MP5: An Open-ended Embodied System in Minecraft via Perception and Planning Collaboration

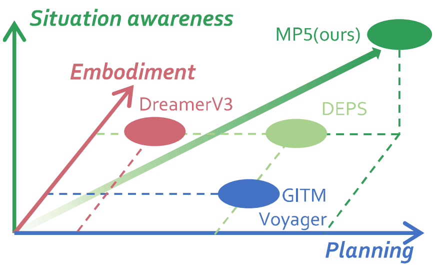

Figure 1. Paradigm innovation.

A. Discussion of *MP5*

As shown in Fig. 1, existing methods usually follow the paradigm that an agent should be with the ability of planning (*e.g.*, GITM & Voyager), embodiment (*e.g.*, DreamerV3), or both of them (*e.g.*, DEPS). To be different, *MP5* introduces a new paradigm that these components (*i.e.*, embodiment, planning, and execution as well), should be enhanced with the awareness of the situation (*i.e.*, rich contextual and procedural information w.r.t. the task), which is more human-like and has the potential to solve more difficult open-end context- and process-dependent tasks (as evaluated in Sec. 4.2.3). **(2)** Technically, *MP5* comprises five interactive modules to meet the requirement of the new paradigm, with an MLLM-based *active perception* scheme to fulfil the situation awareness. To our best knowledge, *MP5* is the first embodied system in Minecraft that is capable of situation-aware planning and action execution. **(3)** Only our method can solve Context-Dependent Tasks (in Tab. 1) by leveraging the situation-aware planning and execution. We constructed a detailed benchmark (in Sec. 4) to explore how agents complete tasks required complex reasoning and constrained by extensive environmental information.

We have provided a comparison of setups and their consequences in Tab. 1. It tells that the proposed *MP5* applies ego-centric RGB images, just utilizes a restrained set of human-defined primitives, but can solve the most challenging context-dependent and process-dependent tasks.

B. Implementation Details

B.1. Percipient

B.1.1 Data Collection

For data collection, we use Minedojo [4] 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 [4] observation space such as Lidar. In order 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 [4]. 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

Table 1. Explicit comparisons of setups and consequences.

Method	Observation Space		Action Space			Instruct Following	Situation-aware Plan	Situation-aware Execution	Tasks Agents Can Perform		
	Info Type	Not Omniscient	Action Type	Action Num	Primitive Library Size				Process (Long-Horizon Tasks)	Context (High Env. Info Tasks)	Process & Context (Combination of 2 Tasks)
DreamerV3	RGB	✓	Original MineRL	25	0	✗	✗	✗	✓	✗	✗
DEPS	RGB	✓	Compound (MineDojo)	42	0	✓	✗	✗	✓	✗	✗
GITM	Text	✗	Manual (MineDojo)	9	9 + corner cases	✓	✗	✗	✓	✗	✗
Voyager	Text	✗	JavaScript APIs	N/A	All	✓	✗	✗	✓	✗	✗
***MP5*(ours)**	RGB	✓	Compound (MineDojo)	10	4	✓	✓	✓	✓	✓	✓

Table 2. Comparison of Observation Spaces Among Different Methods

Method	Perceptual Observation	Status Observation
GITM [19]	LiDAR rays $10 \times 10 \times 10$ Voxels	life statistics GPS, inventory, equipment
DreamV3 [5]	Ego-View RGB	life statistics inventory, equipment
VPT [1]	Ego-View RGB	$\varnothing$
DEPS [17]	Ego-View RGB $3 \times 3 \times 3$ Voxels	Compass GPS, equipment
JARVIS-1 [16]	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

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 [10] to curate a list of instructions to obtain $500K$ image-text instruction-following data.

B.1.2 MineLLM training details

MineLLM combines the image visual encoder from MineCLIP [4] and the large language models from Vicuna-13B-v1.5 [2]. 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 [7], 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 [4] and large language model from Vicuna [2], we collect $500K$ image-text instruction-following data on the MineDojo [4] following the method detailed in Section B.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 [6] parameters to all projection layers of the self-attention layers in the LLM. Only the parameters of the Alignment Net and the LoRA [6] module are optimized during training. Multimodal tokens are decoded by the LLM model and the corresponding LoRA [6] parameters.

