MCU: An Evaluation Framework for Open-Ended Game Agents

Xinyue Zheng [1 *] **Haowei Lin** [2 *] **Kaichen He** [2] **Zihao Wang** [2] **Qiang Fu** [3] **Haobo Fu** [3] **Zilong Zheng** [1 ‡]
Yitao Liang [2 ‡]

Abstract

Developing AI agents capable of interacting with open-world environments to solve diverse tasks is a compelling challenge. However, evaluating such open-ended agents remains difficult, with current benchmarks facing scalability limitations. To address this, we introduce *Minecraft Universe* (MCU), a comprehensive evaluation framework set within the open-world video game Minecraft. MCU incorporates three key components: (1) an expanding collection of 3,452 composable atomic tasks that encompasses 11 major categories and 41 subcategories of challenges; (2) a task composition mechanism capable of generating infinite diverse tasks with varying difficulty; and (3) a general evaluation framework that achieves 91.5% alignment with human ratings for open-ended task assessment. Empirical results reveal that even state-of-the-art foundation agents struggle with the increasing diversity and complexity of tasks. These findings highlight the necessity of MCU as a robust benchmark to drive progress in AI agent development within open-ended environments. Our evaluation code and scripts are available at `https://github.com/CraftJarvis/MCU`.

1. Introduction

Developing AI agents capable of interacting with dynamic environments—often referred to as "open-world" in the literature (Parmar et al., 2023)—to solve diverse tasks remains a long-standing challenge in Artificial Intelligence (Kejriwal et al., 2024). Among the various environments used to study AI agents, games have emerged as a prominent choice, as they provide real-world challenges within programmable simulators, offering valuable opportunities for real-world simulation (Bruce et al., 2024; Reed et al., 2022; Raad et al., 2024). Compared to other digital environments such as web/apps (Zhou et al., 2023; Qin et al., 2025; Lin et al., 2024), mobile platforms (Pan et al., 2024; Wang et al., 2024), and coding IDEs (Xu et al., 2022; Jimenez et al., 2024; Huang et al., 2023), games present a higher degree of control complexity, similar to that encountered by physical robotic agents (Nasiriany et al., 2024; Wang et al., 2023b; Zhou et al., 2024b). While compared to robotics, games support long-horizon planning tasks and enable safer and more efficient testing within sandbox environments.

A crucial aspect of open-ended game agents is their *generalizability*. The ultimate goal of developing AI agents is to deploy them in real, open-world environments, where they must solve tasks robustly across drastically different situations. With this in mind, early game agents have been studied in procedurally generated Atari-like environments with diverse configurations (Cobbe et al., 2020). Recent efforts shift toward complex environments with greater freedom (Fan et al., 2022; Hafner, 2021; Team et al., 2021), among which Minecraft stands out. Minecraft is a video game that provides procedurally generated open-world environments with a state space exceeding the number of atoms in the universe (we elaborate on the benefits of using Minecraft for open-ended game agents in Section 2.1). This vast environment allows agents to tackle infinite open-ended tasks using human-like actions in diverse situations.

Despite the numerous advantages of Minecraft as an experimental environment, we have identified several limitations in existing benchmarks that impede a comprehensive evaluation of agents' generalizability. These limitations include low task quality (Fan et al., 2022), insufficient diversity (Hafner, 2021), and the lack of automatic evaluation suites (Cai et al., 2024c). We further illustrate these issues in Figure 2. Based on these issues, we find that Minecraft agents lack a unified benchmark, and each agent is evaluated on a distinct set of tasks (see Section 4 for further discussion). To address this, we introduce *Minecraft Universe* (**MCU**), an advanced evaluation framework in Minecraft (Figure 1). MCU encompasses thousands of composable

* Equal contribution. ‡ Equal contribution as co-corresponding authors. [1]Beijing Institute for General Artificial Intelligence (BIGAI), Beijing, China [2]Institute for Artificial Intelligence, Peking University, Beijing, China [3]Tencent AI Lab, Shenzhen, China. Correspondence to: Zilong Zheng <zlzheng@bigai.ai>, Yitao Liang <yitaol@pku.edu.cn>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

tasks and provides a scalable automatic evaluation system, designed based on the following key principles:

High Task Diversity Task diversity plays a crucial role in: (1) simulating challenges across various scenarios to thoroughly evaluate generalizability, and (2) harnessing the full potential of open-ended game agents capable of solving a wide range of tasks. MCU emphasizes two key aspects of task diversity in its implementation: (1) **Intra-task diversity:** We utilize a large language model (LLM) to dynamically generate task initialization conditions—such as biome, weather, and player states—introducing variability and randomness that closely mirror real gameplay scenarios (Figure 1a). (2) **Inter-task diversity:** We collect 3,452 atomic tasks within Minecraft, spanning a broad spectrum of challenges, including precise control (*e.g., combat, building*), logical reasoning (*e.g., crafting, trading*), and knowledge application (*e.g., tool use, animal care*) (Figure 1b). These tasks can further be composed to generate new tasks on a combinatorial scale as detailed in Section 2.3.

High Task Quality Minecraft imposes numerous constraints that render unchecked tasks intractable (Fan et al., 2022; Yang et al., 2024). For example, the task "mine diamond with a wooden pickaxe" is infeasible because diamonds cannot be mined with such a tool. Similarly, the task "design and build a transportation system for your city" from the MineDojo benchmark (Fan et al., 2022) presents an extreme challenge even for humans. Other issues include repetitive tasks (Figure 2). To mitigate these issues, MCU filters tasks from a wide range of data sources, ensuring adherence to high-quality standards (Figure 1b left). Additionally, in the LLM-based task configuration generator, we incorporate a series of soft constraints within the prompt, and implement a refinement mechanism based on self-reflection (Shinn et al., 2023) and the Minecraft simulator's feedback by executing the generated configuration. Further implementation details are provided in Section 2.4.

Automated Evaluation Open-ended tasks (Stanley et al., 2017; Standish, 2003) inherently lack clear success signals and often depend on human evaluation or manually designed metrics (Dubois et al., 2024), which hinders scalability. To address this issue, MCU introduces an automated evaluation system (*AutoEval*) based on vision-language model (VLM) that fulfills two key objectives: (1) producing evaluation results that closely align with human judgments, and (2) offering multi-dimensional assessments that go beyond simple success rates for a comprehensive evaluation for open-ended tasks (Figure 1c), while the designed "task progress" dimension is a process-supervised counterpart of 0-1 success rate. We also show in Figure 1e that *AutoEval* is cost-efficient.

Enduring Challenges To ensure that MCU remains a long-term benchmark for agent development, we adopt two key strategies: (1) designing tasks with varying levels of difficulty, where increasing complexity introduces additional challenges such as adverse weather conditions and misleading factors. While state-of-the-art models achieve moderate success on simpler tasks, they struggle with more complex scenarios (Figure 1d). (2) Enabling the composition of atomic tasks into more intricate tasks. This approach exponentially increases both the number and complexity of tasks, ensuring that MCU continues to provide a lasting challenge.

2. MCU Benchmark

In this section, we first provide an overview of Minecraft and its game simulator, followed by a detailed outline of the construction process for the MCU benchmarking pipeline.

2.1. Minecraft Environment

Minecraft is a voxel-based 3D video game that, due to its popularity and wide variety of mechanics, has become a prominent domain for RL research (Oh et al., 2016; Tessler et al., 2017). Much of the prior work focuses on small, custom-built Minecraft environments with tasks such as navigation (Arumugam et al., 2019; Matiisen et al., 2019), block placement (Trott et al., 2019; Alaniz, 2018), combat (Udagawa et al., 2016), and other similar activities (Shu et al., 2017). More recent efforts have shifted towards studying the full, unmodified human action space, which encompasses tasks like drag-and-drop inventory management and item crafting. In this work, we employ unmodified Minecraft version 1.16.5 as our testing environment (Guss et al., 2019b), utilizing mouse and keyboard inputs as the action space and a 640×360 RGB image as the observation. The specifics of the action space will be detailed in Table 6.

As mentioned in Section 1, Minecraft serves as a powerful experimental environment due to its unparalleled diversity, complexity, and open-ended nature, which enable creative gameplay and countless possibilities. Below, we outline the key features that make Minecraft particularly suitable for open-ended game agent development:

1. **Vast State Space.** Minecraft provides an extraordinarily large state space, as illustrated in Table 7 with an intuitive comparison. Its expansive maps, functional blocks, and diverse mobs result in an immense number of possible configurations. This makes Minecraft an ideal platform for studying the generalizability of agents. Additionally, we present illustrative experiments in Appendix D to demonstrate the importance of vast state spaces for agent generalization.
2. **High Complexity Support.** Minecraft supports tasks

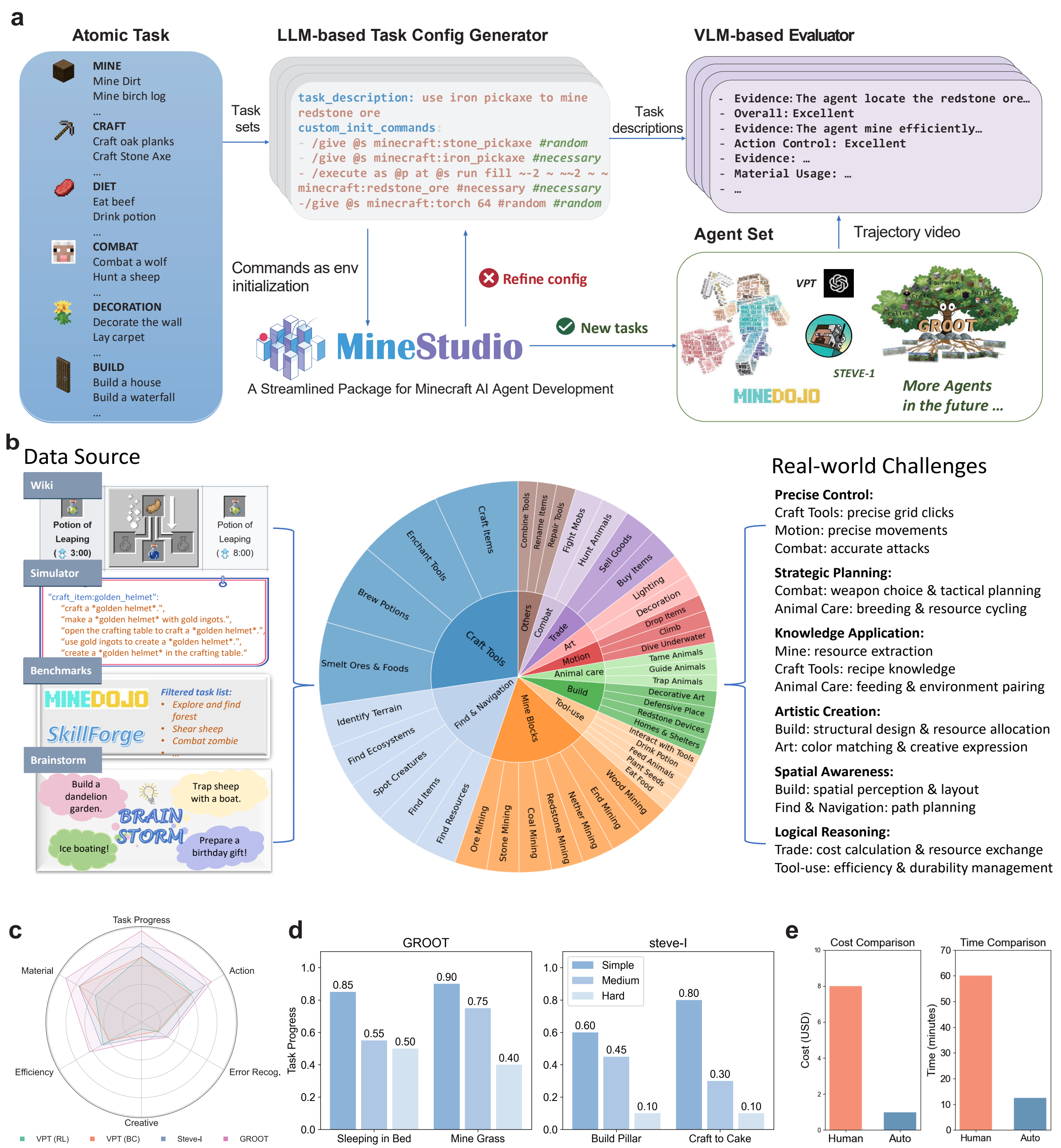

Figure 1: An overview of MCU. ***a. Benchmarking pipeline.*** MCU includes two main components: ***task generation*** *and* ***trajectory evaluation***. The LLM-based task configuration generator instantiates the environment with the necessary prerequisites, random factors, and task descriptions for diverse atomic tasks. These configurations are verified using an environment simulator. The VLM-based evaluator assesses each task trajectory in video form across multiple dimensions, providing comprehensive performance insights. MCU offers a model-agnostic evaluation interface based on Minestudio (Cai et al., 2024a), making it suitable for various agents. ***b. Task category distribution.*** The atomic task set is sourced from the Minecraft wiki, in-game data, existing benchmarks, and brainstorming sessions. It spans 11 major categories and 41 subcategories, ensuring high ***inter-task diversity***. For readers unfamiliar with Minecraft, we illustrate the real-world challenges associated with different task categories to provide context. ***c. Multi-dimensional capabilities.*** MCU evaluation shows that SOTA agents have made progress in overall task completion and material usage, but still have obvious limitations in creativity and error recognition. ***d. Intra-task generalizability.*** Varying difficulty levels within the same task lead to performance degradation, which tests the agent's intra-task generalization capability. ***e. Human vs AutoEval.*** The automatic evaluation (*AutoEval*) of MCU is 8.1× more cost-effective and 4.8× more efficient in labeling 30 samples. Best viewed zoomed in.

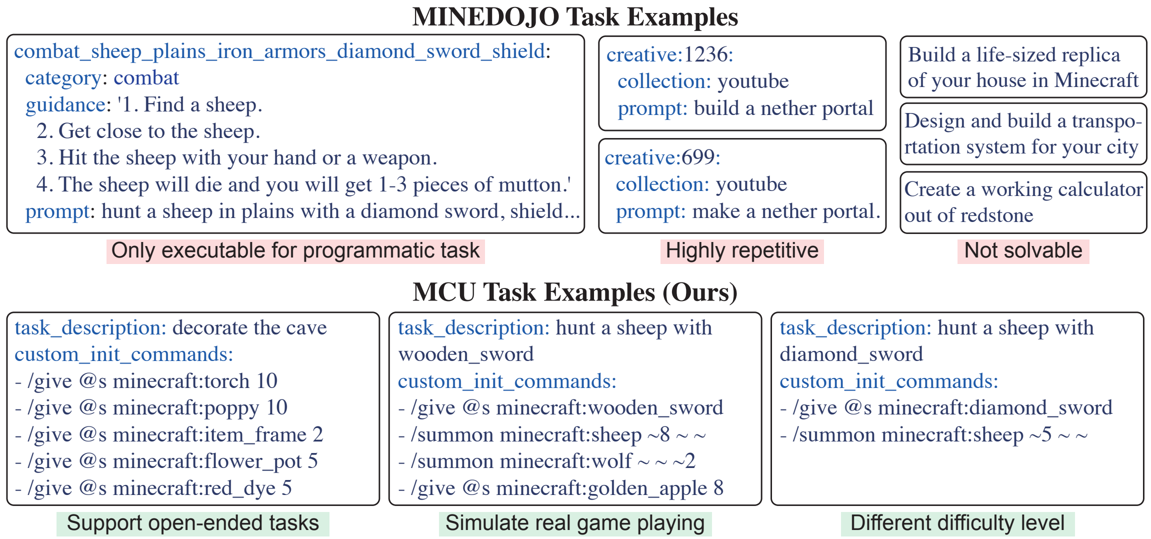

Figure 2: A comparison between the "tasks" in MCU and Minedojo (Fan et al., 2022). We investigate the task list provided by MineDojo and identify several issues. For example, only programmatic tasks that have clear reward signal can be executable in the benchmark; many tasks in their list are repetitive (both No.1236 and No.699 are "build nether portal"); and a large amount of tasks in the creative tasks are not solvable even by human. To address this, our MCU benchmark can create executable configurations for open-ended tasks, and ensure intra-task and inter-task diversity to simulate real game playing in different difficulty levels, while preserving solvability of tasks.

requiring advanced problem-solving skills. For instance, *obtain diamond* task (Guss et al., 2019a) involves over 20 sub-goals, often takes hours to complete, and demands the ability to remember terrain and resource locations—showcasing the game's capability to facilitate complex task development and execution.

