Open-World Skill Discovery from Unsegmented Demonstration Videos

Supplementary Material

A. Discussion and Future work

Computational Cost. Our method takes approximately 1 minute to process a 5-minute video on an NVIDIA RTX 3090Ti, due to the need to predict actions across the entire trajectory. Since data preprocessing is a one-time cost, spending only 1/20 of the video duration with 4 GPUs is acceptable. Also, it is more efficient than those methods relying on human or VLM annotations while ensuring quality.

Failure Case Due to Occasional Assumption Violation. Our method is occasionally unstable in action-intensive scenes like combat, where performance drops on task `combat spiders` (29% to 20%). See Appendix E for visualization of this failure case. This is likely due to a violation of Assumption 3.2 (Skill Confidence), as indicated by sharp prediction loss fluctuations. However, such issues are not observed elsewhere. Since overly short segments (< 5 frames) comprise less than 10% of all videos, overall effectiveness remains unaffected.

Choice of Loss Term. Currently our method detects changes in $P(a_t|o_{1:t})$. A possible alternative is to use $P(o_{t+1}|o_{1:t})$. We don't use it because it does not directly reflect skill changes as $P(a_t|o_{1:t})$ does, and current models [53] for predicting future observations suffer from hallucinations. We leave the attempt of this alternative to future work.

Scalability and Broader Applicability. Since our method is label-free, it has the potential to segment and train on the numerous gameplay trajectories on YouTube. We also anticipate the potential application of our method to other larger datasets and other open-world video games beyond Minecraft.

B. Proofs

In this section, we provide a detailed proof of Theorem 3.4, regarding the bounds of relative predictive probability under scenarios of skill transition and non-transition. We prove the lower bound and upper bound respectively in the following two subsections.

B.1. Proof of Upper Bound Under Skill Non-Transition

$$
\begin{aligned}
P\left(\frac{P(a_{t+1}|o_{1:t+1})}{(\prod_{i=1}^{t} P(a_i|o_{1:i}))^{1/t}} > \frac{(K-1)c}{K}\right) &> P\left(P(a_{t+1}|o_{1:t+1}) > \frac{(K-1)c}{K}\right) \\
&> P\left(P(\pi_{t+1}=\pi_t|o_{1:t+1})\pi_t(a_{t+1}|o_{1:t+1}) > \frac{(K-1)c}{K}\right) && \text{(Eq. (6))} \\
&> P(\pi_t(a_{t+1}|o_{1:t+1}) > c) && \text{(Assumption 3.1)} \\
&= P(\pi_{t+1}(a_{t+1}|o_{1:t+1}) > c) \\
&> 1-\delta && \text{(Assumption 3.2)}
\end{aligned}
$$

B.2. Proof of Lower Bound Under Skill Transition

We first magnify the likelihood ratio between the next action and the average action history,

$$\begin{aligned}
\frac{P(a_{t+1}|o_{1:t+1})}{(\prod_{i=1}^{t} P(a_i|o_{1:i}))^{1/t}} &< \frac{P(\pi_{t+1}=\pi_t|o_{1:t+1})\pi_t(a_{t+1}|o_{1:t+1}) + P(\pi_{t+1}\neq\pi|o_{1:t+1})}{(\prod_{i=1}^{t} P(a_i|o_{1:i}))^{1/t}} && \text{(Eq. (6))} \\
&< \frac{P(\pi_{t+1}=\pi_t|o_{1:t+1})\pi_t(a_{t+1}|o_{1:t+1}) + P(\pi_{t+1}\neq\pi|o_{1:i})}{\frac{K-1}{K}(\prod_{i=1}^{t} \pi_{i-1}(a_i|o_{1:i}))^{1/t}} && \text{(Eq. (6) and Assumption 3.1)} \\
&< \frac{\pi_t(a_{t+1}|o_{1:t+1}) + \frac{1}{K}}{\frac{K-1}{K}(\prod_{i=1}^{t} \pi_{i-1}(a_i|o_{1:i}))^{1/t}} && \text{(Assumption 3.1)} \\
&= \frac{K}{K-1}\frac{\pi_t(a_{t+1}|o_{1:t+1})}{\prod_{i=1}^{t} \pi_t(a_i|o_{1:i}))^{1/t}} + \frac{1}{(K-1)\prod_{i=1}^{t} \pi_i(a_i|o_{1:i}))^{1/t}} \\
&< \frac{Km}{2(K-1)} + \frac{1}{(K-1)\prod_{i=1}^{t} \pi_i(a_i|o_{1:i}))^{1/t}} && \text{(Assumption 3.3)}
\end{aligned}$$

