

# Minecraft in Reinforcement Learning

Kushal Khangaonkar, Rahul Shah

## 1. Extended Abstract

Extensive work has been done within the past year to build Minecraft agents with internet-scale knowledge. In Nvidia's paper [1], these agents can take in open-ended as well as language-prompted tasks and use their new model MineCLIP to find the correlation of an action with the text instruction. This correlation is then used as a reward signal to train a policy network to perform the task.

The survival task (essentially the main goal of minecraft) doesn't have a well-defined reward, nor is there a clear language instruction to use with MineCLIP. With recent advances in language models, we propose using a language model to generate a prompt for the agent to perform several smaller scale tasks in success. We experimented with a couple Large Language models, including Cohere.ai's and OpenAI's state of the art models. We had good success with GPT in processing observations to create new instructions. Periodically generating a new instruction with GPT provides us a way to create a reward signal with MineCLIP, with the underlying goal being to survive as long as possible.

We systematically utilized the recently released ChatGPT system from OpenAI and autonomously queried it to help lead our agent to survival. This was done with the intent and goal for the agent to be able to survive longer when given a prompt from a language model than when given a prompt from a human. For this to happen, we realized that not only would ChatGPT's prompts need to be more meticulous and specific, but also that ChatGPT needed to have relevant context from the agent's observations. This needs to be standardized between training and testing, so we created a new dataset of Minecraft observations and ChatGPT prompts.

To train agents, we base our experiments off of Proximal Policy Optimization (PPO)[3]. We choose this algorithm as it runs quickly and works well on continuous control tasks. Our work can be easily applied to any other RL algorithm that makes use of a neural network to estimate  $\pi$ , whether it is a regular policy gradient method or an actor-critic method. In training our agents, we use the MineCLIP reward signal to train our agents to perform the tasks that ChatGPT prompts them to do. We also use a reward signal that is based on the agent's survival time, which is the main goal of the game. We found that the MineCLIP reward signal was more effective at training the agent to perform the tasks that ChatGPT prompts it to do, but the survival time reward signal was more effective at training the agent to survive longer.

Overall, our results were mixed: We had good success with ChatGPT in processing observations to create new instructions, but we were unable to train an agent that really learned how to survive. We hypothesize a variety of reasons for this, but mainly we believe that the training times needed to be extremely long for the agent to see all the different situations it might need to. A day in minecraft is 24k timesteps, so even a 1 million timesteps is not very many days. Additionally, the observation space is tremendous, and we observed that the agent was experiencing a variety of observations that are necessary to learn how to survive.

Future work would involve refining the training process to include more variety in terms of what equipment/items the agent sees/uses, what mobs/monsters it interacts with, and many more crucial aspects of survival. In theory, an agent that is capable of surviving infinitely is quite similar to the idea of a language model that can perform many tasks zero shot. It seems like GPT + MineCLIP generated rewards are a reasonable way to do this, but like Large Language Models, the training data and overall train time needs to be extremely long in order to have a variety of capabilities.

Additionally, we are interested in exploring how this style of language based instruction algorithms works in smaller, easily trainable environments. It made sense to use MineDojo to easily utilize the novel MineCLIP model (fine-tuning a new CLIP model would have been hard in the allotted timeframe of a semester).

## 2. Complexity of the Minecraft Survival Task

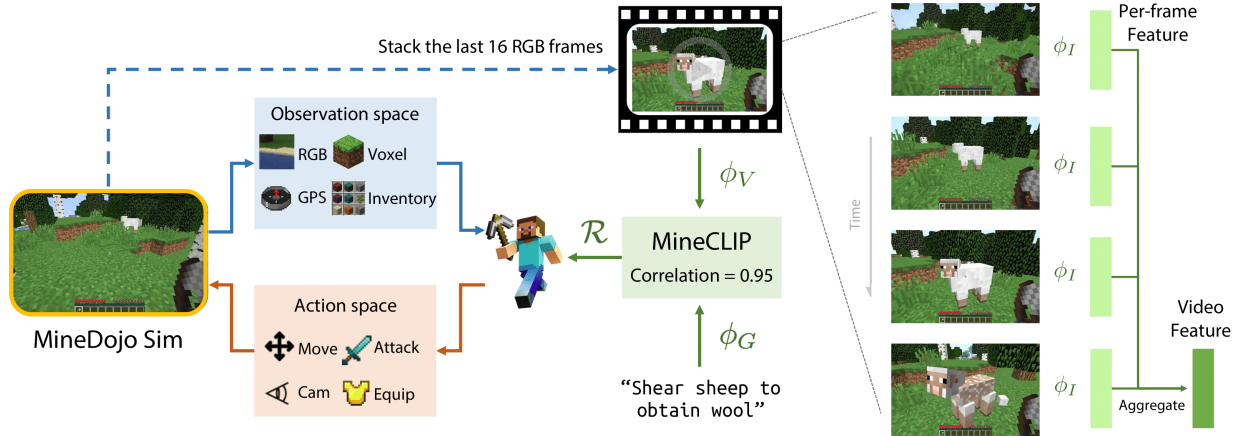


Fig. 1. An overview of the MineCLIP Pipeline – MineDOJO Paper[1]