3. **Open-Endedness.** Minecraft encourages unrestricted exploration, allowing players to engage in a wide spectrum of challenges. These range from defeating the ender dragon, which requires long-horizon decision-making (Jin et al., 2023), to building a house with precise control and creativity (Zhang et al., 2020). This open-ended nature fosters the development of agents capable of handling diverse and dynamic objectives.

2.2. Integration with MineStudio

To create a robust and user-friendly benchmark, we develop MCU based on MineStudio (Cai et al., 2024a), an open-source software package designed specifically to facilitate agent development in Minecraft. MineStudio provides intuitive APIs, extensive documentation and tutorials, making it an ideal foundation for our benchmarking framework.

MineStudio offers users extensive customization options by allowing them to inherit from the `MinecraftCallback` class. This facilitates functionalities such as issuing cheat commands, logging episodes, and overriding observations. Leveraging this flexibility, our task configuration pipeline generates the necessary data for class `CommandsCallback`, `SummonMobsCallback` and `FastResetCallback` to initialize the environment. To ensure generality, MineStudio unifies the agent inference pipeline. Consequently, our evaluation pipeline observes only the generated trajectories, which are consistently formatted by `RecordCallback` for compatibility across diverse agents. Additionally, we employ `RewardsCallback` to support user-defined metrics, such as task success rates and our *AutoEval* metrics, and to enable RL training with MCU evaluation results.

2.3. Atomic Tasks: Fundamental Testing Units in MCU

We introduce the concept of *atomic tasks* as the fundamental testing units in MCU. The inspiration for atomic tasks stems from unit testing in software development (Olan, 2003), where a system is decomposed into smaller, independently verifiable units. Similarly, atomic tasks are designed to isolate and evaluate specific agent capabilities.

An *atomic task* $\mathcal{T}$ is defined solely by its goal g, independent of the methods, tools, or specific environmental conditions. During evaluation, an atomic task is instantiated, which induces a task-specific initial state distribution $\mathcal{P}(s_0|g)$ (see Section 2.4). For example, the atomic task `mine stone` is purely goal-driven. Across different evaluation batches, it may manifest as `mine stone with a wooden pickaxe`, `mine stone during a rainy day`, or other variations. Regardless of these instances, the core goal remains consistent, ensuring a reliable test of whether the agent has genuinely acquired the underlying capability. This property makes atomic tasks an effective tool for evaluating an agent's ability to achieve goals under diverse and

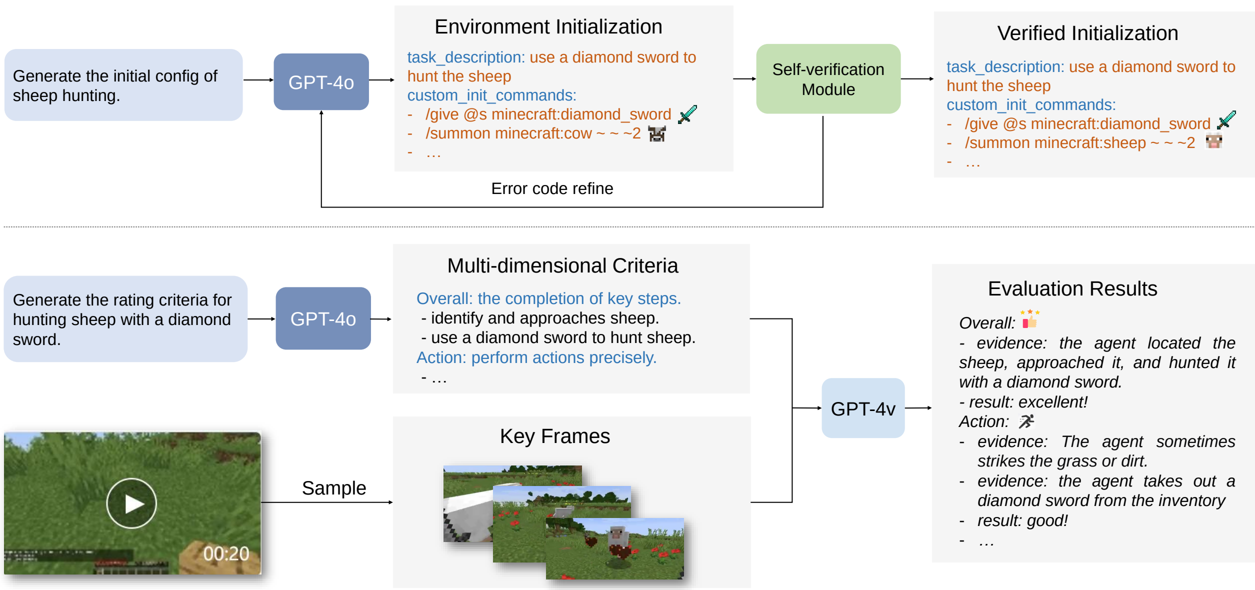

Figure 3: Automatic task generation and evaluation pipeline. **Top: Task Generation.** GPT-4o generates and verifies the environment initialization configuration for each atomic task, producing executable instructions compatible with Minestudio APIs. **Bottom: Automatic Evaluation.** A vision-language model (GPT-4o with vision) assesses task performance by analyzing sampled video frames against criteria generated by GPT-4o, providing detailed evidence and results. The GPT-4o used is `gpt-4o-2024-08-06`.

potentially unseen conditions.

Atomic tasks can also be combined to form more complex tasks using logical operators such as "and" ($\bigwedge$) and "or" ($\bigvee$), or by introducing constraints such as "when," "where," and "how." For instance, an agent could be tasked with `[mine oak log]` `or` `[mine grass]` `{bare-handed}` `and` `{then}` `[craft sticks]`, where `[]` denotes individual atomic tasks. This compositional approach enables the exploration of a vast task space by leveraging the combination of simple goals to create more challenging scenarios. We also incorporate this feature in MCU task generation pipeline as detailed in Appendix E.2

To pursue task diversity, as outlined in Section 1, we have collected *3,452 atomic tasks*[1], each represented as a textual description. These tasks serve as the fundamental building blocks for task synthesis in MCU. They may be later composed (discussed in the previous paragraph) and instantiated using an LLM-based task configuration generator (see Section 2.4). The atomic tasks are categorized into 11 major categories and 41 sub-categories, covering diverse challenges encountered in Minecraft. These tasks are sourced through the following pathways: (1) Distilling high-quality tasks from existing benchmarks, such as MineDojo (Fan et al., 2022) and SkillForge (Cai et al., 2024c). (2) Extracting tasks from the Minecraft Wiki[2]. (3) Synthesizing tasks using in-game data from the Minecraft simulator (e.g., generating tasks like `craft item X` if item "X" is craftable). (4) Brainstorming innovative tasks with input from domain experts and LLMs. Further details on the collection process for atomic tasks are provided in Appendix B.1.

[1]The dataset is continuously expanding.

[2]https://minecraft.wiki/

2.4. Task Configuration Generation

To execute automatic or compositional tasks in the Minecraft environment, tasks must be *configured* by specifying parameters such as spawn biomes, player states, inventories, and surrounding mobs. We formalize this configuration process as sampling an initial state s_0 from a task-specific distribution $\mathcal{P}(s_0|\mathcal{T})$. To simulate realistic gameplay, evaluate agents effectively on a given task $\mathcal{T}$, and ensure intra-task diversity (Section 1), we propose a scalable task configuration pipeline powered by LLMs. This pipeline incorporates an automatic verification mechanism to ensure task validity based on two criteria: (1) the sampled initial state s_0 exhibits high diversity, and (2) the environment includes all elements necessary to complete the task (i.e., s_0 is compatible with $\mathcal{T}$). By leveraging the knowledge and creativity of LLMs, this approach generates diverse and scalable task scenarios, addressing the limitations of manually predefined configurations. The implementation details are as follows.

Config Generator As detailed in Section 2.2, we utilize three callbacks from MineStudio to initialize the task environment. A powerful LLM GPT-4o (Achiam et al., 2023) is prompted with a task description and few-shot examples of the parameter format to populate these callbacks, generating executable configurations and detailed task descriptions for text-conditioned agents and evaluation in Section 2.5. For instance, the atomic task `mine diamonds` may include the description "mine diamonds with an iron pickaxe", with the initial state s_0 providing necessary resources (e.g., diamond ores and an iron pickaxe) to eliminate preparatory subtasks. To enhance task diversity, random factors such as ore placement and item arrangements are introduced to prevent predictability. The prompt also includes soft constraints to address LLM limitations, such as numerical insensitivity or confusion with game-specific rules. For crafting tasks requiring precise materials (e.g., 3 wool blocks and 3 wooden planks), surplus resources are instructed to be generated to account for LLM inaccuracies. Soft constraints also ensure environmental integrity by avoiding the generation of inaccessible structures (e.g., via the `/fill` command). Complete prompt details are provided in Appendix G.1.

Verification To ensure the quality of the generated task configurations, the config generator incorporates a self-verification mechanism. Initially, the generated configuration is re-validated by the LLM using the Reflxion technique (Shinn et al., 2023). If any errors are detected—such as summoning cows only for the task `hunt sheep` (Figure 3)—the config generator is prompted to regenerate the configuration. The generated configuration is also validated using the MineStudio Simulator to ensure executability. Error logs from the simulator are also utilized to guide further regeneration of the configuration if necessary.

2.5. Automatic Evaluation (*AutoEval*)

Automatic evaluation is critical for machine learning benchmarks but challenging to design for AI agents. This problem is framed as defining a scoring function f that maps a task description $\mathcal{T}$ and an agent's trajectory $traj = \{s_0, a_0, s_1, a_1, \ldots, s_T\}$ to a normalized score reflecting the trajectory's quality with respect to $\mathcal{T}$. In digital agent benchmarks, success rate based on annotated criteria is commonly used for tasks with clear, objective metrics (e.g., coding, OS operations). However, in open-ended games like Minecraft, defining a single programmatic metric is often infeasible. To address this, we propose a VLM-based multi-dimensional evaluation framework for MCU, comprising two components (Figure 3): (1) criteria generation: creating clear, task-specific evaluation dimensions; and (2) scoring with criteria: leveraging predefined criteria to infer quality scores from agent performance videos using VLM.

Criteria Generation Preliminary experiments reveal that directly prompting GPT-4o (as a VLM) to score trajectories without task-specific criteria leads to suboptimal performance (Table 1). To address this, we introduce a criteria generation pipeline that provides detailed scoring instructions. Specifically, we define six key dimensions for evaluating agent performance in Minecraft. Prior to evaluation, GPT-4o is prompted to generate task-specific criteria for each dimension. The dimensions are as follows, with an example shown in Figure 3 (bottom): (1) **Task Progress:** Assesses critical steps and factors required for task completion. (2) **Action Control:** Evaluates the avoidance of unrelated or unnecessary actions. (3) **Material Usage:** Measures the selection and application of materials. (4) **Task Efficiency:** Focuses on minimizing repetitions and optimizing strategies. (5) **Error Recognition:** Assesses the ability to identify and correct errors. (6) **Creative Attempts:** Recognizes innovative approaches in task execution. Please check Appendix G.1 for the prompts of criteria generation.

Scoring with Criteria As described in Section 2.2, the agent's rollout trajectories are recorded in video format. To balance resource efficiency and evaluative effectiveness under specific research conditions, we extract one frame every 30 frames from the video. During the evaluation phase, the sampled frames, along with task-specific criteria, are input into the VLM. The VLM assesses each dimension by identifying supporting evidence from the video, providing evidence and explanations, and subsequently assigning a score. For each criterion, we define five scoring intervals: *very poor*, *poor*, *fair*, *good*, and *excellent*, corresponding to scores of 0, 0.25, 0.5, 0.75, and 1, respectively. Please check Appendix G.5 for the prompts of scoring.

3. Experiments

In this section, we first demonstrate the effectiveness of *AutoEval*. Subsequently, we assess the capabilities of state-of-the-art agents using MCU and provide insights for the development of future open-ended Minecraft agents.

3.1. Validity of Automatic Evaluation

Experimental Setup To validate the effectiveness of automatic evaluation, we compare the results of automatic evaluation methods with human annotations. We also evaluate a baseline approach, MineCLIP (Fan et al., 2022), which also focuses on automatic evaluation. Our evaluation is conducted under two distinct settings: (1) *comparative evaluation*, where the quality of two trajectories of same task is compared; and (2) *absolute rating*, where a score is assigned to a single trajectory. For *AutoEval*, the comparative prediction for two trajectory is given by comparing the scores.

Dataset We collected 500 trajectories spanning 60 tasks, derived from both agent rollouts and human gameplay videos. To evaluate the quality of these trajectories, we engaged 20 expert Minecraft players to provide annotations. The players were tasked with performing both compara-

Table 1: F1 scores for predicting the better human-annotated trajectory across different task categories. The compared methods include MineCLIP (Fan et al., 2022), our *AutoEval* on both open-access models (MiniCPM (Yao et al., 2024), JarvisVLA (Li et al., 2025)) and closed API-based models (GPT-4o). The highest score for each task category is **bolded**.

Method	Survive	Build	Craft	Mine	Explore	Average
MineCLIP	11.0	45.0	44.0	73.0	0.0	34.6
AutoEval (MiniCPM)	65.0	43.0	80.0	59.0	53.0	60.0
AutoEval (JarvisVLA)	73.0	62.0	73.0	84.0	65.0	71.4
AutoEval (GPT-4o)	**100.0**	**85.0**	**62.0**	**73.0**	**100.0**	**84.0**

Table 2: F1 scores for predicting the better human-annotated trajectory across various dimensions (denoted using abbreviations).

Progress	Action	Error Rec.	Creative	Efficiency	Material	Average
84.0	96.0	86.0	100.0	92.0	91.0	**91.5**

tive evaluations and absolute ratings on randomly sampled trajectory pairs for the same task or individual trajectories. Each player contributed 1 hours of work. Details regarding the annotation process are provided in Appendix C.2.

Comparative Evaluation In this setting, participants are asked to vote on the comparative quality of the trajectory videos (denoted as *A* and *B*), selecting from the following options: *A is better*, *B is better*, *tie*, or *both are bad*. We filtered trajectory pairs annotated with first two options for automatic evaluation, resulting in 236 pairs. As shown in Table 1, our methodology exhibits a significant improvement over MineCLIP, a CLIP model (Radford et al., 2021) fine-tuned on large-scale Minecraft video frames. MineCLIP, however, struggles to capture complex event-level semantics with an average F1 score of only 34.6 (compared to 84.0 of *AutoEval*). For intricate tasks such as `craft`, which demand precise attention to detail and the recognition of subtle elements, the performance of *AutoEval* is slightly reduced. We leave further improvements in this for future work. Nevertheless, as demonstrated in Table 2, our automated evaluation metric achieves an average agreement rate of 91.5% with human assessments across all dimensions. This highlights the effectiveness of the criteria-enhanced *AutoEval* for multi-dimensional evaluations.

Absolute Rating For absolute rating, we collected a total of 227 individual ratings across approximately 50 distinct tasks. The Pearson and Kendall correlation coefficients between *AutoEval* and human ratings are presented in Table 3, demonstrating a strong overall positive correlation. However, the correlation varies across different dimensions. For instance, objective dimensions such as *task progress* exhibit high agreement between human evaluators and VLM assessments, with a Pearson correlation of 0.78. In contrast, more subjective dimensions, such as *creativity*, show a lower correlation. Additionally, we compute the inter-rater agreement for scoring the same trajectory, revealing a higher *Pearson correlation* for task progress (0.83) and a lower correlation for creativity (0.69).

