MCU: A TASK-CENTRIC FRAMEWORK FOR OPEN-ENDED AGENT EVALUATION IN MINECRAFT

Anonymous authors
Paper under double-blind review

ABSTRACT

To pursue the goal of creating an open-ended agent in Minecraft, an open-ended game environment with unlimited possibilities, this paper introduces a novel task-centric framework named **MCU** for Minecraft agent evaluation.[1] The MCU framework leverages the concept of *atom tasks* as fundamental building blocks, enabling the generation of diverse or evan arbitrary tasks. Within the MCU framework, each task is measured with 6 distinct difficulty scores (time consumption, operational effort, planning complexity, intricacy, creativity, novelty). These scores offer a multi-dimensional assessment of a task from different angles, and thus can reveal an agent's capability on specific facets. The difficulty scores also serve as the feature of each task, which creates a meaningful task space and unveils the relationship between tasks. For practical evaluation of Minecraft agents employing the MCU framework, we maintain two custom benchmarks, comprising tasks meticulously designed to evaluate the agents' proficiency in *high-level planning* and *low-level control*, respectively. We show that MCU has the high expressivity to cover all tasks used in recent literature on Minecraft agent, and underscores the need for advancements in areas such as creativity, precise control, and out-of-distribution generalization under the goal of open-ended Minecraft agent development.

1 INTRODUCTION

In artificial intelligence (AI), an *agent* is a computer program or system that is designed to perceive its environment, make decisions and take actions to solve a specific task or set of tasks. On top of that, an *open-ended* agent is an agent that possesses the capabilities to solve *arbitrary* tasks that are feasible and can be solved by humans. The open-ended agent has crucial difference with task-specific agent or multi-task agent, which can only handle a limited spectrum of tasks.

Building an open-ended agent has long been a holy grail for AI researchers. Minecraft, as the world's best selling game, empowers players to survive, explore, and create within a digital realm remarkably akin to human world. With its immense popularity, expansive environment, and limitless possibilities, Minecraft has risen as an unmatched platform for the developtment of open-ended agents. Recently, we have witnessed the remarkable success of numerous Minecraft agents powered by cutting-edge techniques such as hierarchical Reinforcement Learning (Lin et al., 2021; Mao et al., 2022), large-scale pre-training (Baker et al., 2022; Fan et al., 2022; Lifshitz et al., 2023), and Large Language Models (LLMs) (Zhu et al., 2023; Wang et al., 2023a;b; Lifshitz et al., 2023; Yuan et al., 2023; Nottingham et al., 2023).

However, the pursuit of building a open-ended agent for Minecraft still remains a formidable challenge. The current Minecraft agents are far from solving arbitrary tasks. To approach this holy grail, it is imperative that we analyze the current gap between the existing agents and the ultimate open-ended agent, and then improve the agents accordingly. To be specific, we have to assess the areas where they exhibit shortcomings, e.g., whether they fall short in environment perception, subgoal planning, or precise control.

We find traditional benchmarks are not suitable for this analysis purpose. Traditional benchmarks (Kanervisto et al., 2022; Zhu et al., 2023) mainly focus on "obtain XXX in Minecraft

[1] MCU is the acronym of *Minecraft Universe*, which has nothing to do with *Marvel Cinematic Universe*.

from scratch" (e.g., log, iron, diamond). Minedojo (Fan et al., 2022) proposed the categorization of creative tasks and programmatic tasks based on the ease of evaluation with pre-defined game information. However, it is hard to scale the number of tasks into infinity, and it is also hard to know the essential capability of the tested agents via these evaluation paradigms. On top of that, this paper proposes a task-centric framework to evaluate Minecraft agents, namely **MCU** (**M**ine**C**raft **U**niverse), which boasts the following important merits:

- MCU is the first evaluation framework targeted at **arbitrary tasks**. MCU has a task generation approach that can create infinite tasks via operation of 3000+ collected **Atom Tasks**. Each task proposed in recent works can be covered by MCU. On top of that, MCU can serve as a unified evaluation protocol for Minecraft agent research.
- MCU categorizes the difficulties of tasks into six different dimensions, which enable us to assess the agents from different angles. Empirical study shows certain human-defined task categories (e.g., navigation tasks, building tasks) exhibit distinct task difficulty distributions. Composing a feature vector for each task using its difficulty scores, we can construct a meaningful task space that can show semantic relationships between tasks.
- For evaluation convenience, we maintain two benchmarks created by MCU, each characterized by varying difficulty distributions: (i) **HorizonForge** contains long-horizon tasks necessitating efficient planning and subgoal selection. (ii) **SkillForge** is composed of short-horizon tasks that demand a wide spectrum of skills. MCU also provides filtering options for other researchers to generate new benchmarks by choosing certain tasks with their focus on some particular facets of agent capability.

2 Atom Task, Molecule task, and Benchmark

2.1 Atom Tasks

To facilitate a comprehensive set of tasks for evaluating Minecraft agents, we introduce *atom task*, a basic building block for constructing complex tasks via combination and imposing constraints. The idea of using atom task to test agents' capabilities is motivated by *unit testing* (Olan, 2003) in software testing. Just as unit testing dissects a desired outcome into individual units to verify that minimal functionality operates correctly, we adopt a similar approach. We follow prior works (Wang et al., 2023b; Zhu et al., 2023) to assume that any complex task can be decomposed into a finite sequence of simple tasks, each serving a distinct function – these are our atom tasks. Then we design the basic data structure of atom tasks as follows (see Figure 3 (left) for the data structure of an atom task "Craft iron_sword" as an example).

Initialization In contrast to conventional evaluation methods, which necessitate agents to begin entirely from scratch, devoid of any resources, and at random spawn points, atom tasks employ a different approach. Atom tasks kickstart agents with essential prerequisite conditions for the goal, including their location (e.g., specific biome), summoned entities (e.g., creatures), and access to essential resources (e.g., building materials). For instance, when subjected to the atom task "Craft an iron_sword", the agent is equipped with 2 iron ingots, 1 stick, and a crafting table. This initialization method empowers the agent to focus exclusively on the core functionality demanded by the atom task, untangled from the complexities of other tasks.

