JARVIS-1: Open-world Multi-task Agents with Memory-Augmented Multimodal Language Models

Zihao Wang[1,2], **Shaofei Cai**[1,2], **Anji Liu**[3], **Xiaojian Ma**[3], **Yitao Liang**[1,4]*
Team CraftJarvis
[1]Institute for Artificial Intelligence, Peking University
[2]School of Intelligence Science and Technology, Peking University
[3]Computer Science Department, University of California, Los Angeles
[4]Beijing Institute for General Artificial Intelligence (BIGAI)
`{zhwang,caishaofei}@stu.pku.edu.cn`
`liuanji@cs.ucla.edu,xiaojian.ma@ucla.edu,yitaol@pku.edu.cn`

Abstract

Achieving human-like planning with multimodal observations in an open world is a key milestone for more functional generalist agents. We introduce **JARVIS**-1, an open-world multi-task agent that can perceive multimodal input (visual observations and human instructions), generate sophisticated plans, and perform embodied control, all within the popular yet challenging Minecraft universe. Specifically, we develop **JARVIS**-1 on top of pre-trained Vision-Language Models, which maps visual observations and textual instructions to plans. The plans will be ultimately dispatched to the goal-conditioned controllers. We outfit **JARVIS**-1 with a multimodal memory, which facilitates planning using both pre-trained knowledge and its actual game survival experiences. Additionally, **JARVIS**-1 can rectify its own planning errors through retrospection, *i.e. self-debug*. In our experiments, **JARVIS**-1 exhibits nearly perfect performances across over 200 varying tasks in Minecraft, ranging from entry to intermediate levels. **JARVIS**-1 has achieved a completion rate of 12.5% in the long-horizon diamond pickaxe task. This represents a significant increase up to 5 times compared to previous records.

1 Introduction

Creating sophisticated agents that can accomplish myriad of tasks in complex domains remains a pivotal milestone towards generally capable artificial intelligence [42, 8, 3, 6, 58]. Recent advancements have shown a trend towards employing a hierarchical goal execution architecture [49, 22, 21], and leveraging large language models (LLMs) as the high-level planner to generate action plans that will be ultimately executed by low-level instruction-following controllers. Albeit the fruitful progress they have yielded in many robotics [22] and even open-world environments like Minecraft [14, 19], today's agents built with these approaches are still struggling with three major issues: 1) perceive the world from multimodal sensory observations, such as images, videos in addition to natural language instructions and feedback for planning; This is mostly due to the inability of LLM-based planners on processing multimodal data [21, 53]; 2) perform consistent and accurate long-term planning. This requires multi-round, knowledge, and reasoning-intensive dialogues, which remain great challenges to LLMs [22]; 3) learn and evolve in a life-long fashion. This calls out the need for agents to propose their own tasks and self-improve. Addressing these issues will unleash the full planning potential of LLM-based agents, and expedite the development of more generalist agents.

---

[*] Corresponding Author.

2nd Workshop on Agent Learning in Open-Endedness (ALOE) at NeurIPS 2023.

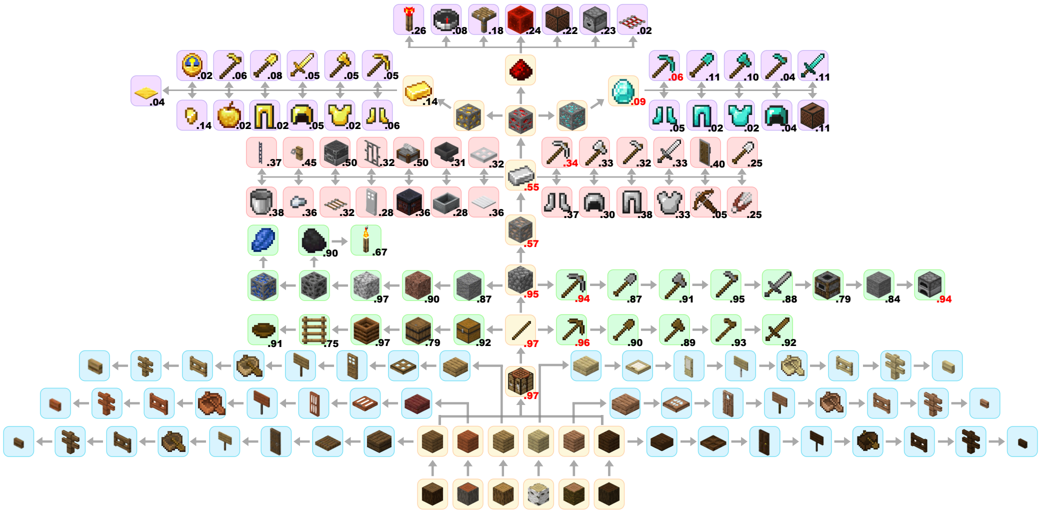


Figure 1: **How does JARVIS-1 unlock the technology tree of the Minecraft universe.** **JARVIS**-1 can consistently obtain high-level items on the main tech-tree of the overworld in Minecraft, such as diamond, redstone, and golden items, which require collecting over 10 different intermediate items. **JARVIS**-1 not only outperforms the previous state-of-the-art VPT [4] (6% vs. 2.5% reliability) on diamond pickaxe, but also can craft almost all diamond items in overworld including diamond chestplate.

In this work, we introduce **JARVIS**-1, a brand new agent that can robustly produce plans for long-horizon tasks from multimodal user and environment inputs, and translate them into motor control in Minecraft, a popular yet challenging open-world testbed for generalist agents. To be specific, we chain a multimodal foundation model MineCLIP [14] and an LLM [8] together, the resulting multimodal language model (MLM) allows our agent to better understand the tasks, situations, and environmental feedback. To further enhance the correctness and consistency of planning, especially on long-horizon tasks, we propose to augment the agent with a *multimodal memory*, which stores both the scenarios and actual plans of the successful planning experiences in the past. By retrieving the relevant memory entries, the planning skill of our MLM-based agent can be strengthened from the agent's own interactions with the environment in an in-context manner. Finally, **JARVIS**-1 is able to evolve throughout the gameplay by proposing tasks on its own (*i.e*. self-instruct) as a means of exploration and saving the obtained experiences in the multimodal memory, therefore facilitating better reasoning and planning. This self-improving ability sparks its potential for a higher level of autonomy.

Our main evaluations are conducted in Minecraft, with more than 200 tasks selected from the Minecraft Universe Benchmark [30], with no demonstration provided. The tasks cover a broad spectrum from the early game (*e.g*. `ObtainCraftingTable`) to intermediate and even challenging long-horizon tasks (*e.g*. `ObtainDiamondPickaxe`). A glimpse of what **JARVIS**-1 is able to achieve can be found in Figure 1. **JARVIS**-1 exhibits strong performances on these tasks, representing an up to 5× increase to the previous records. Our ablative analysis then offers a detailed account of how **JARVIS**-1 approaches this significant progress and becomes the first agent that can robustly obtain the diamond pickaxe with up to 12.5% success rate. What is even more surprising is that, without the need for additional training, **JARVIS**-1 demonstrates a continuous increase in performance as game time increases in long-horizon tasks. Moreover, **JARVIS**-1 has demonstrated its potential of *self-improve* in an exploratory life-long learning experiment, where it needs to propose tasks to progressively explore the world, collect experiences, and sharpen its planning skill using these experiences stored in the multimodal memory.

In summary, **JARVIS**-1 pilots the effort towards a human-like multi-task and autonomous agent in an open-world, embodied environment like Minecraft. We would like to share the key takeaways of what we have learned during its development as follows:

- **From LLMs to MLMs.** The capability of perceiving multimodal sensory input is critical to planning in a dynamic and open-world world. **JARVIS**-1 enables this by chaining a multimodal foundation model together with an LLM. Compared to LLM "blindly" produces plans, MLM is able to natively understand the current situation and plan accordingly. Further, rich environmental

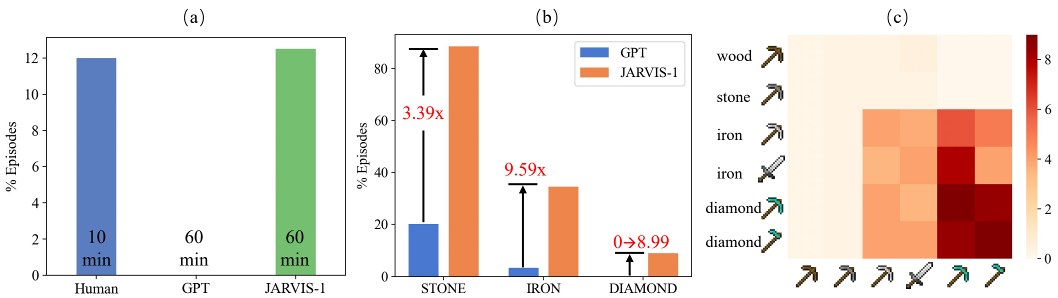


Figure 2: **Challenges in open-world environments and how does JARVIS-1 tackle them**. **(Left)** With situation-aware planning, **JARVIS**-1 substantially improves the success rate on the challenging `ObtainDiamond` task, compared to the baseline (GPT) without it. Note: Due to resource constraints, we can only provide human results of 10-min gameplay; **(Middle)** As task complexity increases (STONE→IRON→DIAMOND), **JARVIS**-1 exhibits more significant advantages thanks to interactive planning; **(Right)** Success rate gain (indicated by the color depth) on selected tasks (x-axis) given in-context experiences on other tasks (y-axis) retrieved from the multimodal memory. With life-long learning and memory, **JARVIS**-1 can utilize prior experiences on relevant tasks for better planning.

feedback can be obtained through multimodal perception, therefore helping the *self-check* and *self-explain* of the planner spot and fix possible bugs in the plans, enabling stronger interactive planning.

- **Multimodal memory.** Early research has suggested the crucial role that memory mechanisms can serve in the functioning of generalist agents. By outfitting **JARVIS**-1 with a multimodal memory, we effectively allow it to plan with both pretrained knowledge and its actual experiences in the world, therefore bringing significant improvement to planning correctness and consistency. Compared to canonical RL or planning agents with exploration, no additional model update is needed as the MLM in **JARVIS**-1 makes it possible to leverage these experiences in an in-context manner.
- **Self-instruct and self-improve.** A sign of generalist agents is the capacity to proactively acquire new experiences and continuously improve themselves. We have demonstrated how **JARVIS**-1 effectively traverses the environment by executing tasks autonomously generated through its *self-instruct* mechanism. With multimodal memory teaming up with experiences from the explorations, we have observed consistent improvement, especially in accomplishing more complicated tasks. Ultimately, this aspect of autonomous learning in **JARVIS**-1 signifies an evolutionary step towards generalist agents that can learn, adapt, and improve over time with minimal external intervention.

2 Challenges for Agents in open-world Environments

Compared to canonical scenarios with relatively small scale, simple dynamics, and limited tasks, open-world environments impose substantial challenges to building agents that can accomplish a diverse set of tasks [14, 18, 17, 23, 9, 9, 49, 10]. In this section, we will review three major challenges we've identified during the development of **JARVIS**-1.

