of blocks are shown in the given <image> ?
+
Response
The blocks shown in the image are water, grass ...
Percipient
Env. Info.
Patroller
Question
What time of day does the <image> depict in the Minecraft world?
+
Response
The image depicts daytime in the Minecraft world.
Percipient
Env. Info.
Patroller
Result
I have seen a pig during the daytime near the water with grass next to it. You can execute the next action.
Env. Info.
Temporary Env. Info. Set
Ask question(s) of significant necessity
Answer the question
Add answer to Env. Info.
Query Env. Info. Or Return
Perceived env info.

Figure 3. A demonstration of the process of Active Perception scheme. Temporary Env. Info. Set saves information collected in the current scenario, so it should be reset at the beginning of Active Perception scheme. Performer then invokes Patroller to start asking Percipient questions with respect to the description of the sub-objective and the current execution action round by round. The responses of Percipient are saved in Temporary Env. Info. Set and are also gathered as the context for the next question-answering round. After finishing asking all significant necessary questions, Patroller will check whether the current execution action is complete by analyzing the current sub-objective with Perceived env info. saved in Temporary Env. Info. Set, therefore complex Context-Dependent Tasks could be solved smoothly.

of plans/actions based on the visual evidence so that the process-dependent tasks are solved with fewer chances of context-dependent execution failures. Moreover, Percipient can understand open-ended visual concepts, therefore it allows *MP5* to solve tasks that are never seen before. Based on these merits, *MP5* is able to solve context-dependent and process-dependent tasks in an open-world embodied environment.

How does *MP5* function? In Fig. 2, upon receiving a high-level task, *MP5* first utilizes the Parser to generate a sequence of short-horizon sub-objectives, as a list of rich instructions in natural languages. The feasibility of the generated sub-objectives is augmented by retrieving an external Knowledge Memory (*e.g.*, online wiki, user tips, and *etc.*). To one sub-objective, Planner schedules the action sequence that is grounded by the environmental information gathered by the active perception scheme. In this case, Performer will execute the actual actions by explaining the action sequence that is adapted to the embodied environment, via frequent interaction with the active perception scheme. Once there are execution failures (determined by Patroller), Planner will re-schedule the action sequence of the current sub-objective, or even update the following sub-objectives if some necessary sub-objectives are missing. Otherwise, the agent will go to the next sub-objective and schedule new action sequences, whilst the successful action sequence of the current sub-objective will be stored in the external memory of Planner (called Performer Memory), along with the agent situation when it was planned. In the end, the agent will stop when the final sub-objective of the task has been reached.

3.2. Active Perception

Let's take the example shown in Fig. 3 to demonstrate how the active perception scheme works. In this example, the active perception scheme is communicated with Performer to enable an embodied action execution.

At first, Performer invokes Patroller to start asking Percipient questions with respect to the description of the sub-objective and the current execution action, while simultaneously resetting the set of environmental information to be gathered. Then Patroller progressively asks Percipient whether the observed image contains necessary items/factors (*e.g.*, mobs 🐖 🐄 🐑, blocks ▨ ▨ ▨, time ☀ ☾) related to recent sub-objective (*e.g.*, pig 🐖) and the executing action (*e.g.*, "find pig 🐖"). The responses of Percipient are also progressively gathered and act as the context for the next question-answering round. Note that in each round, Patroller also checks whether all the necessary items/factors have been collected - If yes, Patroller stops the interaction and returns all the environmental information as natural language, and invokes Performer to execute the next action. If Patroller eventually fails to gather enough items/factors, it will tell Performer what items/factors are missing in the observed images, which suggests Performer keeps executing the current action. Please also check the example shown in Fig. 2.

Similarly, active perception used in situation-aware planning is similar to what is explained here, except that the applied instructions do not contain the executable action. For more details please check the Sup. D.

3.3. Perception-aware Planning and Execution

Situation-aware Planning. Given one sub-objective, Planner will generate the action sequence based on the description of the situation, such as the objective-conditioned envi-

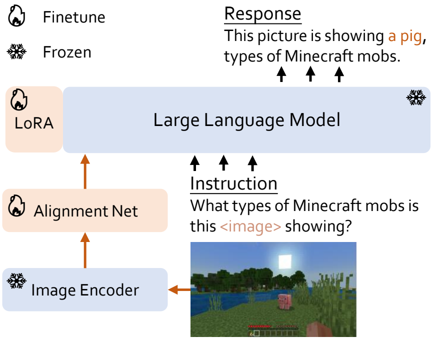

Finetune
Frozen
Response
This picture is showing a pig, types of Minecraft mobs.
LoRA
Large Language Model
Alignment Net
Instruction
What types of Minecraft mobs is this <image> showing?
Image Encoder

Figure 4. The model architecture of MineLLM. Image is encoded by a pre-trained vision encoder and decoded by LLM. Only the parameters of Alignment Net and LoRA are trainable.

ronmental information from the active perception scheme, the inventory status and localization, and *etc*. Moreover, Planner will retrieve previous successful action sequences as the demonstration prompt to augment the aforementioned planning results. If the active perception scheme fails to find the key items/factors about the current sub-objective in the observed image, the generated action sequences will include more actions to reach them. Moreover, if Performer encounters execution failures determined by Patroller (such as failure of "equip wooden sword 🗡"), Planner will reschedule the action sequence or even update the following sub-objectives, with the help of external memories.

Embodied Action Perception. As indicated in Sec. 3.2, Performer would like to communicate with the active perception scheme in every round of action execution, so as to enhance the ego-centric awareness of the agent. The new action will be executed if Patroller identifies necessary environmental information in the observed images that matches both the sub-objective and the goal of the current action. Otherwise, the current action is kept executing until encountering execution failures or the end of the episode. The successful action sequence about one sub-objective will be stored in the Performer Memory, together with necessary situational information of the agent when it was planned. For more details about the planning and execution process, please check Sup. F.2 and Sup. A.2.

3.4. Implementation Details

Percipient. The network of Percipient is depicted in Fig. 4. Images are processed by a frozen vision encoder MineCLIP [9], whose features are projected by an Alignment Net(we use two-layer MLP like LLaVA-1.5 [19]) to the same feature space as the text embeddings of the applied LLM (we use Vicuna-13B-v1.5 [5]). Then the vision and text tokens are concatenated to feed into a LoRA-based fine-tuned LLM [13]. We add LoRA [13] parameters to all projection layers of the self-attention layers in the LLM. Only the parameters of the Alignment Net and the LoRA module are optimized during training. The construction of the training data with respect to Percipient is in the Sup. A.1.

Parser, Planner, and Patroller. We utilize OpenAI's GPT-4 [25] as LLMs in Parser, Patroller, and Planner. We also evaluate other alternatives of GPT-4 [25], such as open-source models like Vicuna-13B-v1.5 [5] and LLaMA2-70B-Chat [32] in Sup.C.3.

Performer. It is important to clarify that the actions generated by Planner are not low-level commands such as keyboard and mouse operations [1], but a set of simple actions (such as equip, move, craft). Inspired by GITM [41], we implement these actions appropriately through basic operations provided by the MineDojo [9] simulator. For more details, please check the Sup. A.2.

4. Experiments

At first, we depict the setup of the Minecraft simulation environment that we build and validate *MP5*, and give the definition of the evaluated tasks and how to set them in Sec. 4.1. In Sec. 4.2, we present the quantitative and qualitative performance of *MP5*, as well as in-depth discussions on these tasks, and demonstrate that *MP5* can even successfully accomplish tasks that are more open-ended and never seen before. At last, we investigate how different modules affect the performance of *MP5* and analyze the impact of various module choices within our system in Sec. 4.3.

4.1. Experimental Setup

4.1.1 Environment Setting

We employ MineDojo [9] as the Minecraft simulation environment to build and validate *MP5*. We capture player ego-view images provided by MineDojo [9] as input of *MP5*, and further construct a dataset for training MineLLM. As for the output of *MP5*, we encapsulate MineDojo's [9] actions to create our own action space.

4.1.2 Task Setting

To evaluate how our *MP5* can organically integrate perception information with planning and execution, we mainly define two types of tasks: *Context-Dependent Tasks* and *Process-Dependent Tasks* as illustrated in Tab. 1 and Tab. 2.

Context-Dependent Tasks. primarily study how Active Perception enables the agent to better perceive low-level context information in the environment. We first establish 6 aspects of environmental information derived from the Minecraft game environment: *[Object, Mob, Ecology, Time, Weather, Brightness]*. Each aspect has multiple options. For example, pigs 🐖, cows 🐄, and sheep 🐑 are all elements belonging to Mob. Based on this, we define 16 tasks and organize their difficulty into four levels by taking into account the

Table 1. *Context-Dependent Tasks*. 16 tasks are defined and divided into 4 difficulty levels based on the minimum number of information types needed. Underlines label the environmental information, reflecting the complexity varies at each level.

Task Level	Example Task
Easy	Find a tree
Mid	Find a tree in the forest
Hard	Find a tree in the forest during the nighttime
Complex	Find a pig near a grass in the forest during the daytime

Table 2. *Process-Dependent Tasks*. 25 tasks are defined and divided into 5 difficulty levels based on incrementally increasing reasoning steps. A higher difficulty level implies that the agent needs to engage in longer reasoning and planning with the environment.