Difficulties In our endeavor to elucidate the characteristics of each atom task, we assess them across six distinct dimensions of difficulty: time consumption, operational effort, planning complexity, intricacy, creativity, and novelty. While each atom task boasts its own unique functionality, these common difficulty metrics provide us with valuable insights into the interplay between different atom tasks. For instance, navigation tasks often demand substantial time investments but involve relatively lower intricacy, whereas combat tasks typically require significant operational effort. A comprehensive explanation of how we quantify each dimension of difficulty can be found in Section 3.

Pre-tasks Within the context of an atom task, the concept of "pre-tasks" delineates a collection of essential atom tasks that must be completed before the focal atom task can be achieved. These pre-tasks are indicative of the planning complexity inherent to the atom task. For instance, when

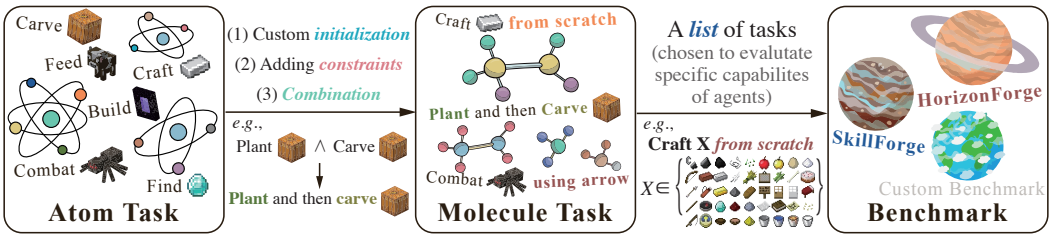

Figure 1: The taxonomy of atom task, molecule task, and benchmark in MCU framework. **Atom tasks** are the basic components of MCU, which can test a minimal ability of Minecraft agent by resolving prerequisite dependencies for the task. By initializing the agents with custom conditions (craft iron ingot *from scratch*), adding constraints to the task (e.g., combat a spider *using arrow*), or combining different tasks (e.g., plant pumpkin *and then* carve pumpkin), atom tasks can be exploited to generate infinite **molecule tasks**. By collecting a list of tasks, custom **benchmarks** can be created to test particular facets of agents' capabilities. They serve as targeted evaluation tools, helping us gain insights into the agent's performance in particular domains or scenarios.

considering the atom task "craft a wooden sword", its pre-tasks encompass tasks like "find oak trees", "chop oak trees", "craft oak planks", "craft sticks", and "craft a crafting table". It is important to note that pre-tasks need not follow a strictly linear sequence; for example, "find oak trees" could be substituted with "find birch trees", demonstrating the existence of multiple viable paths to the same task. Together, the pre-tasks of an atom task construct a dependency graph (DG) wherein the atom task serves as the highest topological order node. Refer to Figure 2 for an illustration of the dependency graph for the "craft bed" task.

Metrics To comprehensively evaluate the agent's performance, it is imperative to establish a set of metrics tailored to each individual atom task. The flexibility of metric design allows us to create diverse evaluation criteria, each uniquely suited to the nature of the task at hand. For instance, beyond the commonly used success-rate metric, we have the option to quantify vertical axis displacement for tasks like "dig down", assess task-solving efficiency through time consumption analysis, or even employ model-based metrics such as MineCLIP (Fan et al., 2022) and human rating metrics. Designing better metrics for each task is also set as future work for MCU.

Description In addition to the attributes mentioned earlier, a precise task description is essential for defining the task's objectives. This description serves as the task's identifier and can take various forms, such as a simple phrase (e.g., "Craft an iron sword"), a complex sentence, or even human demonstrations. It's important to note that the description isn't limited to textual information alone, as a single modality may not adequately convey the intricacies of human intentions. Consequently, we also allow the use of gameplay videos and trajectories as means of describing tasks. These descriptions offer a clear and comprehensive definition of the task's goals and potential steps for task completion.

Instantiation. Based on the attributes listed above, we curate a comprehensive list of 3000+ atom tasks (which is continually updated). The curation process draws from multiple sources, such as distillation of the Minedojo task list, extraction from Minecraft wiki, and innovative brainstorming. We also curate a small set of game-play trajectories for each task as description for the task, which can be leveraged for agent learning or evaluation. The list is also carefully cleaned to fulfill the desiderata raised in section 5. In the next section, we show how to generate complex tasks and customized benchmarks.

2.2 Molecule Tasks

One of the key merits for atom task is the potential to generate infinite tasks. In this section, we describe how to compose customized tasks, namely *molecule tasks* by leveraging atom tasks. We define three operations on atom tasks to derive molecule tasks.

Custom Initialization The initialization of atom tasks provides the agents with the necessary conditions (as noted by Section 2.1), while we can tailor the initialization to create a broader range

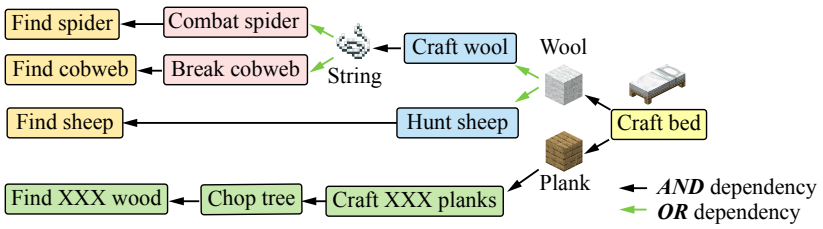

Figure 2: The illustration of an atom task "Craft bed" and its pre-tasks. The bed is crafted via planks and wools, and the wools can be obtained by hunting sheep or crafting with strings. And the strings can be obtained from combating spider or breaking cobweb. The planks are crafted with chopped woods (There are many types of wood in Minecraft, and we use "XXX" to denote that arbitrary types of woods are allowed). An agent has to plan and select the efficient task routes to prepare necessary materials for the target task. Items like string, wool, and plank are also introduced as nodes in the graph to show the complete dependencies. The black arrows represent the AND dependency, e.g., to craft bed the agent needs both the wools and the planks. And the green arrows denote the OR dependency, e.g., the string can be obtained by combating spider or breaking cobweb.

of molecule tasks. As an example, we can double the number of tasks by employing a **naive initialization** approach, wherein we initiate the tested agent with an empty inventory and a random spawn point. This necessitates the agent to plan and solve the pre-tasks associated with the given atom task.