The training objective of Percipient is defined as:

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

where θ_a and θ_l correspond to the learnable parameters of the Alignment Net and LoRA [6]. The $\mathcal{I}$ is the image representation produced by the visual encoder from MineCLIP [4] and $\theta = \{\theta_a, \theta_l, \theta_m, \theta_v\}$, where θ_m and θ_v are frozen parameters of MineCLIP [4] and Vicuna-13B-v1.5 [2]. 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 [6]) 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 [6] 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 [4] pre-trained ViT-B/16 [3] as visual encoder, the number of vision tokens are 196 and length of text tokens after vision tokens are limited to 400 in training.

B.2. Memory

Inspired by the Skill library of Voyager [15], memory is utilized in two parts of *MP5* to perform Retrieval-Augmented Generation (RAG [8]). 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 [15].

B.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 [4], and some are user tips from Reddit. Specifically, we convert commonly used knowledge into corresponding text embeddings using OpenAI's text-embedding-ada-002 [11] 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 [8]. 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 [8].

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

B.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 2.

B.4. Action Space

The Performer module executes action sequences, which consist of actions falling within the action space outlined in Tab 3. These actions are brief combinations formed by the MineDojo [4] 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.

C. Environment Setting

Our Minecraft experimental environment is based on the MineDojo [4] 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 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

Table 3. The Definition of the Action Space we use in MineDojo [4] Simulator

Name	Arguments	Description	Corresponding MineDojo [4] 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	$\varnothing$
Place	object	Place an inventory item on the ground.	place	Begins only once the environmental conditions required are met

Table 4. 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

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 Section D.1, we select different environment settings in MineDojo [4]. For *Context-Dependent Tasks*, we uniformly adopt the environment in MineDojo [4] with the creative “task_id” of “0”. For *Process-Dependent Tasks*, we uniformly adopt the environment with the “task_id” of “harvest”, “target_names”

Table 5. 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	✓	✓	✓	✓	✓	✓

as "diamond", and "spawn_rate" as "1". This is why obtaining redstone is more difficult than obtaining diamond, as described in Section D.2.2.

D. Task Details and Experiment Results

D.1. Task Details

D.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. 4. 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. 5. 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.

D.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 6, 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

Table 6. 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*	

Process-Dependent Tasks and took the average success rate as the final reported performance.

D.2. Success Rates of All Task

D.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 7, including ours, GPT-4V [12], and LLaVA-1.5 [9], 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.

D.2.2 Process-Dependent Tasks

We report the success rates of different methods for all tasks comprehensively and in detail in Table 10, 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 2 depicts the game-playing steps corresponding to each milestone object (*e.g.*, log, plank, stick, *etc.*) obtained by the agent.

More evaluations compared with ReAct and Reflextion. ReAct and Reflextion are designed to augment the planning ability of LLMs, thus direct comparison with *MP5* is not quite fair. But we can indirectly compare ReAct and Reflextion in this way: *MP5* can replace Parser, Planner, Patroller, and Memory modules with ReAct or Reflextion, where their results are depicted in Tab. 9. Without changing Percipient and Performer, ReAct merges the Parser and Planner, removes Memory in *MP5*, thus it loses the Patroller's error detection and Planner's triggers for active perception and re-planning. Reflexion extends ReAct by reintroducing error detection and re-planning abilities. The unnecessary integration of the Parser and Planner, as well as the lack of