Task Level	Reasoning Step	Example Task
Basic	1-3	craft crafting table
Wooden	4-5	craft wooden sword
Stone	6-9	mine stone
Iron	10-11	smelt iron ingot
Diamond	>11	obtain diamond

number of information elements that require perception, as is shown in Tab. 1. For example, Easy tasks necessitate the perception of only one element, whereas Complex tasks involve the perception of 4 to 6 elements. We rigorously assess *MP5*'s proficiency in environmental context perception across these 16 tasks. In *Context-Dependent Tasks*, our environment details are predetermined (*e.g.*, biomes , weather , and *etc.*), as certain targets are exclusive to specific environments. Without this environmental specificity, the agent might never encounter the intended target. We retain each observation of active perception throughout the task, using them as references to ascertain the agent's successful completion of the task.

Process-Dependent Tasks. focus on exploring the contributions of situation-aware planning, embodied action execution, and the integration with Active Perception in accomplishing long-term tasks while constantly perceiving the environment and dynamically adjusting actions. We select 25 tasks from the technology tree and define their difficulty levels as Basic level to Diamond level based on the number of reasoning steps required to complete the tasks. All environmental factors (*e.g.*, biomes , weather , and *etc.*) are randomized in *Process-Dependent Tasks*. More details can be found in Sup.C.1.

4.1.3 Evaluation Metrics

For different tasks, the agent's initial position and environment seed are randomized. The agent begins in survival mode, commencing with an empty inventory, and faces the challenge of hostile mob generation. It starts from scratch, with a game time limit of 10 minutes, a time period equivalent to 12,000 steps at a control frequency of 20Hz. More details can be found in Sup. B.

For the *Context-Dependency Tasks*, each assignment is open-ended. Therefore, we conduct manual evaluations when the agent determines it has completed the task or exceeds the time limit. Two cases are ruled as failures: 1)There is an observation that meets all the conditions, but the agent does not end the task; 2) The last observation does not meet all the conditions, yet the agent ends the task. Otherwise, we believe that the agent correctly perceives all the context according to the task and determines that the task is successfully completed. For the *Process-Dependent Tasks*, any accidental deaths of the agent during the game are counted as failures, as are instances where the agent does not accomplish the task within the time limit.

In practice, we conduct 50 games on *Context-Dependent Tasks* and 30 games on *Process-Dependent Tasks*, averaging the success rates for both. The results are grouped according to the previously defined difficulty levels, and report the group means. For detailed definitions of the evaluation, please refer to Sup. C.

4.2. Main Results

4.2.1 Results of Context-Dependent Tasks

In *Context-Dependent Tasks*, we primarily investigate how to enhance an agent's perception of context information within the environment. We demonstrate the performance difference between Active Perception and other perception methods. We compare them with pre-trained multi-modal large language models LLaVA-1.5 [19] and GPT-4V [24], and analyze the performance of both active and fine-grained global perception on the tasks in Tab. 3. Although fine-grained global perception can obtain comprehensive perceptual information, due to the lack of objective-conditioned attention, the objective-related information obtained may be lacking or incorrect. Active perception only focuses on objective-related information and ignores other useless information, so that more accurate objective-related information can be obtained and better performance in *Context-Dependent Tasks* can be achieved. For the comparison, we use MineLLM, which is fine-tuned on the Minecraft instruction dataset we collect, slightly better than GPT-4V [24], which is trained on massive data, and substantially better than LLaVA-1.5 [19], which is not fine-tuned on instruction data. The complete results of *Context-Dependent Tasks* can be found in Sup.C.2.

Table 3. Performance on *Context-Dependent Tasks*. We compare the success rate of different Methods and different Perception strategies. We set up special prompt to make the output of the caption as comprehensive as possible, this perception method is called Fine-Grained Global Perception. We use A to denote Active Perception, and G to denote Fine-Grained Global Perception.

Method	Strategy	Average Success Rate(%)			
		Easy	Mid	Hard	Complex
LLaVA-1.5 [19]	G	47.5	22.5	5.0	0.0
	A	72.5	50.0	11.0	0.0
GPT-4V [24]	G	97.5	85.0	75.0	60.0
	A	**100.0**	94.5	92.5	87.5
***MP5*(Ours)**	G	90.0	82.5	77.5	67.5
	A	98.5	**94.5**	**93.0**	**91.0**

Table 4. Performance on *Process-Dependent Tasks*. We compare the success rate when interacting or not interacting with the environment during the planning or execution. w/o P. and w/o E. indicates non-situation-aware planning and non-embodied action execution.

Method	Average Success Rate(%)				
	Basic	Wooden	Stone	Iron	Diamond
MP5 w/o P.	0.00	0.00	0.00	0.00	0.00
MP5 w/o E.	92.00	86.00	68.67	45.33	14.00
MP5	96.00	88.67	76.00	52.00	22.00

4.2.2 Results of Process-Dependent Tasks

In *Process-Dependent Tasks*, we report the performance of the agent in completing long-horizon tasks by continuously perceiving the environment context and dynamically adjusting its actions. We also investigate the agent's behavior in scenarios of non-situation-aware planning and non-embodied action execution. The complete results of *Process-Dependent Tasks* can be found in Sup.C.2.

In considering the landscape of related works [1, 12, 33, 35, 41], we refrain from making direct comparisons due to the substantial variations in the **observation space**, **action space**, **environmental setup**, and **game termination conditions**. Notably, VPT [1] emulates human players' keyboard and mouse controls, DreamerV3 [12] is trained from scratch for diamond collection in a modified Minecraft environment with altered block-breaking mechanics using world models, DEPS [35] integrates LLM planning and a learning-based control policy based on MineDojo [9] actions, GITM [41] employs privileged information such as lidar perception, and Voyager [33] utilizes purely text-based information perception in collaboration with the Mineflayer API for action. Given that our experiments aim to showcase the system's capability to adapt both process-dependent reasoning and complex context-understanding tasks, our focus turns to presenting two key insights drawn from the system's performance, as detailed below.

Embodied action execution is critical for open-ended tasks. Comparing *MP5* w/o E. and *MP5* in Tab. 4, we can observe that when an agent is unable to interact with the environment and access low-level environment contextual information during action execution, it essentially becomes "blind", unable to determine the termination of its actions based on environment. Therefore, the success rate in *Process-Dependent Tasks* is 0.00%.

Situation-aware planning leads to more scenario-appropriate strategies. Comparing *MP5* w/o P. and *MP5* in Tab. 4, we observe that the lack of environment contextual information during the agent's planning process can lead to erroneous or redundant actions, thereby reducing the success rate (for example, the success rate in diamond-level tasks decrease from 22.00% to 14.00%). Consider a scenario where the current sub-objective is "kill a pig". If a pig is already present, the agent should directly execute "move" to approach without the need to first "find" then "move". However, the relatively small decrease in the success rate can be attributed to the dynamic adjustment of perception and action execution offered by embodied action execution. Simultaneously, when errors are detected, the perceived environmental information and the erroneous actions can be fed back to the planner for re-planning.

4.2.3 Open-Ended Tasks

Processing long-horizon reasoning and understanding complex contexts are interconnected in the real world. For simplicity and comparability of the experimental setup, the first two task settings do not consider the intersection of process and context, as we cannot exhaust all combinations that these two task dimensions can form. Therefore, we refer to tasks that incorporate both Process-Dependent and Context-Dependent elements as *Open-Ended Tasks*. Specifically, these tasks require the agent to perceive different information of the environment at multiple stages of completing sub-objectives. As shown in Fig. 5, we present an example of an *Open-Ended Task*, named *"Dig a block of sand near the water at night with a wooden shovel"*. We conduct extensive validations on this type of task, proving that *MP5* can complete long-sequential tasks in challenging environments. More demonstrations and experimental results of *Open-Ended Tasks* can be found in Sup.E.3.

4.3. Ablation Study

We conduct ablation studies to evaluate the effectiveness of various modules. The experimental setup and the associated success rates are in Sec. 4.1. More detailed ablation studies are listed in Sup.C.3. The following paragraphs present the analyses derived from our ablation studies.

Model pre-trained on massive data of Minecraft can better comprehend the Minecraft appearance styles. We conduct ablation studies on the multi-modal large language

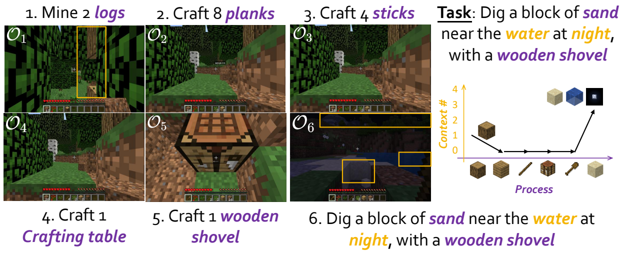


Figure 5. Screenshots of *"Dig a block of sand near the water at night with a wooden shovel"*. In *Open-Ended Tasks*, the agent needs to better integrate low-level context information and high-level decision-making, making it extremely challenging.