Combination Similar to logical operators, we can use the operator AND and OR to combine different atom tasks into a more complex molecule task. Note that the molecule task generated by naive initialization can be expressed by the combination of pre-tasks of the atom task and the given atom task itself. For example, the molecule task with naive initialization for "craft oak planks" can be expressed as:

$$\text{CraftOakPlanks (n)} = \text{FindOakTrees (a)} \wedge \text{ChopOakTrees (a)} \wedge \text{CraftOakPlanks (a)}, \qquad (1)$$

where "craft oak planks", "find oak trees", "chop oak trees" are denoted by CraftOakPlanks, FindOakTrees, ChopOakTrees, respectively. "(n)" represents molecule task with naive initialization of the corresponding atom task, and "(a)" represents the atom task. [2]

Constraints We also allow imposing constraints to an atom task. The constraints can be about when (e.g., at night, at morning, in the first day after spawning), where (e.g., in a cave, in dessert, in the Nether), or how (e.g., bare-hand, with an iron sword, using arrows). The difficulty of some tasks may be sensitive to the constraints, such as the the difficulty of combat tasks, while some are not. The constraints may push the agents to solve tasks with creativity or generalization ability.

Remarks. The data structure defined in Section 2.1 also applies to molecule tasks. The composition of molecule tasks is flexible: We can not only construct a molecule task by combining atom tasks, but can also define the five attributes introduced in Section 2.1 from scratch in a top-down approach, especially when the desired task is complicated and involves many atom tasks. A feasible top-down approach to define the data structure of a molecule task can be adopted by first identifying its atom tasks (the atom tasks that combine to the molecule task). Then the *initialization* of this molecule task can be derived by combining the initialization of all of its atom tasks. For *difficulties*, we can follow the methods in section 3 to compute difficulty scores for each dimension. For *pre-tasks* of the molecule task, we can simply choose the union set of all the pre-tasks of its atom tasks. For *metrics*, we can borrow the metrics of its atom tasks, or design customized metrics for a molecule task. For *description*, we can easily derive the text description from atom tasks, and collect exemplar trajectories of solving the task.

[2]Some atom tasks may conflict in initialization, and thus they are illegal to be combined. E.g., "climb the mountain" and "boating" have different spawn points. We only consider feasible molecule tasks in combination.

2.3 BENCHMARKS

With the molecule task generation approach, MCU can generate infinite benchmarks with arbitrary task lists. To provide unified benchmarks for researchers to test their agents and improve the fairness of comparing different baselines on the same settings, we maintain two benchmarks created by MCU.

The two benchmarks focus on two important facets: *high-level planning* and *low-level control*, which are well acknowledged by communities as the two key challenges in Minecraft agent development (Cai et al., 2023; Lifshitz et al., 2023). Given a task, high-level planning aims to decompose the difficult task into executable sub-tasks, and low-level control aims to solve the sub-tasks via learned policies. We design the benchmark for the evaluation of high-level planning named **HorizonForge**, which consists of complex tasks with diverse horizons. And We devise the benchmark for the evaluation of *low-level control* named **SkillForge**, which focuses on assessing specific skills that a Minecraft agent possess.

HorizonForge The HorizonForge benchmark contains 976 molecule tasks, and each task is the naive initialization version (see Section 2.2 for details) of an atom task "Craft X" ("X" denotes all the items in Minecraft that can be crafted). To solve a crafting task from scratch, the agent has to collect all the necessary materials, which usually requires resolution of complex dependency connections between each other. The agent has to plan the feasible routes, decide the sub-goal sequence, and solve each sub-tasks accordingly. Therefore, the crafting tasks (with naive initialization) is suitable for evaluating the capability to plan an efficient sub-goal route. We also devise a test environment for community to test the pure plan capability based on text instructions, which is denoted as **HorizonForge-text**. In this environment, the agent only needs to output the atom task it plans to solve without conducting low-level control. The details of HorizonForge-text will be presented in Appendix D.

SkillForge The SkillForge benchmark contains 35 atom tasks that can be roughly categorized into six functional groups: 5 combat tasks, 5 crafting tasks, 5 building tasks, 10 exploitation tasks,[3] 5 mining tasks, and 5 navigation tasks. There are also 8 molecule tasks in SkillForge, result in a total task number of 43. All the tasks in SkillForge are typical skills in Minecraft and have diverse difficulty distributions, which can test the low-level control ability of Minecraft agents. The `Mining` section requires the agent to be aware of its target item, and exploit the available tools to collect the target items. The `Navigation` category tests the navigation proficiency, understanding of diverse environments, and intrinsic motivation for exploration. The `Crafting` category evaluates the knowledge of Minecraft crafting mechanics, which requires the agent to convert materials into functional items by harnessing the pre-defined recipes in Minecraft. The `Exploitation` category investigates the knowledge of exploitation of specific tools in various scenarios. The `Combat` category assesses the ability to hunting and fighting in order to fulfill basic survival needs. The `Building` category evaluates the aptitude in structural reasoning, spatial organization, and the ability to interact with and manipulate the environment to create specific architectures. The full task list can be found in Figure 3.

We also design a filter option for research community to design their benchmarks by selecting certain tasks from our collected tasks based on the task difficulty (as introduced in Section 3).