In the original MineDojo paper, no agent was trained on the survival task, probably because its an extremely complicated task for an agent to learn. They define the task as sparse reward function with a reward of +1 given for every day survived (Optional larger reward for N days survived for trainer specified N). Similar to the other programmatic tasks, they attach a text instruction for survival given a certain number of days:

'Survive for 2 days.'

Of course, it is not viable to use MineCLIP rewards with this instruction, because this instruction is too abstract for the types of data MineCLIP was trained on. If we examine the success of the MineCLIP model on programmatic tasks, its most effective at providing rewards for very granular tasks (eg. combat against a spider or a zombie), and small tasks like “milk a cow”.

Table 1: Our novel MINECLIP reward model is able to achieve competitive performance with manually written dense reward function for Programmatic tasks, and significantly outperforms the CLIP<sub>OpenAI</sub> method across all Creative tasks. Entries represent percentage success rates averaged over 3 seeds, each tested for 200 episodes. Success conditions are precise in Programmatic tasks, but estimated by MineCLIP for Creative tasks.

Group	Tasks	Ours (Attn)	Ours (Avg)	Manual Reward	Sparse-only	CLIP <sub>OpenAI</sub>
	Milk Cow	<b>64.5 ± 37.1</b>	6.5 ± 3.5	62.8 ± 40.1	0.0 ± 0.0	0.0 ± 0.0
	Hunt Cow	<b>83.5 ± 7.1</b>	0.0 ± 0.0	48.3 ± 35.9	0.3 ± 0.4	0.0 ± 0.0
	Shear Sheep	12.1 ± 9.1	0.6 ± 0.2	<b>52.3 ± 33.2</b>	0.0 ± 0.0	0.0 ± 0.0
	Hunt Sheep	8.1 ± 4.1	0.0 ± 0.0	<b>41.9 ± 33.0</b>	0.3 ± 0.4	0.0 ± 0.0
	Combat Spider	80.5 ± 13.0	60.1 ± 42.5	<b>87.5 ± 4.6</b>	47.8 ± 33.8	0.0 ± 0.0
	Combat Zombie	47.3 ± 10.6	<b>72.3 ± 6.4</b>	49.8 ± 26.9	8.8 ± 12.4	0.0 ± 0.0
	Combat Pigman	1.6 ± 2.3	0.0 ± 0.0	<b>13.6 ± 9.8</b>	0.0 ± 0.0	0.0 ± 0.0
	Combat Enderman	0.0 ± 0.0	0.0 ± 0.0	0.3 ± 0.2	0.0 ± 0.0	0.0 ± 0.0
	Find Nether Portal	37.4 ± 40.8	<b>89.8 ± 5.7</b>	N/A	N/A	26.3 ± 32.6
	Find Ocean	33.4 ± 45.6	<b>54.3 ± 40.7</b>	N/A	N/A	9.9 ± 14.1
	Dig Hole	<b>91.6 ± 5.9</b>	88.1 ± 13.3	N/A	N/A	0.0 ± 0.0
	Lay Carpet	97.6 ± 1.9	<b>98.8 ± 1.0</b>	N/A	N/A	0.0 ± 0.0

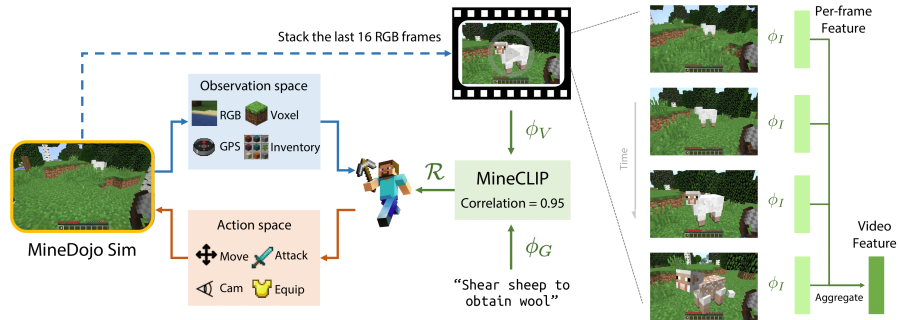
Fig. 2. Results from the MineDOJO Paper for context[1]

Even without considering effective use of MineCLIP, It is hard to construct a comprehensive instruction that captures the entirety of a agent's survival campaign. In order to survive indefinitely, one has to procure a reliable food source, mine a variety of resources, collect many different items, all while being able to survive the mobs and monsters trying to kill the agent. This led us to consider breaking down the survival task into smaller tasks to calculate reward from.