Table 5. Success rates for different MLLMs and pre-trained visual encoders in the percipient on *Context-Dependent Tasks*

Method	Visual Encoder	Average Success Rate(%)			
		Easy	Mid	Hard	Complex
LLaVA-1.5 [19]	CLIP [29]	72.50	50.00	11.00	0.00
MineLLM	CLIP [29]	95.00	90.00	87.00	80.00
MineLLM	MineCLIP [9]	98.50	94.50	93.00	91.00

Table 6. Success rates for different LLMs as zero-shot Planner on *Process-Dependent Tasks*

Planner	Average Success Rate(%)				
	Basic	Wooden	Stone	Iron	Diamond
Vicuna-13B-v1.5 [5]	1.33	0.00	0.00	0.00	0.00
GPT-3.5-turbo [22]	95.33	86.67	42.00	2.67	0.00
GPT-4 [25]	96.00	88.67	76.00	52.00	22.00

model (MLLM) part within *Context-Dependent Tasks* in Tab. 5, comparing the performance outcomes of different MLLMs and different pre-trained visual encoders in the percipient. We find the performance of the open-source model LLaVA-1.5 [19] to be relatively weak, with a success rate of merely 50.00% at the Mid level and 11.00% on the Hard level. This is primarily due to the model's training predominantly on real-world data, causing it to struggle with the pixel-style image recognition characteristic of Minecraft. We also discover that, when the visual encoder is frozen, the MineLLM with CLIP [29] as its visual encoder consistently performs worse across all levels compared to MineLLM with MineCLIP's [9] pre-trained single image visual encoder. It may caused by, in the case of a frozen visual encoder, a visual encoder pretrained on massive data of Minecraft can align with pixel-style images more rapidly.

Enhanced reasoning ability results in improved planning. We compare the performance of open-source large language models, OpenAI's GPT-3.5-turbo [22] in Tab. 6, and GPT-4 [25] as zero-shot Planners on *Process-Dependent Tasks*. We find that as the models' inferential capabilities increase, the Planner produces better results by planning in a situation-aware method, yielding more concise and accurate execution actions. The Vicuna-13B-v1.5 [5] model, when used as a Planner, struggles to produce effective plans, achieving only a 1.33% accuracy rate at the Basic level. GPT-4 [25] exhibits the best performance, attaining a 22.00% success rate at the Diamond level, whereas both Vicuna-13B-v1.5 [5] and GPT-3.5-turbo [22] score 0.00%.

Table 7. Success rates on different modules within *Process-Dependent Tasks*: We study the roles of the Performer Memory (PM) and the check part of Patroller (P), with 'RD' denoting *"Random Drop"* setting. ✓denotes the inclusion of the module or setting, and ✗ indicates its absence.

PM	P	RD	Average Success Rate(%)				
			Basic	Wooden	Stone	Iron	Diamond
✗	✓	✗	96.00	87.33	67.33	47.33	16.67
✓	✗	✓	70.00	7.33	0.67	0.00	0.00
✓	✓	✓	87.33	76.67	45.33	18.67	1.33
✓	✓	✗	96.00	88.67	76.00	52.00	22.00

Leveraging memory leads to better planning. In our Performer Memory, we store previously successful sub-objectives and their corresponding execution actions. When planning in similar scenarios, Performer Memory can provide the Planner with similar execution action plans for completing the sub-objectives. While the plans may not be identical, they can effectively assist the Planner in performing situation-aware planning. Comparing the first and last rows of Tab. 7, we find that without the Performer Memory, the success rate of tasks at all levels decreases (Diamond level drops from 22.00% to 16.67%). However, the decrease is not significant as the Performer Memory primarily serves a reference function, with specific action planning still heavily reliant on the Planner's capabilities.

Robustness is essential in open-world settings. To enhance the robustness evaluation of our system, we introduce a *"Random Drop"* setting. In this setting, we randomly discard one complete sub-objective from the inventory at the start of each new sub-objective, which deliberately induces execution errors for the agent. Comparing the second and third lines in Tab. 7, we observe the critical role of the Patroller in recognizing feedback errors. The Patroller's ability to integrate current environmental information with error information is essential for enabling the planner to re-plan. The significance of this robustness is evident when examining the success rates. Without the Patroller's robustness, the agent's success rate on the Wooden level plummets from 76.67% to 7.33%, while success rates on the Iron, and Diamond levels drop to 0.00%. Details regarding the *"Random Drop"* setting can be found in Sup.C.3.

5. Conclusion

In this paper, we propose a novel multi-modal embodied system termed *MP5* which is driven by frequently ego-centric scene perception for task planning and execution. In practice, it is designed by integrating five functional modules to accomplish task planning and execution via actively acquiring essential visual information from the scene. The experimental results suggest that our system represents an effective integration of perception, planning, and execution, skillfully crafted to handle both context- and process-dependent tasks within an open-ended environment.

Limitation and Future Work. Despite the impressive results of our approach, two major limitations need to be clarified. Firstly, the reliance on GPT-3.5-turbo [22] or GPT-4 [25] limits the system's usability, as not everyone has access to these APIs. Secondly, the scope of the applied simulation platform is limited. Despite showing promising performance in Minecraft, we haven't extended our exploration to other simulation platforms, which is a potential area for further research.

References

[1] Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022. 2, 3, 5, 7

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 2

[3] Shaofei Cai, Zihao Wang, Xiaojian Ma, Anji Liu, and Yitao Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13734–13744, 2023. 2

[4] Keqin Chen, Zhao Zhang, Weili Zeng, Richong Zhang, Feng Zhu, and Rui Zhao. Shikra: Unleashing multimodal llm's referential dialogue magic. *arXiv preprint arXiv:2306.15195*, 2023. 2

[5] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2023. 2, 5, 8, 1, 7, 9, 11

[6] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023. 2

[7] Ziluo Ding, Hao Luo, Ke Li, Junpeng Yue, Tiejun Huang, and Zongqing Lu. Clip4mc: An rl-friendly vision-language model for minecraft. *arXiv preprint arXiv:2303.10571*, 2023. 2

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2

[9] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35: 18343–18362, 2022. 2, 3, 5, 7, 8, 1, 4

[10] Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, et al. Llama-adapter v2: Parameter-efficient visual instruction model. *arXiv preprint arXiv:2304.15010*, 2023. 2

[11] William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: a large-scale dataset of minecraft demonstrations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2442–2448, 2019. 2, 3

[12] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023. 2, 7

[13] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 5, 1, 2

[14] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022. 2

[15] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018. 1

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020. 2

[17] Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *arXiv preprint arXiv:2306.00937*, 2023. 2

[18] Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:2112.04907*, 2021. 2

[19] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744*, 2023. 5, 6, 7, 8

[20] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023. 2

[21] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, pages 2661–2670. PMLR, 2017. 2, 3

[22] OpenAI. Introducing chatgpt. 2022. 8, 9, 1, 11

[23] OpenAI. New and improved embedding model. 2022. 2

[24] OpenAI. Gpt-4v(ision) system card. 2023. 6, 7

[25] R OpenAI. Gpt-4 technical report. *arXiv*, pages 2303–08774, 2023. 5, 8, 9

[26] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022. 2

[27] Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. Kosmos-2: Grounding multimodal large language models to the world. *arXiv preprint arXiv:2306.14824*, 2023. 2

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 2

[29] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 8

[30] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022. 2

[31] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 2

[32] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 2, 5, 7, 11

[33] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. 2, 7

[34] Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv:2311.05997*, 2023. 2

[35] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023. 2, 7

[36] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022. 8

[37] Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, et al. mplug-owl: Modularization empowers large language models with multimodality. *arXiv preprint arXiv:2304.14178*, 2023. 2

[38] Zhenfei Yin, Jiong Wang, Jianjian Cao, Zhelun Shi, Dingning Liu, Mukai Li, Lu Sheng, Lei Bai, Xiaoshui Huang, Zhiyong Wang, et al. Lamm: Language-assisted multi-modal instruction-tuning dataset, framework, and benchmark. *arXiv preprint arXiv:2306.06687*, 2023. 2

[39] Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*, 2023. 2

[40] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023. 2

[41] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023. 2, 3, 5, 7

MP5: A Multi-modal Open-ended Embodied System in Minecraft via Active Perception

Supplementary Material

The supplementary document is organized as follows:

- Sec. A: Percipient, Memory, Observation Space and Action Space.
- Sec. B: Environment Setting.
- Sec. C: Task Details, Sucess Rates of All Tasks and Ablation Study.
- Sec. D: Different Strategy of Active Perception.
- Sec. E: Applications Demonstration of *MP5*.
- Sec. F: Interactions in *MP5*.

A. Implementation Details

A.1. Percipient

A.1.1 Data Collection

For data collection, we use Minedojo [9] to obtain Minecraft snapshots which contain a wide array of details within the agent's surroundings, including blocks, biomes, mobs and *etc*. Following the environment creation, we enable our agent to perform a rotation on the spot, capturing snapshots from 12 distinct perspectives spaced 30 degrees apart. For each of these snapshots, we record the ground-truth information about the agent's surroundings by leveraging the data available in the MineDojo [9] observation space such as Lidar. To ensure the exact correspondence between the ground-truth information and the image, the information corresponding to the Field of View region of the image is screened from the Lidar as the ground-truth information of the image.

To compile a comprehensive dataset encompassing various conditions and terrains in Minecraft, we implement a two-step data collection process: acquiring data related to different biomes and gathering data on different mobs. In the first step dedicated to gathering data on diverse biomes, we collect information from all 60 biomes available in MineDojo [9]. For each biome, we sample 20 environments, resulting in a total of $7.2K$ images. In the second phase of gathering data for various mobs, our focus is on collecting images of 9 commonly found mobs in the Minecraft world: zombies, skeletons, creepers, spiders, cows, chickens, sheep, pigs, and wolves. We specifically choose 30 representative biomes from the available 60 Minecraft biomes for this data batch. Among these 9 types of mobs, the first four exclusively appear during the night, while the remaining five can be encountered both during the daytime and nighttime. For the mobs that appear in both periods, each mob type is generated across 30 biomes, with 20 environment samples (10 during the daytime and 10 during the nighttime). This results in the creation of $36K$ images for these five mobs. As for the mobs exclusive to nighttime, they are generated in 30 biomes, with 10 nighttime environment samples per biome and 12 images per environment sample, culminating in the generation of $7.2K$ images.