3 TASK DIFFICULTY

As noted in Section 5, it is crucial for us to know the genuine capabilities a Minecraft agent possess, which cannot be revealed by the naive success-rate metric on specific tasks. To address this, we propose a set of difficulty dimensions to depict the tasks in MCU. For each difficulty, We also design a heuristic score to quantify it, which makes it convenient to computed for MCU-generated tasks.

Time Consumption Time consumption measures the horizon of a task. Due to the sparse reward, it is hard for agents to learn long-horizon decisions efficiently. Hierarchical RL from demonstrations (Lin et al., 2021; Mao et al., 2022) has been explored to leverage the task structure to accelerate

[3]Ten exploitation tasks are included (while other categories only contain five tasks) because this category contains diverse skills which are important for Minecraft agent.

the learning process. However, learning from unstructured demonstrations without domain knowledge remains challenging. Also, the policy model (usually based on Transformer architecture) is expensive to scale up with long action sequence generation (Zaheer et al., 2020; Choromanski et al., 2020; 2023). To measure this difficulty, the heuristic score designed for time consumption is defined as the average numbers of frames for solving the task.

Operational Effort The operational effort measures the entropy of the action distribution. When a task requires strong operational effort in Minecraft, it requires skillful operations of keyboard and mouse, and thus requires agents to solve the task with complex actions. The heuristic score is defined as the average *permutation entropy* (PE) of the action sequence in solving the task (which can be obtained from exemplar trajectories from *description* of a task). PE is a time series tool which provides a quantification measure of the complexity of a dynamic system (Bandt & Pompe, 2002), and the detailed definition of PE will be shown in Appendix A.

Planning Complexity The planning complexity is defined to measure the complexity of the pre-tasks. Considering the dependency graph (DG) of the task with its pre-tasks, if there exist many long paths that lead to the final task, it will be difficult to plan and select the optimal routes of sub-tasks for the agent when solving the task. The process of finding a pre-task routes towards the target task is the same as dependency resolution on DG (Ossher et al., 2010). We approximate the planning complexity as the complexity of finding a feasible resolution path. The typical algorithm is based on topology sorting (Kahn, 1962) with time complexity of $\mathcal{O}(E) + \mathcal{O}(V)$, where E and V are the number of edges and vertices. We define the heuristic score of planning complexity as $E + V$.

Intricacy The intricacy aims to measuring the precision required in agent control. The intricate tasks, such as "activate redstone circuits" and "craft specific items" are error-intolerant, which requires the agent to precisely finish the task with the precise task knowledge and have the ability to debug when something wrong happens. It is hard to design an intuitive heuristic for this difficulty, and we choose human-rating as an alternative.

Creativity The open-endness of Minecraft allows the agent to solve tasks which requires creativity. For example, an agent can build architectures, decorate the surroundings, and design functional circuits in Minecraft. Creativity is also a representative capability of a open-ended while the current works fail to focus on as shown in Section 4.3. Intuitively, we define creativity of a task as "the size of solution space" for the task. E.g., there are many solutions of "building a house" (high creativity), while only a single way to "Eat apple" (low creativity). It is also hard to find a practical heuristic for estimating creativity, therefore we also choose human-rating for expediency.

Novelty The ability to generalize to unseen (or out-of-distribution, OOD) scenarios is a crucial capability of open-endeds, which is typically considered weak in deep learning models. Due to the high diversity of Minecraft environment and infinite nature of tasks, it is impractical to curate training data that cover all possible cases. While existing Minecraft agents were usually trained on large-scale web data (Baker et al., 2022; Lifshitz et al., 2023), it is meaningful to see if the agent can adapt to OOD scenarios with necessary generalization ability. The in-distribution (IND, versus OOD) is defined as the distribution of trajectories that commonly adopted when playing Minecraft, e.g., survival and play-through. Some tasks that are useless for play-through but valid in Minecraft will be considered novel as there are less learning materials for agents, such as "commit suicide" or "kill a villager". We also opt for human-rating to evaluate the novelty of a task.

To compare different tasks on specific difficulty dimension, we devise a method to calibrate the heuristic scores into a normalized value in $[0, 5]$. The details are presented in Appendix C.

4 Analysis

In this section, we make some important analysis of MCU. Section 4.1 shows that MCU covers all the tasks used to test the Minecraft agents in previous literature, Section 4.2 takes SkillForge as an example to show how the different tasks are related in MCU by visualizing the task difficulty space, and Section 4.3 points out that the existing benchmarks exhibit a bias of task difficulty.