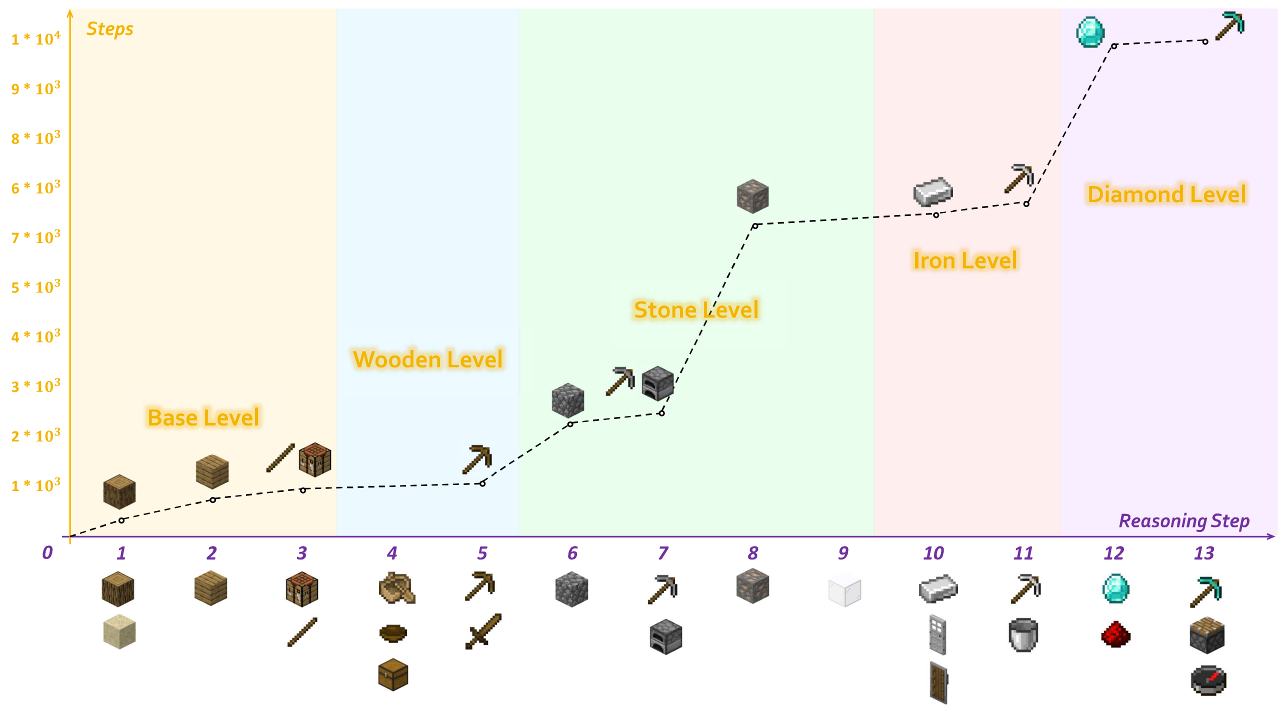

Figure 2. The game-playing steps corresponding to the acquisition of different milestone objects by the agent during the completion of the *craft diamond pickaxe* challenge. The varying background colors denote the level of the *Process-Dependent Tasks* in which the milestone objects are located.

Table 7. 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$_A$ [12]	GPT-4V$_G$ [12]	LLaVA-1.5$_A$ [9]	LLaVA-1.5$_G$ [9]
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

Table 8. 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 [4]	MineLLM+CLIP [13]	LLaVA-1.5 [9]+CLIP [13]
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

Table 9. Comparison with ReAct and Reflexion.

Success Rate (%)	Method *MP5*	*MP5*(ReAct)	*MP5*(Reflexion)
Basic	**96.00**	40.00	52.67
Wooden	**88.67**	10.00	21.33
Stone	**76.00**	0.00	2.00
Iron	**52.00**	0.00	0.00
Diamond	**22.00**	0.00	0.00

situation-aware planning, resulted in a 0% success rate for these two baseline methods at the Iron level. Since Reflection has the capability to detect errors and re-plan accordingly, it leads to a higher success rate than ReAct. Specifically, it is twice as effective at the Wooden level compared to ReAct.

D.3. Ablation Study

D.3.1 Context-Dependent Tasks

We conduct ablation studies on the multi-modal large language model (MLLM) part within *Context-Dependent Tasks* in 8, comparing the performance outcomes of different MLLMs and different pre-trained visual encoders in the percipient.

D.3.2 Process-Dependent Tasks

In this section, we present detailed results from our ablation experiments. Table 12 shows the performance of the agent in *MP5* after the removal of various modules. Table 13 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 [14] and Vicuna-13B-v1.5-16k [2]. Table 14 further explores the contribution of the Memory components to the agent's performance, including Knowledge Memory and Performer Memory. Table 15 investigates the robustness gain brought by the check part of the Patroller under *"Random Drop"* conditions.

As seen from the results in Table 12, 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 13 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 rea-

Table 10. 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

soning 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 [14] Vicuna-13B-v1.5-16k [2] 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 14, 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 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 15 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

Table 11. 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 [2]	S	100	95	75	45
	M	100	100	95	80
GPT-3.5-turbo [10]	S	100	100	85	70
	M	100	100	100	100

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.

E. 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)[18] 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. 11, We can observe that Multi-round Generation using COT[18] generates better corresponding environment information query and thus have a higher success rate on the *Context-Dependent Tasks*.

F. Applications

F.1. Obtain Diamond Pickaxe

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

F.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.

F.3. Open-Ended Tasks

We demonstrate a *Open-Ended Tasks* "*Dig a block of sand under the water with a wooden shovel during the daytime on a sunny day*" in Video 3.

G. 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.

G.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 3. The corresponding screenshots are illustrated in Figure 5.

G.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 4. The corresponding screenshots are illustrated in Figure 6.