The data obtained from both the first and second stages contribute to a comprehensive dataset totaling $50K$ images, and we prompt ChatGPT [22] to curate a list of instructions to obtain $500K$ image-text instruction-following data.

A.1.2 MineLLM training details

MineLLM combines the image visual encoder from MineCLIP [9] and the large language models from Vicuna-13B-v1.5 [5]. Images are processed by the frozen vision encoder, whose features are projected by a two-layer MLP named Alignment Net to the same feature space as the text embeddings of the applied LLM. Instructions are tokenized by SentencePiece tokenizer [15], and then the vision and text tokens are concatenated to feed into the LLM model. To better align the feature space of visual image encoder from MineCLIP [9] and large language model from Vicuna [5], we collect $500K$ image-text instruction-following data on the MineDojo [9] following the method detailed in Appendix A.1.1, for the purpose of training MineLLM. Each training instance consists of an image $\mathcal{I}$ and a multi-turn conversation data $(\boldsymbol{x}_1, \boldsymbol{y}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{y}_n)$, where $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are the human's instruction and the system's response at the i-th turn. To train MineLLM efficiently, we add LoRA [13] parameters to all projection layers of the self-attention layers in the LLM. Only the parameters of the Alignment Net and the LoRA [13] module are optimized during training. Multimodal tokens are decoded by the LLM model and the corresponding LoRA [13] parameters.

The training objective of Percipient is defined as:

$$\mathcal{L}\left(\theta_a, \theta_l\right) = \prod_{i=1}^{n} p_\theta\left(\boldsymbol{y}_i \mid \boldsymbol{x}_{<i}, \boldsymbol{y}_{<i-1}, f\left(\mathcal{I}\right)\right), \qquad (1)$$

where θ_a and θ_l correspond to the learnable parameters of the Alignment Net and LoRA [13]. The $\mathcal{I}$ is the image repre-

Table 8. Comparison of Observation Spaces Among Different Methods

Method	Perceptual Observation	Status Observation
GITM [41]	LiDAR rays $10 \times 10 \times 10$ Voxels	life statistics GPS, inventory, equipment
DreamV3 [12]	Ego-View RGB	life statistics inventory, equipment
VPT [1]	Ego-View RGB	$\varnothing$
DEPS [35]	Ego-View RGB $3 \times 3 \times 3$ Voxels	Compass GPS, equipment
JARVIS-1 [34]	Ego-View RGB	life statistics GPS, inventory, equipment location status (biome, weather, *etc.*)
***MP5*(ours)**	Ego-View RGB $3 \times 3 \times 3$ Voxels	life statistics GPS, inventory, equipment

sentation produced by the visual encoder from MineCLIP [9] and $\theta = \{\theta_a, \theta_l, \theta_m, \theta_v\}$, where θ_m and θ_v are frozen parameters of MineCLIP [9] and Vicuna-13B-v1.5 [5]. It is worth noting that during the training process, only system message responses denoted as $\boldsymbol{y}_i$, require loss computation. Note that the loss is only computed from the part of system responses during training.

while training MineLLM, trainable parameters(*i.e.*, θ_a from the Alignment Net and θ_l from LoRA [13]) are optimized by Adam optimizer with a learning rate initialized to be $5e-4$, and scheduled using a linear decay scheduler. The rank of LoRA [13] modules is set to 32. We train all parameters in a one-stage end-to-end fashion with 8 A100 GPUs. Each GPU process 2 samples every iteration and the effective batch size is set to 128 by gradient accumulation. Input images are resized to be 224×224 and we use MineCLIP [9] pre-trained ViT-B/16 [8] as visual encoder, the number of vision tokens are 196 and length of text tokens after vision tokens are limited to 400 in training.

A.2. Memory

Inspired by the Skill library of Voyager [33], memory is utilized in two parts of *MP5* to perform Retrieval-Augmented Generation (RAG [16]). The Parser employs Knowledge Memory to decompose tasks into sub-objectives, while the Planner, when planning an action sequence for a specific sub-objective, may refer to similar action sequences provided by Performer Memory. The implementation details are similar to those of Voyager [33].

A.2.1 Knowledge Memory

For Knowledge Memory, we actually adopt a vector database method (*e.g.*, Chroma, FAISS, *etc.*) to store frequently used knowledge. This knowledge mainly comes from three sources: part of it is from the online wiki, another part is from the crafting recipes of items in MineDojo [9], and some are user tips from Reddit. Specifically, we convert commonly used knowledge into corresponding text embeddings using OpenAI's text-embedding-ada-002 [23] and store them in a vector database. When decomposing sub-objectives requires the retrieval of relevant knowledge, we also convert the corresponding descriptions of these sub-objectives into corresponding text embeddings. We then perform a search match in the database and select the most similar piece of knowledge. If the similarity score at this time is below 0.05 (the lower the score, the more similar), it is directly taken as the result of the RAG [16]. Of course, there will also be cases where the similarity scores are all above 0.05. This indicates that there is currently no such type of knowledge in the database. In this case, we manually supplement this type of knowledge and add it to the database as the result of the RAG [16].

A.2.2 Performer Memory

For Performer Memory, we record the task description of each successful sub-objective and its corresponding successful action sequence. Specifically, Performer Memory consists of two parts. One part is a vector database used to store the sub-objective task descriptions and their corresponding positions in the sub-objective sequence. The other part is a JSON file where the key is the position of the sub-objective in the sub-objective sequence, and the value corresponds to the sub-objective task description and its successful action sequence. When we need to find similar action sequences, similar to Knowledge Memory, we convert the current sub-objective's task description into corresponding text embeddings and retrieve the 2 closest matches from the vector library. We then extract the corresponding successful

Table 9. The Definition of the Action Space we use in MineDojo [9] Simulator

Name	Arguments	Description	Corresponding MineDojo [9] Actions	Action Conditions Based on Environmental Information
Find	object	Travel across the present terrain in search of an object	forward, jump move left and right	Halt only when the object is in Ego-View RGB image
Move	object	Move to the target object until it is within striking distance	forward, jump move left and right	Halt only when the object is in the surrounding $3 \times 3 \times 3$ Voxels
Craft	object materials platform	Craft a certain number of objects with materials in the inventory using the platform	craft, attack use, place	Begins only once the environmental conditions required are met
Mine	object tool	Harvest a single block using tool from surroundings	attack	Begins only once the environmental conditions required are met
Equip	object	Equip a given object from the current inventory.	equip	Begins only once the environmental conditions required are met
Fight	object tool	Attack a nearby entity using the specified tool	attack	Begins only once the environmental conditions required are met
Dig-Up	tool	Ascend directly by jumping and placing blocks	jump, place	Halt only when the agent can see the sky
Dig-Down	y-level tool	Descend using the specified tool to dig your way through if necessary	attack	Halt only when the agent reach the specified y-level
Use	object	Use the item held in the main hand	use	∅
Place	object	Place an inventory item on the ground.	place	Begins only once the environmental conditions required are met

objective sequences from the JSON file using their positions in the sub-objective sequence.

A.3. Observation Space

In order to allow the system to more closely resemble an embodied agent rather than emulating a game player unlocking the tech tree, we significantly limited environmental information, endeavoring to enable the agent to perceive through Ego-View RGB images as much as possible.

Our Observation Space primarily consists of two components: one is the Perceptual Observation, and the other is the Status Observation. The Perceptual Observation includes Ego-View Minecraft-style RGB images and $3 \times 3 \times 3$ Voxels that the agent encounters. The Status Observation includes some associated auxiliary textual information(*e.g.*, the current agent's life statistics, GPS location, inventory, and equipment information). Notably, to make the system more resemble an embodied agent, we have obscured a large amount of environmental information (*e.g.*, the current biome, weather, and whether the sky is visible that human players can learn by pressing F3). This encourages the agent to perceive through the current RGB image rather than directly knowing a lot of the current environmental information.

To more clearly demonstrate our Observation Space, we list the differing Observation Spaces of related works in the table below, as shown in Table 8.

A.4. Action Space

The Performer module executes action sequences, which consist of actions falling within the action space outlined in Tab 9. These actions are brief combinations formed by the MineDojo [9] API, with frequent interactions with the environment occurring within each action.

For example, the action of "Find" can be described as a directionless forward motion, initiating a jump when encountering obstacles. If the obstacle proves insurmountable, the action adapts by implementing a left or right turn, followed by the continuation of forward motion. This process involves minimal human intervention or design. During the execution of the "Find" action, there is a fixed frequency at which the current Ego-View RGB images are analyzed to ascertain whether the required object (*e.g.*, a block, a type of mob, *etc.*) has been in sight.

B. Environment Setting

Our Minecraft experimental environment is based on the MineDojo [9] simulation platform, which provides a unified observation and action space to foster the development of intelligent agents capable of multitasking and continuous learning to adapt to new tasks and scenarios.

In our experiments, the *position* at which the agent begins its game, as well as the *seed* used to generate the environment, are both randomized. This introduces an element of unpredictability and variety into the experimental setup, ensuring that the agent will encounter a wide range of scenarios

Table 10. Full *Context-Dependent Tasks*. 16 tasks are defined and divided into 4 difficulty levels based on the minimum number of information types needed. Underlines label the environmental information, reflecting the complexity varies at each level.