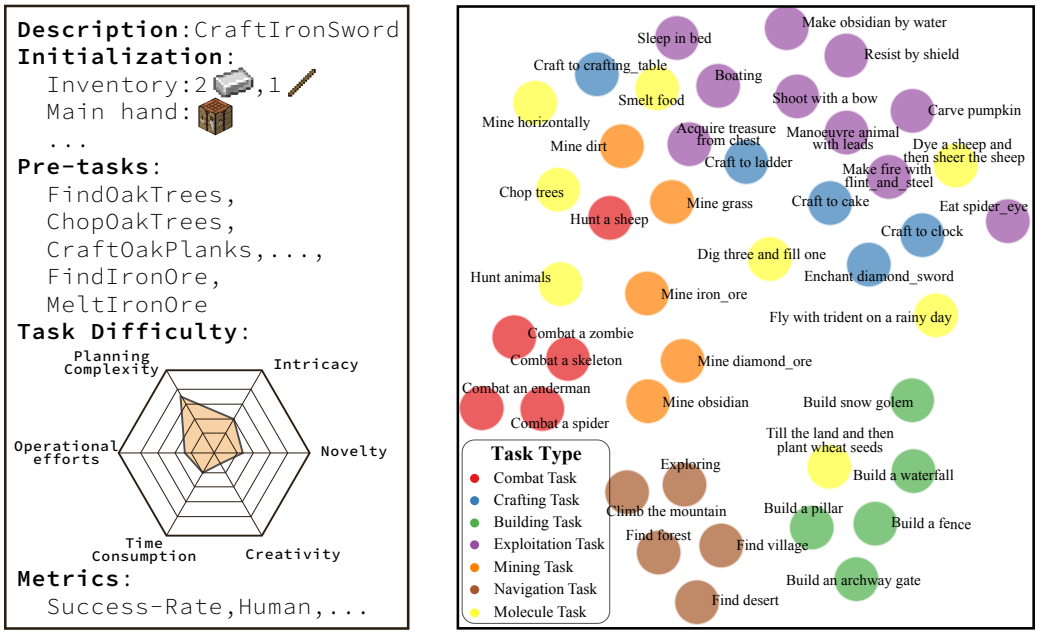

Figure 3: **Left figure**: Data structure for an Atomic Task: "Craft an iron_sword. In this figure, we provide a concise textual description of the task and list the pre-requisite tasks without displaying the full dependency graph for simplicity. **Right figure**: Visualization of the SkillForge Task Space using t-SNE. Each task is depicted by a difficulty feature vector $\mathbf{v} \in \mathbb{R}^6$, as defined in Section 3. Notice the clustering effect in the space, where tasks of the same category tend to group together (except for the molecule tasks).

4.1 Expressivity

We first validate that our proposed framework MCU has the strong expressivity which can cover all the tasks used in recent works to test the Minecraft agent. We maintain a paper list that contains 16 paper about Minecraft agent. The papers were public before September, 2023, which are also listed in Appendix B. As there are so many papers that adopted Minecraft as test bed for their algorithms, we cannot curate the papers thoroughly. These papers are chosen based on their relatedness with our goal of building an open-ended agent for Minecraft, and many old papers that simplified the environment of Minecraft are excluded (Tessler et al., 2017; Oh et al., 2016; Parashar et al., 2017).

The tasks used by these papers are mainly about obtaining diamond (Hafner et al., 2023; Mao et al., 2022; Lin et al., 2021; Baker et al., 2022), unlocking Minecraft Techtree (Nottingham et al., 2023; Zhu et al., 2023; Wang et al., 2023a;b), and specific skills (Malato et al., 2022; Cai et al., 2023; Fan et al., 2022; Guss et al., 2019; Lifshitz et al., 2023). As our curated atom task list covers the obtainment, exploitation, navigation of all items in Minecraft, the tasks to interact with items are all covered by MCU. There are also various tasks about specific skills in Minecraft, and thus cover the left of previous tasks. It is also noteworthy that traditional used tasks are only a small fraction of currently curated tasks in MCU, which shows the strong expressivity of MCU.

4.2 Task Space Visualization

An interesting application of our proposed task difficulty is to create a feature space for tasks. A task can be represented as a feature vector $\mathbf{v} \in \mathbb{R}^{\mathbf{6}}$, and the value for each dimension is the calibrated score of a difficulty. Taking the task list of SkillForge as an example, we visualize the task space in 2D space using t-SNE (Van der Maaten & Hinton, 2008) as the right subfigure in Figure 3.

The figure shows that tasks of the same category tend to group together in the projected 2D space. For example, the building tasks are clustered at the bottom right in the figure, and the navigation tasks clustered at the bottom center of the figure. The clustering effect indicates that each task category has a specific difficulty distribution. For example, the navigation tasks tend to have more time

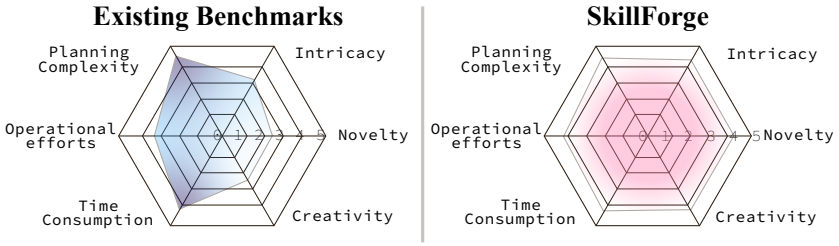

Figure 4: The difficulty distributions of existing benchmarks and SkillForge. Darker shades indicate a higher concentration of tasks with the corresponding difficulty scores, while lighter shades signify the converse trend.

consumption and creativity and less intricacy, while the crafting tasks have less time consumption and creativity and more planning complexity.

Also, we can understand the task difference within a category from the intra-category variance. For instance, the mining tasks are distributed in a long vertical area in the figure, which roughly forms a sequence "Mine dirt - Mine grass - Mine iron_ore - Mine diamond_ore - Mine obsidian". We know that this sequence follows the same order as the time consumption or planning complexity of these tasks. Although the task distribution within the same category is similar (thus they group together), there does exist variance within the same category (e.g., time consumption and planning complexity in mining tasks). This can help us to understand the tasks within the same category better.

Above all, the clustering effect of tasks offers inspirations for us in Minecraft agent evaluation:

- **Categorization improves evaluation efficiency**. As our final goal is to build an open-ended agent that can solve arbitrary tasks, a large number of tasks are necessary in evaluation to demonstrate the open-endedness. However, it is extremely expensive to run a comprehensive test even only on all the atom tasks (more than 3000) during the agent development. As we know from the clustering effect, tasks within the same category tend to assess the similar capabilities of an agent, therefore we can choose the representative tasks for a cluster (instead of all the tasks) to conduct efficient evaluation.
- **Intra-category variance helps representative task selection.** To select representative tasks within a category for evaluation, it is better to select the tasks that are dissimilar to each other to cover diverse facets. As analyzed above, the intra-category variance shown in the task space helps to identify the representative tasks. For example, "Eat spider_eye" (high novelty), "Resist by shield" (high planning complexity), "Sleep in bed" (low time consumption), "Acquire treasure from chest" (low planning complexity) are representatives of exploitation tasks.[4]

Note that the tasks in SkillForge are also selected based on the idea mentioned above.

4.3 Bias of Existing Benchmarks

Figure 4 offers a visual representation of task difficulty distributions within existing benchmarks (mentioned in Section 4.1) and SkillForge. The darker areas in Figure 4 indicate that there are more tasks with the corresponding difficulty scores. Our analysis reveals a pronounced inclination in existing benchmarks towards tasks characterized by long temporal horizons and complex planning requirements. In sharp contrast, our proposed SkillForge benchmark boasts a more diverse and evenly spread task difficulty distribution. This diversity contributes significantly to a comprehensive evaluation of an agent's capabilities.