Table 3: The Pearson correlation and Kendall's τ between *AutoEval* and human ratings across different dimensions.

Dimension	Pearson		Kendall's τ	
	Coefficient	P-value	Coefficient	P-value
Task Progress	0.78	1.70×10^{-22}	0.71	1.94×10^{-19}
Action	0.76	6.22×10^{-19}	0.67	1.68×10^{-16}
Error Recog.	0.68	1.10×10^{-12}	0.62	4.40×10^{-10}
Creativity	0.63	5.90×10^{-9}	0.56	1.18×10^{-7}
Efficiency	0.75	1.10×10^{-16}	0.67	1.02×10^{-14}
Material	0.70	2.28×10^{-16}	0.63	2.29×10^{-13}

3.2. Evaluating Existing Agents with MCU

Agents We evaluate four powerful foundation agents in Minecraft, all supported by MineStudio: (1) VPT (BC), a behavior cloning agent initially pre-trained on YouTube Minecraft videos and further fine-tuned on refined early-game contractor data; (2) VPT (RL), a reinforcement learning fine-tuned model based on VPT (BC) that maximizes the reward for obtaining diamonds in Minecraft; (3) STEVE-I (Lifshitz et al., 2023), which follows text instructions to complete tasks; and (4) GROOT (Cai et al., 2024c), which solves tasks demonstrated by a reference video. GROOT receives video instructions, STEVE-I receives text instructions, and VPT operates without instructions. We exclude agents that simplify the Minecraft environment by deviating from the native action space. More details on these models can be found in Cai et al. (2024c).

Experimental Setup While MCU enables scalable task evaluation without extensive human annotation, we carefully select a small yet diverse set of 30 atomic and 5 compositional tasks to illustrate our experimental conclusions without introducing excessive complexity. A comprehensive evaluation of a total of 150 tasks, including 90 randomly sampled atomic tasks and 60 compositional tasks, is deferred to Appendix E. The 30 atomic tasks are drawn from six major categories, ensuring inter-task diversity while maintaining a moderate difficulty level suitable for Minecraft junior players. The difficulty of each task is of *simple* mode, and each task is evaluated using 30 random seeds. To highlight the importance of intra-task diversity, we select six atomic tasks (1-2 from each category), assessing both *simple* and *hard* modes of these tasks seperately. Each difficulty level is evaluated with 10 random seeds.

3.2.1. INTER-TASK GENERALIZATION

We present the *AutoEval*-generated "task progress" scores for the inter-task generalization experiments in Table 4, while the multi-dimensional performance, averaged over all tasks, is visualized in Figure 1c. Notably, the task progress metric is closely related to the commonly used "task suc-

Table 4: Task progress for 35 tasks. Performance table across all tasks in simple mode. **Abbreviations: Exp** = exploring, **FD** = find diamond, **FF** = find forest, **FV** = find villages, **CM** = climb mountain; **Crv** = carve, **CO** = compose obsidian, **Drk** = drink, **F&S** = flint&steel, **Slp** = sleep; **Ck** = cake, **Clk** = clock, **CT** = craft table, **DS** = diamond sword, **Ld** = ladder; **DO** = diamond ore, **Dt** = dirt, **Gr** = grass, **IO** = iron ore, **Ob** = obsidian; **End** = enderman, **Shp** = sheep, **Skl** = skeletons, **Spd** = spiders, **Zmb** = zombies; **BP** = build pillar, **Cv** = cave, **NP** = nether portal, **SG** = snow golem; **Wf** = waterfall; **CTS** = crafting table from scratch, **MDS** = mine diamond from scratch, **D&S** = dye and shear sheep, **T&P** = till and plant seeds, **PAG** = prepare a gift. Best values within the same task are **bolded**.

Agent	Navigation Task						Tool-use Task						Crafting Task					
	Exp	FD	FF	FV	CM	**Avg**	Crv	CO	Drk	F&S	Slp	**Avg**	Ck	Clk	CT	DS	Ld	**Avg**
GROOT	0.90	0.56	0.75	0.60	**0.60**	0.72	0.20	**0.10**	**0.40**	0.10	**0.85**	**0.33**	0.35	0.60	0.45	**0.75**	0.25	0.48
Steve-I	**0.95**	0.50	**0.95**	**0.90**	0.35	0.73	**0.45**	0.00	0.35	**0.20**	0.10	0.22	**0.80**	**0.70**	0.45	0.20	**0.70**	**0.57**
VPT (BC)	0.90	**0.65**	0.87	0.75	**0.60**	**0.75**	0.25	0.00	**0.40**	0.10	0.45	0.24	0.45	0.35	0.30	0.50	0.45	0.41
VPT (RL)	0.70	0.58	0.55	0.50	0.35	0.54	0.15	**0.10**	0.35	0.15	0.25	0.20	0.70	0.62	**0.50**	0.30	0.62	0.55

Agent	Mining Task						Combating Task						Building Task						Compositional Task					
	DO	Dt	Gr	IO	Ob	**Avg**	End	Shp	Skl	Spd	Zmb	**Avg**	BP	Cv	NP	SG	Wf	**Avg**	CTS	MDS	D&S	T&P	PAG	**Avg**
GROOT	**0.81**	0.70	0.90	0.56	**0.40**	**0.67**	**0.30**	0.50	**0.56**	0.50	**0.75**	0.53	0.40	**0.20**	**0.45**	**0.65**	0.15	**0.38**	0.45	**0.71**	0.05	**0.50**	0.19	0.23
Steve-I	0.35	**0.85**	**0.95**	**0.20**	0.35	0.54	0.05	0.30	0.40	**0.75**	0.42	**0.54**	**0.60**	0.10	0.30	0.05	0.05	0.13	0.45	0.35	**0.30**	0.20	**0.30**	0.13
VPT (BC)	0.30	0.30	0.50	0.15	0.38	0.33	0.25	**0.55**	0.55	0.35	0.50	0.36	0.00	0.02	0.35	0.00	0.20	0.11	0.30	0.30	0.10	0.00	0.25	0.13
VPT (RL)	0.15	0.35	0.37	0.05	0.35	0.22	0.15	0.35	0.35	0.25	0.30	0.28	0.10	0.02	0.40	0.13	**0.25**	0.23	**0.90**	0.50	0.00	0.00	**0.30**	**0.34**

Table 5: Performance changes of GROOT for selected tasks in simple and hard modes. Each result is averaged over 10 seeds.

Task	**Task Progress**			**Action Control**		
	Simple	**Hard**	Δ	**Simple**	**Hard**	Δ
Build Nether Portal	0.45	0.30	-0.15	0.50	0.40	-0.10
Mine Iron Ore	0.56	0.60	0.04	0.44	0.55	0.11
Craft to Cake	0.35	0.32	-0.03	0.31	0.25	-0.06
Combat Skeletons	0.56	0.30	-0.26	0.25	0.20	-0.05
Carve Pumpkin	0.20	0.15	-0.05	0.35	0.25	-0.10
Sleep in bed	0.85	0.50	-0.35	0.40	0.30	-0.10
Average	0.50	0.36	-0.14	0.38	0.33	-0.05

cess rate" in agent benchmarks. However, unlike success rate, which is a binary (0-1) outcome-based criterion, task progress is a process-supervised metric. This allows for a more fine-grained assessment of an agent's performance beyond simply determining whether a task was completed.

We observe that agents often struggle to reliably complete many tasks (e.g., `combat` and `build`). However, they still exhibit nonzero task progress, indicating partial progress as determined by the LLM. This suggests that while agents may not always achieve full task success, they are capable of making incremental advancements toward task completion. Among the evaluated agents, we note that VPT (RL), which is specifically RL-tuned to maximize rewards for obtaining diamonds, performs well on tasks aligned with this objective (e.g., crafting a crafting table from scratch). However, it significantly underperforms on tasks unrelated to its target. This highlights the importance of inter-task diversity in assessing the generalizability of agents. While generalist agents such as GROOT and Steve-I demonstrate better open-endedness, they struggle with specific tasks (e.g., crafting particular items), indicating that future efforts should focus on improving performance in these areas. Additionally, we observe that task progress for compositional tasks is lower than for atomic tasks, underscoring the persistent challenge of MCU. Furthermore, as shown in Figure 1c, current agents fall short in creativity, error recognition, and efficiency, suggesting important directions for future research aimed at improving these aspects.

3.2.2. Intra-task Generalization

The task progress and action control performance of intra-task generalization experiments for GROOT, the best agent among the compared agents, are presented in Table 5. The results indicate that for the same task, slight changes in the task situation lead to a significant drop in task progress performance. For example, consider the very simple task of `sleep in bed`, which only requires the agent to identify a bed and right-click the mouse. In *simple* mode, the bed is placed directly in front of the agent on a plain with no surrounding objects. However, in *hard* mode, both the bed and the agent are inside a house, requiring the agent to correctly identify the bed and interact with it. We observed failure cases where the agent mistakenly interacted with a chest in the room or left the house instead. This suggests that GROOT does not learn the skill robustly, a phenomenon not previously reported in the Minecraft agent literature.

Nevertheless, some tasks exhibited relatively stable performance across difficulty levels, such as `mine iron ore`, `craft to cake`, and `combat skeletons`. This may be because certain categories of tasks (e.g., building, tool-use, and crafting) are more susceptible to difficulty variations due to increased environmental complexity, while others are less affected. Overall, enhancing intra-task diversity of tasks is beneficial to test the robust generalzability of agents.

4. Related Work

Benchmarks in Minecraft MineDojo (Fan et al., 2022) introduces a suite of 1,560 creative tasks defined by natural language instructions. However, it suffers from significant redundancy and overly complex tasks, complicating practical evaluation, as shown in Figure 2. BEDD (Milani et al., 2023) defines five tasks covering various aspects of Minecraft, primarily designed for the MineRL BASALT competition. By decomposing the evaluation framework, BEDD facilitates detailed assessments of agent performance across subgoals and attributes such as human likeness. However, its reliance on human ratings limits scalability. Other works on Minecraft agents (Wang et al., 2023d;c; Cai et al., 2024c; Yuan et al., 2023; Baker et al., 2022; Lifshitz et al., 2023) lack a unified benchmark, with each agent evaluated on its own task set. We argue that establishing a standardized benchmark is a high priority for advancing Minecraft agent development.

Open-ended Agents in Minecraft Many agents have been developed to interact with Minecraft environments. Some methods focus on using Imitation Learning or Reinforcement Learning to learn various skills in open-world Minecraft to complete **short-horizon tasks** (Baker et al., 2022; Cai et al., 2024c; Lifshitz et al., 2023; Jiang et al., 2025; Zhao et al., 2024a; Cai et al.; Yuan et al., 2024; Cai et al., 2024b; Jiang & Lu, 2025). Baker et al. (2022) generates action labels from Minecraft videos on YouTube using IDM and performs imitation learning to obtain an unconditional policy capable of completing various tasks. Lifshitz et al. (2023) uses MineCLIP as a text encoder to obtain a text-conditioned multitask policy based on unconditional VPT (Baker et al., 2022). Minecraft also supports the testing of **long-horizon programmatic tasks**, so some methods accomplish long-horizon tasks by using large language models as planners, combined with skill policies (Wang et al., 2023d; Zhou et al., 2024a; Chen & Gao, 2024; Yuan et al., 2023; Qin et al., 2024; Zheng et al., 2023b). Wang et al. (2023d) has designed an agent pipeline based on GPT, which interactively completes tasks from environmental feedback through the self-explanation and zero-shot planning capabilities of LLMs. In order to further enhance the long-horizon capabilities of LLM Agents, some methods have explored efficient explicit memory mechanisms to support agents retrieve and improve from previous interaction trajectories (Wang et al., 2023c; Zhu et al., 2023; Wang et al., 2023a; Park et al., 2024; Li et al., 2024). Unlike designing complex agent pipelines, the rise of VLA (Zitkovich et al., 2023; Kim et al., 2024) has inspired researchers to use end-to-end VLM as a policy to directly fulfill human instructions in Minecraft (**?**Zhao et al., 2024b). Some methods use a code-as-policy approach (Liang et al., 2022), using MineFlayer (PrismarineJS, 2024) as a language-conditioned policy, combined with a pretrained LLM to accomplish various tasks to avoid agents trapped on short-horizon skills lacking when executing long-horizon tasks (Wang et al., 2023a; Yang et al., 2025; Liu et al., 2024b;a). Finally, there are some methods focused on completing **open-ended creative tasks** in Minecraft, such as building and decoration, which differ from the traditional programmatic object-centric tasks and often cannot be directly defined by rule-based rewards (Guo et al., 2024; Zhang et al., 2023).

LLM-as-a-Judge The advancement of large language models (LLMs) has demonstrated remarkable performance in instruction following, query understanding, and response generation. This capability has motivated the use of LLMs as judges (Zheng et al., 2023a), leveraging their ability to score and rank model outputs. The strong performance of LLMs (Brown et al., 2020), combined with well-designed assessment pipelines (Li et al., 2023; Beigi et al., 2024; Bai et al., 2024), enables fine-grained and detailed judgments for various evaluation applications, addressing the limitations of traditional evaluation methods that require extensive human annotation. Recent efforts have also explored the use of LLMs and vision-language models (VLMs) to evaluate AI agents (Pan et al., 2024; Zhuge et al., 2024). Compared to previous work, MCU is the first to apply this paradigm to open-ended game agent task generation and evaluation. We argue that LLM-based task generators have the potential to create diverse tasks that are crucial for evaluating open-ended game agents, given the inherent open-endedness of LLMs (Hughes et al., 2024). Furthermore, using the same LLM (or VLM) as a judge ensures more consistent evaluation criteria compared to crowdsourcing approaches (Zhou et al., 2023).

5. Conclusion

We introduce *Minecraft Universe* (MCU), a scalable evaluation framework for open-ended game agents in Minecraft. MCU features 3,452 diverse and composable atomic tasks, a dynamic task composition mechanism to ensure sustained challenges, and an automated evaluation system with over 90% human alignment. Empirical results indicate that state-of-the-art agents struggle with tasks exhibiting high inter-task and intra-task diversity. To support standardized benchmarking, we release MCU-Turbo, a curated subset of 100 tasks with structured difficulty settings, as detailed in Appendix F. We hope MCU serve as robust foundations for advancing open-world agent research.

Acknowledgement

This work was funded in part by the National Science and Technology Major Project (2022ZD0114902), and National Natural Science Foundation of China (62376031). We acknowledge other members in CraftJarvis team for their data labeling support and constructive discussions, and thank Dr. Wenzheng Feng for his valuable insights.