Task Level	Task id	Task description
Easy	1-1	Find a tree
	1-2	Find a grass
	1-3	Find a cow
	1-4	Find a pig
Mid	2-1	Find a tree in the forest
	2-2	Find a grass near a pig
	2-3	Find a cow in the desert
	2-4	Find a pig during the nighttime
Hard	3-1	Find a tree in the forest during the nighttime
	3-2	Find a grass near a pig in the plains
	3-3	Find a cow in the desert during the daytime
	3-4	Find a pig during the nighttime in a rainy day
Complex	4-1	Find a tree in the forest during the nighttime in a sunny day
	4-2	Find a pig near a grass in the forest during the daytime
	4-3	Find a cow near the water in the desert during the daytime in sunny day
	4-4	Find a pig during the daytime on the plains with a grass next to it, the weather is sunny day and the brightness is sufficient

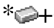
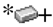
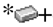
Table 11. Details of *Context-Dependent Tasks* Environment Information content.

Task Level	Task id	Num of Info.	Object	Creature	Ecology	Time	Weather	Brightness
Easy	1-1	1	✓					
	1-2	1	✓					
	1-3	1		✓				
	1-4	1		✓				
Medium	2-1	2	✓		✓			
	2-2	2	✓	✓				
	2-3	2		✓	✓			
	2-4	2		✓		✓		
Hard	3-1	3	✓		✓	✓		
	3-2	3	✓	✓	✓			
	3-3	3		✓	✓	✓		
	3-4	3		✓		✓	✓	
Very Hard	4-1	4	✓		✓	✓	✓	
	4-2	4	✓	✓	✓	✓		
	4-3	5	✓	✓	✓	✓	✓	
	4-4	6	✓	✓	✓	✓	✓	✓

and challenges. The agent is set to start in *survival mode*, the most challenging and interactive mode available. Unlike creative or adventure modes, survival mode represents a test of the agent's ability to strategize, and make quick decisions. The agent is also confronted with the complication of *hostile mob* generation. The agent begins its game with an *empty inventory*, meaning it must actively mine and craft the objects. To simulate a real Embodied Agent, environmental factors(*e.g.*, time, weather, *etc.*) change over time. At night, the agent does not have night vision, and the items in the inventory will be cleared upon death.

To better evaluate *Context-Dependent Tasks* and *Process-Dependent Tasks*, which are defined in detail in Appendix C.1, we select different environment settings in MineDojo [9]. For *Context-Dependent Tasks*, we uniformly adopt the environment in MineDojo [9] with the creative "task_id" of "0". For *Process-Dependent Tasks*, we uniformly adopt the environment with the "task_id" of "harvest", "tar-

Table 12. Detailed Definition of *Process-Dependent Tasks*. 25 tasks are defined and divided into 5 difficulty levels based on incrementally increasing reasoning steps. A higher difficulty level implies that the agent needs to engage in longer reasoning and planning with the environment.

Task Level	Task	reasoning step	Object	Final recipe	Tools/Platforms
Basic level	mine log	1		-	-
	mine sand	1		-	-
	craft planks	2		1*	-
	craft stick	3		2*	-
	craft crafting table	3		4*	-
Wooden level	craft bowl	4		3*	
	craft boat	4		5*	
	craft chest	4		8*	
	craft wooden sword	5		2* +1*	
	craft wooden pickaxe	5		3* +2*	
Stone level	mine cobblestone	6		-	
	craft furnace	7		8*	
	craft stone pickaxe	7		3* +2*	
	mine iron ore	8		-	
	smelt glass	9		1*	
Iron level	smelt iron ingot	10		1*	
	craft shield	11		1*+6*	
	craft bucket	11		3*	
	craft iron pickaxe	11		3*+2*	
	craft iron door	11		6*	
Diamond level	obtain diamond	12		-	
	mind redstone	12		-	
	craft compass	13		1* +4*	
	craft diamond pickaxe	13		3* +2*	
	craft piston	13		1* +1*+4* +3*	

get_names" as "diamond", and "spawn_rate" as "1". This is why obtaining redstone is more difficult than obtaining diamond, as described in Appendix C.2.2.

C. Task Details and Experiment Results

C.1. Task Details

C.1.1 Context-Dependent Tasks

Context-Dependent Tasks primarily study how Active Perception enables the agent to better perceive low-level context information in the environment. We first establish 6 aspects of environmental information derived from the Minecraft game environment: *[Object, Mob, Ecology, Time, Weather, Brightness]*. Each aspect has multiple options. For example, pigs , cows , and sheep are all elements belonging to Mob. Based on this, we define 16 tasks and organize their difficulty into 4 levels by taking into account the number of information elements that require perception, as is shown in Tab. 10. Easy tasks necessitate the perception of only one element, Mid tasks include 2 perception elements, Hard tasks contain 3 elements, whereas Complex tasks involve the perception of 4 to 6 elements. Each task at the same level has different environment information content, the amount of environment information contained in each task, and the corresponding specific environment information is shown in Tab. 11. Finally, we rigorously assess *MP5*'s proficiency in environmental context perception across these 16 tasks.

As the main paper states, our initial environmental details are predetermined (*e.g.*, biomes) in order to reduce the agent's exploration time, otherwise, the agent may fail to find the corresponding scenario within the time limit. We defined ten initial biome, each of which used random seeds to generate five different environments to test each task, so each task was tested in 50 different scenarios and the success rate was calculated to verify *MP5*'s generalization ability. In order to align as much as possible with the experimental Settings of other methods, we did not modify the terrain to simplify the task.

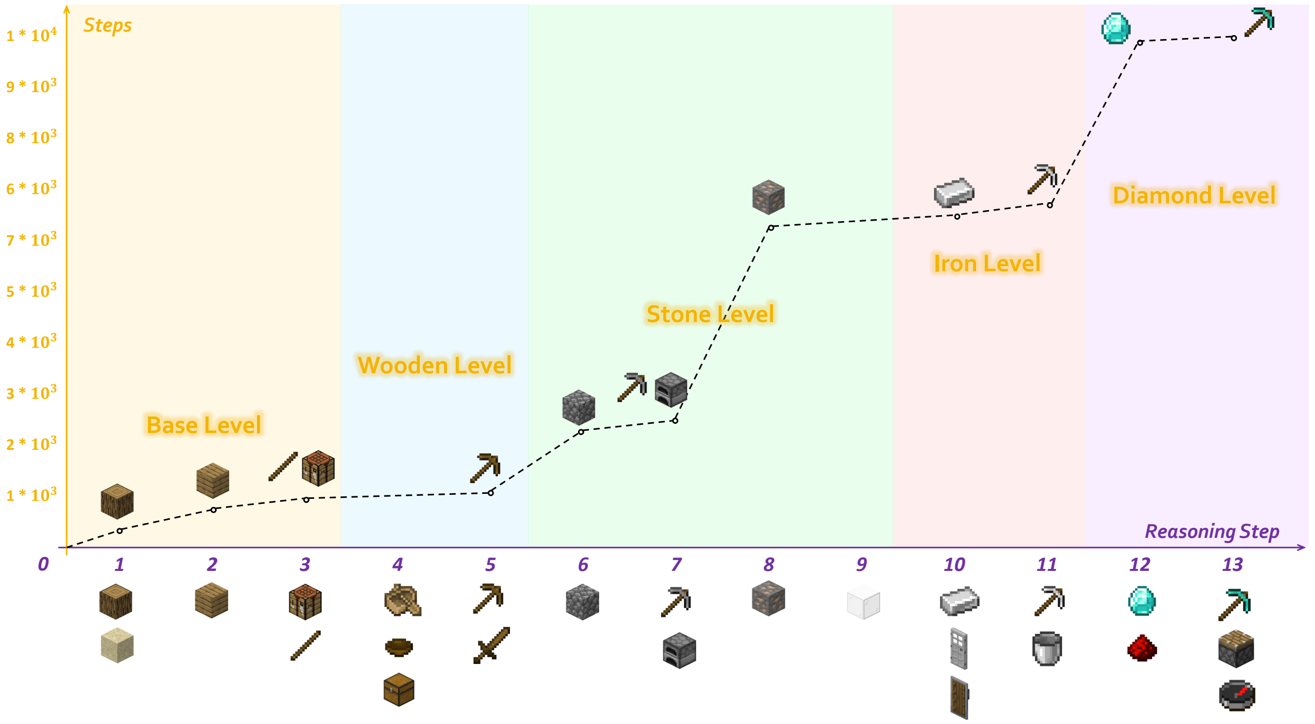


Figure 6. The game-playing steps corresponding to the acquisition of different milestone objects by the agent during the completion of the *craft diamond pickaxe* challenge. The varying background colors denote the level of the *Process-Dependent Tasks* in which the milestone objects are located.

C.1.2 Process-Dependent Tasks

Process-Dependent Tasks primarily investigate situation-aware planning and embodied action execution, incorporating contributions from Active Perception and other modules that continuously perceive the environment and dynamically adjust their actions to accomplish long-horizon tasks. In Table 12, we list the names of all tasks in *Process-Dependent Tasks*, their reasoning steps, object icons, the final recipe, and the required tools/platforms. The reasoning step refers to the number of sub-objectives that need to be completed in order to finish the entire task. Given that the agent's environment information(*e.g.*, biome, weather, *etc.*) is randomly initialized, there may be execution errors requiring replanning, thus potentially necessitating the completion of additional sub-objectives, which means more reasoning steps may be required. We consider only the most basic scenarios and select 25 tasks based on the required reasoning steps in increasing order. These tasks are then divided into 5 difficulty levels.