### 3. Strategies for generating subtasks

Unlike other tasks in the Minedojo knowledge base, we cannot define sub-tasks before training because the situation of the minecraft agent is infinitely variable. It is necessary to generate subtask instructions on demand, which is why we propose using a Large Language Model (LLM) to generate instructions periodically to feed to MineCLIP.

Our new idea is to periodically update the language task by passing in observations and context to an LLM using the following pipeline. Effectively this would be updating  $\phi_G$ , which is a constant in the original pipeline as shown in Fig 1.



In order to generate a good language task, we discussed what observations would make sense as context for the LLM, and also what was going to be feasible for the model to actually respond to (passing in RGB frames makes little sense). Since some key observations in Minecraft are actually text based, a good LLM should be able to recognize the observations as variables when outputting the next language goal.

Possible numerical observations we considered:

**Health [0, 20]: Agent dies once this reaches 0.**

**Hunger (Food) [0, 20]: Agent begins taking starvation damage once this reaches 0.**

**Oxygen level [0, 300]: Agent begins taking suffocation damage once this reaches 0.**

Damage Amount for each possible damage source (hunger, projectile, etc.) [0, inf]: How much damage the agent took in the timestep.

Light Level [0,15]: More mobs spawn when light levels are low.

Possible text/boolean observations we considered:

**Inventory: agent can hold 36 items (with various counts of each item)**

**Equipment: agent can wear armor and currently be using certain tools/items.**

**Biome: What the landscape surrounding the agent is (e.g. Forest, Desert, Jungle)**

Is Sleeping: If the agent is sleeping

**Is Crafting Table Nearby: does the agent have access to a crafting table**

**Is Furnace Nearby: does the agent have access to a furnace**

Boolean for whether a damage source is currently damaging the agent

Changes in inventory items (by name and value)

Bolded items are the ones that worked best with our LLM prompting experiments.

### 4. Experiments with LLMs

We spent most of the project working with Cohere.ai's xlarge model for generation, and eventually used ChatGPT by OpenAI when it was released for the public for a few weeks. With Cohere, we worked on constructing a prompt structure to capture the necessary observations.

## 4.1 Cohere.AI

Our initial experiments with Cohere proved that the model was very difficult to get successful results with for this task. We hypothesize that Cohere’s training data did not have very much Minecraft related content (certainly compared to any OpenAI model). This caused it to struggle with interpreting observations we gave. As a result, we tried to use few shot examples to try and “teach” Cohere to generate prompts in the style we wanted.

One key positive takeaway is that with this few-shot prompting (N=4 in our trials) cohere seems to do a good job recognizing the observations being given. For example if part of the prompt contains:

```
"\nEquipment: '64 wood', '37 cobblestone',
'Iron Sword', 'Iron Pickaxe', 'Iron Axe',
'Wooden Axe'"
```

Cohere returns:

1. Collect resources from nearby trees, by chopping them with the 'Iron Axe'.
2. Use 'Iron Pickaxe' to collect ores
3. Use 'Iron Sword' to kill nearby mobs

However this was very inconsistent: Sometimes it wouldn’t use any of the context we had given, and even worse, sometimes it would use context from the few-shot examples. Other outputted goals would simply not be reasonable goals for an agent to do with the overarching goal being to survive as long as possible, for example:

1. Pick up resources to build a chest and a crafting table
2. Create a chest
3. Find the nearest village and give all the collected resources to the villagers

See Appendix A. for an example few-prompt with Cohere.

Initially, we tried to have Cohere generate multiple steps to be used in sequence, because we thought generating one step would cause the model to be overdetailed in its response and we could simply use the 3 instructions one after the other in our training. Later when we started working with ChatGPT, we moved to an architecture that passed in the previous two instructions to generate only one new response.