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

[2] 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, 8, 9, 10, 13

[3] 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

[4] 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. 1, 2, 3, 4, 8

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

[6] 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. 2

[7] 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. 2

[8] 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, 3

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

[10] OpenAI. Introducing chatgpt. 2022. 2, 10, 13

[11] OpenAI. New and improved embedding model. 2022. 3

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

[13] 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

[14] 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. 8, 9, 13

[15] 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, 3

[16] 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

[17] 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

[18] 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. 10

[19] 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

Table 12. 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 13. More detailed success rates for different LLMs as 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 [10]	LLaMA2-70B-Chat [14]	Vicuna-13B-v1.5-16k [2]
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 14. 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 15. 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 3. 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 4. Dialogue of task “*craft wooden pickaxe*” while re-planning

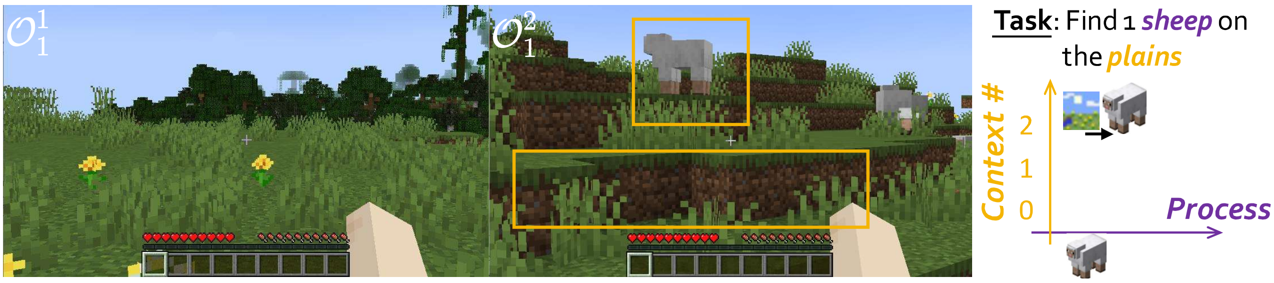

Figure 5. The corresponding screenshots for the dialogue of task "*Find 1 sheep* on the plains"

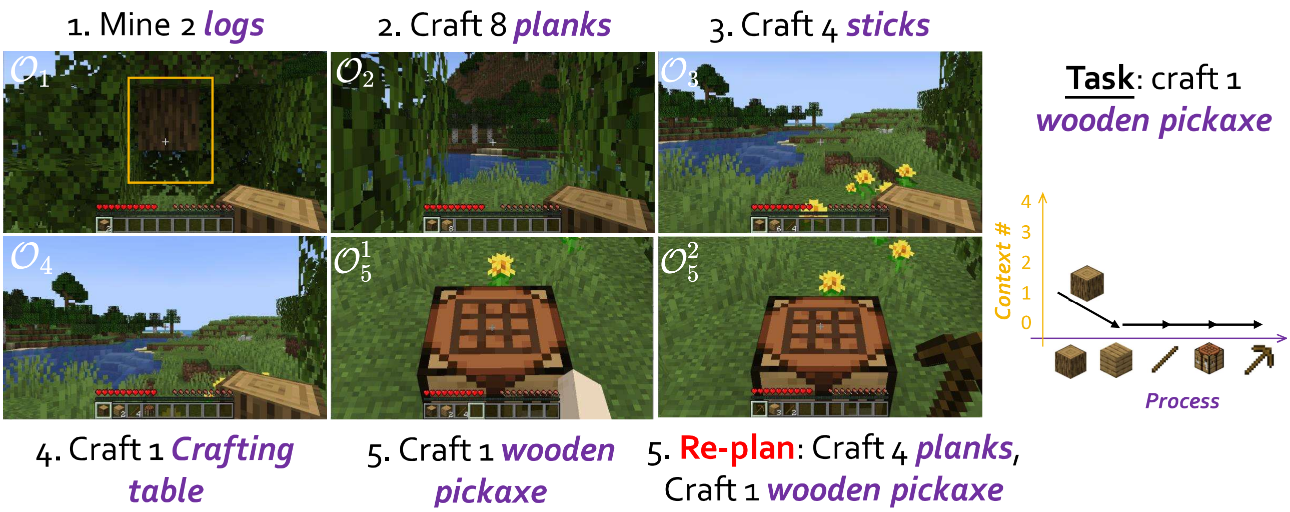

Figure 6. The corresponding screenshots for the dialogue of task "*Find 1 sheep* on the plains" while re-planning