Therefore,

$$\begin{aligned}
&P\left(\frac{P(a_{t+1}|o_{1:t+1})}{(\prod_{i=1}^{t} P(a_i|o_{1:i}))^{1/t}} < \frac{Km}{2(K-1)} + \frac{1}{c(K-1)}\right) \\
&> P\left(\frac{Km}{2(K-1)} + \frac{1}{(K-1)\prod_{i=1}^{t} \pi_i(a_i|o_{1:i}))^{1/t}} < \frac{Km}{2(K-1)} + \frac{1}{c(K-1)}\right) \\
&= P\left(\prod_{i=1}^{t} \pi_i(a_i|o_{1:i}))^{1/t} > c\right) \\
&> \prod_{i=1}^{t} P(\pi_i(a_i|o_{1:i}) > c) \\
&> (1-\delta)^t > 1 - t\delta && \text{(Assumption 3.2)}
\end{aligned}$$

C. Minecraft Environment

Minecraft is an extremely popular sandbox game that allows players to freely create and explore their world. This game has infinite freedom, allowing players to change the world and ecosystems through building, mining, planting, combating, and other methods. It is precisely because of this freedom that Minecraft becomes an excellent AI testing benchmark. In this game, AI agents need to face situations that are highly similar to the real world, making judgments and decisions to deal with various environments and problems. By using Minecraft, AI researchers can more conveniently simulate various complex and diverse environments and tasks, thereby improving the practical value and application of AI technology.

Our Minecraft environment is a hybrid between MineRL [19] and the MCP-Reborn (https://github.com/Hexeption/MCP-Reborn) Minecraft modding package. Unlike the regular Minecraft game, in which the server (or the "world") always runs at 20Hz while the client's rendering speed can typically reach 60-100Hz, the frame rate is fixed at 20 fps for the client in our experiments. 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.

C.1. Minecraft Game World Setting

We choose Minecraft version 1.16.5's survival mode as our experiment platform. In this mode, 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.

C.2. Observation Space

The environmental observation space consists of two parts. The first part is the raw pixels from the Minecraft game that players would see, including overlays such as the hotbar, health indicators, and animations of a moving hand in response to attack or

"use" actions. The rendering resolution of Minecraft is 640x360; however, in our experiments, we resize images to 128x128 for better computational efficiency while maintaining discernibility. The second part includes auxiliary information about the agent's current environment, such as its location and weather conditions. Human players can obtain this information by pressing F3. The specific observation details we include are shown in Tab. 5.

Sources	Shape	Description
pov (raw pixel)	(640, 360, 3) to (128,128,3)	Ego-centric RGB frames.
player_pos	(5,)	The coordinates of (x,y,z), pitch, and yaw of the agent.
location_stats	(9,)	The environmental information of the agent's current position, including `biome_id`, `sea_level`, `can_see_sky`, `is_raining` etc.
inventory	(36,)	The items in the current inventory of the agent, including the `type` and corresponding `quantity` of each item in each slot. If there is no item, it will be displayed as `air`.
equipped_items	(6,)	The current equipment of the agent, including `mainhand`, `offhand`, `chest`, `feet`, `head`, and `legs` slots. Each slot contains `type`, `damage`, and `max_damage` information.
event_info	(5,)	The events that occur in the current step of the game, including `pick_up` (picking up items), `break_item` (breaking items), `craft_item` (crafting items using a crafting table or crafting grid), `mine_block` (mining blocks by suitable tools), and `kill_entity` (killing game mobs).

Table 5. The observation space we use in Minecraft.

During the actual inference process, the controllers (VPT [3], GROOT [6], STEVE-1 [29]) only perceive the raw pixels. The agents (JARVIS-1 [42], OmniJarvis [43]) can access auxiliary information from the environment to generate the text condition of the controller.