This imbalance in difficulty distribution suggests that traditional agents excel in planning-centric tasks but may fall short in areas such as precise control intricacies and out-of-distribution (OOD) generalization capabilities. This finding underscores the need for the deliberate design of agents with enhanced proficiency in these specific areas.

[4]Also note that the shared property of exploitation tasks is low creativity, as the usage of different tools are pre-defined in Minecraft.

5 RELATED WORK

There is still not a unified benchmark for evaluating Minecraft agents. Currently, the prevailing evaluation method is what we refer to as "programmatic evaluation". This approach involves calculating programmatic evaluation metrics by continuously monitoring the environment's state throughout each episode. For instance, these metrics may encompass the success rate in obtaining specific in-game objects, as illustrated in studies such as Zhu et al. (2023); Wang et al. (2023b).

Nonetheless, due to the inherently open-ended nature of the Minecraft environment, numerous tasks prove challenging to assess solely through the observation of in-game data. In an effort to tackle this issue, Minedojo introduced an alternative benchmark suite consisting of 1560 creative tasks defined through natural language instructions (Fan et al., 2022). Despite its promising approach, this benchmark has not gained widespread adoption, primarily due to the following reasons:

(1) **Redundancy**: The Minedojo task list exhibits significant redundancy. For instance, the task "build a nether portal" is repeated nine times. There are other instances of repetitive tasks, such as "create a silverfish statue," "design a city with distinct districts," and "construct an automatic gold farm," to name just a few.

(2) **Impracticality**: A substantial fraction of the tasks within Minedojo presents formidable challenges that even adept human players would find exceedingly impractical. For instance, tasks such as "Craft a life-size replica of the Titanic", "Construct a faithful replica of the White House", and "Erect a cathedral" surpass the practicality threshold for effective evaluation of Minecraft agents. These tasks are excessively demanding and do not serve as suitable test cases.

(3) **Disconnectedness**: Within the Minedojo task list, each task is treated in isolation, devoid of discernible connections or relationships with other tasks. Consequently, when an agent successfully accomplishes some tasks, it remains unclear what specific capabilities it possesses.

Hence, it is of paramount importance to devise a benchmark that rectifies these issues by eliminating redundancy, aligning task difficulty with practicality, and elucidating the essential capabilities required for each task. We aim to address these issues through the introduction of MCU, which is elaborated upon in Section 2 and Section 3.

6 CONCLUSION

We present MCU, a task-centric framework for open-ended agent evaluation in Minecraft. MCU introduces the concept of *atom tasks*, which act as the fundamental building blocks for generating infinite tasks and benchmarks within Minecraft landscape. Through the design of six difficulty scores (time consumption, operational effort, planning complexity, intricacy, creativity, and novelty), the MCU framework creates a meaningful task space. These scores not only define the complexity of tasks but also shed light on the relationships among tasks in Minecraft. Moreover, they serve as a guiding compass for selecting representative tasks when creating benchmarks. To facilitate practical evaluations, we maintain two unified benchmarks (SkillForge and HorizonForge). Through our analysis, we have uncovered an important insight: existing benchmarks tend to exhibit bias towards specific levels of difficulty, which is not sufficient for development of an open-ended agent. This discovery opens up exciting avenues for enhancing agent creativity, precision in control, and out-of-distribution generalization within the Minecraft world.

Furthermore, our approach to generating an endless array of tasks serves as a source of inspiration for the ongoing journey in AI research. In our pursuit of achieving Artificial General Intelligence (AGI), it is imperative to assess AGI's capacity to tackle a wide spectrum of real-world tasks. This also presents an intriguing challenge for the research community: how to design meaningful benchmarks that will guide the development of AGI in the future? As the open-ended Minecraft environment offers a reasonable simulation for the complexities of the real world, we believe that our MCU framework holds the potential to provide valuable insights in this endeavor.

REFERENCES

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. 1, 6, 7, 12, 13

Christoph Bandt and Bernd Pompe. Permutation entropy: a natural complexity measure for time series. *Physical review letters*, 88(17):174102, 2002. 6

Shaofei Cai, Zihao Wang, Xiaojian Ma, Anji Liu, and Yitao Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13734–13744, 2023. 5, 7, 13

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020. 6

Krzysztof Marcin Choromanski, Shanda Li, Valerii Likhosherstov, Kumar Avinava Dubey, Shengjie Luo, Di He, Yiming Yang, Tamas Sarlos, Thomas Weingarten, and Adrian Weller. Learning a fourier transform for linear relative positional encodings in transformers. *arXiv preprint arXiv:2302.01925*, 2023. 6

Ziluo Ding, Hao Luo, Ke Li, Junpeng Yue, Tiejun Huang, and Zongqing Lu. Clip4mc: An rl-friendly vision-language model for minecraft. *arXiv preprint arXiv:2303.10571*, 2023. 13

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, 7, 9, 13

William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019. 7, 13

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023. 7, 13

Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962. 6

Anssi Kanervisto, Stephanie Milani, Karolis Ramanauskas, Nicholay Topin, Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, Wei Yang, et al. Minerl diamond 2021 competition: Overview, results, and lessons learned. *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 13–28, 2022. 1, 13

Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *arXiv preprint arXiv:2306.00937*, 2023. 1, 5, 6, 7, 13

Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:2112.04907*, 2021. 1, 5, 7, 13

Federico Malato, Florian Leopold, Amogh Raut, Ville Hautamäki, and Andrew Melnik. Behavioral cloning via search in video pretraining latent space. *arXiv preprint arXiv:2212.13326*, 2022. 7, 13

Hangyu Mao, Chao Wang, Xiaotian Hao, Yihuan Mao, Yiming Lu, Chengjie Wu, Jianye Hao, Dong Li, and Pingzhong 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*, pp. 38–51. Springer, 2022. 1, 5, 7, 13

Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep?: Embodied decision making using language guided world modelling. *arXiv preprint arXiv:2301.12050*, 2023. 1, 7, 13

Junhyuk Oh, Valliappa Chockalingam, Honglak Lee, et al. Control of memory, active perception, and action in minecraft. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2016. 7

Michael Olan. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*, 19(2): 319–328, 2003. 2

Joel Ossher, Sushil Bajracharya, and Cristina Lopes. Automated dependency resolution for open source software. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 130–140. IEEE, 2010. 6

Priyam Parashar, Bradley Sheneman, and Ashok K Goel. Adaptive agents in minecraft: A hybrid paradigm for combining domain knowledge with reinforcement learning. In *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Visionary Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*, pp. 86–100. Springer, 2017. 7

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017. 7

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 7

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*, 2023a. 1, 7, 13

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*, 2023b. 1, 2, 7, 9, 13

Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*, 2023. 1, 13

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020. 6

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. 1, 2, 7, 9, 13

A PERMUTATION ENTROPY

Following Baker et al. (2022), we use the same action space that includes almost all actions directly available to human players, such as key-presses, mouse movements, and clicks. In this section, we describe how to measure permutation entropy (PE) of an action sequence in order to compute the operational effort of a task.

Suppose there are N possible feasible actions in total, and each action is denoted by a number from $\{1, 2, \cdots, N\}$. Given an action sequence $\mathbf{X} = \{x_t : t = 1, \cdots, T\}$, for a chosen segment length D, we compose a vector of length D consisting of the subsequent actions, namely:

$$\mathbf{x}_i^D = (x_i, x_{i+1}, \cdots, x_{i+D-1}), \quad i \in \{1, 2, \cdots, N - D + 1\}$$

For each constructed vector, $\mathbf{x}_i^D$, by comparing the magnitudes of the entries in the vector, each segment $\mathbf{x}_i^D$ is mapped onto one of the $D!$ possible ordinal patterns $\{\pi_i : i = 1, \cdots, D!\}$, where we have ignored the possibility of equal values in action sequences.[5] To illustrate, if $D = 3$, there are six distinct ordinal patterns which are given as:

$$\begin{aligned}
\pi_1 &: x_i < x_{i+1} < x_{i+2}, \\
\pi_2 &: x_i < x_{i+2} < x_{i+1}, \\
\pi_3 &: x_{i+1} < x_i < x_{i+2}, \\
\pi_4 &: x_{i+2} < x_i < x_{i+1}, \\
\pi_5 &: x_{i+1} < x_{i+2} < x_i, \\
\pi_6 &: x_{i+2} < x_{i+1} < x_i.
\end{aligned}$$

The probability of each ordinal pattern is estimated by the number of the constructed segments that are mapped into π_i, divided by the total number of the segments:

$$p^D(\pi_i) = \frac{\#\{\mathbf{x}_t^D : \mathbf{x}_t^D \text{ has ordinal pattern } \pi_i\}}{N - D + 1}, \quad i \in \{1, \cdots, D!\}$$

Applying the definition of Shannon's Entropy, the permutation entropy for the action sequence $\mathbf{X}$, with chosen segment length D is defined as

$$PE^D(\mathbf{X}) = \sum_{i=1}^{D!} \frac{-p^D(\pi_i) \ln(\pi^D(\pi_i))}{\ln D!}$$

The value of permutation entropy is bounded between 0 and 1, where 0 indicates a completely predictable action dynamic, and 1 indicates a completely stochastic dynamic. In navigation tasks, there are a large sequence of action as "forward", thus the ordinal patterns in $\mathbf{X}$ will have low entropy. While in combat tasks, there will be diverse actions to attack, resist, hide, move, and thus result in higher entropy. As permutation entropy ignores the factor of sequence length, therefore it is orthogonal to another metric, time consumption, adopted in our difficulty dimensions.

[5] If there are equal values, we rank them in occurrence order. E.g., (1,2,1) is equivalent to (1,3,2), and (3,3,1) is equivalent to (2,3,1).

B LITERATURE OF MINECRAFT AGENT

Table 1: The papers of Minecraft agent studied in this paper and their task list used to evaluate the agent. For simplicity, we use "XXX" to denote many items in Minecraft.

Paper	Task list
Cai et al. (2023)	Mine oak wood, Hunt sheep, Hunt cow, Hunt pig, Mine dirt, Mine sand, Mine birch wood, Mine oak_leaves, Mine birch_leaves, Obtain wool, Mine grass, Mine poppy, Mine orange_tulip, Combat spider, Combat polar bear, Hunt chicken, Hunt donkey, Hunt horse, Hunt wolf, Hunt llama, Hunt mushroom cow
Wang et al. (2023b)	Minecraft TASK101 (Craft XXX, Equip XXX, Mine diamond)
Fan et al. (2022)	Milk cow, Hunt cow, Shear sheep, Hunt Sheep, Combat spider, Combat zombie, Combat pigman, Combat enderman, Find Nether_portal, Find ocean, Dig hole, Lay carpet
Baker et al. (2022)	Mine log, Craft planks, Craft crafting_table, Craft wooden_pickaxe, Mine cobblestone, Craft stone_pickaxe, Mine iron_ore, Craft furnace, Smelt to Iron Ignot, Craft Iron_pickaxe, Mine diamond, Craft diamond_pickaxe
Mao et al. (2022)	Mine diamond
Kanervisto et al. (2022)	Mine diamond
Guss et al. (2019)	Navigation, Chop tree, Obtain iron_pickaxe, Obtain diamond, Obtain cookedMeet, Obtain bed, Survival
Lin et al. (2021)	Mine diamond
Lifshitz et al. (2023)	Dig as far as possible, Get dirt, Look at the sky, Break leaves, Chop a tree, Collect Seeds, Break a flower, Go explore, Go swimming, Go underwater, Open inventory, Get dirt, Chop down a tree, Break tall grass
Zhu et al. (2023)	Minecraft Technology Tree (Obtain XXX)
Yuan et al. (2023)	Obtain stick, Obtain crafting_table, Obtain bowl, Obtain chest, Obtain trap_door, Obtain sign, Obtain wooden_pickaxe, Obtain furnace, Obtain stone_stairs, Obtain stone_slab, Obtain cobblestone_wall, Obtain lever, Obtain torch, Obtain stone_pickaxe, Obtain milk_bucket, Obtain wool, Obtain beef, Obtain mutton, Obtain bed, Obtain painting, Obtain carpet, Obtain item_frame, Obtain cooked_beef, Obtain cooked_mutton
Ding et al. (2023)	Obtain milk, Obtain wool, Obtain leaf, Obtain sunflower, Hunt cow, Hunt sheep
Wang et al. (2023a)	Minecraft Technology Tree (Obtain XXX)
Hafner et al. (2023)	Mine diamond
Nottingham et al. (2023)	Minecraft Technology Tree (Obtain XXX)
Malato et al. (2022)	Find cave, Build animal_pen, Make waterfall, Build house