For evaluation, we consider an Agent's accidental death in the game (*e.g.*, being burned by lava, killed by a hostile mob, *etc.*) as a failure, as well as not achieving the objective within the time limit (*e.g.*, exceeding the 10 minute game limit, or API request timeout, *etc.*). We conduct 30 games of *Process-Dependent Tasks* and took the average success rate as the final reported performance.

C.2. Success Rates of All Tasks

C.2.1 Context-Dependent Tasks

We report the success rates of different methods and perception strategies for all tasks comprehensively and in detail in Table 13, including ours, GPT-4V [24], and LLaVA-1.5 [19], using both Active Perception strategy and Fine-Grained Global Perception strategy. This table also presents the detailed results of the "Main Results" section under "*Context-Dependent Tasks*" in the main text.

C.2.2 Process-Dependent Tasks

We report the success rates of different methods for all tasks comprehensively and in detail in Table 15, including ours, non-situation-aware planning, and non-embodied action execution. This table also presents the detailed results of the "Main Results" section under "*Process-Dependent Tasks*" in the main text. The parts with a gray background in the table represent the average success rate for the current level.

To better demonstrate the practical performance of *MP5* in *Process-Dependent Tasks*, we select *craft diamond pickaxe* with a reasoning step of 13 as the challenge. Figure 6 depicts the game-playing steps corresponding to each mile-

Table 13. Detailed Performance on *Context-Dependent Tasks*. Method$_A$ means the method uses the Active Perception strategy, and Method$_G$ means the method uses the Fine-Grained Global Perception strategy. The parts with a gray background in the table represent the average success rate for the current level.

Task Level	Task id	Success rate(%)					
		MP5$_A$	*MP5*$_G$	GPT-4V$_A$ [24]	GPT-4V$_G$ [24]	LLaVA-1.5$_A$ [19]	LLaVA-1.5$_G$ [19]
Easy	1-1	98.0	94.0	100.0	100.0	88.0	56.0
	1-2	100.0	92.0	100.0	100.0	68.0	44.0
	1-3	98.0	88.0	100.0	96.0	76.0	42.0
	1-4	98.0	86.0	100.0	94.0	58.0	48.0
	Average	98.5	90.0	100.0	97.5	72.5	47.5
Mid	2-1	98.0	90.0	98.0	82.0	56.0	28.0
	2-2	96.0	82.0	90.0	86.0	52.0	14.0
	2-3	92.0	88.0	94.0	88.0	44.0	22.0
	2-4	92.0	84.0	96.0	84.0	48.0	26.0
	Average	94.5	86.0	94.5	85.0	50.0	22.5
Hard	3-1	94.0	80.0	96.0	80.0	12.0	8.0
	3-2	98.0	78.0	92.0	74.0	8.0	0.0
	3-3	90.0	76.0	90.0	74.0	10.0	6.0
	3-4	90.0	76.0	92.0	72.0	14.0	6.0
	Average	93.0	77.5	92.5	75.0	11.0	5.0
Complex	4-1	92.0	74.0	90.0	64.0	0.0	0.0
	4-2	92.0	70.0	88.0	60.0	0.0	0.0
	4-3	86.0	64.0	84.0	58.0	0.0	0.0
	4-4	94.0	62.0	88.0	58.0	0.0	0.0
	Average	91.0	67.5	87.5	60.0	0.0	0.0

stone object (*e.g.*, log, plank, stick, *etc.*) obtained by the agent.

C.3. Ablation Study

C.3.1 Context-Dependent Tasks

We conduct ablation studies on the multi-modal large language model (MLLM) part within *Context-Dependent Tasks* in 14, comparing the performance outcomes of different MLLMs and different pre-trained visual encoders in the percipient.

C.3.2 Process-Dependent Tasks

In this section, we present detailed results from our ablation experiments. Table 17 shows the performance of the agent in *MP5* after the removal of various modules. Table 18 demonstrates the impact on the results when the Planner is replaced by large language models with inconsistent reasoning capabilities, including open-source models like LLaMA2-70B-Chat [32] and Vicuna-13B-v1.5-16k [5]. Table 19 further explores the contribution of the Memory components to the agent's performance, including Knowledge Memory and Performer Memory. Table 20 investigates the robustness gain brought by the check part of the Patroller under *"Random Drop"* conditions.

As seen from the results in Table 17, the agent's success rate in completing *Process-Dependent Tasks* significantly decreases after the removal of any modules, with the success rate at the Diamond level falling to 0.00% for all except when the Patroller is removed. The Percipient mainly provides the agent with visual input, the Memory primarily provides the agent with relevant knowledge, the Parser simplifies the difficulty of online task decomposition for the agent, and the Patroller ensures that each action is sufficiently checked for successful execution.

Table 18 presents detailed results from the Planner ablation experiments in the "Ablation Study" section of the main text. From this, we can discern that LLMs with stronger reasoning capabilities demonstrate better understanding when faced with a wide variety of text information inputs, thereby facilitating more effective planning. The poor performance of open-source large models like LLaMA2-70B-Chat [32] Vicuna-13B-v1.5-16k [5] is due to their inadequate ability to process long and diverse types of text information. This inadequacy is evident at the Wooden level, where the success rate has already plummeted to 0.00%.

As can be seen from the results in Table 19, both types of Memory can enhance the agent's actions, particularly the Knowledge Memory. Without the Knowledge Memory, the agent fails to mine iron due to its inability to recognize

Table 14. Detailed Ablation on *Context-Dependent Tasks*. The parts with a gray background in the table represent the average success rate for the current level.

Task Level	Task id	Success rate(%)		
		MineLLM+MineCLIP [9]	MineLLM+CLIP [29]	LLaVA-1.5 [19]+CLIP [29]
Easy	1-1	98.0	98.0	88.0
	1-2	100.0	94.0	68.0
	1-3	98.0	96.0	76.0
	1-4	98.0	92.0	58.0
	Average	98.5	95.0	72.5
Mid	2-1	98.0	94.0	56.0
	2-2	96.0	88.0	52.0
	2-3	92.0	88.0	44.0
	2-4	92.0	90.0	48.0
	Average	94.5	90.0	50.0
Hard	3-1	94.0	90.0	12.0
	3-2	98.0	90.0	8.0
	3-3	90.0	84.0	10.0
	3-4	90.0	84.0	14.0
	Average	93.0	87.0	11.0
Complex	4-1	92.0	82.0	0.0
	4-2	92.0	84.0	0.0
	4-3	86.0	78.0	0.0
	4-4	90.0	76.0	0.0
	Average	91.0	80.0	0.0

where iron ore is more likely to be located. Consequently, the success rates for both Iron and Diamond levels are 0.00%. The Knowledge Memory can help the agent more easily understand the acquisition methods of some items, while the Performer Memory can provide similar scenarios for the agent to reference, thereby easing the pressure in the planning process.

Table 20 primarily studies the robustness brought about by the check part of the Patroller. *"Random Drop"* is a specific setting that forces the Agent into execution errors. More specifically, when the agent successfully completes tasks with the reasoning step greater than 4, it will randomly discard one item from either log, planks, or stick present in its inventory. This situation can lead the agent to commit execution errors due to insufficient material, specifically when it is completing sub-objectives of higher reasoning steps that require logs, planks, or sticks as materials. The check part of the Patroller can detect the cause of these errors during execution and use it as feedback for re-planning. With the *"Random Drop"* enabled and the check part of the Patroller disabled, the agent even struggles to complete tasks at the stone level are 0.00% effectively.

D. Different Strategy of Active Perception

In order to improve the quality of the Active Perception Query generated by Patroller, we use Chain-of-Thought(COT)[36] to design a process of multiple rounds of query generation, Patroller can generate the next most important problem based on the current problem and task description, until the agent judges that all problems have been produced. We conduct experiences to compare Single-round Generation and Multi-round Generation in Tab. 16, We can observe that Multi-round Generation using COT[36] generates better corresponding environment information query and thus have a higher success rate on the *Context-Dependent Tasks*.

E. Applications

E.1. Obtain Diamond Pickaxe

We demonstrate a case of the popular *Process-Dependent Tasks* "*craft diamond pickaxe*" challenge in Video 1.

E.2. Discovery

We demonstrate a complex level *Context-Dependent Tasks* "*Find a pig on the plains with grass and water next to it during a sunny day with sufficient brightness*" in Video 2.

E.3. Open-Ended Tasks

We demonstrate a *Open-Ended Tasks* "*Dig a block of sand under the water with a wooden shovel during the day-*

Table 15. Detailed Performance on *Process-Dependent Tasks*. We compare the success rate when interacting or not interacting with the environment during the planning or execution. The parts with a gray background in the table represent the average success rate for the current level.