Impact Statement

This paper presents research aimed at advancing the field of open-ended game agents, contributing to the broader understanding of adaptive and autonomous artificial intelligence in dynamic environments. While our work has the potential to influence various domains, including AI-driven decision-making and interactive entertainment, we do not identify any specific societal consequences that require immediate attention or emphasis at this time.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 5

Alaniz, S. Deep reinforcement learning with model learning and monte carlo tree search in minecraft. *arXiv preprint arXiv:1803.08456*, 2018. 2

Arumugam, D., Lee, J. K., Saskin, S., and Littman, M. L. Deep reinforcement learning from policy-dependent human feedback. *arXiv preprint arXiv:1902.04257*, 2019. 2

Bai, Y., Ying, J., Cao, Y., Lv, X., He, Y., Wang, X., Yu, J., Zeng, K., Xiao, Y., Lyu, H., et al. Benchmarking foundation models with language-model-as-an-examiner. In *Advances in Neural Information Processing Systems*, volume 36, 2024. 9

Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. Video pretraining (vpt): Learning to act by watching unlabeled online videos. In *Advances in Neural Information Processing Systems*, 2022. 9, 18

Beigi, A., Tan, Z., Mudiam, N., Chen, C., Shu, K., and Liu, H. Model attribution in llm-generated disinformation: A domain generalization approach with supervised contrastive learning. In *2024 IEEE 11th International Conference on Data Science and Advanced Analytics*, pp. 1–10. IEEE, 2024. 9

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. In *Advances in neural information processing systems*, volume 33, pp. 1877–1901, 2020. 9

Bruce, J., Dennis, M., Edwards, A., Parker-Holder, J., Shi, Y., Hughes, E., Lai, M., Mavalankar, A., Steigerwald, R., Apps, C., Aytar, Y., Bechtle, S., Behbahani, F., Chan, S., Heess, N., Gonzalez, L., Osindero, S., Ozair, S., Reed, S., Zhang, J., Zolna, K., Clune, J., de Freitas, N., Singh, S., and Rocktäschel, T. Genie: Generative interactive environments, 2024. URL https://arxiv.org/abs/2402.15391. 1

Cai, S., Zhang, B., Wang, Z., Ma, X., Liu, A., and Liang, Y. Groot-1.5: Learning to follow multi-modal instructions from weak supervision. In *Multi-modal Foundation Model meets Embodied AI Workshop@ ICML2024*. 9

Cai, S., Mu, Z., He, K., Zhang, B., Zheng, X., Liu, A., and Liang, Y. Minestudio: A streamlined package for minecraft ai agent development, 2024a. URL https://arxiv.org/abs/2412.18293. 3, 4

Cai, S., Zhang, B., Wang, Z., Lin, H., Ma, X., Liu, A., and Liang, Y. Groot-2: Weakly supervised multimodal instruction following agents. *arXiv preprint arXiv:2412.10410*, 2024b. 9

Cai, S., Zhang, B., Wang, Z., Ma, X., Liu, A., and Liang, Y. Groot: Learning to follow instructions by watching gameplay videos. In *The Twelfth International Conference on Learning Representationss*, 2024c. 1, 5, 7, 9, 18

Chen, J. Y. and Gao, T. Apt: Architectural planning and text-to-blueprint construction using large language models for open-world agents. *arXiv preprint arXiv:2411.17255*, 2024. 9

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020. 1

Dubois, Y., Li, C. X., Taori, R., Zhang, T., Gulrajani, I., Ba, J., Guestrin, C., Liang, P. S., and Hashimoto, T. B. Alpacafarm: A simulation framework for methods that learn from human feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2024. 2

Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D.-A., Zhu, Y., and Anandkumar, A. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *Advances in Neural Information Processing Systems Datasets and Benchmarks*, 2022. 1, 2, 4, 5, 6, 7, 8, 18

Guo, Y., Peng, S., Guo, J., Huang, D., Zhang, X., Zhang, R., Hao, Y., Li, L., Tian, Z., Gao, M., et al. Luban: Building open-ended creative agents via autonomous embodied verification. *arXiv preprint arXiv:2405.15414*, 2024. 9

Guss, W. H., Codel, C., Hofmann, K., Houghton, B., Kuno, N., Milani, S., Mohanty, S., Liebana, D. P., Salakhutdinov, R., Topin, N., et al. Neurips 2019 competition: the minerl competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 1 (8), 2019a. 4

Guss, W. H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M., and Salakhutdinov, R. Minerl: A large-scale dataset of minecraft demonstrations. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019b. 2

Hafner, D. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021. 1

Huang, D., Bu, Q., Zhang, J. M., Luck, M., and Cui, H. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023. 1

Hughes, E., Dennis, M. D., Parker-Holder, J., Behbahani, F., Mavalankar, A., Shi, Y., Schaul, T., and Rocktäschel, T. Position: Open-endedness is essential for artificial superhuman intelligence. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. 9

Jiang, H. and Lu, Z. Visual grounding for object-level generalization in reinforcement learning. In *European Conference on Computer Vision*, pp. 55–72. Springer, 2025. 9

Jiang, H., Yue, J., Luo, H., Ding, Z., and Lu, Z. Reinforcement learning friendly vision-language model for minecraft. In *European Conference on Computer Vision*, pp. 1–17. Springer, 2025. 9

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. 1

Jin, E., Hu, J., Huang, Z., Zhang, R., Wu, J., Fei-Fei, L., and Martín-Martín, R. Mini-behavior: A procedurally generated benchmark for long-horizon decision-making in embodied ai. *arXiv preprint arXiv:2310.01824*, 2023. 4

Kejriwal, M., Kildebeck, E., Steininger, R., and Shrivastava, A. Challenges, evaluation and opportunities for open-world learning. *Nature Machine Intelligence*, 6(6):580–588, 2024. 1

Kim, M. J., Pertsch, K., Karamcheti, S., Xiao, T., Balakrishna, A., Nair, S., Rafailov, R., Foster, E., Lam, G., Sanketi, P., et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. 9

Li, M., Wang, Z., He, K., Ma, X., and Liang, Y. Jarvis-vla: Post-training large-scale vision language models to play visual games with keyboards and mouse. *arXiv preprint arXiv:2503.16365*, 2025. 7

Li, R., Patel, T., and Du, X. Prd: Peer rank and discussion improve large language model based evaluations. *Transactions on Machine Learning Research*, 2023. 9

Li, Z., Xie, Y., Shao, R., Chen, G., Jiang, D., and Nie, L. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. *arXiv preprint arXiv:2408.03615*, 2024. 9

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022. 9

Lifshitz, S., Paster, K., Chan, H., Ba, J., and McIlraith, S. Steve-1: A generative model for text-to-behavior in minecraft. In *Advances in Neural Information Processing Systems*, 2023. 7, 9, 25

Lin, K. Q., Li, L., Gao, D., Yang, Z., Bai, Z., Lei, W., Wang, L., and Shou, M. Z. Showui: One vision-language-action model for generalist gui agent. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024. 1

Liu, S., Li, Y., Zhang, K., Cui, Z., Fang, W., Zheng, Y., Zheng, T., and Song, M. Odyssey: Empowering minecraft agents with open-world skills. *arXiv preprint arXiv:2407.15325*, 2024a. 9

Liu, S., Yuan, H., Hu, M., Li, Y., Chen, Y., Liu, S., Lu, Z., and Jia, J. Rl-gpt: Integrating reinforcement learning and code-as-policy. *arXiv preprint arXiv:2402.19299*, 2024b. 9

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019. 2

Milani, S., Kanervisto, A., Ramanauskas, K., Schulhoff, S., Houghton, B., and Shah, R. Bedd: The minerl basalt evaluation and demonstrations dataset for training and benchmarking agents that solve fuzzy tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 3*, 2023. 8, 20

Nasiriany, S., Maddukuri, A., Zhang, L., Parikh, A., Lo, A., Joshi, A., Mandlekar, A., and Zhu, Y. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems*, 2024. 1

Oh, J., Chockalingam, V., Lee, H., et al. Control of memory, active perception, and action in minecraft. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2016. 2

Olan, M. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*, 19(2):319–328, 2003. 4

Pan, J., Zhang, Y., Tomlin, N., Zhou, Y., Levine, S., and Suhr, A. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*, 2024. 1, 9

Park, J., Cho, J., and Ahn, S. Mr. steve: Instruction-following agents in minecraft with what-where-when memory. *arXiv preprint arXiv:2411.06736*, 2024. 9

Parmar, J., Chouhan, S., Raychoudhury, V., and Rathore, S. Open-world machine learning: applications, challenges, and opportunities. *ACM Computing Surveys*, 2023. 1

PrismarineJS. mineflayer: Create Minecraft bots with a powerful, stable, and high level JavaScript API. https://github.com/PrismarineJS/mineflayer, 2024. 9

Qin, Y., Zhou, E., Liu, Q., Yin, Z., Sheng, L., Zhang, R., Qiao, Y., and Shao, J. Mp5: A multi-modal open-ended embodied system in minecraft via active perception. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 9

Qin, Y., Ye, Y., Fang, J., Wang, H., Liang, S., Tian, S., Zhang, J., Li, J., Li, Y., Huang, S., et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025. 1

Raad, M. A., Ahuja, A., Barros, C., Besse, F., Bolt, A., Bolton, A., Brownfield, B., Buttimore, G., Cant, M., Chakera, S., et al. Scaling instructable agents across many simulated worlds. *arXiv preprint arXiv:2404.10179*, 2024. 1

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021. 7

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

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. 2, 6

Shu, T., Xiong, C., and Socher, R. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*, 2017. 2

Standish, R. K. Open-ended artificial evolution. *International Journal of Computational Intelligence and Applications*, 3(02):167–175, 2003. 2

Stanley, K. O., Lehman, J., and Soros, L. Open-endedness: The last grand challenge you've never heard of. https://www.uber.com/blog/research, 2017. 2

Team, O. E. L., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021. 1

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D., and Mannor, S. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017. 2

Trott, A., Zheng, S., Xiong, C., and Socher, R. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. In *Advances in Neural Information Processing Systems*, volume 32, 2019. 2

Udagawa, H., Narasimhan, T., and Lee, S.-Y. Fighting zombies in minecraft with deep reinforcement learning. Technical report, Technical report, Technical report, Stanford University, 2016. 2

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. In *Agent Learning in Open-Endedness Workshop*, 2023a. 9

Wang, J., Xu, H., Ye, J., Yan, M., Shen, W., Zhang, J., Huang, F., and Sang, J. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024. 1

Wang, L., Ling, Y., Yuan, Z., Shridhar, M., Bao, C., Qin, Y., Wang, B., Xu, H., and Wang, X. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*, 2023b. 1

Wang, Z., Cai, S., Liu, A., Jin, Y., Hou, J., Zhang, B., Lin, H., He, Z., Zheng, Z., Yang, Y., Ma, X., and Liang, Y. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *ArXiv*, abs/2311.05997, 2023c. URL https://api.semanticscholar.org/CorpusID:265129059. 9, 18

Wang, Z., Cai, S., Liu, A., Ma, X., and Liang, Y. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023d. 9, 18

Xu, F. F., Vasilescu, B., and Neubig, G. In-ide code generation from natural language: Promise and challenges. *ACM Transactions on Software Engineering and Methodology*, 31(2):1–47, 2022. 1

Yang, J., Dong, Y., Liu, S., Li, B., Wang, Z., Tan, H., Jiang, C., Kang, J., Zhang, Y., Zhou, K., et al. Octopus: Embodied vision-language programmer from environmental feedback. In *European Conference on Computer Vision*, pp. 20–38. Springer, 2025. 9

Yang, Y., Sun, F.-Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 2

Yao, Y., Yu, T., Zhang, A., Wang, C., Cui, J., Zhu, H., Cai, T., Li, H., Zhao, W., He, Z., et al. Minicpm-v: A gpt-4v level mllm on your phone. *arXiv preprint arXiv:2408.01800*, 2024. 7

Yuan, H., Zhang, C., Wang, H., Xie, F., Cai, P., Dong, H., and Lu, Z. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*, 2023. 9, 18

Yuan, H., Mu, Z., Xie, F., and Lu, Z. Pre-training goal-based models for sample-efficient reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. 9

Zhang, C., Cai, P., Fu, Y., Yuan, H., and Lu, Z. Creative agents: Empowering agents with imagination for creative tasks. *arXiv preprint arXiv:2312.02519*, 2023. 9

Zhang, Z., Kayacan, E., Thompson, B., and Chowdhary, G. High precision control and deep learning-based corn stand counting algorithms for agricultural robot. *Autonomous Robots*, 44(7):1289–1302, 2020. 4

Zhao, G., Lian, K., Lin, H., Fu, H., Fu, Q., Cai, S., Wang, Z., and Liang, Y. Optimizing latent goal by learning from trajectory preference. *arXiv preprint arXiv:2412.02125*, 2024a. 9

Zhao, Z., Ma, K., Chai, W., Wang, X., Chen, K., Guo, D., Zhang, Y., Wang, H., and Wang, G. Do we really need a complex agent system? distill embodied agent into a single model. *arXiv preprint arXiv:2404.04619*, 2024b. 9

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023a. 9

Zheng, S., Liu, J., Feng, Y., and Lu, Z. Steve-eye: Equipping llm-based embodied agents with visual perception in open worlds. *arXiv preprint arXiv:2310.13255*, 2023b. 9

Zhou, E., Qin, Y., Yin, Z., Huang, Y., Zhang, R., Sheng, L., Qiao, Y., and Shao, J. Minedreamer: Learning to follow instructions via chain-of-imagination for simulated-world control. *arXiv preprint arXiv:2403.12037*, 2024a. 9

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. 1, 9

Zhou, Z., Atreya, P., Lee, A., Walke, H., Mees, O., and Levine, S. Autonomous improvement of instruction following skills via foundation models. *arXiv preprint arXiv:2407.20635*, 2024b. 1

Zhu, X., Chen, Y., Tian, H., Tao, C., Su, W., Yang, C., Huang, G., Li, B., Lu, L., Wang, X., et al. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023. 9

Zhuge, M., Zhao, C., Ashley, D., Wang, W., Khizbullin, D., Xiong, Y., Liu, Z., Chang, E., Krishnamoorthi, R., Tian, Y., et al. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*, 2024. 9

Zitkovich, B., Yu, T., Xu, S., Xu, P., Xiao, T., Xia, F., Wu, J., Wohlhart, P., Welker, S., Wahid, A., Vuong, Q., Vanhoucke, V., Tran, H., Soricut, R., Singh, A., Singh, J., Sermanet, P., Sanketi, P. R., Salazar, G., Ryoo, M. S., Reymann, K., Rao, K., Pertsch, K., Mordatch, I., Michalewski, H., Lu, Y., Levine, S., Lee, L., Lee, T.-W. E., Leal, I., Kuang, Y., Kalashnikov, D., Julian, R., Joshi, N. J., Irpan, A., Ichter, B., Hsu, J., Herzog, A., Hausman, K., Gopalakrishnan, K., Fu, C., Florence, P., Finn, C., Dubey, K. A., Driess, D., Ding, T., Choromanski, K. M., Chen, X., Chebotar, Y., Carbajal, J., Brown, N., Brohan, A., Arenas, M. G., and Han, K. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The 7th Conference on Robot Learning*, 2023. 9

A. Minecraft Environment Setting

In the regular Minecraft game, the server (or ”world”) always runs at 20Hz while the client’s rendering speed can typically reach 60-100Hz. To ensure consistency with the server, the frame rate is fixed at 20 fps for the client. The action and observation spaces in our environment are identical to what a human player can operate and observe on their device when playing the game. These details will be further explained in subsequent subsections. Additionally, diagnostic information such as in-game stats, contents of the agent’s inventory, and whether any in-game GUI is open is provided by the environment. This information can only be used for tracking, recording, and evaluating purposes but cannot serve as inputs to evaluated agents.

A.1. Minecraft Game World Setting

We have chosen to conduct the test in Minecraft version 1.16.5’s survival mode. During this open-world experiment, the agent may encounter situations that result in its death, such as being burned by lava or a campfire, getting killed by hostile mobs, or falling from great heights. When this happens, the agent will lose all its items and respawn at a random location near its initial spawn point within the same Minecraft world or at the last spot it attempted to sleep. Importantly, even after dying, the agent retains knowledge of its previous deaths and can adjust its actions accordingly since there is no masking of policy state upon respawn.

Figure 4: Minecraft game observation.

A.2. Observation Space

The observation space for a human player is limited to the raw pixels visible on the display screen. It does not include any hidden information from the game world, such as hidden blocks or nearby mobs. Additionally, any information contained in the pixels must be perceived by the model rather than directly given, including inventories and health indicators. Human players can access this information by pressing F3, which should be considered part of the game screen. There are no restrictions on optional parameters that human players can adjust in the display settings, such as field of view, GUI scale (controlling the size of in-game GUI), and brightness. The rendering resolution of Minecraft is 640x360; however, it is recommended to resize images to lower resolutions for better discernibility and computational efficiency.

A.3. Action Space

The action space is also consistent with human-playing settings, i.e., mouse and keyboard controls. These actions include key presses, mouse movements, and clicks. The specific binary actions that are triggered by keypress are shown in Table Table 6. In addition to actions triggered by keypresses, the action space also includes mouse movements. Similar to human gameplay, when there are no in-game GUIs open, moving the mouse along the X and Y axes changes the agent’s yaw and pitch respectively. However, when a GUI is open, camera actions shift the position of the mouse cursor. The mouse movements are relative and adjust their position or camera angle based on their current state. More details can be found at https://minecraft.fandom.com/wiki/Controls

Table 6: Binary actions included in the action space.