2.1 Challenge I: Situation-Aware Planning

In an open world, there could be various possible paths towards an open-world goal. However, not all of them are plausible or equally efficient given a certain *situation* (location, inventory status, *etc*.). For example, building a bed can be done through collecting wool from sheeps , haunting spiders for strings , or trading with villagers . Depending on the current location and its proximity to these subjects, some options can be more viable and more efficient than others. Further, the agent's own situation can also change throughout the episode, *e.g*. day and night shifts, weather conditions (bringing different types of danger), tool usage (it can be broken). To this end, the plan needs to be constantly updated based off the current situation. Figure 2 (left) shows that when attempting the "ObtainDiamondPickaxe" task with a GPT-based planner that produces plans only at the beginning without looking at the current situation, the agent failed to complete the task as opposed to human players and **JARVIS**-1, which perform situation-aware planning from time to time. We've observed that many failures coming from this were attributed to the agent's inability to adapt to the changing situations including entering a new biome, the tool being used becoming broken, *etc*.

2.2 Challenge II: Task Complexity

The second challenge comes from the higher task complexity in open-world environments. Due to the richness of terrains, objects, and action space, tasks in open-world domains usually require substantially long planning horizons as well as good accuracy and precision. For example, the task `ObtainEnchantingTable` includes more than 20 different sub-goals and therefore demand significantly longer reasoning steps. Meanwhile, many of these sub-goals have to be achieved precisely with the exact object name, quantities, and preconditions, e.g., `mine 3 obsidian with diamond pickaxe`, `craft 1 diamond pickaxe from 3 diamonds and 2 sticks on crafting table`; otherwise, the subsequent sub-goals won't be executed due to unfulfilled preconditions. To tackle this, we may refer to some approaches in LLM reasoning, *e.g*. self-debugging [11] and turning the planning into an interactive fashion. In Figure 2 (Middle), we've shown that as the complexity of the task increases, our **JARVIS**-1, which uses interactive planning [49] to mitigate the aforementioned issues (details can be found in Section 3.2), elicits more significant advantages over the baseline (GPT) planner.

2.3 Challenge III: Life-long Learning

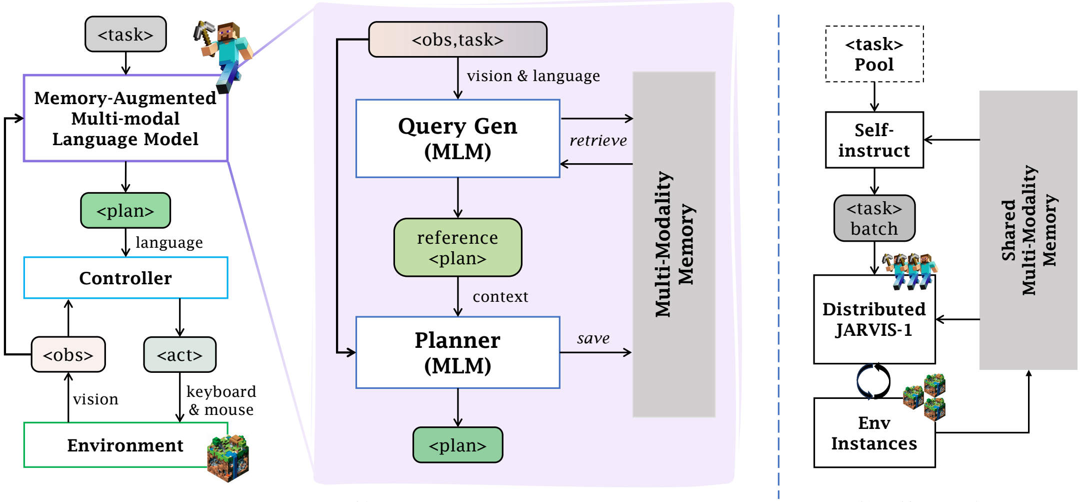
Finally, being open world often implies offering an infinite number of tasks. Clearly, it is difficult for an agent to master all tasks or generalize to arbitrary tasks without additional learning. To this end, agents in an open world should be able to learn novel tasks while completing existing tasks, *i.e*. life-long learning. Furthermore, as many open-world agents employ large models [49, 54, 47, 59], canonical gradient-based learning could be extremely inefficient given the number of new tasks and experiences to learn. Our MLM-based **JARVIS**-1 tackles this by adopting a memory to save all the experiences on past tasks. By retrieving memory entries relevant to the newly-coming task and putting them into the context as a reference, **JARVIS**-1 is able to accumulate more experiences as the game continues and strengthen its own planning skills without gradient update. As illustrated in Figure 2 (Right), for instance, both `ObtainDiamondPickaxe` and `ObtainDiamondAxe` require gathering almost identical materials. Therefore, they can help each other by using the experiences from the other task. Compared to completing these challenging tasks without any prior experiences, memory-based in-context life-long learning in **JARVIS**-1 can bring significant advantages.

3 Multi-task Agent with Memory-Augmented MLM

This section details the architecture of the proposed **JARVIS**-1 agent. We begin with an overview of the modular agent design in Section 3.1. Next, we elaborate on how to implement an interactive planning scheme with a multimodal language model, which helps with more accurate plans, especially on complex and long-horizon tasks in Section 3.2. Finally, we show how to augment this planning framework with a multimodal memory to allow **JARVIS**-1 to strengthen its planning skill throughout the episode by in-context life-long learning in Section 3.3 and Section 3.4.

3.1 Overview

We aim to develop an agent capable of solving long-horizon instruction-following tasks using image observations and human-aligned actions. To accomplish this, we propose a multi-modal agent including an interactive **planner**, a goal-conditioned **controller**, and a **multimodal memory** of multimodal experiences. Upon receiving a task and the current observation, **JARVIS**-1 first utilizes the MLM to generate a multimodal query (**query gen**) that retrieves relevant planning experiences from the memory. These experiences will then be used along with the planning instruction to prompt the MLM-based planner. Leveraging its own pretrained knowledge as well as the retrieved reference plans, the planner will ultimately produce a series of K short-horizon goals $g_1, \ldots, g_K$ to be executed by the controller. Once the plan is successfully executed, it will be stored in the memory along with the task and the agent situation when it was planned. We also empower **JARVIS**-1 with life-long learning by combining *self-instruct*, where **JARVIS**-1 will propose some tasks for itself to complete as a means of exploration; and *self-improve*, where multiple **JARVIS**-1 agents will be running in



(a) JARVIS-1 architecture

(b) Self-Improving

Figure 3: **Architecture of JARVIS-1 and its self-improving mechanism**. (a) **JARVIS**-1 comprises a memory-augmented multimodal language model (MLM) that produces plans and a low-level action controller. **JARVIS**-1 also utilizes a multimodal memory to store and obtain experiences as references for planning. (b) **JARVIS**-1 can strengthen its own planning skills through exploration with its own proposed tasks (*self-instruct*) and a growing memory that helps with better planning on tasks that have been (partially) visited before.

parallel to gather experiences, therefore helping with better planning later. We provide an illustration in Figure 3.

3.2 Interactive Planning with MLM

As we have mentioned in Section 2.1 and Section 2.2, the primary challenges for planning in Minecraft come from the requirement of being able to plan for long-horizon tasks under dynamic observations. Confirmed by many prior arts [49, 47, 54], this makes it exceptionally hard to utilize canonical symbolic planners, which can be much less flexible. To this end, we take a multimodal language model (MLM) as zero-shot **planner** and combine it with an interactive planning framework to tackle these challenges.

Situation-aware planning with MLM. To achieve situation-aware planning, the planner must take the current observation into account, in addition to the task instruction [21, 53]. Specifically, we begin with translating the multimodal observation into text descriptions. As opposed to letting the MLM caption the scene directly, we first extract keywords of Minecraft items (e.g., "`acacia tree`", "`sheep`") from Minecraft wiki and utilizing GPT [8] to generate sentences that describe these observations. For example, a generated sentence could be "I can see sheep in the acacia plains". Then the MLM will retrieve the condition sentence according to current visual observation during planning. Additional situation details including biome and inventory status are also converted into text using templates. Finally, we prompt the MLM again (the language part only) into a plan given the task instruction and all the aforementioned textual situation descriptions. Compared to end-to-end alternatives [5, 20], we find our composable usage of MLM provides higher quality situation descriptions and ultimately, plans with much less hallucination.

Planning with self-check. Our first layer of shield to ensure the correctness of plans involves *self-check*. Similar to self-debugging[11], given an initial plan, we ask **JARVIS**-1 to progressively simulate the plan execution, predict the resulting state after each step (primarily the state of inventory), and evaluate them. By verifying if these states satisfy the goal's precondition, **JARVIS**-1 can proactively identify potential plan flaws. Compared to the canonical planner where the agent has to encounter the error first before making a remedy, this upfront plan verification could mitigate the need for the agent to recover (re-plan) from more challenging situations due to plan failure. For instance, if an agent starts digging underground without sufficient wood, it would typically have to return to the surface, which substantially lowers the chance of completing the task.

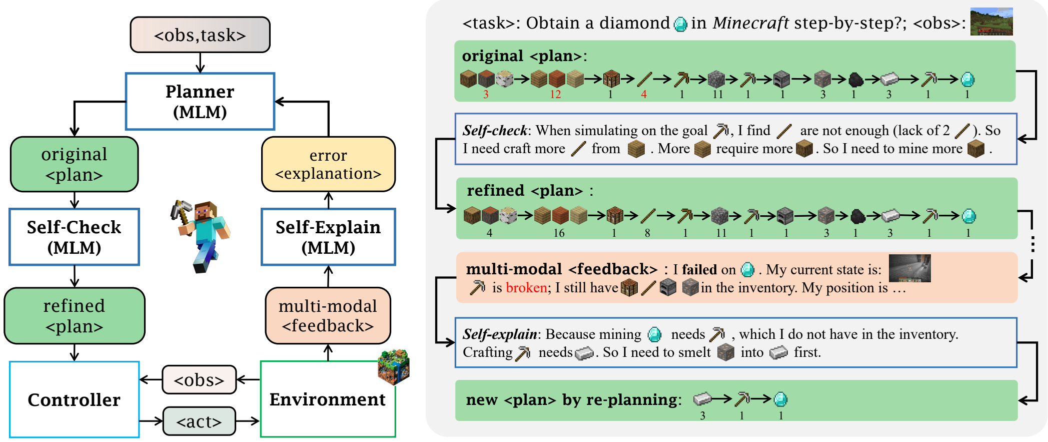

<obs,task>
Planner (MLM)
original <plan>
error <explanation>
Self-Check (MLM)
Self-Explain (MLM)
refined <plan>
multi-modal <feedback>
Controller
<obs>
<act>
Environment
<task>: Obtain a diamond in Minecraft step-by-step?; <obs>:
original <plan>:
3 12 1 4 1 11 1 1 3 1 3 1 1
Self-check: When simulating on the goal, I find are not enough (lack of 2). So I need craft more from. More require more. So I need to mine more.
refined <plan> :
4 16 1 8 1 11 1 1 3 1 3 1 1
multi-modal <feedback> : I failed on. My current state is: is broken; I still have in the inventory. My position is …
Self-explain: Because mining needs, which I do not have in the inventory. Crafting needs. So I need to smelt into first.
new <plan> by re-planning:
3 1 1

Figure 4: **Interactive planning in JARVIS-1**. After receiving the current task instruction and observation, **JARVIS**-1 will produce an initial plan, which will go through *self-check* to get possible bugs (marked in red) fixed. Further, in case any error (also marked in red) occurs during the execution of the refined plan, **JARVIS**-1 will try to reason about the next move from the environmental feedback via *self-explain*. Interleaving self-check and self-explain significantly boosts the correctness and robustness of **JARVIS**-1 planning.