Task Level	Object	Success rate(%)		
		MP5(Ours)	non-situation-aware planning	non-embodied action execution
Basic level	log	96.67	93.33	0.00
	sand	96.67	93.33	0.00
	planks	96.67	93.33	0.00
	stick	96.67	90.00	0.00
	crafting table	93.33	90.00	0.00
	Average	96.00	92.00	0.00
Wooden level	bowl	93.33	90.00	0.00
	boat	93.33	90.00	0.00
	chest	90.00	90.00	0.00
	wooden sword	86.67	80.00	0.00
	wooden pickaxe	80.00	80.00	0.00
	Average	88.67	86.00	0.00
Stone level	cobblestone	80.00	73.33	0.00
	furnace	80.00	73.33	0.00
	stone pickaxe	80.00	70.00	0.00
	iron ore	60.00	50.00	0.00
	glass	80.00	76.67	0.00
	Average	76.00	68.67	0.00
Iron level	iron ingot	56.67	50.00	0.00
	shield	56.67	50.00	0.00
	bucket	53.33	43.33	0.00
	iron pickaxe	50.00	40.00	0.00
	iron door	43.33	43.33	0.00
	Average	52.00	45.33	0.00
Diamond level	diamond ore	30.00	20.00	0.00
	mind redstone	20.00	16.67	0.00
	compass	16.67	10.00	0.00
	diamond pickaxe	23.33	10.00	0.00
	piston	20.00	13.33	0.00
	Average	22.00	14.00	0.00

Table 16. Performance on Active Perception Query Generation with different Round Strategy. S means Single-round Generation and M means Multi-round Generation.

planner	Strategy	Average Generation Rate(%)			
		Easy	Mid	Hard	Complex
Vicuna-13B-v1.5 [5]	S	100	95	75	45
	M	100	100	95	80
GPT-3.5-turbo [22]	S	100	100	85	70
	M	100	100	100	100

time *on a sunny day*" in Video 3.

F. Interactions in *MP5*

Here we illustrate the interactions between the internal modules of *MP5* during Active Perception and Re-planning, presented in the form of dialogue text.

F.1. Active Perception

In this part, we demonstrate the communication process among situation-aware planning, embodied action execution, and active perception scheme when facing the task of "*Find 1 sheep* 🐑 on the plains 🏞", as shown in Figure 7. The corresponding screenshots are illustrated in Figure 9.

F.2. Re-planning

In this part, we depict the situation when facing the task of "*craft wooden pickaxe*⛏" with a shortfall of 1 plank 🟫. In this case, the Patroller identifies the cause of the execution error and instructs the Planner to re-plan, as shown in Figure 8. The corresponding screenshots are illustrated in Figure 10.

Table 17. Success rates on different modules within *Process-Dependent Task*. The parts with a gray background in the table represent the average success rate for the current level.

Task Level	Object	Success rate(%)				
		MP5(Ours)	w/o Percipient	w/o Memory	w/o Parser	w/o Patroller
Basic level	log	96.67	0.00	90.00	96.67	86.67
	sand	96.67	0.00	90.00	96.67	73.33
	planks	96.67	0.00	80.00	96.67	83.33
	stick	96.67	0.00	76.67	96.67	73.33
	crafting table	93.33	0.00	76.67	90.00	73.33
	Average	96.00	0.00	82.67	95.33	78.00
Wooden level	bowl	93.33	0.00	66.67	80.00	66.67
	boat	93.33	0.00	66.67	70.00	66.67
	chest	90.00	0.00	66.67	70.00	63.33
	wooden sword	86.67	0.00	40.00	63.33	60.00
	wooden pickaxe	80.00	0.00	40.00	60.00	60.00
	Average	88.67	0.00	56.00	68.67	63.33
Stone level	cobblestone	80.00	0.00	10.00	50.00	60.00
	furnace	80.00	0.00	3.33	0.00	60.00
	stone pickaxe	80.00	0.00	0.00	0.00	56.67
	iron ore	60.00	0.00	0.00	0.00	40.00
	glass	80.00	0.00	0.00	0.00	43.33
	Average	76.00	0.00	2.67	10.00	52.00
Iron level	iron ingot	56.67	0.00	0.00	0.00	36.67
	shield	56.67	0.00	0.00	0.00	36.67
	bucket	53.33	0.00	0.00	0.00	30.00
	iron pickaxe	50.00	0.00	0.00	0.00	26.67
	iron door	43.33	0.00	0.00	0.00	20.00
	Average	52.00	0.00	0.00	0.00	30.00
Diamond level	diamond ore	30.00	0.00	0.00	0.00	10.00
	mind redstone	20.00	0.00	0.00	0.00	3.33
	compass	16.67	0.00	0.00	0.00	0.00
	diamond pickaxe	23.33	0.00	0.00	0.00	3.33
	piston	20.00	0.00	0.00	0.00	3.33
	Average	22.00	0.00	0.00	0.00	4.00

Table 18. More detailed success rates for different LLMs as zero-shot Planners on *Process-Dependent Tasks*. The parts with a gray background in the table represent the average success rate for the current level.

Task Level	Object	Success rate(%)			
		GPT-4(Ours)	GPT-3.5-Turbo [22]	LLaMA2-70B-Chat [32]	Vicuna-13B-v1.5-16k [5]
Basic level	log	96.67	96.67	6.67	3.33
	sand	96.67	96.67	3.33	3.33
	planks	96.67	96.67	0.00	0.00
	stick	96.67	96.67	0.00	0.00
	crafting table	93.33	90.00	0.00	0.00
	Average	96.00	95.33	2.00	1.33
Wooden level	bowl	93.33	90.00	0.00	0.00
	boat	93.33	90.00	0.00	0.00
	chest	90.00	90.00	0.00	0.00
	wooden sword	86.67	83.33	0.00	0.00
	wooden pickaxe	80.00	80.00	0.00	0.00
	Average	88.67	86.67	0.00	0.00
Stone level	cobblestone	80.00	66.67	0.00	0.00
	furnace	80.00	50.00	0.00	0.00
	stone pickaxe	80.00	50.00	0.00	0.00
	iron ore	60.00	10.00	0.00	0.00
	glass	80.00	33.33	0.00	0.00
	Average	76.00	42.00	0.00	0.00
Iron level	iron ingot	56.67	6.67	0.00	0.00
	shield	56.67	3.33	0.00	0.00
	bucket	53.33	0.00	0.00	0.00
	iron pickaxe	50.00	3.33	0.00	0.00
	iron door	43.33	0.00	0.00	0.00
	Average	52.00	2.67	0.00	0.00
Diamond level	diamond ore	30.00	0.00	0.00	0.00
	mind redstone	20.00	0.00	0.00	0.00
	compass	16.67	0.00	0.00	0.00
	diamond pickaxe	23.33	0.00	0.00	0.00
	piston	20.00	0.00	0.00	0.00
	Average	22.00	0.00	0.00	0.00

Table 19. Success rates for different parts of Memory on *Process-Dependent Tasks*. The parts with a gray background in the table represent the average success rate for the current level.

Task Level	Object	Success rate(%)			
		All Memory(Ours)	w/o Performer Memory	w/o Knowledge Memory	w/o All Memory
Basic level	log	96.67	96.67	90.00	90.00
	sand	96.67	96.67	90.00	90.00
	planks	96.67	96.67	83.33	80.00
	stick	96.67	96.67	76.67	76.67
	crafting table	93.33	93.33	80.00	76.67
	Average	96.00	96.00	84.00	82.67
Wooden level	bowl	93.33	93.33	70.00	66.67
	boat	93.33	90.00	66.67	66.67
	chest	90.00	90.00	70.00	66.67
	wooden sword	86.67	83.33	43.33	40.00
	wooden pickaxe	80.00	80.00	40.00	40.00
	Average	88.67	87.33	58.00	56.00
Stone level	cobblestone	80.00	73.33	16.67	10.00
	furnace	80.00	73.33	6.67	3.33
	stone pickaxe	80.00	70.00	3.33	0.00
	iron ore	60.00	50.00	0.00	0.00
	glass	80.00	70.00	3.33	0.00
	Average	76.00	67.33	6.00	2.67
Iron level	iron ingot	56.67	53.33	0.00	0.00
	shield	56.67	53.33	0.00	0.00
	bucket	53.33	46.67	0.00	0.00
	iron pickaxe	50.00	43.33	0.00	0.00
	iron door	43.33	40.00	0.00	0.00
	Average	52.00	47.33	0.00	0.00
Diamond level	diamond ore	30.00	23.33	0.00	0.00
	mind redstone	20.00	26.67	0.00	0.00
	compass	16.67	10.00	0.00	0.00
	diamond pickaxe	23.33	20.00	0.00	0.00
	piston	20.00	13.33	0.00	0.00
	Average	22.00	16.67	0.00	0.00

Table 20. Success rates with and without the check part of the Patroller in the presence of *"Random Drop"* Setting on *Process-Dependent Tasks*. The parts with a gray background in the table represent the average success rate for the current level.