## 4.2 ChatGPT

Since Cohere was fairly unsuccessful, we were lucky to be able to start working with ChatGPT to keep experimenting. With ChatGPT we immediately had qualitatively better results as it seemed like ChatGPT had a better understanding of specific Observations we passed in. It also is more consistent with utilizing observations to output clear and specific goals. At zero-shot, ChatGPT was able to output very good understanding of the goals in Minecraft, for example given the following prompt:

```
You are given the following information about a minecraft player's state, and need to
provide them with an instruction for what to do.
Health: 15/20
Food: 20/20
Biome: Jungle
Weather: Rainy
Time: Night
Equipment: '64 wood', '37 cobblestone', 'Iron Sword', 'Iron Pickaxe', 'Iron Axe', 'Wooden
Axe'
Instructions:
1.
```

ChatGPT outputs:

```
Based on the information provided, it sounds like the player is in good health and has
enough food to sustain themselves for the time being. Since it is nighttime in a jungle
```

biome and the weather is rainy, the player may want to consider finding a safe place to spend the night, such as a treehouse or cave. The player can use their wooden axe to gather wood and cobblestone to build a shelter, and their iron tools to defend themselves against any hostile mobs that may be present. Once they have found a safe place to spend the night, they can rest and regain any lost health.

This is extremely detailed and mostly correct, but far too much information for a language goal, so we think few shot is still necessary to help shape the outputted goal into a format that will be successful when inputted into MineCLIP.

See Appendix B. for an example few-shot prompt with ChatGPT

We came up with dozens of example few-shot examples and chose the 4 best based on the following qualitative criteria:

1. Outputted Instructions must be a succinct sentence similar in length to language tasks in the MineDojo knowledgebase
2. Instruction must not invoke and equipment or inventory items that aren't currently possessed by the agent
3. Instruction should not ask the agent to do multiple tasks in one instruction, i.e. collect more than one new item at a time
4. Instruction should reflect the previous instructions and use the observations as essentially the result of the previous instructions.
5. Instruction should prioritize staying alive over creative/expansive actions.

Overall we were able to find decent success in all of these criteria but 3. proved to be somewhat difficult at first. ChatGPT is trained to be a conversational model so it seems to edge towards more detail than less. By adding instructional details like "Next Instruction *limited to a single sentence*" and making our few-shot example instructions all very concise we were able to combat this. However, in doing this, we lost a bit of specificity in the instruction. For example, in a prompt with Food status 5/20, the response was

Collect food until you are at 20/20.

Which isn't a bad instruction, but without the prompt direction pushing for simplicity, the model would have given extra detail like:

Collect food until you are at 20/20. In the Plains biome there are often sheep which offer wool to make a bed with, but also a good source of food. You will need a furnace

It is difficult to experimentally verify which is better, but due to the extra information that doesn't matter, we move forward with the simpler instructions. In future work, we might be able to examine how the MineCLIP rewards for a specific instruction vary with different agent situations and compare between various complexities of instructions.

Currently, we plan on regenerating the task on a set time schedule, i.e. once every  $X$  environment timesteps.  $X$  should be big enough that the agent has enough time to finish a specific task, but also small enough such that the agent isn't left idling and the reward signal becomes meaningless. Additionally, it might make more sense to edge on the side of smaller intervals because its possible the LLM might be able to recognize whether the task is finished or not, and suggest a follow up language goal.

## 5. Training Agents for the Survival Goal

For us to effectively train agents in order to sustain multiple days in Minecraft, we make sure to train under a variety of environments and reward functions. By having our agent train under a diverse set of game worlds varying biomes, structures, as well as the inherent randomness provided by Minecraft itself, we are able to challenge the Agent to choose actions that allow it to survive longer.

We tried various reward functions, each with a policy optimized with PPO for fair comparison:

### 1. Experiment 1:

- Sparse Reward
- PPO with 4 environments
- Average Pooling CNN Policy

### 2. Experiment 2:

- Sparse Reward
- PPO with 4 environments
- CNN Policy with a Frame Stacking of 2 frames

3. Experiment 3:

- Dense Reward 1
  - -1 if you lose health
  - -1 if you lose a point on the food bar (hunger)
  - +1 if you survive a day
- PPO with 4 environments
- CNN Policy with a Frame Stacking of 2 frames

4. Experiment 4:

- Dense Reward 1
  - -100 if you lose health for the first time
  - -1 if you lose health again
  - -50 if you lose a point on the food bar (hunger)
  - -1 if you lose a point on the food bar (hunger) again
  - -1 for each oxygen lost
  - +1 if you survive a day
- PPO with 4 environments
- CNN Policy with a Frame Stacking of 2 frames

5. Experiment 5:

- MineCLIP Reward from 16 frames with initial survival text
- PPO with 4 environments
- MineCLIP video features

6. Experiment 6:

- MineCLIP Reward from 16 frames with various survival text update strategies
- PPO with 4 environments
- CNN Policy with Frame Stacking of 2 frames

7. Experiment 7:

- MineCLIP Reward from 16 frames with various survival text update strategies
- PPO with 4 environments
- MineCLIP video features

each of these had a `target_days` set to 2 and we trained each CNN for 25 to 40 days worth of timesteps.