Action	Human action	Description
forward	W key	Move forward.
back	S key	Move backward.
left	A key	Strafe left.
right	D key	Strafe right.
jump	space key	Jump.
inventory	E key	Open or close inventory and the 2x2 crafting grid.
sneak	shift key	Move carefully in the current direction of motion. In the GUI it acts as a modifier key: when used with an attack it moves item from/to the inventory to/from the Hotbar, and when used with craft it crafts the maximum number of items possible instead of just 1.
sprint	ctrl key	Move fast in the current direction of motion.
attack	left mouse button	Attack; In GUI, pick up the stack of items or place the stack of items in a GUI cell; when used as a double click (attack - no attack - attack sequence), collect all items of the same kind present in inventory as a single stack.
use	right mouse button	Place the item currently held or use the block the player is looking at. In GUI, pick up the stack of items or place a single item from a stack held by the mouse.
drop	Q key	Drop a single item from the stack of items the player is currently holding. If the player presses ctrl-Q then it drops the entire stack. In the GUI, the same thing happens except for the item the mouse is hovering over.
hotbar.[1-9]	keys 1 – 9	Switch active item to the one in a given hotbar cell.
show debug screen	F3 key	See the chunk cache, the memory usage, various parameters, the player's map coordinates, and a graph that measures the game's current frame rate.

A.4. Why Minecraft is Suitable for Open-Ended Agent?

A.4.1. State Space Calculation

Minecraft has an enormous state space. Here, we will estimate how many possibilities are contained within the full state space of Minecraft.

Block State The world in Minecraft consists of different types of blocks, each with potentially multiple states. Let B be the number of block types, and W be the world size, which is approximately $30M \times 30M \times 384$, i.e., the total number of blocks in the world.

The formula for the number of block states is:

$$\text{Block States} = B^W$$

Substituting the values $B = 500$ and $W \approx 3.46 \times 10^{18}$, we get:

$$\text{Block States} = 500^{3.46\times10^{18}}$$

In Procgen environments, the number of block states is estimated based on a pixel space. Block states are calculated based on how many different visual representations (pixels) can be generated. In ALE, the levels are artificially fixed and finite.

Table 7: Comparison of State Space between Different Benchmarks. Unlike other benchmarks that isolate tasks within separate state spaces, which may simplify the learning process, Minecraft integrates all tasks into a shared state space. This requires the agent to generalize without relying on memorizing specific environments. The initial seed reflects the inherent generation capability of procedural generation, while the final state reflects the full range of possibilities contained within the entire game. The final state space of Minecraft (Approx. $10^{10^{20}}$) far exceeds the number of atoms in the universe (Approx. 10^{80}). The constraint is 10^{-2}, assuming that only one percent of the combinations are possible.

Dimension	**Minecraft**	**Procgen/CoinRun**	**ALE/Pitfall**
Initial Seed	2^{64}	2^{32}	Fixed
World Size	$30M \times 30M \times 384 \approx 3.46 \times 10^{18}$	64×64	Predefined Layout
Block Types	500	3	3-5
Block States	$500^{3.46\times10^{18}}$	$3^{64\times64}$	limited
Entities	Mobs: 30+ Types, Health: 0–20 Animals: 30+ Types, Health: 0–10 Villagers: 13 Professions, 5 Levels, 100 Trades	Obstacles: 3 Classes	Obstacles: 8 Classes
Entity Count	10^7	≤ 20	≤ 10
Entity States	$(30 \times 20 + 30 \times 10 + 13 \times 5 \times 100)^{10^7}$ $\approx 2.57 \times 10^{75\times10^7}$	$3^{20} \approx 3.49 \times 10^9$	Limited
Inventory States	36 Slots, 500+ Item Types, Max Stack 64 $(500 \times 64)^{36}$	N/A	N/A
Final State Space *Block × Entity × Inventory States × Constraint*	**Approx.** $10^{10^{20}}$	**Approx.** 10^{99}	Limited, Predefined Levels

Entity State Minecraft contains a wide variety of entities including animals, mobs, and villagers, each with different properties and states. Mobs have more than 30 types, health ranging from 0 to 20, animals have more than 30 types with health from 0 to 10, and villagers have 13 professions, 5 levels, and 100 trades. The entire map contains approximately 10^7 entities. Consider all possible combinations, the number of entity states is:

$$\text{Entity States} \approx (30 \times 20 + 30 \times 10 + 13 \times 5 \times 100)^{10^7} \approx 2.57 \times 10^{75\times10^7}$$

In Procgen environments like CoinRun, the number of entity states is typically smaller due to fewer types of entities and simpler interactions.

Inventory State Minecraft's inventory consists of 36 slots, each capable of holding a variety of items (over 500 types), with a maximum stack size of 64. The formula for the number of inventory states is:

$$\text{Inventory States} = (500 \times 64)^{36}$$

Final State Space The final state space is determined by the combination of block states, entity states, inventory states, and possible constraints (such as game rules). Assuming all these state spaces are independent, the final state space formula is:

$$\text{Final State Space} = \text{Block States} \times \text{Entity States} \times \text{Inventory States} \times \text{Constraint}$$

For Minecraft's final state space, assuming a constraint factor of 10^{-2}, we get:

$$\text{Final State Space} \approx 10^{10^{20}}$$

A.4.2. Further Attributes

Complexity Minecraft presents a highly complex environment composed of diverse elements, including blocks, creatures, terrain, and vegetation. This complexity challenges agents to learn adaptive behaviors across varied tasks, fostering generalization in a dynamic setting.

Open-endedness The open-world nature of Minecraft exposes agents to a vast range of environments, requiring exploration and adaptive navigation. The flexibility to define tasks of varying difficulty enables targeted evaluation of agent capabilities across diverse challenges.

Dynamism and Unpredictability Unlike static benchmarks, Minecraft features dynamic environmental changes such as day-night cycles, emergent entities, and varied terrain. Agents must develop adaptability and robust decision-making to handle unforeseen events, enhancing their generalization to real-world complexities.

Creativity and Innovation Minecraft supports open-ended tasks like construction and decoration, encouraging agents to explore diverse strategies for goal achievement. This fosters innovation and problem-solving in complex, unstructured settings.

Broad Challenge Coverage Minecraft serves as an ideal platform for training and evaluating generalist agents, presenting four key challenges: **Long-horizon Decision Making:** Tasks decompose into flexible subtask sequences, requiring agents to plan beyond immediate actions. For example, acquiring wool may involve killing sheep, crafting from string, or trading with villagers, demanding strategic foresight. **Precise Control:** Building and crafting require fine-grained movement and accurate object manipulation. Tasks like constructing a Nether portal necessitate precise block placement, challenging agents to handle high-dimensional action spaces with stability. **Out-of-distribution Generalization:** The dynamic environment introduces novel scenarios beyond training data. Agents must generalize to unseen conditions, such as avoiding hazards (e.g., lava) or adapting to ecosystem variations. **Compositional Generalization:** Agents should infer new task compositions from learned subskills. For instance, if trained to craft sticks from planks and ladders from sticks, they should generalize to crafting ladders from planks. The vast combinatorial task space in Minecraft makes compositional generalization a crucial challenge.

Community and Resources Minecraft's extensive community provides rich datasets, strategies, and problem-solving techniques. Open-source mods and plugins further enable controlled experimental setups for agent training.

Safe and Controlled Environment Minecraft offers a risk-free virtual world where researchers can precisely manipulate environmental parameters for reinforcement learning, ensuring reproducibility and safety in training agents.

B. Details of Task Generation

B.1. The Source of Atomic Tasks

Filtering from Existing Benchmarks We curated tasks from established benchmarks, including Skill-Forge (Cai et al., 2024c), MineDojo (Fan et al., 2022), and prior Minecraft research (Baker et al., 2022; Yuan et al., 2023; Wang et al., 2023d;c). To refine the selection, we performed deduplication to remove redundant tasks, excluded those too difficult for human players (e.g., constructing highly complex architectures or automated redstone circuits), and eliminated compositional tasks that could be decomposed into two or more atomic tasks.

Minecraft Wiki Resources Additional tasks were sourced from the official Minecraft Wiki, which categorizes various in-game activities. From these, we extracted executable tasks, focusing primarily on the `Advancement` page[3], which contains a curated list of diverse, engaging, and reasonably challenging tasks designed to enhance gameplay.

Synthesis from In-Game Information The Minecraft simulator defines various item properties, such as "craftable," "mineable," "eatable," and "breakable." Leveraging these definitions, we systematically generated atomic tasks, such as `craft item X` if `X` is craftable, and similarly for other properties. This method efficiently scales up the task set while ensuring comprehensive coverage of game elements.

LLM and Expert Brainstorming Beyond structured sources, we incorporated tasks generated through brainstorming sessions with expert Minecraft players and LLMs. This approach was particularly valuable for designing open-ended, creative tasks that pose real-world challenges for agents. Brainstorming was conducted in collaboration with a university Minecraft club.

Since most tasks are derived from the official Minecraft Wiki and in-game data, they are inherently reliable. Nevertheless, all tasks underwent rigorous validation through human inspection and automated scripts. The finalized atomic task list is provided as supplementary material alongside the code.

B.2. Task Configuration

In this section, we provide an overview of the key considerations for configuring a task, as introduced in Section 2.4. This section aims to offer an intuitive understanding of task configuration. For detailed implementation and real-world configurations, please refer to Minestudio Section 2.2 and our integrated codebase.

The initial state of a task encapsulates all the information an agent can utilize based on its intended plan to execute the task. This includes not only the valid input but also any information the agent can derive or perceive, such as the observed 2D pixels of the game scene, inventory items, and coordinates (which can be accessed in-game by pressing F3, particularly the Y-dimension). The inventory $\mathcal{I}$ consists of two components: the necessary items for completing the task, denoted as $\mathcal{I}_n$—without which the agent would not be able to plan and execute the task in a real game—and additional random items, denoted as $\mathcal{I}_r$. Our objective is to manipulate these variables while ensuring that the random elements closely align with the real in-game distribution.

B.2.1. Observation and Coordinate

For a fixed version of the Minecraft game, the observation and coordinate elements are determined by the world seed, the coordinate, and the facing direction. The world seed is entirely independent of other variables and can be chosen arbitrarily. The facing direction remains unchanged from its state before teleportation to the task scene, making it inherently random and not subject to manipulation.

Setting a coordinate as a valid spawn location for a given task requires satisfying certain preconditions, such as specific biome types or other constraints defined by the game. For example, in the `climb the mountain` task, the agent must spawn in a `stony shore` biome, while for a task that involves reaching a village, the spawn location should be near one.

To facilitate reproducible task execution, we also collect a series of coordinate locations for each selected seed, corresponding to the required preconditions. Each (seed, precondition) pair can be mapped to multiple possible locations, allowing flexibility across different tasks. Minestudio allows for initializing game environment given these predefined random seeds.

[3] `https://minecraft.wiki/w/Advancement`

B.2.2. INVENTORIES

The inventory $\mathcal{I}$ consists of two main components: $\mathcal{I}_n$, the essential items required to complete the task, and $\mathcal{I}_r$, a set of random items acting as distractors. Since multiple approaches may exist to accomplish the same goal, $\mathcal{I}_n$ is also treated as a random variable. For instance, an agent may use either an iron pickaxe or a diamond pickaxe to mine diamond ore. To ensure comprehensive testing, we incorporate a variety of possible item sets for $\mathcal{I}_n$.

Regarding $\mathcal{I}_r$, we adjust its presence based on task difficulty. For some tasks, we omit $\mathcal{I}_r$ to reduce complexity. In other cases, we introduce random initial inventories by sampling from game snapshots derived from VPT contractor data. This approach ensures that the test environment remains diverse while maintaining a realistic distribution of inventory items.

C. Human Annotation

C.1. Minecraft Quiz

To get an annotation for multi-dimensional task scores for trajectories used in our experiments (Section 3.2), we designed and distributed a questionnaire to confirm the annotators are familiar with Minecraft. The questionnaire is a quiz, containing five multiple-choice questions with 25 options to test their familiarity with Minecraft; each correctly answered option is worth 1 point. Then we filtered out the questionnaires with a correct rate of less than 75%, and then considered their investigation parts for the remaining questionnaires. The quiz is shown in Table 8. We distributed the questionnaires to the signed up Minecraft annotators, and all of the annotators passed the quiz.

Table 8: The quiz in our questionnaire (only 5 questions are presented), is used to judge the respondents' familiarity with Minecraft. The problems are adapted from Milani et al. (2023).

No.	Question	Options
1	A bed can	A. speed up the night. B. change the respawn location. C. be crafted from drops of a certain animal in the game. D. can be crafted by a furnace, but cannot be crafted by a crafting table.
2	You can acquire EXP when	A. killing hostile mobs. B. mining trees. C. jumping on a coal ore block. D. mining coal. E. enchanting a diamond sword.
3	What mobs can deal damage to the player?	A. Skeletons. B. Zombies. C. Sheep. D. Pigs. E. Creepers. F. Enderman.
4	What items can be eaten?	A. Apples. B. Dirt. C. Beef. D. Wheat. E. Breads. F. Spider eyes.
5	If you mine a block with a bare hand, what kinds of block can drop the corresponding item?	A. Wooden logs. B. Wooden planks. C. Iron ore. D. Coal ore.

C.2. Human Videos For Tasks

Human videos serve two purposes: they are used as reference videos for GROOT and for comparison with the trajectory videos generated by the agent models. For each task, we select three world seeds: 19961103, 20010501, and 12345. For each (task, seed) pair, we manipulate the controllable parameters as described above, resulting in three distinct environment configuration files. For each configuration file, we record a corresponding human video. Additionally, we designate the first configuration file of seed 19961103 as the reference video for GROOT.

C.3. Human rating system

In our dataset, there are a total of 60 tasks, each containing 10 rollout trajectories in video format. These videos capture gameplay records from either humans or various agents.

For the **absolute rating task**, we randomly select one task and present a corresponding video to human raters, who score it across six predefined dimensions.

For the **comparative evaluation task**, we randomly sample two different rollouts from the same task, referred to as *Video A* and *Video B*. These videos may both be from human players, both from agents, or one from a human and the other from an agent. Raters are then asked to compare *Video A* and *Video B* on each dimension to determine which one performs better.

The human rating interfaces are illustrated in Figure 5 and Figure 6. Taking the video comparison website as an example, it is designed to evaluate agent performance by displaying two videos side by side, enabling human raters to directly compare their behaviors within the same task. The interface consists of the following modules:

1. **Task Description Module**: Positioned at the top-right, this module specifies the task to be evaluated (e.g., *Survive Shield: Use a shield to ward off zombies*). It ensures that raters understand the objective before scoring.

2. **Video Display Module**: Two videos are presented side by side, each replaying an agent's gameplay. This layout allows raters to observe agent behaviors, mistakes, or innovative strategies in real-time.

3. **Scoring Panel**: Located below the videos, this panel enables raters to assess agent performance across six dimensions. For each dimension, raters can indicate which agent performed better, mark a tie, or specify that neither agent took relevant actions.

4. **Input and Submission Module**: At the top-center, an input box collects rater identifiers to ensure traceability. A *Submit* button at the bottom sends completed ratings to the database, contributing to the dataset used for benchmarking.