Planning with environment feedback. Next, our interactive planning framework ventures into allowing **JARVIS**-1 to quickly recover from failure by leveraging environment feedback in a closed-loop fashion. The process is illustrated in Figure 4. During plan execution, we feed the feedback to the MLM of **JARVIS**-1 in case there is any execution failure (possibly due to a flawed plan) and utilize its self-explain mechanism [43] to explain the error and locate the bugs in the original plan (we term this as *error explanation*). Finally, the MLM planner of **JARVIS**-1 will produce an improved plan based on both the outside environment feedback and the inside retrospective. Compared to other agents that rely on human intervention or privileged environment information [22, 59], **JARVIS**-1 has the ability to speculate about the reasons why current goals cannot be achieved, without the need for additional information or design.

3.3 Planning with Multimodal Memory in the Loop

To address the life-long learning challenge mentioned in Section 2.3, we equip **JARVIS**-1 with multimodal memory to allow learning from its own past experiences. We will detail the formulation of the retrieval-augmented planning, query generation, and memory layout below.

Retrival-augmented planning. Retrieval-augmented generation (RAG) [27, 36] enhances the quality of responses generated by LLMs by incorporating external sources of knowledge to complement the model's internal representation. We also utilize RAG to enhance **JARVIS**-1's long-term planning capability. Compared to official RAG methods leveraging the external knowledge library, we take the collected multimodal memory as the knowledge library and retrieve the interactive experiences as the demonstration prompt to augment the planning results. The formulation is as follows:

$$p(y \mid x) \approx \sum_{z \in \text{top-k}(p(\cdot \mid x))} p_\eta(z \mid x) p_\theta(y \mid x, z), \tag{1}$$

where x, y, and z denote instruction, plans, and retrieved memory entries respectively, and p_η and p_θ are denoted as retrieval and planning models. Such retrieval-augmented planning method helps **JARVIS**-1 ground the internal knowledge into the open-ended environments efficiently and leverage the historical interaction feedback to solve the hallucination within LLMs and produce more accurate plans.

Multimodal memory. We have demonstrated the layout of our multimodal memory on the right side of Figure 5. From a high level, it is a key-value memory where the keys are multimodal, comprising both the task and the observation (or situation) made when this memory entry was created. The values are the plans that were successfully executed. Note that, since the plans in an open-world environment like Minecraft are situated (see Section 2.1), there could be multiple entries that are with the same task but different observations and plans. As a result, **JARVIS**-1 needs to produce multimodal queries based on the current task and situations to retrieve the relevant memory entries.

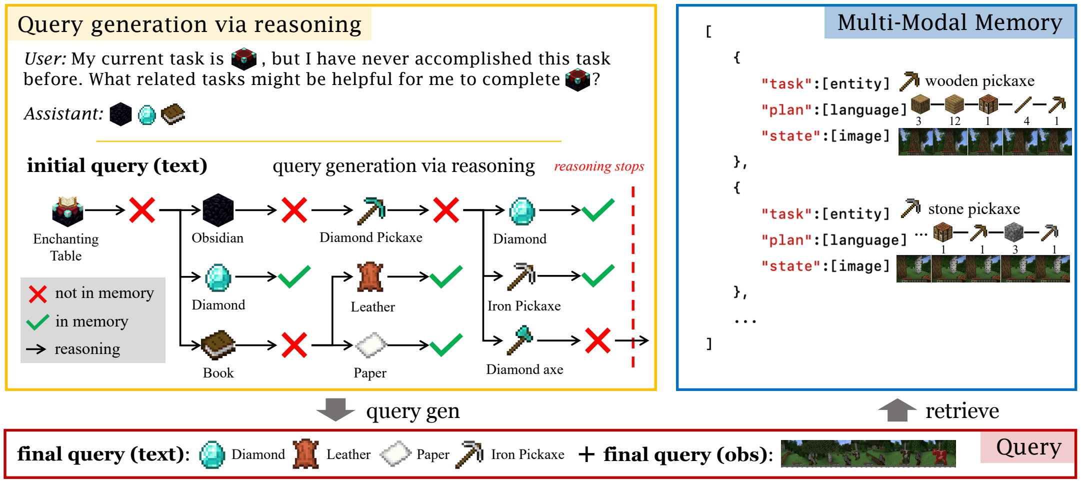

Query generation via reasoning
User: My current task is , but I have never accomplished this task before. What related tasks might be helpful for me to complete ?
Assistant:
initial query (text)
query generation via reasoning
reasoning stops
Enchanting Table
Obsidian
Diamond Pickaxe
Diamond
Diamond
Leather
Iron Pickaxe
Book
Paper
Diamond axe
not in memory
in memory
reasoning
Multi-Modal Memory
[
{
"task":[entity] wooden pickaxe
"plan":[language]
3
12
1
4
1
"state":[image]
},
{
"task":[entity] stone pickaxe
"plan":[language] ...
1
1
3
1
"state":[image]
},
...
]
query gen
retrieve
final query (text): Diamond Leather Paper Iron Pickaxe + final query (obs):
Query

Figure 5: **Query generation in JARVIS-1**. Given the current observation and the task, **JARVIS**-1 will first think backward and figure out the needed intermediate sub-goals. The reasoning will be bounded by a limited depth. The sub-goal that is present in the memory will join the current visual observation to form the final query. Entries that match the text query will be ranked by the perceiving distance of their states to the obs query and only the top entry of each sub-goal will be retrieved.

Query generation via reasoning. When presented with an instruction as a task, we employ query generation via LLM reasoning to decompose the instruction into sub-tasks or related tasks, which will then be used as textual queries to retrieve relevant planning experiences as references for solving the current task. For instance, consider the instruction "craft 1 enchanting table with empty inventory" as shown in Figure 5. **JARVIS**-1 queries the MLMs to identify the tasks that are required for achieving the main task in a backward search fashion, e.g., "obtain book /diamond /obsidian with empty inventory". The search depth is bounded for efficiency. Further, instead of relying solely on retrieval based on the text query [47, 59], we also propose to append the agent's current visual observation to the textual query, resulting in a *multimodal query* to take the situation into account during memory retrieval.

Multimodal retrieval. After obtaining the textual and visual query, we compute the alignment between the query and each trajectory in multimodal memory. We first use the text encoder of CLIP model to compute the embedding of the query and task key of each entry in memory. We select the memory entries with similarity higher than the confidence threshold as the candidate entries. Then we will compute the visual state embedding of query and states in candidate entires. Then we sort the candidate entries with the visual embedding similarities, which can be formed as:

$$p_\eta(z \mid x) \propto \text{CLIP}_v(s_z)^\top \text{CLIP}_v(s_x), \tag{2}$$

where s_z and s_x are the visual key of memory entries and visual query, respectively. Finally, we retrieve the plan of top-k candidate entries as reference prompt z.

3.4 Self-improving Agents

Learning in Minecraft with memory. The remaining issue now is where the aforementioned multimodal memory comes from. Inspired by the life-long learning scheme in many close-world and open-world reinforcement learning problems [1, 2, 47], we propose the following learning approach for augmenting the memory in **JARVIS**-1: 1) First, we generate a set of tasks, which form some curricula for the agents to complete as means of exploration of the world. During this process, **JARVIS**-1 produces plans, interacts with the environment, embraces the errors, and stores all these experiences in the memory; 2) After this learning stage, we evaluate **JARVIS**-1 on various tasks. Therefore, **JARVIS**-1 is able to produce better plans with the memory teaming up with the planning experiences. In our experiments, we use this as the default setting for all tasks.

Exploration using self-instruct. The key issue to the success of learning with memory is how to effectively acquire useful experiences given a limited amount of time. We propose to use self-instruct [48] to generate the dynamic curriculum and guide **JARVIS**-1 to learn from the interactions with environments. In each round, we prompt the MLM to consider how capable **JARVIS**-1 is at this point and subsequently select tasks from a task pool to explore. We find that the curriculum almost follows the technical tree-growing direction. To accelerate the learning process, we augment the linear self-instruct to distributed learning in distributed environments with shared memory, *i.e*. speculative execution [26]. Specifically, we generate multiple executable tasks as candidate task batches and provide them to agents with the same memory for verification and execution in various different environments. Meanwhile, experiences are collected into a shared centralized memory. When all exploration tasks have been accomplished, we move to the next round, until the memory reaches a certain capacity.

Life-long learning. We've also observed that the aforementioned learning (where the memory is being filled) can be extended throughout the whole gameplay, where the agent gradually acquires more and more skills. As the gameplay continues, more and more experiences are pouring in, therefore **JARVIS**-1 can find better references for challenging tasks like `ObtainDiamondPickaxe`, resulting in an improved success rate on these tasks. Further, there is no gradient update in this thanks to the memory-augmented MLM, *i.e.* we can do in-context life-long learning. In Section A.2, we offer exploratory experiments to show the potential of such capability of **JARVIS**-1.

4 Experiments

In the experiments, our goal is to 1) evaluate the general performances of **JARVIS**-1 on the challenging Minecraft tasks, especially on its advantages over baselines that do not (fully) address the aforementioned issues in open-world agents; 2) understand the factors that contributes to the general results; 3) explore the potential of **JARVIS**-1 in terms of life-long learning and its benefits to long-horizon tasks. To this end, we will first briefly introduce the evaluation settings, then cover the main comparative results and ablation studies, and conclude with an exploratory trial on long-horizon tasks.

4.1 Experimental Setups

We evaluate **JARVIS**-1 in Minecraft, with tasks selected from the recently introduced Minecraft Universe Benchmark [30]. For the reader's convenience, we provide details on the basic setups below.

Environment setting. To ensure realistic gameplay, the agent needs to utilize observation and action spaces that are similar to those used by humans. Instead of manually designing a custom interface for models to interact with the environment, as done in previous methods such as MineDojo[14], GITM[59], and Voyager[47], we opt for using the native human interface provided by Minecraft. This applies to both the observation and action space. The model operates at a speed of 20 frames per second and is required to use a mouse and keyboard interface when interacting with human GUIs. For more information on the detailed descriptions of the observation and action spaces, please refer to the Appendix.

Task setting. In Minecraft, players have access to thousands of items, each with specific acquisition requirements or recipes. For example, stone-type items can only be obtained using a pickaxe, and two planks can be crafted into four sticks (these requirements are available on the Minecraft Wiki[2]). In survival mode, players must obtain each type of item from the environment or craft/smelt the object item from materials. We choose over 200 tasks from the Minecraft Universe Benchmark [30] for evaluation. These tasks are related to items that can be obtained in the Minecraft overworld. For the convenience of statistics, we have classified them into 11 groups according to recommended categories[3] in Minecraft (see Table 6 in Appendix). Due to the varying complexity of these tasks, we adopt different maximum gameplay durations (Max. Steps) for each task. The limit is determined by the average time the human players need to accomplish the corresponding task. Other details about each task, such as language instruction, maximum steps, evaluation times, biome, and initial inventory when the agent is born into the world can be found in Appendix Table 7-16.