We further improved training by fine-tuning inputs as we noticed that the agent was not prioritizing food as much as needed for survival. Additionally, we noted that the model was not tuned to pay attention to brightness which is important for avoiding monsters. Therefore we engineered our prompting to include few-shot examples where making a torch is an instruction.

---

**Algorithm 1** Training Algorithm with the Goal of Survival. Note that knowledge of common RL algorithms are assumed and thus omitted for brevity.

---

**Require:** Biome  $B$

**Require:** Structures  $S$

**Require:** Randomness entropy  $R$

**Ensure:** Environment  $E \triangleq B \times S \times R$

$\Pi \leftarrow \{\pi_{\theta_1}, \pi_{\theta_2}, \dots, \pi_{\theta_n}\}$

$N \leftarrow n$

**while**  $N \neq 0$  **do**

**for**  $\pi_{\theta_i} \in \Pi$  **do**

**for**  $t = 1, 2, \dots, N$  **do**

      Run previous policy in  $E$  for  $t$  steps

**for**  $f \in \mathcal{F}$  **do**

$\hat{A} \leftarrow \mathbb{E}_f[A]$

**end for**

$\phi \leftarrow \text{NN}(\hat{A})$

$N \leftarrow N - 1$

**end for**

$\mathcal{L} \leftarrow$  original RL loss under the policy

$\rho \leftarrow$  penalties for missing food, low brightness, etc.

$\mathcal{L} \leftarrow \mathcal{L} + \rho$

$\theta'_i \leftarrow \theta'_i - \alpha \nabla_{\theta'_i} \mathcal{L}$

    ▷ This optimization is done over the surrogate tractable Loss  $\mathcal{L}$

**end for**

**end while**

---

Finally, we made sure to skillfully represent the Minecraft inventory at any timestep via a multi-hot encoded vector, using gym's MultiDiscrete space class. [12] This allowed us to easily represent the inventory of the agent, and we were able to use this to fine-tune the model to prioritize food (to prevent hunger) and torches (to prevent a lack of brightness and monsters). We also used this to fine-tune the model to prioritize the agent's health, which is important for survival. We found that this can be a very effective way to fine-tune the model to prioritize the agent's survival, as the agent was able to respond to changes in the environment. However we were not able to see a drastic effect in short-term survival in the plots on the Appendix below. We believe that this is due to the environment being so sparse that the training we were able to run had such limited compute that it was unable to capture an accurate fraction of reality.

## 6. Appendix

### 6.1 Cohere Few-Shot Prompt

A minecraft player is trying to survive for 10 days. You are given the following information about their state, and need to provide them with an instruction for what to do.

Health: 15/20

Food: 5/20

Biome: Desert

Weather: Sunny

Time: Day

Equipment: 'Iron Sword', 'Wooden pickaxe', 'furnace', '32 wood planks', '3 rotten flesh'

Instructions:

1. Find food as soon as possible.
2. Find a chicken and kill it using 'Iron Sword'.
3. Collect the resources and eat them.

You are given the following information about a minecraft player's state, and need to provide them with an instruction for what to do.

Health: 20/20

Food: 20/20

Biome: Forest

Weather: Rainy

Time: Day

Equipment: None

Instructions:

1. Collect wood from nearby trees
2. Create wood planks by using wood in crafting
3. Build a shelter using the wood planks

You are given the following information about a minecraft player's state, and need to provide them with an instruction for what to do.

Health: 1/20

Food: 10/20

Biome: Forest

Weather: Cloudy

Time: Night

Equipment: 'Wooden Sword', '10 Steak'

Instructions:

1. Run away from mobs and monsters
2. Eat Steak until food status is back to 20
3. Hide in an available shelter

You are given the following information about a minecraft player's state, and need to provide them with an instruction for what to do.

Health: input/20

Food: input/20

Biome: input

Weather: input

Time: input

Equipment: input

Instructions:

- 1.