C CALIBRATION OF DIFFICULTY SCORES

For a specific difficulty, we compute the uncalibrated scores for all curated atom tasks and their naive initialization version, which result in a list of scores $\{S_i\}_{i=1}^N$. We then obtain the 5-quantiles of $\{S_i\}_{i=1}^N$ as $Q_0, Q_1, Q_2, Q_3, Q_4, Q_5$.[6] Then we compute the calibrated score as:

$$\tilde{S}_i = \begin{cases} j + \dfrac{S_i - Q_j}{Q_{j+1} - Q_j}, & S_i \in [Q_j, Q_{j+1}), \quad j \in \{0, 1, 2, 3, 4\} \\ 5, & S_i = Q_5 \end{cases}$$

where $\tilde{S}_i$ is the calibrated score of S_i, and we set the maximum score to 5. It is obvious that the calibrated score $\tilde{S}_i$ ranges from $[0, 5]$.

D HORIZONFORGE-TEXT

HorizonForge-text is a simple environment to evaluate the ability of finding a feasible route towards the target task by resolving pre-tasks. Given a target task in HorizonForge, the agent has to choose the candidate actions in each round in order to achieve the final task. Each candidate action is an atom task that can be solved with current conditions (the dependency of the action is resolved).

For example, we show an example of testing agent to solve "craft iron_sword" in HorizonForge-text as Figure 5. At the beginning, the agent is randomly initialized with 3 logs and 8 cobblestones in its

[6] q-quantiles are values that partition a finite set of values into q subsets of equal sizes.

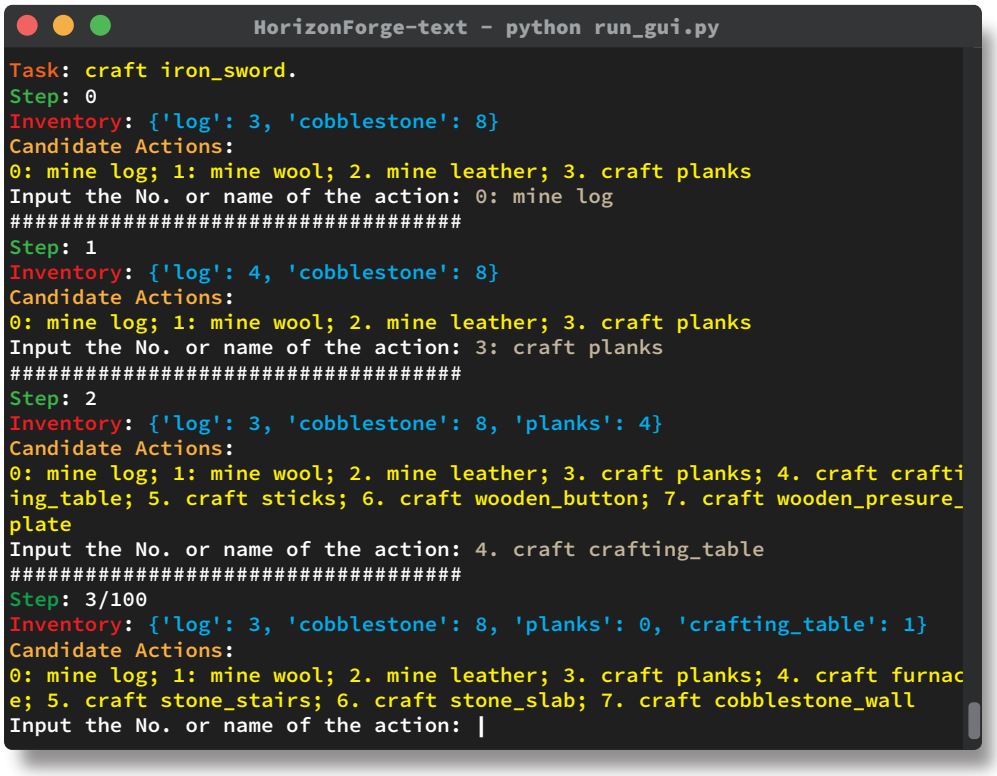

Figure 5: The command-line interface of HorizonForge-text. In this example, the agent is tested with a task "craft iron_sword" from HorizonForge.

inventory, and its feasible actions are "mine log", "mine wool", "mine leather", and "craft planks". Choosing to mine log at step 0, the agent got 1 log and continued to craft planks using logs in step 1. After obtaining planks, more candidate actions were unlock and it chose to craft a crafting table.

HorizonForge-text offers a lightweight testing environment for HorizonForge, designed to simplify the Minecraft gaming environment by removing the need for low-level control. While it excels in evaluation efficiency, it is important to underscore that HorizonForge-text cannot fully supplant the original Minecraft environment, given the vital role that control plays in agent development. Instead, we envision HorizonForge-text as a convenient tool, providing researchers with a user-friendly platform to assess and refine their agents through swift, iterative testing.