Evaluation metrics. By default, the agent always starts in survival mode, with an empty inventory. A task is considered a success when the target object is obtained within a specified time. Due to the open-world nature of Minecraft, the world and initial position that the agent is spawned at could vary

---

[2] https://minecraft.fandom.com/wiki/Minecraft_Wiki

[3] https://minecraft.fandom.com/wiki/Tutorials/Organization#Categories

Table 1: Results of **JARVIS**-1 and baselines on Minecraft. The detailed task instructions, settings and results can be found in the Appendix.

Group	Task	GPT	ReAct	Inner Monologue	DEPS	JARVIS-1
Wood		26.67	45.00	36.67	75.00	**91.55**
	AVG	27.30±14.86	40.31±13.30	60.15±19.41	80.23±17.32	**88.84±16.82**
Wood Var		6.67	36.67	30.00	36.67	**60.47**
	AVG	24.39±11.08	38.13±12.81	53.39±12.86	68.75±12.32	**76.78±12.27**
Stone		20.00	20.00	66.67	75.00	**94.20**
	AVG	20.21±12.32	39.00±12.15	52.86±16.90	69.27±7.78	**88.69±4.87**
Iron		0.00	0.00	3.33	20.00	**33.82**
		3.33	6.67	0.00	20.00	**38.10**
	AVG	3.27±2.85	4.61±3.63	5.20±5.17	16.92±4.69	**34.63±10.61**
Gold		0.00	2.00	2.00	6.00	**14.49**
	AVG	0.00±0.00	0.45±0.60	0.59±0.64	2.20±1.55	**6.85±4.71**
Diamond		0.00	0.00	1.00	2.00	**9.20**
		0.00	0.00	0.00	2.50	**6.22**
	AVG	0.00±0.00	0.35±0.48	0.96±0.67	2.42±1.01	**8.99±2.68**
Redstone		0.00	2.00	0.00	10.00	**22.78**
	AVG	1.04±1.30	1.14±1.18	0.69±1.68	6.02±3.61	**17.51±9.34**
Blocks		16.67	33.33	43.33	53.33	**86.67**
	AVG	45.64±33.88	49.35±30.51	55.71±29.43	58.02±27.68	**80.34±21.09**
Armor		6.67	0.00	10.00	10.00	**30.30**
	AVG	1.36±2.25	0.50±0.88	3.10±4.71	3.71±3.78	**13.44±14.62**
Decoration		15.00	15.00	15.00	25.00	**50.00**
	AVG	17.12±11.59	17.13±9.19	12.03±10.19	29.59±15.94	**46.67±23.39**
Food		13.33	16.67	25.00	16.67	**43.55**
	AVG	9.40±4.29	15.56±6.83	20.78±11.99	22.85±8.15	**46.75±11.16**

a lot. Therefore, we conducted at least 30 tests for each task using different seeds and reported the average success rate to ensure a thorough assessment. Further, since we categorize the tasks into groups, we also report mean and variance values for each group for ease of presentation.

4.2 Main Results

We compare **JARVIS**-1 with other multi-task instruction-following agents based on LLM, including Instruct GPT [21, 40], ReAct [53], Inner Monologue [22], DEPS [49]. Since some methods are not originally experimented in Minecraft, we reproduce them to conform to the Minecraft specification based on prompt and feedback template design. All LLM-based methods access the LLM model through OpenAI API. And all hyper-parameters of LLM including temperature are kept as default.

The average success rates for every task group are listed in Table 1. **JARVIS**-1 achieves the best performance with all meta tasks. It is important to note that in Minecraft, the technology tree can be formed by Group Wood, Stone, Iron, Gold, and Diamond. The tasks become increasingly difficult as you progress through the tree. For more difficult tasks such as obtaining a gold ingot or a diamond, the agents typically need to perform more actions and longer goal sequences in order to complete the task. As a result, the success rate of all agents decreases as the difficulty level increases. It is evident that reasoning methods (ReAct [53] vs. GPT [40, 21]) and interactive re-planning with feedback (Inner Monologue [22] vs. GPT) effectively enhance the agent's task performance in an open world. However, these approaches still face challenges when dealing with long-horizon tasks, specifically in the Iron and Diamond group. DEPS [49], on the other hand, enables agents to accomplish diamond-related tasks through interactive long-horizon planning accompanied by descriptions and explanations. Nevertheless, its reliability remains very low at approximately 2.5%.

In comparison to DEPS [49] without memory, **JARVIS**-1 demonstrates superior performance even in challenging tasks due to its extensive experience. In diamond-related tasks specifically, the success rate has increased by nearly 3 times (8.99% vs 2.42%). And **JARVIS**-1 usually only requires 2-3 rounds of re-planning to generate the correct executable plan, whereas DEPS requires more than 6 rounds. This means that **JARVIS**-1 saves a significant amount of LLM tokens and thinking time, enabling more efficient plan execution and providing additional steps and tokens for handling uncertainty in the environment.

Based on our observations, we have found that the bottleneck for **JARVIS**-1 in tasks involving diamonds often lies with the Controller's inability to perfectly execute short-horizon text instructions generated by LLM. Therefore, it is worth exploring methods for generating plans that are easier for the controller to execute or improving the controller's ability to follow instructions.

5 Conclusion

We propose a multi-task agent **JARVIS**-1 designed for the complex environment of Minecraft, marks a significant advancement in achieving human-like planning within an open-world setting. By leveraging pre-trained Multi-modal Language Models, **JARVIS**-1 not only effectively interprets multimodal inputs but also adeptly translates them into actions. Its integration of a multimodal memory, which draws from both ingrained knowledge and real-time game experiences, enhances its decision-making capabilities. The empirical evidence of its prowess is evident in its impressive performance across a wide array of tasks in Minecraft. Notably, its achievement in the long-horizon diamond pickaxe task, where it achieved a completion rate that surpasses VPT by up to five times, underscores its potential and the strides made in this domain. This breakthrough sets the stage for the future of more versatile and adaptable agents in complex virtual environments.

Acknowledgments

This work is funded in part by the National Key R&D Program of China #2022ZD0160301, a grant from CCF-Tencent Rhino-Bird Open Research Fund, NSF grants #IIS-1943641, #IIS-1956441, #CCF-1837129, an SRA from Meta and a research gift from Amazon Alexa AI, and a gift from RelationalAI. The authors sincerely thank Dr. Rita Zhang, Zhixiang Dai at NVIDIA for the valuable technical support of GPU computing.

References

[1] D. Abel, D. Arumugam, L. Lehnert, and M. Littman. State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 10–19. PMLR, 2018. 7

[2] D. Abel, Y. Jinnai, S. Y. Guo, G. Konidaris, and M. Littman. Policy and value transfer in lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 20–29. PMLR, 2018. 7

[3] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022. 1

[4] B. Baker, I. Akkaya, P. Zhokhov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *arXiv preprint arXiv:2206.11795*, 2022. 2, 15, 16, 17, 20

[5] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023. 5

[6] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022. 1

[7] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*, 2022. 16

[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1, 2, 5

[9] S. Cai, Z. Wang, X. Ma, A. Liu, and Y. Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction. *arXiv preprint arXiv:2301.10034*, 2023. 3, 16

[10] S. Cai, B. Zhang, Z. Wang, X. Ma, A. Liu, and Y. Liang. Groot: Learning to follow instructions by watching gameplay videos. *arXiv preprint arXiv:2310.08235*, 2023. 3, 16, 17

[11] X. Chen, M. Lin, N. Schärli, and D. Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023. 4, 5

[12] I. Dasgupta, C. Kaeser-Chen, K. Marino, A. Ahuja, S. Babayan, F. Hill, and R. Fergus. Collaborating with language models for embodied reasoning. In *NeurIPS Foundation Models for Decision Making Workshop*, 2022. 16

[13] Z. Ding, H. Luo, K. Li, J. Yue, T. Huang, and Z. Lu. Clip4mc: An rl-friendly vision-language model for minecraft. *arXiv preprint arXiv:2303.10571*, 2023. 16

[14] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems Datasets and Benchmarks*, 2022. 1, 2, 3, 8, 16, 17, 20

[15] R. Gong, Q. Huang, X. Ma, H. Vo, Z. Durante, Y. Noda, Z. Zheng, S.-C. Zhu, D. Terzopoulos, L. Fei-Fei, et al. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*, 2023.

[16] R. Gong, Q. Huang, X. Ma, H. Vo, Z. Durante, Y. Noda, Z. Zheng, S.-C. Zhu, D. Terzopoulos, L. Fei-Fei, et al. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*, 2023. 16

[17] W. H. Guss, M. Y. Castro, S. Devlin, B. Houghton, N. S. Kuno, C. Loomis, S. Milani, S. P. Mohanty, K. Nakata, R. Salakhutdinov, J. Schulman, S. Shiroshita, N. Topin, A. Ummadisingu, and O. Vinyals. The minerl 2020 competition on sample efficient reinforcement learning using human priors. *arXiv: Learning*, 2021. 3

[18] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, et al. Neurips 2019 competition: the minerl competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 2019. 3

[19] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019. 1, 19

[20] J. Huang, X. Ma, S. Yong, X. Linghu, et al. An embodied generalist agent in 3d world. *arXiv preprint arXiv:2311.xxxx*, 2023. 5

[21] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *ICML*, 2022. 1, 5, 9, 16

[22] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022. 1, 6, 9, 16

[23] A. Kanervisto, S. Milani, K. Ramanauskas, N. Topin, Z. Lin, J. Li, J. Shi, D. Ye, Q. Fu, W. Yang, W. Hong, Z. Huang, H. Chen, G. Zeng, Y. Lin, V. Micheli, E. Alonso, F. Fleuret, A. Nikulin, Y. Belousov, O. Svidchenko, and A. Shpilman. Minerl diamond 2021 competition: Overview, results, and lessons learned. *neural information processing systems*, 2022. 3

[24] Z. Ke, H. Lin, Y. Shao, H. Xu, L. Shu, and B. Liu. Continual training of language models for few-shot learning. *arXiv preprint arXiv:2210.05549*, 2022. 16

[25] Z. Ke, Y. Shao, H. Lin, T. Konishi, G. Kim, and B. Liu. Continual pre-training of language models. In *The Eleventh International Conference on Learning Representations*, 2022. 16

[26] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023. 8

[27] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020. 6

[28] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022. 16

[29] S. Lifshitz, K. Paster, H. Chan, J. Ba, and S. McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *arXiv preprint arXiv:2306.00937*, 2023. 15, 17

[30] H. Lin, Z. Wang, J. Ma, and Y. Liang. Mcu: A task-centric framework for open-ended agent evaluation in minecraft. *arXiv preprint arXiv:2310.08367*, 2023. 2, 8

[31] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023. 16

[32] Z. Lin, J. Li, J. Shi, D. Ye, Q. Fu, and W. Yang. Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:2112.04907*, 2021. 16

[33] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023. 16

[34] J. Mai, J. Chen, B. Li, G. Qian, M. Elhoseiny, and B. Ghanem. Llm as a robotic brain: Unifying egocentric memory and control. *arXiv preprint arXiv:2304.09349*, 2023. 16