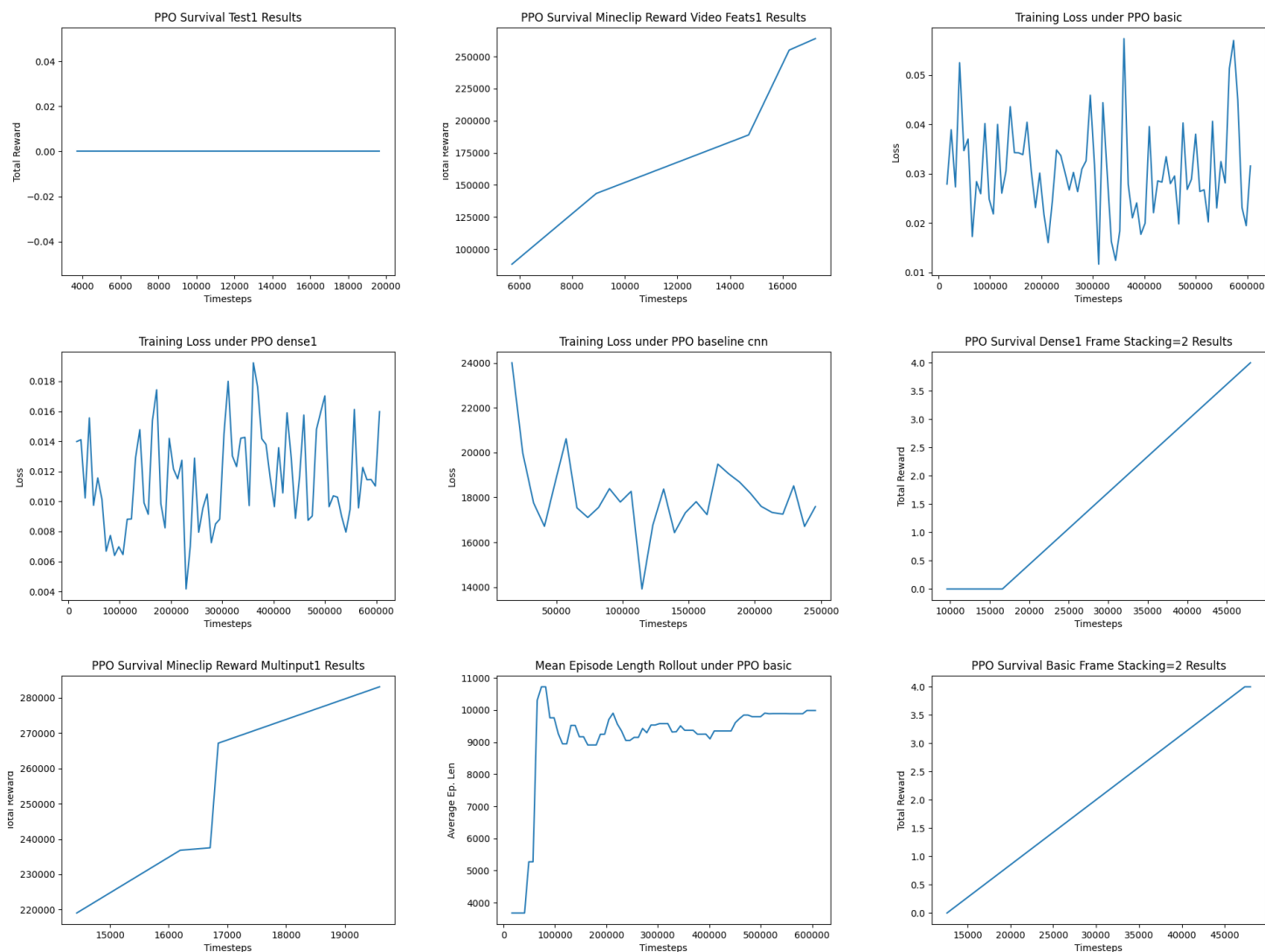
## 6.2 Successful ChatGPT Few-Shot Prompt