Component		Method			
the check part of Patroller		✓	✗	✓	✗
"Random Drop"		✗	✗	✓	✓
Task Level	Object	Success rate(%)			
Basic level	log	96.67	86.67	90.00	90.00
	sand	96.67	73.33	90.00	90.00
	planks	96.67	83.33	86.67	70.00
	stick	96.67	73.33	86.67	50.00
	crafting table	93.33	73.33	83.33	50.00
	Average	96.00	78.00	78.00	70.00
Wooden level	bowl	93.33	66.67	80.00	10.00
	boat	93.33	66.67	83.33	10.00
	chest	90.00	63.33	80.00	10.00
	wooden sword	86.67	60.00	70.00	3.33
	wooden pickaxe	80.00	60.00	70.00	3.33
	Average	88.67	63.33	78.00	7.33
Stone level	cobblestone	80.00	60.00	53.33	3.33
	furnace	80.00	60.00	53.33	0.00
	stone pickaxe	80.00	56.67	50.00	0.00
	iron ore	60.00	40.00	30.00	0.00
	glass	80.00	43.33	40.00	0.00
	Average	76.00	52.00	45.33	0.00
Iron level	iron ingot	56.67	36.67	26.67	0.00
	shield	56.67	36.67	26.67	0.00
	bucket	53.33	30.00	16.67	0.00
	iron pickaxe	50.00	26.67	13.33	0.00
	iron door	43.33	20.00	10.00	0.00
	Average	52.00	30.00	18.67	0.00
Diamond level	diamond ore	30.00	10.00	3.33	0.00
	mind redstone	20.00	3.33	3.33	0.00
	compass	16.67	0.00	0.00	0.00
	diamond pickaxe	23.33	3.33	0.00	0.00
	piston	20.00	3.33	0.00	0.00
	Average	22.00	4.00	1.33	0.00

```
Agent:
     My information is as follows:
     task information:
     - description: Find 1 sheep on the plains.

     current environment information:
     - position: ground

     inventory: {}

Patroller: What Minecraft creatures is this image displaying?
Percipient: There appear to be no creature in the given Minecraft image.
Patroller:
     feedback: There appear to be no creature, so you need to find the specific sheep.
     Success: False
     Suggestion: Start planning the actions.

Planner:
     explanation: None
     thoughts: Since the task is to find a sheep and we are already on the ground, we just need to find the sheep.
     actions: [
             {"times": "1", "actions": [ {"name": "Find", "args": {"obj": "sheep"} }}
     ]

Performer: continue doing "Find" action.
...

Performer: Having completed a move in “Find” action, based on my current view, tell me if I should continue this action or if the next action
is ready to execute.
Patroller: What Minecraft entities does this image represent?
Percipient: This image represents sheep.
Patroller: Could you identify the biome in this Minecraft image?
Percipient: The image appears to be from the forest biome in Minecraft.
Patroller:
     feedback: Because the current biome is not forest biome, you should continue finding the sheep in the specific biome.
     Success: False
     Suggestion: Continue doing the current action.

Performer: Continue doing "Find" action.
...

Performer: Having completed a move in “Find” action, based on my current view, tell me if I should continue this action or if the next action
is ready to execute.
Patroller: What types of Minecraft mobs is this picture showing?
Percipient: This picture is showing sheep, types of Minecraft mobs.
Patroller: What kind of Minecraft biome can you observe in this image?
Percipient: The image presents a clear view of a plains biome in Minecraft.
Patroller:
     feedback: Because there are sheep on the plains, which means you have successfully found the specific sheep.
     Success: True
     Suggestion: Start to planning the next sub-objective actions. If there is no next sub-objective, the task is successfully done.

Performer: Finish doing "Find" action.

Agent: I have found a sheep on the plains.
```

Figure 7. Dialogue of task “*Find 1 sheep* *on the plains*”

```
System:
     You are a helpful planner in Minecraft, capable of planning actions to complete sub-objectives. I will give you environmental
     information and a sub-objective, for which you need to conceive a plan, and then create a list composed of a sequence of various
     actions to complete it.

=========
…
Patroller:
     feedback: Because there are 2 logs in your inventory, this means you have successfully mined 2 logs.
     Success: True
     Suggestion: Start to planning the next sub-objective actions.

…
Patroller:
     feedback: Because there are already 8 planks in your inventory, this means you have successfully crafted 8 planks using 2 logs.
     Success: True
     Suggestion: Start to planning the next sub-objective actions.

…
Patroller:
     feedback: Because there are already 4 sticks in your inventory, this means you have successfully crafted 4 sticks using 2 planks.
     Success: True
     Suggestion: Start to planning the next sub-objective actions.

…
Patroller:
     feedback: Because there is already 1 crafting table in your inventory, this means you have successfully crafted a crafting table.
     Success: True
     Suggestion: Start to planning the next sub-objective actions.

…
Patroller:
     feedback: You do not have enough planks to complete the 'craft' action. You need 3 planks but you only have 2 planks in your
     inventory.
     Success: False
     Suggestion: Mine or Craft enough planks first.

=========
Agent:
     My information is as follows:
     task information:
     - task: wooden pickaxe
     - quantity: 1
     - material: {"planks": 3, "stick": 2}
     - tool: None
     - platform: crafting table
     - tips: 1 wooden pickaxe can be crafted with 3 planks and 2 stick as the material and crafting table as the platform.
     - description: Craft 1 wooden pickaxe using 3 planks and 2 stick on a crafting table as the platform.

     current environment information:
      -      position: ground

     inventory: {"stick": 2, "crafting_table": 1, "log": 1, "planks": 2, "dirt": 6}

     Here are some plans to achieve similar goal for reference:
     - description: Craft 1 crafting table using 4 planks without any tool or platform.
     - actions: [
          {"times": "1", "actions": [ {"name": "Craft", "args": {"obj": {"crafting table": 1}, "materials": {"planks": 4}, "platform":
           null}} ] }
     ]
     - description: Craft 8 planks using 2 logs without needing any tool or platform.
     - actions: [
           {"times": “2", "actions": [ {"name": "Craft", "args": {"obj": {"planks": 4}, "materials": {"log": 1}, "platform":
           null} } ] }
     ]


Planner:
     explanation: You do not have enough planks to complete the "craft" action. You need 3 planks but you only have 2 planks in your
     inventory.
     thoughts: First, I need to craft more planks using the logs in my inventory. Then, I can craft a wooden pickaxe using the planks
     and sticks.
     actions: [
           {"times": "1", "actions": [ {"name": "Craft", "args": {"obj": {"planks": 4}, "materials": {"log": 1}, "platform":
     null} } ] },
          {"times": "1", "actions": [ {"name": "Craft", "args": {"obj": {"wooden pickaxe": 1}, "materials": {"planks": 3, "stick": 2},
     "platform": "crafting table"} } ] }
     ]
```

Figure 8. Dialogue of task “*craft wooden pickaxe*” while re-planning

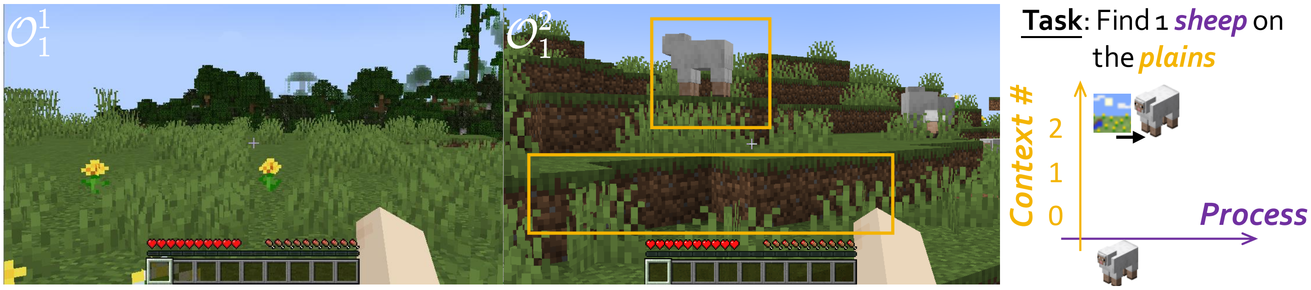


Figure 9. The corresponding screenshots for the dialogue of task "*Find 1 sheep on the plains*"

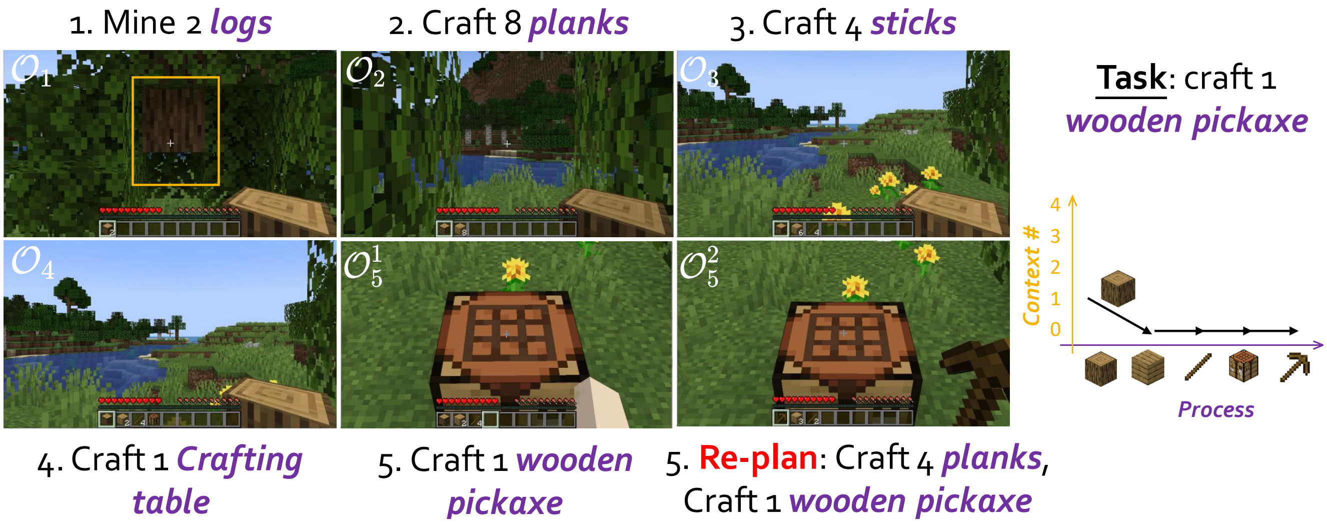


Figure 10. The corresponding screenshots for the dialogue of task "*Find 1 sheep on the plains*" while re-planning