[35] H. Mao, C. Wang, X. Hao, Y. Mao, Y. Lu, C. Wu, J. Hao, D. Li, and P. Tang. Seihai: A sample-efficient hierarchical ai for the minerl competition. In *Distributed Artificial Intelligence: Third International Conference, DAI 2021, Shanghai, China, December 17–18, 2021, Proceedings 3*, pages 38–51. Springer, 2022. 16

[36] Y. Mao, P. He, X. Liu, Y. Shen, J. Gao, J. Han, and W. Chen. Generation-augmented retrieval for open-domain question answering. *arXiv preprint arXiv:2009.08553*, 2020. 6

[37] G. Mialon, R. Dessì, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Rozière, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz, et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

[38] J. Oh, S. Singh, H. Lee, and P. Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, pages 2661–2670. PMLR, 2017. 16

[39] OpenAI. Gpt-4 technical report, 2023. 14, 17

[40] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022. 9, 14

[41] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023. 16

[42] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022. 1

[43] N. Shinn, B. Labash, and A. Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023. 6, 16

[44] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022. 16

[45] H. Sun, Y. Zhuang, L. Kong, B. Dai, and C. Zhang. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023. 16

[46] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 14

[47] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. 4, 5, 7, 8, 14, 16, 17

[48] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2022. 7

[49] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023. 1, 3, 4, 5, 9, 15, 16

[50] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022. 16

[51] Y. Wu, S. Y. Min, S. Prabhumoye, Y. Bisk, R. Salakhutdinov, A. Azaria, T. Mitchell, and Y. Li. Spring: Gpt-4 out-performs rl algorithms by studying papers and reasoning. *arXiv preprint arXiv:2305.15486*, 2023. 16

[52] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. 16

[53] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. 1, 5, 9, 16

[54] H. Yuan, C. Zhang, H. Wang, F. Xie, P. Cai, H. Dong, and Z. Lu. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*, 2023. 4, 5, 16, 17

[55] A. Zeng, A. Wong, S. Welker, K. Choromanski, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022. 16

[56] C. Zhang, K. Yang, S. Hu, Z. Wang, G. Li, Y. Sun, C. Zhang, Z. Zhang, A. Liu, S.-C. Zhu, et al. Proagent: Building proactive cooperative ai with large language models. *arXiv preprint arXiv:2308.11339*, 2023. 16

[57] W. Zhang and Z. Lu. Rladapter: Bridging large language models to reinforcement learning in open worlds. *arXiv preprint arXiv:2309.17176*, 2023. 16

[58] H. Zhao, Z. Cai, S. Si, X. Ma, K. An, L. Chen, Z. Liu, S. Wang, W. Han, and B. Chang. Mmicl: Empowering vision-language model with multi-modal in-context learning. *arXiv preprint arXiv:2309.07915*, 2023. 1

[59] X. Zhu, Y. Chen, H. Tian, C. Tao, W. Su, C. Yang, G. Huang, B. Li, L. Lu, X. Wang, et al. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023. 4, 6, 7, 8, 16, 17

A Experimental Results

A.1 Ablation Studies

A.1.1 JARVIS-1 based on different LMs

We conducted ablation experiments on various Language Models, including OpenAI's ChatGPT [40] and GPT-4 [39]. Among these models, GPT-4 has more parameters and has been proven to outperform ChatGPT in extensive research [47]. We also select the open-source pre-trained LLaMA2 70B (LLaMA2 PT) model [46]. Additionally, we gathered a substantial amount of Minecraft-related text from the internet as training data and further fine-tuned LLaMA2 13B (LLaMA FT). The experiments were conducted on a subset of Minecraft tasks using different language models. Each **JARVIS**-1 learns for 4 epochs of interaction with all task sets and evaluates on task subset across at least 20 seeds. The experimental results are presented in Table 2.

Table 2: **Success rates for different LLMs on Minecraft tasks.**

Task	GPT-4	ChatGPT	LLaMA2 Pretrained	LLaMA2 Finetuned
📦	0.97	0.95	0.55	0.85
⛏	0.96	0.95	0.50	0.85
⛏	0.94	0.90	0.35	0.75
⛏	0.34	0.30	0.05	0.25
💎	0.09	0.05	0.00	0.05

Table 2 demonstrates that ChatGPT, despite having fewer parameters, achieves nearly identical success rates as GPT-4. This suggests that language models equipped with memory can significantly enhance planning abilities. In Minecraft-related tasks, the open-source pre-trained LLaMA2 70B exhibits a notable performance gap compared to OpenAI models, particularly in long-horizon tasks. However, by finetuning LLaMA2 with fewer parameters, its performance on Minecraft tasks improves substantially. This indicates that the open-source model lacks knowledge specific to Minecraft and requires further finetuning for successful completion of such tasks.

A.1.2 Ablation on Memory

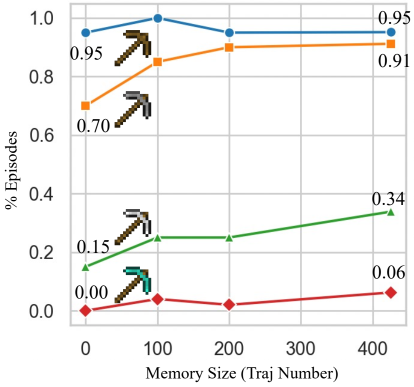


Figure 6: **Success rate by memory size for different items**. We evaluated the performance of **JARVIS**-1 at different memory sizes (representing different learning stages) by measuring the success rate (% Episodes) of completing key items on the Minecraft technology tree. As the learning progressed, we observed an improvement in completion rates for all items, with an increasing number of successful trajectories being included in memory. After 4 epochs of learning, **JARVIS**-1 had accumulated a total of 425 successful trajectories in its memory.

We also conduct ablation experiments on the multimodality memory and retrieval methods. We set **JARVIS**-1 w/o memory module as the baseline agent. We first evaluate **JARVIS**-1's performance with different memory size (representing different learning stages) as shown in Figure 6, which

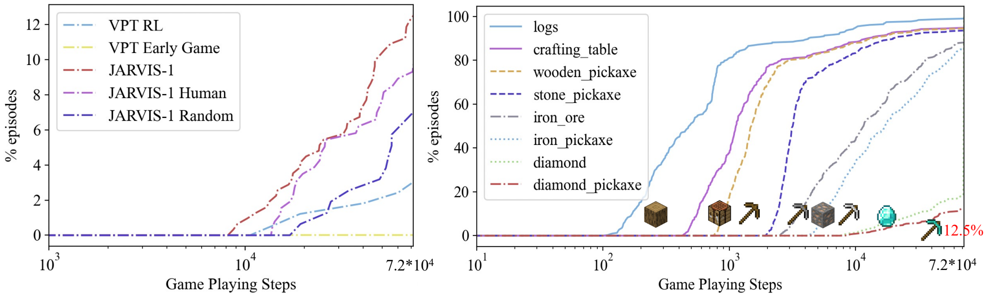


Figure 7: **(Left)** The success rate of different models in the `ObtainDiamondPickaxe` challenge over gameplay time. VPT RL is finetuned from VPT early game with reinforcement learning over 1.4 million episodes. **JARVIS**-1 agent and its varients have interacted with Minecraft with over 4 epochs on all tasks in task pool. Typically, it takes a skilled person over 20 minutes (24,000 steps) to obtain a diamond pickaxe. **(Right)** The success rate of obtaining important intermediate items during the process of synthesizing a diamond pickaxe of **JARVIS**-1. This task has been evaluated over 300 times on different seeds. These curves indicate that as the game progresses, the success rates of obtaining all intermediate items are increasing, which indicates that **JARVIS**-1 is constantly improving its skills.

demonstrates the effectiveness of self-improving within **JARVIS**-1. We further conduct the experiments on a subset of Minecraft tasks using three different retrieval methods: retrieval with instruction embedding only (T), reasoning + retrieval with text embedding (T+R), and reasoning + retrieval with multimodality embedding (M+R). Except the memory and retrieval methods, all others are kept same. The results are listed in Table 3.

Table 3: **Success rates for memory ablation on Minecraft tasks. R** is reasoning process, **T** and **M** represent retrieve with text embedding and multi-modal embedding, respectively.

Memory	Retrieval	⛏	⛏	🚪	💎	🟥
-	-	0.85	0.00	0.05	0.00	0.05
✓	T	0.85	0.05	0.10	0.00	0.10
✓	T+R	0.95	0.25	0.30	0.05	0.20
✓	M+R	0.94	0.34	0.40	0.09	0.24

The experiments show that reasoning before retrieval can effectively improve retrieval accuracy. And retrieval based on multimodal state including vision observation and symbolic information (e.g., inventory, location etc) is better than only considering the text embedding.

A.2 Long-Horizon Challenges

Most concurrent multi-task agents in Minecraft can only handle short-term tasks and struggle with long-horizon tasks like `CraftingDiamondPickaxe`. The VPT foundation model[4] is capable of accomplishing various tasks in Minecraft but lacks the ability to execute human instructions. To address this limitation, Reinforcement Learning is required to fine-tune the VPT foundation model for specific task completion. However, after fine-tuning, VPT may experience a decline in performance for other tasks while focusing on the specified task. In contrast, Steve-1 [29] has implemented goal-conditioned fine-tuning on VPT-earlygame, enabling it to follow human text instructions while maintaining multitasking capabilities. However, Steve-1 primarily focuses on low-level tasks like obtaining dirt, collecting flowers and chopping trees. When it comes to long-horizon tasks such as starting from scratch by obtaining a wooden pickaxe, Steve-1 still encounters difficulties.

DEPS[49] also utilizes LLM as a planner, but it lacks the ability to learn from experience in different tasks and apply that knowledge to new ones. Additionally, DEPS is limited in its re-planning rounds due to the LM's context constraints. The experiments reveal that DEPS has a success rate of less than 50% in generating accurate and executable plans for acquiring diamonds. The probability of DEPS successfully obtaining diamonds in the environment is approximately 0.59%. Consequently, DEPS continues to face challenges when attempting to finish long-horizon tasks within the Minecraft world.

Even human players who have mastered the distribution pattern of diamonds achieve success rates of obtaining diamonds and crafting a diamond pickaxe (which requires at least three diamonds) within 10 minutes at approximately 15% and 12%, respectively. **JARVIS**-1 performs better in the `ObtainDiamondPickaxe` challenge. Compared to the state-of-the-art model, which has undergone RL-finetuned VPT, **JARVIS**-1 has more than doubled the success rate of obtaining a diamond pickaxe (6.22% vs 2.5% within 20 minutes).

To increase the chances of obtaining diamonds, we extended the game playing time to 60 minutes (72000 game-playing steps, as shown in Figure 7). As a result, **JARVIS**-1's success rate in acquiring a diamond pickaxe improved from 6.2% to 12.5%. The graph on the right side of Figure 7 illustrates how the success rate of intermediate milestone items changes over time, indicating that **JARVIS**-1 tends to improve with longer game-playing time. We also conduct two variants of **JARVIS**-1 with different self-improving curriculum: human-written and random-generated. All three **JARVIS**-1 have collect experiences into memory with the curriculum for 4 epochs before evaluation in 60 minutes. The results show that **JARVIS**-1 with GPT-generated curriculum can finish the task within the shortest game-playing steps and achieve the best performance in 60 minutes.