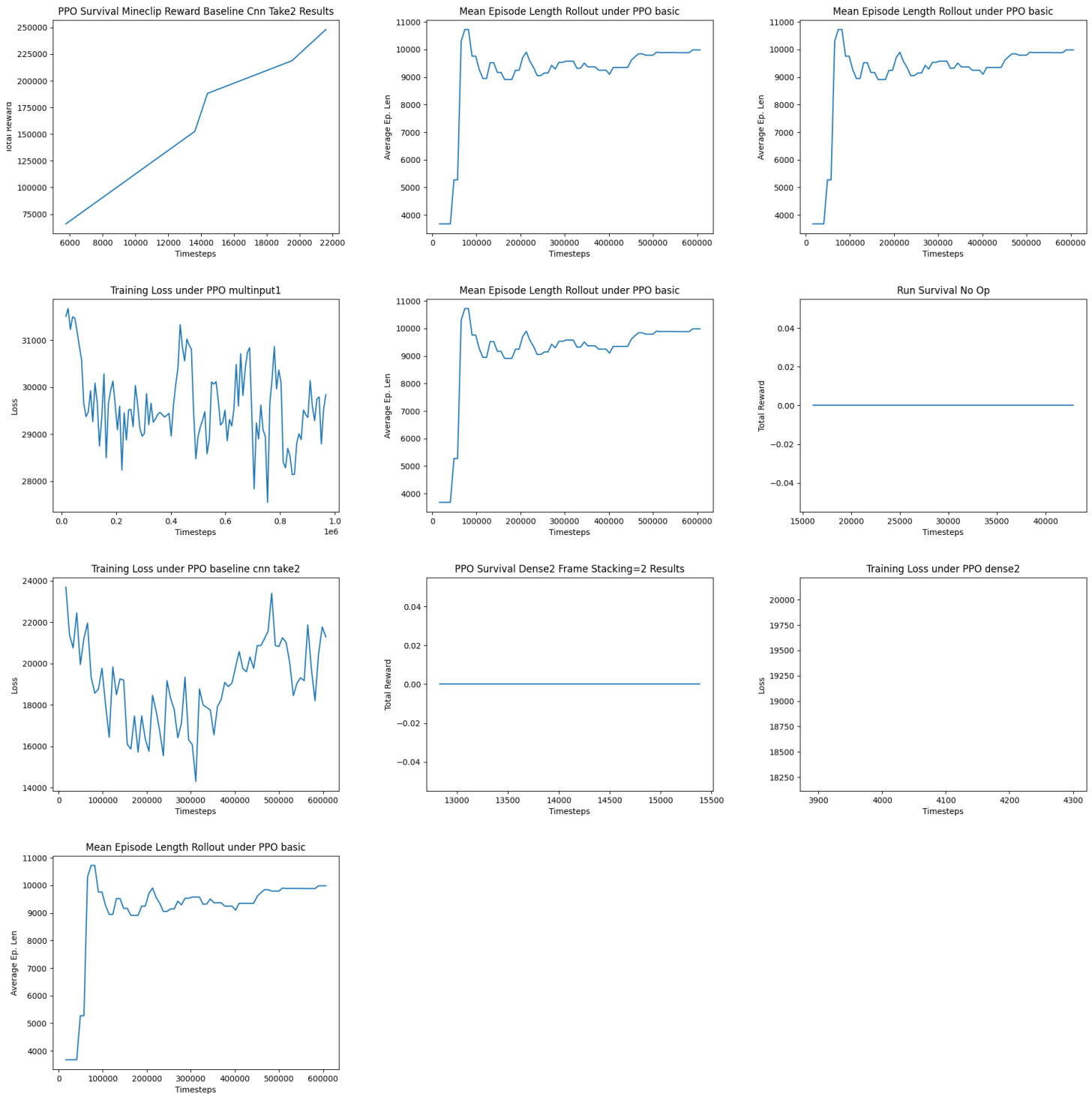
A minecraft player is trying to survive for as long as possible.  
 Given the following information about their state, you must provide them with an instruction for what to do next. Make sure to eat food any time it is not full.  
 Health: 6/20  
 Food: 15/20  
 Oxygen: 300/300  
 Biome: jungle  
 light\_level: 3/15  
 Crafting Table Nearby: False  
 Furnace Nearby: False  
 Inventory: {"jungle logs": 3}  
 Previous Instructions:  
 1. Run away from mobs and monsters until health is recovered.  
 2. Collect food until you are at 20/20.  
 Next Instruction, limited to a single sentence:  
 Find an animal to kill, and eat food to recover health.  
 A minecraft player is trying to survive for as long as possible.  
 Given the following information about their state, you must provide them with an instruction for what to do next. Make sure to eat food any time it is not full.  
 Health: 20/20  
 Food: 18/20  
 Oxygen: 300/300  
 Biome: plains\_m  
 light\_level: 13/15  
 Crafting Table Nearby: True  
 Furnace Nearby: False  
 Inventory: {"coal": 5, "sticks": 3, "wooden pickaxe": 1}  
 Previous Instructions:  
 1. Build a pickaxe using the crafting table with the wood and sticks.  
 2. Find coal and mine it using wooden pickaxe.  
 Next Instruction, limited to a single sentence:  
 Collect food until you are at 20/20.  
 A minecraft player is trying to survive for as long as possible.  
 Given the following information about their state, you must provide them with an instruction for what to do next. Make sure to eat food any time it is not full.  
 Health: 20/20  
 Food: 20/20  
 Oxygen: 300/300  
 Biome: flower\_forest  
 light\_level: 15/15  
 Crafting Table Nearby: False  
 Furnace Nearby:  
 Inventory: {}  
 Previous Instructions:  
 1. Explore the world until you find a tree or source of wood  
 2. Chop down wood until you have 64 wood  
 Next Instruction, limited to a single sentence:  
 Use the wood to build a crafting table  
 A minecraft player is trying to survive for as long as possible.  
 Given the following information about their state, you must provide them with an instruction for what to do next. Make sure to eat food any time it is not full.