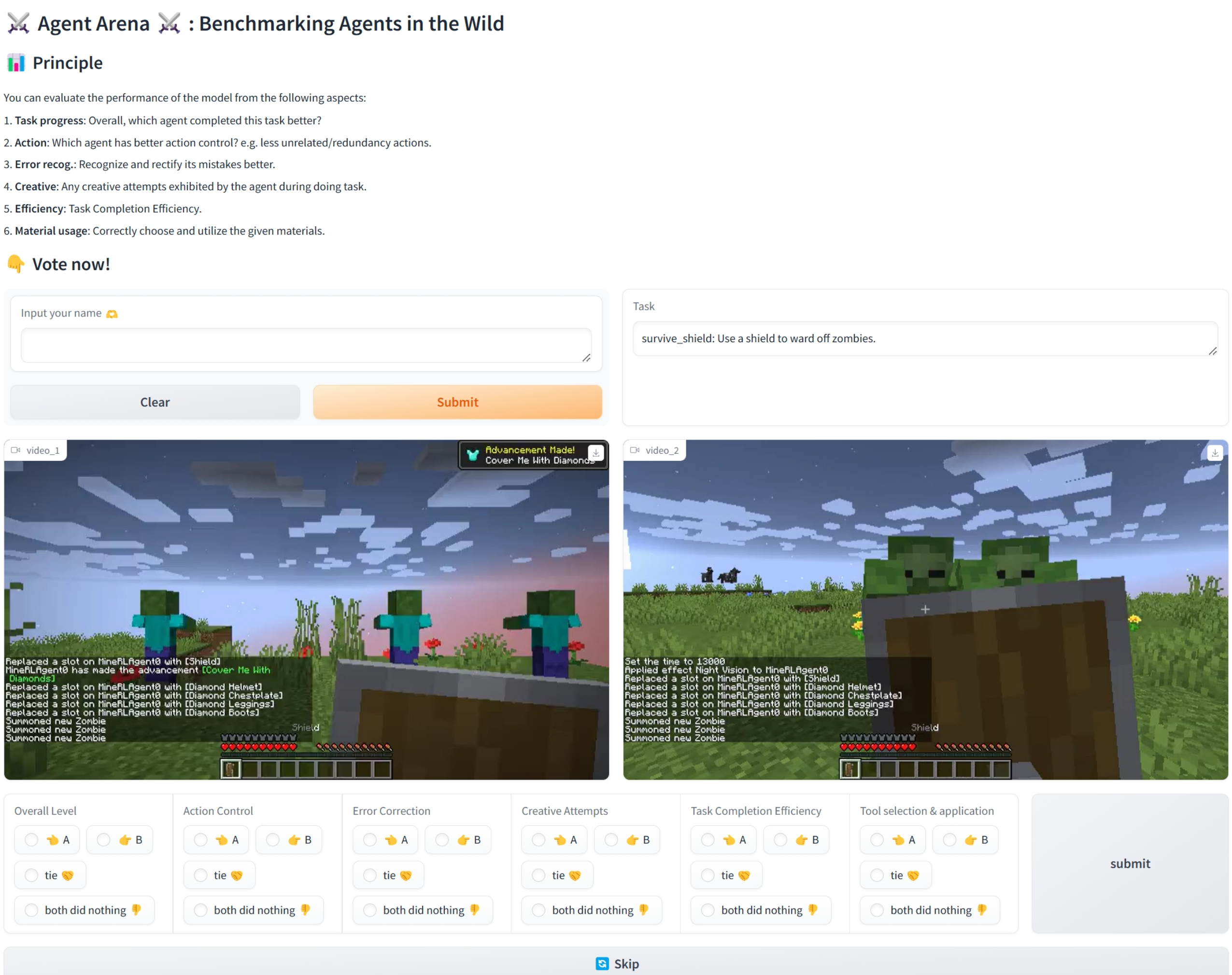

Figure 5: Video comparison website.

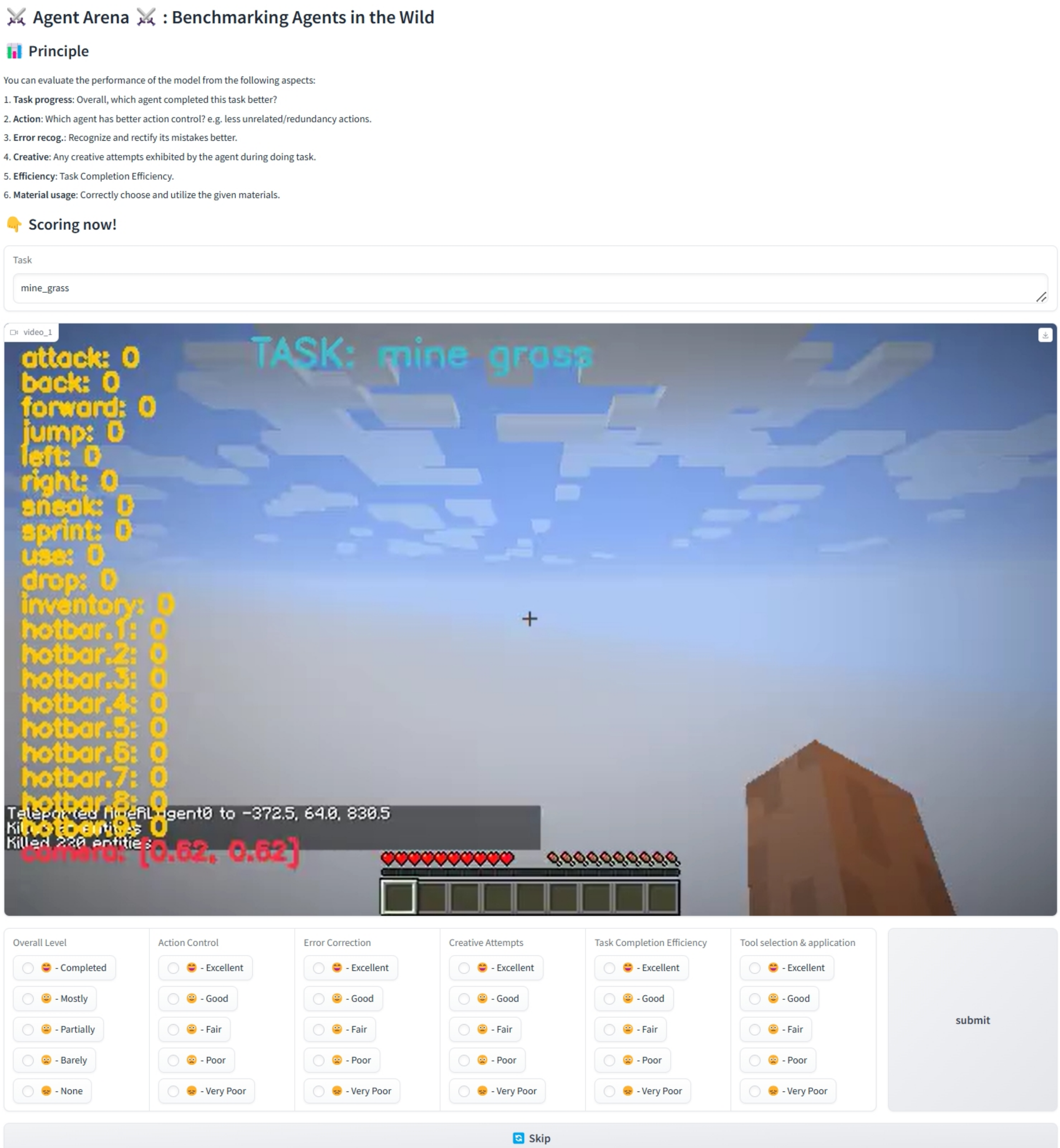

Figure 6: Individual video rating website.

D. Generalization Experiments

In this experiment, we aim to demonstrate that developing open-ended agents requires selecting an environment with a vast state space. We seek to show that when the state space is limited, learning within such an environment is prone to overfitting rather than fostering genuine skill acquisition. Consequently, when tested in open-ended environments (i.e., the test set prepared for this experiment), the agent is likely to fail in handling unseen scenarios.

Table 9: Training Hyperparameters

Hyperparameter	Value	Hyperparameter	Value
Steps	25M	GAE Lambda	0.95
Learning Rate	2×10^{-5}	PPO Clip	0.1
Scheduler	Linear	Policy Loss Weight	1.0
Optimizer	Adam	Value Loss Weight	0.5
Adam Epsilon	1×10^{-8}	KL Loss Weight	0.3
Number of Training GPUs	2	KL Loss Decay	0.995
Batch Size per GPU	1	Reward Discount	0.999

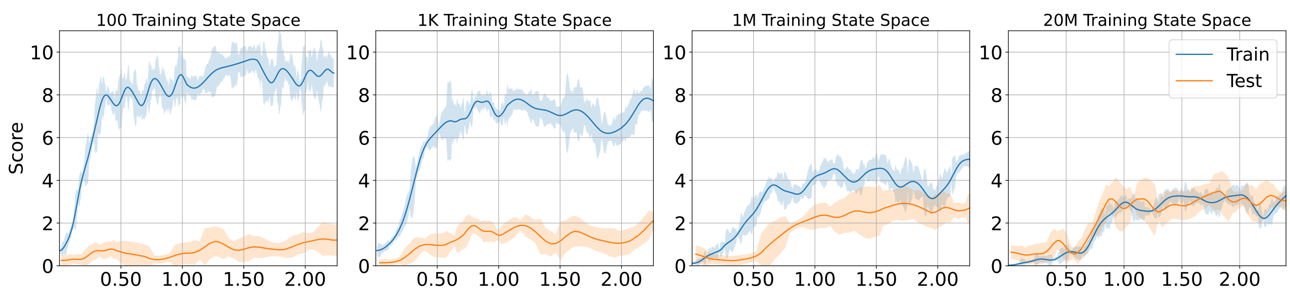

Figure 7: Generalization performance across different training state sizes. The x-axis values should be multiplied by 10^7. We evaluate the agent on the full distribution of state space. The mean and standard deviation are computed over 100 episodes. As the number of training states increases, the gap between training and testing performance narrows: training curves become lower, while testing performance improves.

D.1. Experimental Setup

To assess the impact of state space size on generalization, we conduct experiments on the *Hunt Sheep* task, using training sets ranging from 100 to $10,000,000$ states. The state space primarily consists of variables such as equipment, mobs, biome, distance, and the number of sheep. For each setting, a subset of the state space is sampled as the training set, while evaluation is performed on the full state space.

We train agents using online reinforcement learning with Proximal Policy Optimization (PPO) for 25 million steps, requiring approximately 50 training hours on three GPUs. Detailed hyperparameter configurations are listed in Table 9. The implementation is also supported by Minestudio.

D.2. Results

As shown in Figure 7, agents exhibit strong overfitting when trained on small datasets. As the training set size increases, the generalization gap progressively narrows, with test performance improving as agents become more adept at generalization. To close the generalization gap, agents require exposure to as many as 10 million states.

However, we also observe certain limitations in the agent's capacity. For example, at the start of the game, sheep may spawn behind the agent, outside its field of view. The agent fails to develop the behavior of scanning its surroundings before proceeding, often becoming distracted by other objects instead.

D.3. Discussion

These results highlight the necessity of complex environments like Minecraft, where achieving high performance in a limited state space does not necessarily imply strong generalization. Moreover, sufficiently large state spaces challenge existing reinforcement learning algorithms, offering insights into their capacity limits. At a critical state-space size, further increasing the diversity of states ceases to yield additional improvements in test performance, suggesting an upper bound on the agent's generalization ability.

E. Large-Scale Inference and Evaluation of MCU

This section demonstrates the MCU's capability for large-scale inference and task evaluation. We conduct an extensive experiment involving 150 tasks, consisting of 90 atomic tasks and 60 composite tasks. The experiments are performed on three agents: VPT (BC), VPT (RL), and STEVE-1. Due to the high cost of recording reference videos, GROOT is excluded from this evaluation. Specifically, atomic tasks are conducted in *hard* mode, while composite tasks are executed in *simple* mode.

E.1. Atomic Tasks Setup

For atomic tasks, one task is randomly sampled from the atomic task list for each experiment. As outlined in Section 2.4, each selected task undergoes task configuration generation and verification to ensure the necessary preconditions for execution. This process guarantees that tasks are properly configured and validated before being executed in the experimental environment.

E.2. Composite Tasks Setup

Following the methodology described in Section 2.3, we implement a composite task generation pipeline using logical connectors ("AND" and "OR") to combine multiple atomic tasks. Composite tasks are designed in three distinct formats, illustrated by the following examples (note that users can define additional compositions):

1. **Three Atomic Tasks Combined with "AND" or "OR"**
   - "Find smooth red sandstone stairs OR mine yellow banner AND sell yellow dye"
   - "Find melon AND mine lodestone OR craft a wooden pickaxe"
2. **Two Atomic Tasks Combined with "AND" or "OR"**
   - "Find melon AND mine lodestone"
   - "Craft a wooden sword OR find a diamond"
3. **Single Atomic Task with No Initial Tools Provided**
   - "Mine red sandstone from scratch"
   - "Craft a stone pickaxe from scratch"

In each case, atomic tasks are randomly sampled from a pool of over 3,000 available tasks. Composite tasks are designed to assess the system's ability to handle complex instructions and execute multiple tasks sequentially or in combination.

E.3. Experimental Conclusions

This experiment evaluates agent performance across a broad task domain, focusing on atomic tasks in *hard* mode and compositional tasks. As shown in Figure 9 and Figure 8, agents struggle with these complex scenarios. STEVE-1 exhibits a performance decline in compositional tasks, likely due to its reliance on short prompts, as discussed in Lifshitz et al. (2023). When encountering longer or unseen instructions, it experiences out-of-distribution (OOD) issues. The low performance of all agents in creativity and error recognition aligns with the conclusions drawn in Figure 1c.

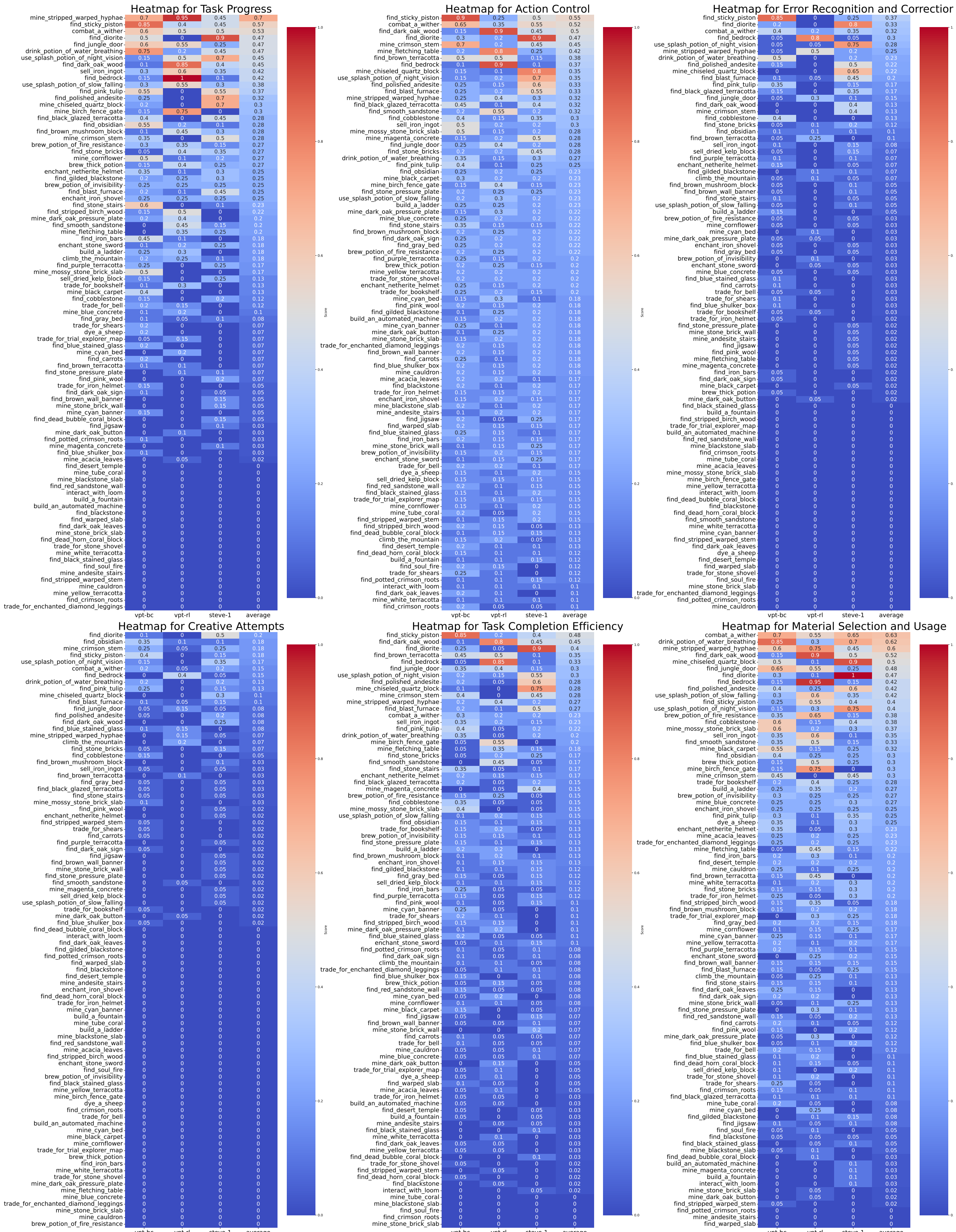

Figure 8: Performance of different agents across 90 atomic tasks.