In contrast, VPT's success rate barely changed when we increased the time from 20 minutes to 60 minutes (from 2.5% to 3%). This can be attributed to Minecraft's durability system where prolonged underground exploration often leads to pickaxe damage. When **JARVIS**-1's pickaxe breaks, it dynamically re-plans based on its current inventory and crafts a new one. However, VPT-RL exhibits perplexing behaviors at this stage by using inappropriate tools for mining stone or crafting unnecessary items. This comparison demonstrates that **JARVIS**-1 possesses superior generalization and planning abilities for long-horizon tasks.

Note that our method is designed to be multi-task in its nature and not finetuned through imitation learning on specific dataset or reinforcement learning.

B Related Works

B.1 Planning with LLM

There have been some methods leveraging the large language model to generate action plans for high-level tasks in embodied environments [55, 12, 34, 33, 56, 57, 16]. [21] decompose natural language commands into sequences of executable actions by text completion and semantic translation, while SayCan generates feasible plans for robots by jointly decoding an LLM weighted by skill affordances from value functions [7]. Some methods also leverage the LLM to produce the program code as plan for better executation [44, 28, 31]. However, the above methods assume that the initial plan from the LLM is correct. When there are bugs in the initial plan, it's difficult for the agent to finish the task successfully. Recent research frequently employs LLM as an interactive planner, harnessing its self-updating capabilities to enhance the plan's executability over time [49, 43, 45]. Inner Monologue [22] pilots the front of interactive planning with LLMs, which introduces the feedback (including success detection and scene description) to the planner. However, we found it could still suffer from accumulative planning errors, especially in long-horizon open-world tasks. ReAct [53] will reason about the agent state before acting, which indicates that various reasoning methods [50, 52, 51] are benefitial for planning. LLM-based planning methods often use the fixed pretrained LLM as the agent, while we focus more on life-long and continual learning for agents in open-world environments [25, 24, 47]. For better leveraging historical interaction between agent and environments, an explicit memory [41, 59] for more historical chatting has been leveraged for bigger storage of agent experiences. However, the above methods usually rely only on a text-based environment and struggle to execute plans in partial-observed visual open-world environments.

B.2 Minecraft Agents

Developing generally capable agents in Minecraft to solve open-world tasks has gained increasing interests [13, 14, 4, 9, 10, 57, 54, 59]. As an early attempt, [38] studied task generalization in a simple Minecraft environment variant. It designed a two-stage pipeline, first mastering the prerequisite skills with parameterization trick, and then learning a meta controller to execute the instructions. Moving to solve complex long-horizon tasks in Minecraft, works [38, 35, 32] explored the hierarchical architecture. In recent years, influenced by the trend of large-scale pre-training paradigms, a group of researchers have emerged, who are utilizing vast amounts of internet knowledge to train intelligent agents. [14] trained a visual-semantic alignment model, MineCLIP, using the correspondences between subtitles and video snippets available on YouTube, and used it to generate intrinsic rewards to guide policy learning. [4] utilizes a pre-trained inverse dynamics model to label actions in YouTube videos which are used to learn a foundation policy VPT through imitation learning. By bridging MineCLIP and VPT, [29] creates a performant instruction-following policy Steve-1 to solve open-world short-horizon tasks using hindsight relabeling and unCLIP tricks. However, Steve-1 can not solve complicated process-oriented tasks due to the expressive capability of its goal space. [10] learns to follow reference videos as the instruction by merely watching gameplay videos, which improves the capacity of goal space and reduces the cost of policy training. All of these methods focus on improving the smoothness and robustness of interaction between policy and environment. Inspired by the powerful language understanding and reasoning capabilities of large language models, researchers have begun to build Minecraft agents based on LLMs. [47] used LLM to guide the agent to explore the Minecraft world by acquiring diverse skills, making novel discoveries, and generating goal proposals. [59] integrated LLM with text-based knowledge and memory to equip the agent with common sense and past experiences for higher reasoning efficiency. [54] used LLM to guide the agent to explore the Minecraft world and interact with the environment with reinforcement learning control policies.

C Implementation Details

C.1 Controller

Tasks in Minecraft are usually related to `mine` and `craft` goals. The `mine` goals require the agent to collect raw materials from the environment using the appropriate tools. The `craft` goals ask the agent to use the recipe to generate new items with existing materials in inventory. The `mine` goals are achieved through STEVE-1[29] with text condition during implementation. The environment can directly executes the `craft` and `smelt` actions (`craft`/`smelt` with argument), which are same as MineDojo [14] .

C.2 Interactive Planner

JARVIS-1 relies on the Multi-modal Language Model for planning, self-checking, and self-explaining, and can accept three types of inputs: visual images, language, and symbolic information (including inventory, located position, home, current life statistics, etc.). Specifically, this is a hybrid model with language processing capabilities derived from the GPT model [39]. The visual ability comes from MineCLIP [14]. We collected approximately 1000 Minecraft text data from the internet and calculated the similarity between the current vision observation and these text data. Text above the similarity threshold will be selected into the GPT model's prompt. Symbolic information is converted into natural language text through a designed template. All modalities are ultimately captured as language and processed by the GPT model.

Different modules in **JARVIS**-1 (e.g. self-check and self-explain) are completed through MLM based on different prompts. The specific prompt design are shown below.

```
System: You are a helper agent in Minecraft. You need to generate the sequences of
     goals for a certain task in Minecraft. Just refer the history dialogue to
    give the plan consist of template. Do not explain or give any other
    instruction.

==========
User: My current inventory has nothing. I current locate in plains. How to obtain
    1 wooden_pickaxe in Minecraft step-by-step?
Assistant: The code for obtaining 1 wooden_pickaxe is as follows:
def craft_wooden_axe(initial_inventory={}):
   mine(obj = {"log":3}, tool = None) # step 1: chop down trees to mine logs
   craft(obj = {"planks":12}, materials = {"log":3}, tool = None) # step 2: craft
       12 planks from 3 logs
   craft(obj = {"stick":4}, materials = {"planks":2}, tool = None) # step 3: craft
        4 sticks from 2 planks
   craft(obj = {"crafting_table":1}, materials = {"planks":4}, tool = None) # step
        4: craft 1 crafting_table from 4 planks
   craft(obj = {"wooden_axe":1}, {"planks": 3, "stick": 2}, "crafting_table") #
       step 5: craft 1 wooden_axe from 3 planks and 2 sticks on crafting table
   return "wooden_axe"
User: [Description] I succeed in step 1, 2, 3, 4, 5.
I finish all steps and I obtain 1 wooden_axe successfully.

==========
User: My current inventory has <inventory>. <visual observation>. How to obtain 1
    stone_sword in Minecraft step-by-step?
Assistant:
```

Listing 1: Planning prompt in **JARVIS**-1

```
System: Extract the action name, action type, goal object, tool and action rank
    from the input text.

User: mine({"log":3}, null); # step 1: chop down trees to mine logs
Assistant:
name: mine_log
text condition: chop down trees to mine logs
action: mine
object_item: log
object_number: 3
tool: null
rank: 1
###

input: craft({"planks":12}, {"log":3}, null); # step 2: craft 12 planks from 3 log
Assistant:
name: craft_planks
condition: craft 12 planks from 3 log
action: craft
object_item: planks
object_number: 12
materials: {"log":3}
tool: null
rank: 2
###
```

Listing 2: Goal parsing prompt in **JARVIS**-1

```
System: Here are some actions that the agent fails to perform in Minecraft. Please
     give the explanation of action execution failure according to the current
    inventory information of the agent.

###
User:
Failed Action: mine({"iron_ore":1}, null); # step 5: dig down to mine iron blocks
Current Inventory: null
Assistant: Because mining iron_ore needs to use the tool stone_pickaxe, but my
    inventory does not have stone_pickaxe. So I need to craft stone_pickaxe first.

###
User:
Failed Action: craft({"stone_pickaxe":1}, {"cobblestone":3, "stick":2}, "
    crafting_table"); # step 1: craft 1 stone_pickaxe from 3 cobblestone and 2
    stick, on crafting_table
Current Inventory: null
Assistant: Because crafting stone_pickaxe needs to have 3 cobblestone and 2 stick
    in inventory, but my inventory does not have cobblestone and stick. So I need
    to mine cobblestone and craft stick first.

###
User:
Failed Action: craft({"stick":4}, {"planks":2}, null); # step 3: craft 4 stick
    from 2 planks first
Current Inventory: null
Assistant: Because crafting stick needs to have planks in inventory, but my
    inventory does not have planks. So I need to craft planks first.

###
```

Listing 3: Self-explaining prompt in **JARVIS**-1

```
System: You are a helper agent in Minecraft. Check the plan whether can be
    finished.

==========
User: My current inventory has nothing. I current locate in plains. My task is to
    obtain 1 wooden axe in Minecraft step-by-step. This is my plan:
def craft_wooden_axe(initial_inventory={}):
   mine(obj = {"log":3}, tool = None) # step 1: chop down trees to mine logs
   craft(obj = {"planks":12}, materials = {"log":3}, tool = None) # step 2: craft
       12 planks from 3 logs
   craft(obj = {"stick":4}, materials = {"planks":2}, tool = None) # step 3: craft
        4 sticks from 2 planks
   craft(obj = {"wooden_axe":1}, {"planks": 3, "stick": 2}, null) # step 4: craft
       1 wooden_axe from 3 planks and 2 sticks
   return "wooden_axe"

Assistant:
My current inventory is empty.
1. I want to mine 3 logs. Mining logs do not require tools. My inventory will has
    3 logs.
2. I want to craft 12 planks from 3 logs. 1 log can produce 4 planks. My inventory
     will add 12 planks, and delete 3 logs. My inventory will has 12 planks.
3. I want to craft 4 stick from 2 planks. 2 planks can produce 4 planks. My
    inventory will add 4 stick and delete 2 planks. My inventory will has 10
    planks, 4 stick.
4. I want to craft 1 wooden pickaxe from 3 planks and 2 sticks. Crafting wooden
    pickaxe requires crafting_table. But I do not have crafting_table in inventory.
     This action will failed.
Return: Step 4 will failed because of lacking of crafting_table.
```

Listing 4: Self-checking prompt in **JARVIS**-1

C.3 Memory

Our memory records every successful trajectory experience of **JARVIS**-1, including the task goals that the agent needs to execute, the actual goal sequence (plan) executed by the agent, and the state (visual observation and symbolic information returned from the environment) when the agent completes the task. In specific implementation, memory is a list where each trajectory experience is encoded as a dictionary, including the keys `task`, `state`, and `plan`.

D Environment Setting

Our Minecraft environment is a hybrid between MineRL [19] and the MCP-Reborn (github.com/Hexeption/MCP-Reborn) Minecraft modding package. Unlike the regular Minecraft game, in which the server (or the "world") always runs at 20Hz and the client runs as fast as rendering

D.1 Observation Space

The environment observations include two parts. One are simply the raw pixels from the Minecraft game that player would see. The overlays including the hotbar, health indicators, and the animation of a moving hand shown in response to the attack or "use" actions are not removed, which are same with the human-playing GUI. Another part is some auxiliary information about the current environment of the agent, including the agent's current location and current weather. These pieces of information can be obtained by human players by pressing F3. The specific observation information we include are shown in Table 4.

Table 4: The observation space we use in Minecraft.