Health: 8/20  
 Food: 20/20  
 Oxygen: 300/300  
 Biome: plains\_m  
 light\_level: 13/15  
 Crafting Table Nearby: True  
 Furnace Nearby: False  
 Inventory: {"coal": 5, "sticks": 3, "wooden pickaxe": 1, "steak": 13}  
 Previous Instructions:  
 1. Build a shelter using the crafting table with the wood and sticks.  
 2. Add a furnace to the shelter using the crafting table.  
 Next Instruction, limited to a single sentence:  
 Hide in the shelter and eat food until health is recovered.

The context for the real question is built off of the observation space in the same style of the few-shot examples.

### 6.3 Plots and Figures





## 6.4 Individual Contributions

- Kushal: Worked on code and experiments for MineDOJO including policy methods
- Rahul: Worked on fine-tuning with Cohere and ChatGPT for diverse pre-training

## References

- [1] MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, Anima Anandkumar. <https://arxiv.org/pdf/2206.08853.pdf>
- [2] Brown, Tom, et al. “Language models are few-shot learners.” *Advances in neural information processing systems* 33 (2020): 1877-1901.
- [3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.
- [4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.
- [5] <https://github.com/kushal1248/285-final-project>
- [6] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *CoRR*, abs/1802.06070, 2018.
- [7] Josh Abramson, Arun Ahuja, Iain Barr, Arthur Brussee, Federico Carnevale, Mary Cassin, Rachita Chhaparia, Stephen Clark, Bogdan Damoc, Andrew Dudzik, Petko Georgiev, Aurelia Guy, Tim Harley, Felix Hill, Alden Hung, Zachary Kenton, Jessica Landon, Timothy Lillicrap, Kory Mathewson, Sona Mokrá, Alistair Muldal, Adam Santoro, Nikolay Savinov, Vikrant Varma, Greg Wayne, Duncan Williams, Nathaniel Wong, Chen Yan, and Rui Zhu. Imitating interactive intelligence. *arXiv preprint arXiv: Arxiv-2012.05672*, 2020.
- [8] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, and Mengyuan Yan. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv: Arxiv-2204.01691*, 2022
- [9] Rami Al-Rfou, Marc Pickett, Javier Snider, Yun hsuan Sung, Brian Strope, and Ray Kurzweil. Conversational contextual cues: The case of personalization and history for response ranking. *arXiv preprint arXiv: Arxiv-1606.00372*, 2016.
- [10] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of Proceedings of Machine Learning Research*, pages 166–175. PMLR, 2017. URL <http://proceedings.mlr.press/v70/andreas17a.html>.
- [11] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn

- Zhou, and Percy Liang. On the opportunities and risks of foundation models. arXiv preprint arXiv: Arxiv-2108.07258, 2021.
- [12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. arXiv preprint arXiv: Arxiv-1606.01540, 2016.
- [13] Stiennon, Nisan, et al. “Learning to summarize with human feedback.” Advances in Neural Information Processing Systems 33 (2020): 3008-3021.
- [14] Gao, Leo, John Schulman, and Jacob Hilton. “Scaling Laws for Reward Model Overoptimization.” arXiv preprint arXiv:2210.10760 (2022).
- [15] The inspiration for this contest comes in part from work by Kenway, Josh, Camille François, Sasha Costanza-Chock, Inioluwa Deborah Raji, and Joy Buolamwini. Bug Bounties For Algorithmic Harms? Lessons from Cybersecurity Vulnerability Disclosure for Algorithmic Harms Discovery, Disclosure, and Redress. Washington, DC: Algorithmic Justice League. January 2022. Available at <https://ajl.org/bugs>.
- [16] Brundage, Miles, Avin, Shahar, Wang, Jasmine, Belfield, Haydn, and Gretchen Krueger et al. “Toward Trustworthy AI Development: Mechanisms for Supporting Verifiable Claims,” April 2020. Available at <https://arxiv.org/abs/2004.07213>. See an earlier instance of such a competition at HackerOne. 2021b. “Twitter Algorithmic Bias.”.
- [17] Rubinovitz, JB, “Bias Bounty Programs as a Method of Combatting Bias in AI,” August 2018. Available at <https://rubinovitz.com/2018/08/01/bias-bounty-programs-as-a-method-of-combatting>.