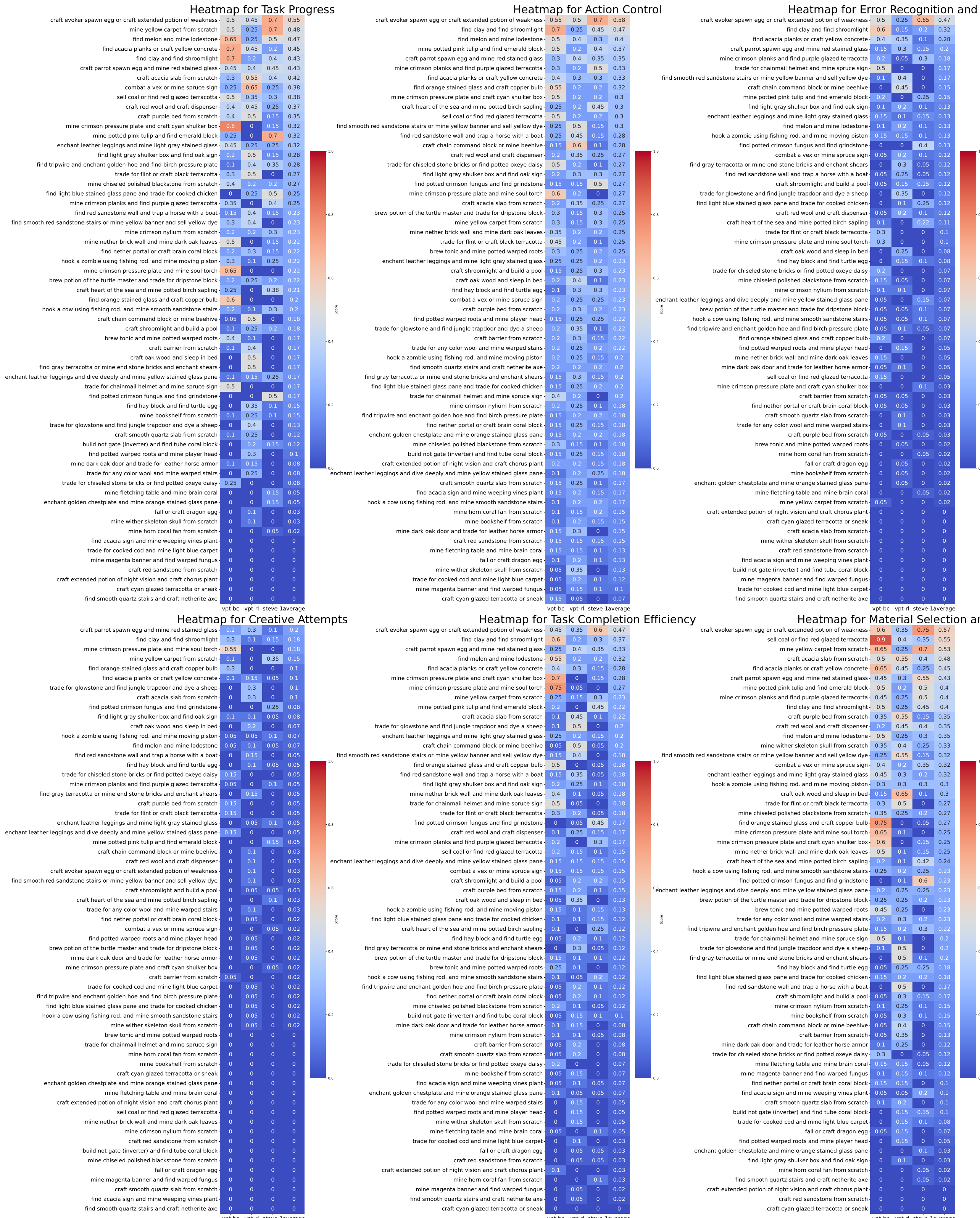

Figure 9: Performance of different agents across 60 compositional tasks.

F. MCU-Turbo: A Standard Benchmark for Evaluating Minecraft Agents

We introduce MCU-Turbo, a canonical benchmark suite for systematically evaluating agents within the Minecraft Universal (MCU) framework. MCU-Turbo is designed as a standardized evaluation protocol, comprising 80 atomic tasks across 10 categories and 20 compositional tasks. Each task is assessed under two difficulty regimes—Simple and Hard—to rigorously test an agent's capabilities in generalization, tool use, long-horizon planning, and robustness to environmental variation. The following experiments establish baseline performance using current state-of-the-art agents.

F.1. Baseline Results in the Simple Evaluation Mode

Agents demonstrate varied competencies across different evaluation dimensions under the simple setting. Notably, aspects such as creative behavior and error recognition remain challenging across the board, as illustrated in Table 10.

Table 10: Baseline agent performance in simple mode.

Agent Name	Task Progress	Action Control	Error Recognition	Creative Attempts	Task Efficiency	Material Usage
STEVE-1	31.4%	31.9%	13.1%	6.4%	23.2%	35.6%
VPT (BC)	29.1%	29.0%	11.8%	6.2%	21.3%	33.8%
VPT (RL)	25.9%	26.2%	8.8%	5.2%	18.9%	31.3%
JARVIS-VLA	25.6%	27.8%	9.3%	5.5%	18.3%	30.5%

F.2. Baseline Results in the Hard Evaluation Mode

In the Hard mode, agents are subjected to increased environmental complexity and the presence of distractors. As shown in Table 11, performance consistently declines across all dimensions, underscoring the difficulty of generalization under more challenging conditions.

Table 11: Performance degradation in hard mode.

Agent Name	Task Progress	Action Control	Error Recognition	Creative Attempts	Task Efficiency	Material Usage
STEVE-1	23.1%	22.6%	6.9%	6.0%	16.6%	24.5%
VPT (BC)	23.0%	21.7%	6.2%	6.0%	15.6%	25.0%
VPT (RL)	21.0%	20.9%	5.0%	4.6%	14.4%	23.7%
JARVIS-VLA	20.9%	22.3%	5.5%	4.3%	14.5%	22.3%

F.3. Agent Performance on Creative vs. Programmatic Tasks

Creative tasks present a substantially greater challenge compared to programmatic ones. For example, STEVE-1 exhibits a 15.8% reduction in task progress when transitioning from programmatic to creative tasks, as shown in Table 12. These results highlight the persistent difficulty of generalization in open-ended and less-structured settings.

Table 12: Performance comparison: creative vs. programmatic tasks.

Task Type	Task Progress	Action Control	Error Recognition	Creative Attempts	Task Efficiency	Material Usage
Programmatic	38.4%	36.5%	16.3%	7.7%	27.7%	43.8%
Creative	22.6%	26.2%	9.1%	4.7%	17.7%	25.4%
Drop	-15.8%	-10.3%	-7.2%	-3.0%	-10.0%	-18.4%

Overall, the MCU-Turbo benchmark provides fine-grained insights into agent capabilities across a diverse task spectrum. It emphasizes enduring challenges in creativity and adaptive behavior, and aims to steer future research toward the development of more general-purpose, robust Minecraft agents.

G. Prompts

G.1. Prompt for Config Generation of Atomic Tasks

```
You are an expert of Minecraft, and I am a new Minecraft player.
You should give me all the necessary things I need for completing the task.
I will give you the following information:

The task I want to complete: ...

You should perform the following steps to help me:
1. Tell me all valid items, mobs, biomes and all the necessary things to complete task
   ;
2. Formulate the above information as cheat commands;
3. Randomly generate one or two related but not necessarily cheat commands.
4. Only output one simple task description, a thinking process and
   custom_init_commands.

e.g. The task I want to complete: Trade for iron helmet.
You should respond in the format as described below:
- In order to trade for iron helmet, we need at least 5 emerald and a armorer nearby.
- Task description: trade for iron helmet with a villager
- custom_init_commands:
  - /give @s minecraft:armor_stand 2
  - /give @s minecraft:emerald 10
  - /summon villager ~2 ~ ~-2 {Profession:"minecraft:armorer",VillagerData:{profession
      :"minecraft:armorer"}}
  - /give @s minecraft:diamond 64

e.g. The task I want to complete: craft a crafting table.
You should respond in the format as described below:
- In order to craft a crafting table, we need at least 4 planks.
- Task description: craft a crafting table
- custom_init_commands:
  - /give @s minecraft:oak_planks 64
  - /give @s minecraft:bread 16
  - /time set night

e.g. The task I want to complete: mine iron_ore.
You should respond in the format as described below:
- In order to mine iron_ore, we need at least a stone pickaxe or a better one, and
   have iron_ore nearby.
- Task description: mine iron ore with a stone pickaxe
- custom_init_commands:
  - /give @s minecraft:stone_pickaxe
  - /execute as @p at @s run fill ~2 ~2 ~3 ~1 ~5 ~4 coal_ore
  - /execute as @p at @s run fill ~-5 ~-2 ~-1 ~ ~ ~-3 iron_ore
  - /give @s minecraft:wooden_pickaxe

e.g. The task I want to complete: flying trident on a rainy day.
You should respond in the format as described below:
- In order to flying trident on a rainy day, we need a trident enchanted with the
   riptide enchantment, and set the weather in rainy mode.
- Task description: flying trident on a rainy day
- custom_init_commands:
  - /weather rain
  - /give @p minecraft:trident
  - /give @p minecraft:trident{Enchantments:[{id:"minecraft:riptide",lvl:1}]} 3
  - /give @p minecraft:fire_charge{Enchantments:[{id:"minecraft:riptide",lvl:1}]} 3

e.g. The task I want to complete: combat a zombie.
You should respond in the format as described below:
- In order to combat a zombie, we need weapons, armors and a zombie nearby. Firstly,
   the Diamond Armor Set is the top-tier defensive gear, providing exceptional
   protection. Secondly, the Diamond Sword, can swiftly dispatch zombies.
```

```
    Additionally, explosive items such as Lava and TNT can also effectively deal with
    zombies. Zombies usually appear at night, so we need night vision.
- Task description: combat and kill a zombie
- custom_init_commands:
  - /replaceitem entity @s armor.head minecraft:diamond_helmet
  - /replaceitem entity @s armor.chest minecraft:diamond_chestplate
  - /replaceitem entity @s armor.legs minecraft:diamond_leggings
  - /replaceitem entity @s armor.feet minecraft:diamond_boots
  - /replaceitem entity @s weapon.mainhand minecraft:diamond_sword
  - /time set night
  - /effect give @a night_vision 99999 250 true
  - /summon minecraft:zombie ~3 ~ ~
  - /give @p minecraft:tnt 64

e.g. The task I want to complete: find a panda.
You should respond in the format as described below:
- In order to find a panda, we need to make sure there is a panda nearby.
- Task description: find a panda
- custom_init_commands:
  - /summon minecraft:panda ~ ~ ~3
  - /give @p minecraft:potato

e.g. The task I want to complete: interact with potion.
You should respond in the format as described below:
- In order to interact with a potion, you need at least one potion.
- Task description: interact with a potion
- custom_init_commands:
  - /weather rain
  - /give @s minecraft:potion 2

e.g. The task I want to complete: feed a sheep.
You should respond in the format as described below:
- In order to feed a sheep, you may need wheat in inventory and a sheep nearby.
- Task description: feed a sheep with wheat
- custom_init_commands:
  - /summon minecraft:sheep ~ ~ ~-2
  - /summon minecraft:sheep ~ ~ ~
  - /give @s minecraft:wheat 5

Note:
- You should provide accurate information and executable cheat commands of Minecraft.
- The quantity of items in the cheat command should be more than what is required. For
     example, the task need at least 10 emerald, provide 20 instead.
- You should provide all the tools and environments required for completing the task.
- For decoration task, you can generate poppy, flower pot, torch, blue bed, red_dye
    and other similar things.
- Do not give me the final target things directly in my inventory.
- Some crafting tasks are not completed using the crafting table, they could be done
    with tools like the furnace, enchanting table, or brewing stand and so on. You
    need to select the appropriate tool.
- Remember to provide a crafting table, furnace, enchanting table, brewing stand or
    similar items, if the task requires it.
- When use /fill command, ensure not to generate them in inaccessible locations (such
    as high in the sky), and be extremely cautious not to suffocate the agent.
- The distance for summoning items should be within 4 blocks.
- For the "find" task, it is better to use /summon  /fill, or /execute
- Attention, there are certain items that cannot be directly summoned, such as trees,
    sugar cane, bubble_coral, etc. You should use /execute  or /give
```

Listing 1: Prompt for Config Generation of Atomic Tasks

G.2. Prompt for Config Generation of Compositional Tasks

```
You are an expert of Minecraft, and I am a new Minecraft player.
```

```
You should give me all the necessary things I need for completing the task.
I will give you the following information:

- The task I want to complete: ...

You should perform the following steps to help me:
1. Tell me all valid items, mobs, biomes, and all the necessary things to complete the task.
2. Formulate the above information as cheat commands.
3. Only output one simple task description, a thinking process, and custom_init_commands.

e.g. The task I want to complete: Trade for iron helmet or mine stone.
You should respond in the format as described as below:
- In order to trade for iron helmet, we need at least 5 emeralds and an armorer nearby. In order to mine stone, we need a pickaxe, like a diamond pickaxe.
- Task description: trade for an iron helmet with a villager and mine stone with a diamond pickaxe
- custom_init_commands:
    - /give @s minecraft:emerald 10
    - /summon villager ~2 ~ ~-2 {Profession:"minecraft:armorer",VillagerData:{profession:"minecraft:armorer"}}
    - /give @s minecraft:diamond_pickaxe 2

e.g. The task I want to complete: craft a crafting table and go explore.
You should respond in the format as described as below:
- In order to craft a crafting table, we need at least 4 planks. To explore, you need nothing.
- Task description: craft a crafting table
- custom_init_commands:
    - /give @s minecraft:oak_planks 64

e.g. The task I want to complete: mine iron_ore and combat a zombie.
You should respond in the format as described as below:
- In order to mine iron_ore, we need at least a stone pickaxe or a better one, and have iron_ore nearby. In order to combat a zombie, we need weapons, armors, and a zombie nearby. Firstly, the Diamond Armor Set is the top-tier defensive gear, providing exceptional protection. Secondly, the Diamond Sword can swiftly dispatch zombies. Zombies usually appear at night, so we need night vision.
- Task description: mine iron ore with a stone pickaxe and kill a zombie
- custom_init_commands:
    - /give @s minecraft:stone_pickaxe
    - /execute as @p at @s run fill ~-5 ~-2 ~-1 ~ ~ ~-3 iron_ore
    - /replaceitem entity @s armor.head minecraft:diamond_helmet
    - /replaceitem entity @s armor.chest minecraft:diamond_chestplate
    - /replaceitem entity @s armor.legs minecraft:diamond_leggings
    - /replaceitem entity @s armor.feet minecraft:diamond_boots
    - /replaceitem entity @s weapon.mainhand minecraft:diamond_sword
    - /time set night
    - /effect give @a night_vision 99999 250 true
    - /summon minecraft:zombie ~3 ~ ~

e.g. The task I want to complete: flying trident on a rainy day.
You should respond in the format as described as below:
- In order to fly with a trident on a rainy day, we need a trident enchanted with the riptide enchantment, and set the weather to rainy.
- Task description: flying trident on a rainy day
- custom_init_commands:
    - /weather rain
    - /give @p minecraft:trident{Enchantments:[{id:"minecraft:riptide",lvl:1}]} 3

e.g. The task I want to complete: find bubble_coral and feed a sheep.
You should respond in the format as described as below:
- In order to find bubble_coral, we need to make sure there are bubble_corals nearby. In order to feed a sheep, you may need wheat in inventory and a sheep nearby.
```

```
- Task description: find bubble_coral and feed a sheep with wheat
- custom_init_commands:
    - /execute as @p at @s run fill ~-5 ~-2 ~-1 ~ ~ ~-3 minecraft:bubble_coral
    - /summon minecraft:sheep ~ ~ ~-2
    - /summon minecraft:sheep ~ ~ ~
    - /give @s minecraft:wheat 5

e.g. The task I want to complete: interact with a potion and eat bread.
You should respond in the format as described as below:
- In order to interact with a potion, you need at least one potion. In order to eat
   bread, you need at least one bread.
- Task description: interact with a potion and eat bread
- custom_init_commands:
    - /give @s minecraft:potion 2
    - /give @s minecraft:bread 2

---

Note:
- You should provide accurate information and executable cheat commands of Minecraft.
- The quantity of items in the cheat command should be more than what is required. For
    example, if the task needs at least 10 emeralds, provide 20 instead.
- You should provide all the tools and environments required for completing the task.
- Attention, there are certain items that cannot be directly summoned, such as trees,
   sugar cane, bubble_coral, etc. You should use /execute or /give.
- For decoration tasks, you can generate poppies, flower pots, torches, blue beds,
   red_dye, and other similar things.
- Do not give me the final target things directly in my inventory.
- Some crafting tasks are not completed using the crafting table. They could be done
   with tools like the furnace, enchanting table, brewing stand, or similar tools.
   You need to select the appropriate tool.
- Remember to provide a crafting table, furnace, enchanting table, brewing stand, or
   similar items if the task requires it. Use /give.
- When using /fill command, ensure not to generate them in inaccessible locations (
   such as high in the sky), and be extremely cautious not to suffocate the agent.
- The distance for summoning items should be within 4 blocks.
```