Sources	Shape	Description
pov	(640, 360, 3)	Ego-centric RGB frames.
player_pos	(5,)	The coordinates of (x,y,z), pitch, and yaw of the agent.
location_stats	(9,)	The environmental information of the agent's current position, including `biome_id`, `sea_level`, `can_see_sky`, `is_raining` etc.
inventory	(36,)	The items in the current inventory of the agent, including the `type` and corresponding `quantity` of each item in each slot. If there is no item, it will be displayed as `air`.
equipped_items	(6,)	The current equipment of the agent, including `mainhand`, `offhand`, `chest`, `feet`, `head`, and `legs` slots. Each slot contains `type`, `damage`, and `max_damage` information.
event_info	(5,)	The events that occur in the current step of the game, including `pick_up` (picking up items), `break_item` (breaking items), `craft_item` (crafting items using a crafting table or crafting grid), `mine_block` (mining blocks by suitable tools), and `kill_entity` (killing game mobs).

Note that no high-level observations like voxels and lidar information in Minedojo [14] can be accessed by agents. During the actual inference process, controller only perceive the raw pixels and interact with the environment, which is same with VPT [4] models. The agent will access information from the environment to generate the text condition of the controller.

The environment observations are simply the raw pixels from the Minecraft game that a human would see. Unlike MineRL, we do not remove overlays like the hotbar, health indicators, and the animation of a moving hand shown in response to the attack or "use" actions. The field of view is 70 degrees, which corresponds to the Minecraft default. GUI scale (a parameter controlling the size of the in-game GUI) is set to 2, and brightness is set to 2 (which is not a Minecraft default, but is very frequently used in online videos). The rendering resolution is 640x360, which is downsampled to 128x128 before being input to the models. We empirically found 128x128 to be the smallest resolution for which in-game GUI elements are still discernible, and then chose that to minimize compute costs. Whenever an in-game GUI is open, we additionally render an image of a mouse cursor at the appropriate mouse position to match what a human player's operating system does (Fig. 12).

D.2 Action Space

We design a hybrid action space. Some are directly available to human players, including keypresses, mouse movements, and clicks, which are similar to VPT [4]. The keypresses and clicks are binary functional actions, including `forward`, `jump`, `use` and `attack` etc. In addition to the binary (on/off) keypress actions, our action space also includes mouse movements, as with human gameplay. As with human gameplay, when in-game GUIs are not open, mouse X and Y actions change the agent's yaw and pitch, respectively. When a GUI is open, camera actions move the mouse cursor. Mouse movements are relative (i.e. they move the mouse or camera relative to the current position, and thus their effect depends on the current position). In Minecraft, interacting with the inventory requires precise mouse movements for tasks like crafting and smelting, while mining and navigating the world can be accomplished with broader mouse actions. To be enable to achieve both the same action space, we abstract the `craft` and `smelt` action with GUI into functional binary actions, which are same as MineDojo [14]. The detailed action space are described in Table 5.

Table 5: The action space we use in Minecraft.

Index	Action	Human Action	Description
1	Forward	key W	Move forward.
2	Back	key S	Move backward.
3	Left	key A	Strafe left.
4	Right	key D	Strafe right.
5	Jump	key Space	Jump. When swimming, keeps the player afloat.
6	Sneak	key left Shift	Slowly move in the current direction of movement.
7	Sprint	key left Ctrl	Move quickly in the direction of current motion.
8	Attack	left Button	Destroy blocks (hold down); Attack entity (click once).
9	Use	right Button	Interact with the block that the player is currently looking at.
10	hotbar.[1-9]	keys 1 - 9	Selects the appropriate hotbar item.
11	Yaw	move Mouse X	Turning; aiming; camera movement.Ranging from -180 to +180.
12	Pitch	move Mouse Y	Turning; aiming; camera movement.Ranging from -180 to +180.
13	Equip	-	Equip the item in main hand from inventory.
14	Craft	-	Execute a crafting recipe to obtain new item.
15	Smelt	-	Execute a smelting recipe to obtain new item.

D.3 Rules

We choose to conduct the test in survival mode of Minecraft 1.16.5. For each environment reset, we have added the following rules:

- `/difficulty peaceful`: Set the difficulty of the environment to peaceful mode.
- `/gamerule doDaylightCycle false`: Set the environment to daytime forever.
- `/gamerule keepInventory true`: Set agent to not drop items upon death. We have added a time limit for each task, within which if the player dies, they will respawn at the spawn point and retain their previous inventory contents.
- `/effect give @a night_vision 99999 250 true`: In order to facilitate the display of agent behavior, we have added night vision effects to the agent.

E Results and Details of 200+ tasks in Minecraft Universe Benchmark

We first list the 11 task groups encompassing over 200 Minecraft tasks in Table 6.

Table 6: Characteristics of 11 task groups encompassing over 200 Minecraft tasks.

Group	Task Num.	Max. Steps	Initial Inventory	Biome	Language Instruction
Wood	34	12k	null	Plains/Forest	Pick up a `wooden_pickaxe`.
Wood-Variants	43	12k	null	Savanna/Jungle/Taiga	Pick up a `acacia_boat`.
Stone	10	12k	iron_axe	Plains/Forest	Craft a `furnace` given an iron axe.
Iron	22	12k	iron_axe	Plains/Forest	Smelt and craft an `iron_door` given an iron axe.
Gold	9	36k	iron_axe	Plains/Forest	Smelt and craft an `golden_axe` given an iron axe.
Diamond	7	36k	iron_axe	Plains/Forest	Dig down to mine diamond and craft `diamond_pickaxe`.
Redstone	7	36k	iron_axe	Plains/Forest	Mine redstone and make `dropper` given an iron axe.
Blocks	15	12-36k	iron_axe	Plains/Forest	Dig down to mine `lapis_lazuli` block.
Armor	17	12-36k	iron_axe	Plains/Forest	Craft `diamond_boots` given an iron axe and equip it.
Decoration	17	12k	iron_axe	Flower Forest	Obtain the `bed` and dye it `red`.
Food	9	12k	iron_axe	Plains	Kill sheep to obtain `mutton` and cook it.

We then list the evaluation task set below with details including task name, maximum steps, initial inventory, biome, and language instructions. We also show the evaluation times across different seeds and successful episode rates. Note that all tasks are evaluated in Minecraft 1.16.5 Survival Mode.

Table 7: The results of our agent on various tasks in the `Wood` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
wooden_shovel	12000	null	Plains/Forest	0.9028	72	Pick up a wooden_shovel given nothing.
wooden_pickaxe	12000	null	Plains/Forest	0.9516	62	Pick up a wooden_pickaxe given nothing.
wooden_axe	12000	null	Plains/Forest	0.8909	55	Pick up a wooden_axe given nothing.
wooden_hoe	12000	null	Plains/Forest	0.9318	44	Pick up a wooden_hoe given nothing.
stick	12000	null	Plains/Forest	1	86	Pick up a stick given nothing.
wooden_sword	12000	null	Plains/Forest	0.9242	66	Pick up a wooden_sword given nothing.
composter	12000	null	Plains/Forest	0.7872	47	Pick up a composter given nothing.
barrel	12000	null	Plains/Forest	0.7544	57	Pick up a barrel given nothing.
crafting_table	12000	null	Plains/Forest	0.9706	68	Pick up a crafting_table given nothing.
chest	12000	null	Plains/Forest	0.9155	71	Pick up a chest given nothing.
ladder	12000	null	Plains/Forest	0.9737	76	Pick up a ladder given nothing.
bowl	12000	null	Plains/Forest	0.9149	47	Pick up a bowl given nothing.
oak_wood	12000	null	Forest	0.9868	76	Pick up a oak_wood in Forest.
oak_slab	12000	null	Forest	0.9506	81	Pick up a oak_slab in Forest.
oak_planks	12000	null	Forest	0.9659	88	Pick up a oak_planks in Forest.
oak_log	12000	null	Forest	1	65	Pick up a oak_log in Forest.
oak_button	12000	null	Forest	0.9153	59	Pick up a oak_button in Forest.
oak_door	12000	null	Forest	0.8732	71	Pick up a oak_door in Forest.
oak_fence	12000	null	Forest	0.8	60	Pick up a oak_fence in Forest.
oak_fence_gate	12000	null	Forest	0.9322	59	Pick up a oak_fence_gate in Forest.
oak_trapdoor	12000	null	Forest	0.8861	79	Pick up a oak_trapdoor in Forest.
oak_boat	12000	null	Forest	0.9074	54	Pick up a oak_boat in Forest.
oak_sign	12000	null	Forest	0	0	Pick up a oak_sign in Forest.
birch_wood	12000	null	Forest	0.9474	57	Pick up a birch_wood in Forest.
birch_slab	12000	null	Forest	0.9231	65	Pick up a birch_slab in Forest.
birch_planks	12000	null	Forest	0.9714	70	Pick up a birch_planks in Forest.
birch_log	12000	null	Forest	0.9833	60	Pick up a birch_log in Forest.
birch_button	12000	null	Forest	0.9245	53	Pick up a birch_button in Forest.
birch_door	12000	null	Forest	0.8431	51	Pick up a birch_door in Forest.
birch_fence	12000	null	Forest	0.8	30	Pick up a birch_fence in Forest.
birch_fence_gate	12000	null	Forest	0.9355	62	Pick up a birch_fence_gate in Forest.
birch_trapdoor	12000	null	Forest	0.9524	63	Pick up a birch_trapdoor in Forest.
birch_boat	12000	null	Forest	0.8906	64	Pick up a birch_boat in Forest.
birch_sign	12000	null	Forest	0.9	60	Pick up a birch_sign in Forest.

Table 8: The results of our agent on various tasks in the `Stone` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
stone_shovel	12000	iron_axe	Plains/Forest	0.8514	74	Craft a stone_shovel given an iron_axe.
stone_pickaxe	12000	iron_axe	Plains/Forest	0.9118	68	Craft a stone_pickaxe given an iron_axe.
stone_axe	12000	iron_axe	Plains/Forest	0.9123	57	Craft a stone_axe given an iron_axe.
stone_hoe	12000	iron_axe	Plains/Forest	0.9459	74	Craft a stone_hoe given an iron_axe.
stone	12000	iron_axe	Plains/Forest	0.8413	63	Craft a stone given an iron_axe.
charcoal	12000	iron_axe	Plains/Forest	0.8947	76	Craft a charcoal given an iron_axe.
smoker	12000	iron_axe	Plains/Forest	0.7867	75	Craft a smoker given an iron_axe.
stone_sword	12000	iron_axe	Plains/Forest	0.8831	77	Craft a stone_sword given an iron_axe.
furnace	12000	iron_axe	Plains/Forest	0.942	69	Craft a furnace given an iron_axe.
torch	12000	iron_axe	Plains/Forest	0.9	30	Craft a torch given an iron_axe.