C.3. Action Space

Our action space includes almost all actions directly available to human players, such as keypresses, mouse movements, and clicks. It consists of two parts: the mouse and the keyboard. When in-game GUIs are not open, the mouse movement is responsible for changing the player's camera perspective. When a GUI is open, it moves the cursor. The left and right buttons are responsible for attacking and using items. The keyboard is mainly responsible for controlling the agent's movement. We use the same joint hierarchical action space as VPT [3], which combines button space and camera space. Button space encodes all combinations of possible keyboard operations (excluding mutually exclusive actions such as `forward` and `back`) and a flag indicating whether the mouse is used, resulting in a total of 8461 candidate actions. The camera space discretizes the range of one mouse movement into 121 actions. Therefore, the action head of the agent is a multi-classification network with 8461 dimensions and a multi-classification network with 121 dimensions. We also filter out frames (both the observation and action) with null action as VPT [3].

In addition, we abstract the crafting and smelting actions with GUI into functional binary actions, which are the same as MineDojo (Fan et al., 2022). These advanced actions are only used by the agents (JARVIS-1 [42], OmniJarvis [43]). The detailed action space is described in Tab. 6.

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

Table 6. The action space we use in Minecraft.

D. Experiment Details

In this section, we provide our experiment details, including benchmark, training details for GROOT and STEVE-1, how we use event-based information as external auxiliary information, and the length pruning algorithm for sub-trajectories.

D.1. Minecraft Skill Benchmark

MCU [30] is a diverse benchmark that can comprehensively evaluate the mastery of atomic skills by agents in Minecraft. Since our method is an improvement on dataset processing instead of a new model architecture, it cannot help agents learn new skills that they are completely incapable of acquiring previously. Therefore, we choose 10 early game skills from the original benchmark that the original agent can already grasp at a basic level. Besides, we add `use torch`, which is also a useful skill in Minecraft. We also add `find and collect wood`, an enhanced version of the skill `collect wood`, which requires the agent to start from the plains instead of the forest, aiming to test its ability to explore and find trees.

Details of the 12 skills in our early game benchmark are shown in Tab. 7. For each skill, we include the evaluation metric and a brief description of what the skill is. For skills "Sleep" and "Use bow", GROOT performs very well on the original metric, so we design a manual metric that better assesses its actual performance. STEVE-1 can take text as prompts, which are also listed in the table. For some of the skills, it is tricky to find proper text prompts, so we use visual prompts instead.

D.2. Training Details

The modified hyperparameters for GROOT and STEVE-1 are listed in Appendix D.2. We adjust parallel GPUs, gradient accumulation batches, and batch sizes to better align with our available computing resources. To speed up convergence without compromising performance, we double the learning rate for GROOT. The total number of frames for STEVE-1 is also modified, as we change the training dataset from a mixed subset of 8.x (house building from scratch), 9.x (house building from random materials), and 10.x (obtaining a diamond pickaxe) to the full 7.x dataset (early game), which better suits our benchmark tasks. [1]

All models are trained parallelly on four NVIDIA RTX 4090Ti GPUs. We follow the same policy training pipeline as the original paper, except for the modified hyperparameters mentioned above. GROOT is trained for three epochs of our dataset. STEVE-1, however, is only trained for 1.5 epochs of our dataset, because we observe that the model starts to overfit on the dataset if it is trained further.

D.3. Event-Based Information

Within the Minecraft environment, we focus on events in the categories "use item", "mine block", "craft item", and "kill entity", as they reflect the player's primary activities. Multiple events may occur simultaneously and are recorded as a set.

[1] OpenAI released five subsets of contractor data: 6.x, 7.x, 8.x, 9.x, and 10.x.