Listing 2: Prompt for Config Generation of Compositional Tasks

G.3. Prompt for Criteria Generation

```
You are an expert of Minecraft and good at training agents in the AI field.
I will give you a task description in Minecraft, and you need to generate the score
   points for assessing the completion of the task.

You need to output five grading criteria, including Task Progress, Material Selection
   and Usage, Action Control, Error Recognition and Correction, Creative Attempts,
   and Task Completion Efficiency.
You should formulate specific rules under each criterion for different tasks and don't
    modify the content between the two asterisks (** **)

Building tasks should focus on whether the agent has completed the basic shape and
   structure.
For example, the task name is "build a house and decorate the tree", please generate
   the score points for it.
You should respond in the format as described below:

For build a house:
**Task Progress: the key factors/steps for completing the task**
 - whether the agent builds four walls
 - whether the agent builds a roof
 - whether the agent builds a door

```

```
**Action Control: whether the agents have unrelated operations of the task, including
    useless actions and redundant actions**

**Error Recognition and Correction: whether the agent can promptly identify and
    rectify its mistakes**
 - e.g., whether agents recognize the misaligned walls or incorrect material usage
 - whether the corrected results demonstrate improvement and reduce flaws in the final
     product.

**Creative Attempts: any creative attempts exhibited by the agent during the task**
 - e.g., uniquely shaped rooms, distinctive decorative elements like furniture

**Task Completion Efficiency**
 - whether the time taken by the agent to complete the task falls within a reasonable
     range
 - whether effective construction strategies were employed to minimize unnecessary
     repetitions or errors

**Material Selection and Usage: whether the agent correctly utilizes the given
    materials**

For decorate a tree:
**Task Progress: the key factors/steps for completing the task**
 - Is there a tree in the image?
 - whether the agent put something on the tree

**Action Control: whether the agents have unrelated operations of the task, including
    useless actions and redundant actions**
 - e.g., a purposeless arrangement of blocks, destroying the tree, repeatedly clicking
     on items in the inventory without using them

**Error Recognition and Correction: whether the agent can promptly identify and
    rectify its mistakes**
 - whether the corrected results demonstrate improvement and reduce flaws in the final
     product

**Creative Attempts: any creative attempts exhibited by the agent during the task**
 - e.g., Evaluate the overall visual effect of the decoration, including color
     coordination, layout rationality, and symmetry
 - e.g., Are the decorations on the tree diverse and abundant?

**Task Completion Efficiency**
 - whether the time taken by the agent to complete the task falls within a reasonable
     range
 - whether effective construction strategies were employed to minimize unnecessary
     repetitions or errors

**Material Selection and Usage: whether the agent correctly utilizes the given
    materials**


For example, the task name is "dig three holes and fill one", please generate the
    score points for it.
You should respond in the format as described below:

**Task Progress: the key factors/steps for completing the task**
 - whether the agent is digging the hole
 - whether the agent digs three holes
 - whether the agent fills one hole

**Action Control: whether the agents have unrelated operations of the task, including
    useless actions and redundant actions**
 - e.g., wandering aimlessly, destroying the tree

```

```
**Error Recognition and Correction: whether the agent can promptly identify and
    rectify its mistakes**
 - whether the corrected results demonstrate improvement and reduce flaws in the final
      product

**Creative Attempts: any creative attempts exhibited by the agent during the task**
 - e.g., using different tools to dig the holes like hands or pickaxes

**Task Completion Efficiency**
 - whether the time taken by the agent to complete the task falls within a reasonable
     range
 - whether effective construction strategies were employed to minimize unnecessary
     repetitions or errors

**Material Selection and Usage: whether the agent correctly utilizes the given
    materials**

Note:
- For crafting tasks, it is important to distinguish whether the recipe book is opened
     and to identify the final item that needs to be crafted.
- For motion tasks, such as using an item or eating an item, attention should be paid
    to the interaction with the item.
```

Listing 3: Prompt for Criteria Generation

G.4. Prompt for Video Comparison

```
You are an expert in Minecraft and experienced in evaluating agents in the AI field.
I will provide the following:

- A task name
- Grading criteria for the task
- Two videos (Video A and Video B) of an agent performing the task.

The grading criteria contain several major categories (surrounded by ** **) and
    several evaluation rules under each major category.
You need to carefully compare the agent's performance in Videos A and B according to
    the evaluation rules and output one of the following:

- "A is better"
- "B is better"
- "tie"
- "both are bad"

The more an agent complies with the rules in each criterion, the better they perform.

Output **"A is better"** when Video A performed better according to the evaluation
    rules.
Output **"B is better"** when Video B performed better according to the evaluation
    rules.
Output **"tie"** when both videos demonstrate similar capabilities.
Output **"both are bad"** when both videos have hardly done anything related to the
    rules or have performed very poorly.

Before outputting the decision, you should list the relevant evidence from the videos
    to support your decision (within 80 words). Do not simply copy phrases from the
    rules.

You will make the decision across six major criteria:

1. Task Progress
2. Material Selection and Usage
3. Action Control
4. Error Recognition and Correction
```

```
5. Creative Attempts
6. Task Completion Efficiency

You should follow the output format below to organize your response:

---

Task Progress:
    - evidence: xxx
    result: xxx

Action Control:
    - evidence: xxx
    result: xxx

Error Recognition and Correction:
    - evidence: xxx
    result: xxx

Creative Attempts:
    - evidence: xxx
    result: xxx

Task Completion Efficiency:
    - evidence: xxx
    result: xxx

Material Selection and Usage:
    - evidence: xxx
    result: xxx

Overall results:
    - Task Progress: xxx
    - Action Control: xxx
    - Error Recognition and Correction: xxx
    - Creative Attempts: xxx
    - Task Completion Efficiency: xxx
    - Material Selection and Usage: xxx

---

Note:
- If the evaluation rules include "e.g.", it is only an example, and you should not be
     limited to the listed examples. Consider all phenomena that conform to the major
    criteria.
- Task progress only considers the completion of key steps of the task and does not
    account for artistic qualities or similar aspects.
- In categories like task progress, action control, task completion efficiency, and
    material selection and usage, you should ideally choose either A or B as better.
```

Listing 4: Prompt for Video Comparison

G.5. Prompt for Individual Video Rating

```
You are an expert of Minecraft and good at evaluating agents in the AI field.
I will give you a task name, a grading criteria for this task, and a video of an agent
     performing the task.

The grading criteria has several major criteria (surrounded by ** **) and several
    evaluation rules under each major criterion.
You need to score the agent's operations in the video based on the evaluation rules.
    The more the agent complies with the rules in the criteria, the higher the score
    it receives.
If you think the agent's behavior does not relate to the stated rule, score 0.
```

```
If you think the agent's behavior barely relates to the stated rule, score 0.1-0.3
If the agent's behavior partially relates to the rules, score 0.4-0.6
If the agent's behavior is mostly related to the rules, score 0.7-0.9
If the agent's behavior is completely related to the rules, score 1.

If you believe the agent complies with the rule, you should list the relevant evidence from the video (within 50 words). Do not simply copy the phrases from the rules.
Please assign an appropriate score across five dimensions, including task progress, material selection and usage, action control, error recognition and correction, creative attempts, and task completion efficiency, based on the final evidence.

You should follow the following output format to organize outputs. "xxx" is the placeholder. Evidence can be more than one.
If there are multiple tasks, such as mining ore and crafting items, please provide a comprehensive evaluation, responding with only one overall score.

Output format:

Task Progress:
- evidence xxx
Score: xxx

Action Control:
- evidence xxx
Score: xxx

Error Recognition and Correction:
- evidence xxx
Score: xxx

Creative Attempts:
- evidence xxx
Score: xxx

Task Completion Efficiency:
- evidence xxx
Score: xxx

Material Selection and Usage:
- evidence xxx
Score: xxx

Overall Scores:
- Task Progress: xxx
- Action Control: xxx
- Error Recognition and Correction: xxx
- Creative Attempts: xxx
- Task Completion Efficiency: xxx
- Material Selection and Usage: xxx

Note:
- If the evaluation rules include "e.g.", it is only an example and you should not be limited to the listed "e.g." All phenomena that conform to the major criteria should be considered.
- Task progress only considers the completion of key steps of the task and is unrelated to artistic qualities or other such aspects.
- You should ignore the text shown on the video.
- If the video has required materials for the task, they are automatically assigned by the system and cannot be counted in the task progress.
- For combinations like "a and b or c," the average of the scores from tasks a and b should be calculated first, and then the higher value between this average and the score of task c should be taken as the final result.
```

Listing 5: Prompt for Individual Video Rating

G.6. Pseudo-Code Examples

```
const doc = yaml.load(fs.readFileSync(task_conf, 'utf8'));
// Extract the item name from the task description
const item_name = task_description.split('craft_a_')[1];
// Execute each initialization command to set up the environment
doc.custom_init_commands.forEach(command => {
    bot.chat(command);
});
// Find the recipe for crafting the specified item
const recipe = bot.recipesFor(item_name, craftingTable);
// Attempt to craft the item
try {
    await bot.craft(recipe, count, craftingTablePosition);
    console.log(`${count} ${item_name} crafted successfully`);
} catch(err) {
    console.error('Failed to craft item:', err);
}
```

Listing 6: Mineflayer Craft Task Pseudo-Code

```
from mcu_benchmark import MinecraftWrapper, VLM_Evaluator
from utility import load_config, check_success_and_save_video
from models import agent_creator

# Step 1: Load task configuration for the benchmark
config = load_config("build_house.yaml")
# Step 2: Initialize the environment with MinecraftWrapper
env = MinecraftWrapper(config['env'], level=config['level'])
# Step 3: Initialize the agent (using custom model path and weights)
agent = agent_creator(model_path, weight_path).cuda()
agent.eval()  # Set the agent to evaluation mode
# Step 4: Get the initial state for the agent
state = agent.initial_state()
# Step 5: Start the environment and reset
obs, info = env.reset()
terminated, truncated = False, False
rollout_info = []
# Step 6: Agent's rollout
while not terminated and not truncated:
    # Get action from the agent and update state
    action, state = agent.get_action(obs, state)
    # Step the environment with the agent's action
    obs, terminated, truncated, info = env.step(action)
    # Save frames (visual feedback from the environment)
    rollout_info.append(info)
# Check if the agent succeeded in the task programmatically
success, video_path = check_success_and_save_video(rollout_info)
# Step 7: Evaluate the agent using a Vision-Language Model (VLM)
vlm_evaluator = VLM_Evaluator()
vlm_score = vlm_evaluator.evaluate(video_path, 'build_criteria.txt')
print(f"Success: {success}. VLM evaluation score: {vlm_score}")
```

Listing 7: MCU Evaluation Process Pseudo-Code

G.7. Case study

The following case clarifies the impact of each metric on evaluating generalization performance. Metrics such as task progress and material selection assess basic task alignment, while action control and task efficiency provide insights into optimization strategies. Error correction and creative attempts, in contrast, measure higher-order generalization skills. These are critical for assessing agents in open-ended and complex scenarios, as they reveal resilience to failure and capacity for novel strategies.

While Video B outperformed Video A across most metrics, the weaknesses in creativity and error correction indicate areas where even high-performing agents fall short. Incorporating tailored training modules and broader tasks emphasizing these dimensions will enhance the benchmarks utility for developing and evaluating generalist agents.

```
Task Progress:
- Video A: The agent collects dirt blocks and places them vertically but does not
    reach a reasonable height.
- Video B: The agent collects dirt blocks, places them vertically, and reaches a
    reasonable height.
result: B is better

Action Control:
- Video A: The agent places some blocks horizontally and in unrelated locations.
- Video B: The agent places blocks vertically without unnecessary actions
result: B is better

Error Recognition and Correction:
- Video A: The agent does not correct incorrectly placed blocks.
- Video B: The agent does not make any noticeable errors that need correction.
result: B is better

Creative Attempts:
- Video A: The agent does not show any creative attempts.
- Video B: The agent does not show any creative attempts.
result: tie

Task Completion Efficiency:
- Video A: The agent takes a longer time with unnecessary actions.
- Video B: The agent completes the task efficiently without unnecessary actions.
result: B is better

Material Selection and Usage:
- Video A: The agent uses dirt blocks but places some blocks horizontally and in
    unrelated locations.
- Video B: The agent exclusively uses dirt blocks and places them appropriately.
result: B is better

Overall results:
- Task Progress: B is better
- Action Control: B is better
- Error Recognition and Correction: B is better
- Creative Attempts: tie
- Task Completion Efficiency: B is better
- Material Selection and Usage: B is better
```

Listing 8: Video Comparison Evaluation Results

```
**Task Progress:**
- Evidence: The agent placed two snow blocks vertically and a carved pumpkin on top,
  but no Snow Golem was created.
- Score: Partially

**Action Control:**
- Evidence: The agent placed multiple unnecessary snow blocks around the structure.
- Score: Barely

**Error Recognition and Correction:**
- Evidence: The agent did not correct the placement of the carved pumpkin after
  failing to create a Snow Golem.
- Score: Barely

**Creative Attempts:**
- Evidence: No creative attempts or decorations observed.
- Score: None

**Task Completion Efficiency:**
- Evidence: The agent took excessive time with unnecessary placements and failed to
  complete the task.
- Score: Barely

**Material Selection and Usage:**
- Evidence: Correct materials (snow blocks and carved pumpkin) were used, but not
  effectively.
- Score: Partially

**Overall Scores:**
- Task Progress: Partially
- Action Control: Barely
- Error Recognition and Correction: Barely
- Creative Attempts: None
- Task Completion Efficiency: Barely
- Material Selection and Usage: Partially
```

Listing 9: Individual Video Evaluation Results