Table 9: The results of our agent on various tasks in the `Iron` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
iron_axe	12000	null	Plains/Forest	0.3333	60	Smelt and craft an iron_axe.
iron_pickaxe	12000	iron_axe	Plains/Forest	0.3382	68	Smelt and craft an iron_pickaxe.
iron_shovel	12000	iron_axe	Plains/Forest	0.338	71	Smelt and craft an iron_shovel.
iron_sword	12000	iron_axe	Plains/Forest	0.3288	73	Smelt and craft an iron_sword.
iron_trapdoor	12000	iron_axe	Plains/Forest	0.3151	73	Smelt and craft an iron_trapdoor.
iron_door	12000	iron_axe	Plains/Forest	0.2836	67	Smelt and craft an iron_door.
iron_ingot	12000	iron_axe	Plains/Forest	0.5479	73	Smelt and craft an iron_ingot.
bucket	12000	iron_axe	Plains/Forest	0.381	42	Smelt and craft a bucket.
rail	12000	iron_axe	Plains/Forest	0.3226	62	Smelt and craft a rail.
minecart	12000	iron_axe	Plains/Forest	0.2833	60	Smelt and craft a minecart.
smithing_table	12000	iron_axe	Plains/Forest	0.3611	72	Smelt and craft a smithing_table.
tripwire_hook	12000	iron_axe	Plains/Forest	0.45	60	Smelt and craft a tripwire_hook.
chain	12000	iron_axe	Plains/Forest	0.3729	59	Smelt and craft a chain.
iron_bars	12000	iron_axe	Plains/Forest	0.3208	53	Smelt and craft an iron_bars.
hopper	12000	iron_axe	Plains/Forest	0.3077	65	Smelt and craft a hopper.
iron_nugget	12000	iron_axe	Plains/Forest	0.3582	67	Smelt and craft an iron_nugget.
heavy_weighted_pressure_plate	12000	iron_axe	Plains/Forest	0.358	81	Smelt and craft a heavy_weighted_pressure_plate.
blast_furnace	12000	iron_axe	Plains/Forest	0.5	60	Smelt and craft a blast_furnace.
shears	12000	iron_axe	Plains/Forest	0.25	64	Smelt and craft a shears.
stonecutter	12000	iron_axe	Plains/Forest	0.5	60	Smelt and craft a stonecutter.
iron_hoe	12000	iron_axe	Plains/Forest	0.3214	56	Smelt and craft an iron_hoe.
crossbow	12000	iron_axe	Plains/Forest	0.047	63	Smelt and craft a crossbow.

Table 10: The results of our agent on various tasks in the `Gold` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
golden_pickaxe	36000	iron_axe	Plains/Forest	0.0526	77	Smelt and craft a golden_pickaxe.
golden_shovel	36000	iron_axe	Plains/Forest	0.0822	73	Smelt and craft a golden_shovel.
golden_sword	36000	iron_axe	Plains/Forest	0.0476	85	Smelt and craft a golden_sword.
golden_hoe	36000	iron_axe	Plains/Forest	0.058	69	Smelt and craft a golden_hoe.
golden_axe	36000	iron_axe	Plains/Forest	0.0469	64	Smelt and craft a golden_axe.
golden_apple	36000	iron_axe	Plains/Forest	0.02	76	Smelt and craft a golden_apple.
clock	36000	iron_axe	Plains/Forest	0.02	77	Smelt and craft a clock.
gold_nugget	36000	iron_axe	Plains/Forest	0.1444	91	Smelt and craft a gold_nugget.
gold_ingot	36000	iron_axe	Plains/Forest	0.1449	70	Smelt and craft a gold_ingot.

Table 11: The results of our agent on various tasks in the `Diamond` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
diamond_pickaxe	36000	iron_axe	Plains/Forest	0.0622	692	Dig down to mine diamond and craft diamond_pickaxe.
diamond_shovel	36000	iron_axe	Plains/Forest	0.1136	88	Dig down to mine diamond and craft diamond_shovel.
diamond_sword	36000	iron_axe	Plains/Forest	0.1134	97	Dig down to mine diamond and craft diamond_sword.
diamond_hoe	36000	iron_axe	Plains/Forest	0.0441	68	Dig down to mine diamond and craft diamond_hoe.
diamond_axe	36000	iron_axe	Plains/Forest	0.0986	71	Dig down to mine diamond and craft diamond_axe.
diamond	36000	iron_axe	Plains/Forest	0.092	728	Dig down to mine diamond and craft diamond.
jukebox	36000	iron_axe	Plains/Forest	0.1053	79	Dig down to mine diamond and craft jukebox.

Table 12: The results of our agent on various tasks in the `Redstone` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
piston	36000	iron_axe	Plains/Forest	0.1772	79	Mine redstone and make piston.
redstone_torch	36000	iron_axe	Plains/Forest	0.2584	89	Mine redstone and make redstone_torch.
redstone_block	36000	iron_axe	Plains/Forest	0.2469	81	Mine redstone and make redstone_block.
activator_rail	36000	iron_axe	Plains/Forest	0.0159	63	Mine redstone and make activator_rail.
compass	36000	iron_axe	Plains/Forest	0.0759	79	Mine redstone and make compass.
dropper	36000	iron_axe	Plains/Forest	0.2278	79	Mine redstone and make dropper.
note_block	36000	iron_axe	Plains/Forest	0.2239	67	Mine redstone and make note_block.

Table 13: The results of our agent on various tasks in the `Blocks` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
diorite	12000	iron_axe	Plains/Forest	0.9	30	Dig down to mine diorite block.
andesite	12000	iron_axe	Plains/Forest	0.9667	30	Dig down to mine andesite block.
granite	12000	iron_axe	Plains/Forest	0.8667	30	Dig down to mine granite block.
coal	12000	iron_axe	Plains/Forest	0.6667	30	Dig down to mine coal block.
lapis_lazuli	12000	iron_axe	Plains/Forest	0.8667	30	Dig down to mine lapis_lazuli block.
iron_ore	12000	iron_axe	Plains/Forest	0.5667	30	Dig down to mine iron_ore block.
gold_ore	36000	iron_axe	Plains/Forest	0.27	30	Dig down to mine gold_ore block.
cobblestone	12000	iron_axe	Plains/Forest	0.9667	30	Dig down to mine cobblestone block.
gravel	12000	iron_axe	Plains/Forest	0.9667	30	Dig down to mine gravel block.
oak_log	12000	iron_axe	Plains/Forest	0.9667	30	Chop down tree and mine oak_log block.
birch_log	12000	iron_axe	Plains/Forest	0.8718	39	Chop down tree and mine birch_log block.
acacia_log	12000	iron_axe	Plains/Forest	0.5	30	Chop down tree and mine acacia_log block.
jungle_log	12000	iron_axe	Plains/Forest	0.9333	30	Chop down tree and mine jungle_log block.
dark_oak_log	12000	iron_axe	Plains/Forest	0.9	30	Chop down tree and mine dark_oak_log block.
spruce_log	12000	iron_axe	Plains/Forest	0.9333	30	Chop down tree and mine spruce_log block.

Table 14: The results of our agent on various tasks in the `Armor` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
shield	12000	iron_axe	Plains/Forest	0.3939	66	Craft shield and equip it.
leather_helmet	12000	iron_axe	Plains/Forest	0.0508	59	Craft leather_helmet and equip it.
leather_chestplate	12000	iron_axe	Plains/Forest	0.0312	32	Craft leather_chestplate and equip it.
leather_leggings	12000	iron_axe	Plains/Forest	0.0588	34	Craft leather_leggings and equip it.
leather_boots	12000	iron_axe	Plains/Forest	0.087	23	Craft leather_boots and equip it.
iron_chestplate	12000	iron_axe	Plains/Forest	0.3333	30	Craft iron_chestplate and equip it.
iron_boots	12000	iron_axe	Plains/Forest	0.3667	30	Craft iron_boots and equip it.
iron_leggings	12000	iron_axe	Plains/Forest	0.3788	66	Craft iron_leggings and equip it.
iron_helmet	12000	iron_axe	Plains/Forest	0.303	33	Craft iron_helmet and equip it.
diamond_helmet	36000	iron_axe	Plains/Forest	0.0429	70	Craft diamond_helmet and equip it.
diamond_chestplate	36000	iron_axe	Plains/Forest	0.0149	68	Craft diamond_chestplate and equip it.
diamond_leggings	36000	iron_axe	Plains/Forest	0.02	73	Craft diamond_leggings and equip it.
diamond_boots	36000	iron_axe	Plains/Forest	0.0533	75	Craft diamond_boots and equip it.
golden_helmet	36000	iron_axe	Plains/Forest	0.0533	75	Craft golden_helmet and equip it.
golden_chestplate	36000	iron_axe	Plains/Forest	0.02	78	Craft golden_chestplate and equip it.
golden_leggings	36000	iron_axe	Plains/Forest	0.0159	89	Craft golden_leggings and equip it.
golden_boots	36000	iron_axe	Plains/Forest	0.0617	81	Craft golden_boots and equip it.

Table 15: The results of our agent on various tasks in the `Decoration` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
yellow_dye	12000	iron_axe	Flower Forest	0.2333	30	Obtain the yellow_dye.
red_dye	12000	iron_axe	Flower Forest	0.6364	33	Obtain the red_dye.
light_gray_dye	12000	iron_axe	Flower Forest	0.6667	27	Obtain the light_gray_dye.
pink_dye	12000	iron_axe	Flower Forest	0.6667	39	Obtain the pink_dye.
orange_dye	12000	iron_axe	Flower Forest	0.4857	35	Obtain the orange_dye.
white_dye	12000	iron_axe	Flower Forest	0.1471	34	Obtain the white_dye.
white_bed	12000	iron_axe	Flower Forest	0.5	36	Obtain the white_bed.
item_frame	12000	iron_axe	Flower Forest	0.2143	28	Obtain the item_frame.
painting	12000	iron_axe	Flower Forest	0.5484	31	Obtain the painting.
white_wool	12000	iron_axe	Flower Forest	0.8235	34	Obtain the white_wool.
white_carpet	12000	iron_axe	Flower Forest	0.6857	35	Obtain the white_carpet.
white_banner	12000	iron_axe	Flower Forest	0.0968	31	Obtain the white_banner.
yellow_wool	12000	iron_axe	Flower Forest	0.0625	32	Obtain the yellow_wool.
red_wool	12000	iron_axe	Flower Forest	0.6571	35	Obtain the red_wool.
light_gray_wool	12000	iron_axe	Flower Forest	0.6098	41	Obtain the light_gray_wool.
pink_wool	12000	iron_axe	Flower Forest	0.4	25	Obtain the pink_wool.
orange_wool	12000	iron_axe	Flower Forest	0.5	36	Obtain the orange_wool.

Table 16: The results of our agent on various tasks in the `Food` group.

Task	Max. Steps	Initial Inventory	Biome	Success Rate	Eval Times	Language Instruction
apple	12000	iron_axe	Plains	0.5	30	Chop down tree to obtain apple.
cooked_chicken	12000	iron_axe	Plains	0.3562	73	Kill chicken to obtain chicken and cook it.
cooked_mutton	12000	iron_axe	Plains	0.4355	62	Kill sheep to obtain mutton and cook it.
cooked_porkchop	12000	iron_axe	Plains	0.3968	63	Kill pig to obtain porkchop and cook it.
cooked_beef	12000	iron_axe	Plains	0.2857	63	Kill cow to obtain beef and cook it.
chicken	12000	iron_axe	Plains	0.5667	30	Kill chicken to obtain chicken.
beef	12000	iron_axe	Plains	0.6333	30	Kill cow to obtain beef.
mutton	12000	iron_axe	Plains	0.5667	30	Kill sheep to obtain mutton.
porkchop	12000	iron_axe	Plains	0.4667	30	Kill pig to obtain porkchop.