Skill	Metric	Description	Text Prompt for STEVE-1
Use furnace	Craft item cooked mutton	Given a furnace and some mutton and coal, craft a cooked mutton.	-
Hunt Sheep	Kill entity sheep	Summon some sheep before the agent, hunt the sheep.	-
Sleep in bed	**STEVE-1**: Use item white bed. **GROOT**: Sleep in bed properly.	Given a white bed, sleep on it.	Sleep in bed.
Use torch	Use item torch	Give some torches, use them to light up an area. Time is set at night.	Use a torch to light up an area.
Use boat	Use item birch boat	Given a birch boat, use it to travel on water. The biome is ocean.	-
Use bow	**STEVE-1**: Use item bow. **GROOT**: Use item bow 20%, Shoot in distance 40%, take aim 40%	Given a bow and some arrows, shoot the sheep summoned before the agent.	-
Collect stone	Mine block stone (cobblestone, iron, coal, diamond)	Given an iron pickaxe, collect stone starting from cave. Night vision is enabled.	Collect stone.
Collect seagrass	Mine block seagrass (tall seagrass, kelp)	Given an iron pickaxe, collect seagrass starting from ocean.	-
Collect wood	Mine block oak (spruce, birch, jungle, acacia) log	Given an iron pickaxe, collect wood starting from **forest**.	Chop a tree.
Find and collect wood	Mine block oak (spruce, birch, jungle, acacia) log	Given an iron pickaxe, **find** and collect wood starting from **plains**.	Chop a tree.
Collect dirt	Mine block dirt (grass block)	Given an iron pickaxe, collect dirt starting from plains.	Collect dirt.
Collect grass	Mine block grass (tall grass)	Given an iron pickaxe, collect grass starting from plains.	Collect grass.

Table 7. Details of 12 atomic skills in our Minecraft skill benchmark for testing GROOT and STEVE-1.

Hyperparameter	Value
Learning Rate	0.00004
Parallel GPUs	4
Accumulate Gradient Batches	1
Batch Size	8

Hyperparameter	Value
Parallel GPUs	4
Accumulate Gradient Batches	4
Batch Size	4
n_frames	100M

Table 8. Modified hyperparameters for training controllers. **(Left)** GROOT. **(Right)** STEVE-1.

Only the final step of a repeating sequence of sets of events is marked as positive. For example, in the sequence: ("use item iron pickaxe", "mine block iron ore"), ("use item iron pickaxe", "mine block iron ore"), ("use item iron pickaxe", "mine block diamond ore"), ("use item torch"), ("use item torch"), the second, third, fifth steps will be marked as positive. Additionally, for any step t that includes a "kill entity" event, we mark $t + 16$ as positive instead of t, because there will be a short death animation that follows the entity's death, and we want it to be included in the same segment.

D.4. Length Pruning Algorithm

STEVE-1 forces a minimum and maximum length of trajectories in its method, so we need a length pruning algorithm. In our implementation, the length of each trajectory is pruned as follows: If a trajectory is too short, it is merged with subsequent trajectories until it meets the minimum length requirement. If this results in a trajectory exceeding the maximum length, it

is truncated, and the remainder forms the beginning of the next trajectory. For instance, given a minimum and maximum length of 15 and 200, sub-trajectories of lengths 12, 12, 6, 196, and 37 are adjusted to 24, 200, and 39. Here, the first two sub-trajectories are merged, while the 6 is combined with 196 but truncated at 200, with the remaining 2 merged into the next trajectory. There might be more sophisticated strategies, but we use this straightforward algorithm for simplicity.

E. Examples of Skill Videos

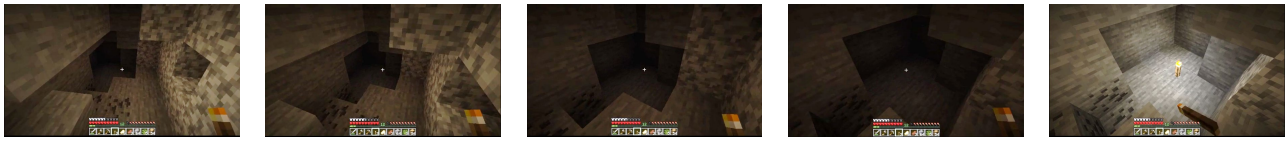
We sample one video segmented by our method for each skill in the benchmark and an extra video showing a failure case, presenting each video with five screenshots. The first and last screenshots correspond to the first and last frames of the video, while the other three are manually selected to best illustrate the progression of the skill.

- smelt food

- hunt sheep

- sleep

- use torch



- use boat

- use bow

- collect stone

- collect seagrass

- collect wood

- collect dirt

- collect grass

- combat spider (failure case: overly short)