DDL Final Report
ReLearnCraft: Reinforcement Learning with Behavioral Cloning

Jason Nguyen, Arya Shetty, Vineal Sunkara

Problem/Task

Minecraft is a 3D sandbox game where players are free to explore and creatively express themselves by placing and destroying blocks. With Minecraft being such an open game, it is challenging to develop a simple policy to perform specific tasks, not to mention the player's limited sightlines and vast action space. Despite this, we sought to create an agent, in the place of a human player, capable of performing tasks within Minecraft.

Referenced Paper/Code

We aimed to recreate the results from OpenAI's VPT (Video PreTraining) paper[1], which trained a foundation model on 70k hours of labeled Minecraft gameplay and fine-tuned it for tasks through imitation learning and reinforcement learning. The data was acquired by collecting contractor recordings where a player accomplishes a goal in Minecraft. An inverse dynamics model is then trained on the contractor data, allowing them to label any Minecraft video data input automatically. This method yielded most of their 70k hours of labeled gameplay[2]. The repository associated with this paper provided these foundation models and scripts for behavioral cloning and evaluation.

Methodology

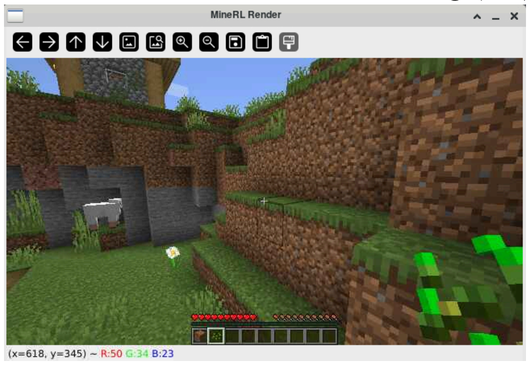
Using labeled video gameplay from MineRL's BASALT (Benchmark for Agents that Solve Almost-Lifelike Tasks) competition dataset[2], we started by fine-tuning the foundation models to get task-oriented weights using imitation learning/behavioral cloning[3]. With these weights, we could further direct the agent using reinforcement learning. The agent would attempt to maximize its reward based on outputs from a given MineRL scenario, providing the agent with a positive feedback loop aligned with human goals. For example, if the scenario dictates the agent to find a cave within the game, the rewards would be directly tied to how close the agent is to an identifiable cave. As a result, it may try unique solutions to maximize rewards, similar to how a human can think critically about the most optimal solution forward.

The VPT repository from the OpenAI paper provided us with most of the structure and code used in the original paper; however, the provided scripts are stripped-down versions designed for consumer hardware, which need to adhere to memory restrictions and only utilize one GPU. With OpenAI's foundation models' complexity, the original script for behavioral cloning could only train so many batches, resulting in poor results after fine-tuning. Our specific model's architecture included multi-headed attention for the memory and giving the agent context of the situation it's in, an IMPALA (Importance Weighted Actor-Learner Architecture) CNNs for extracting features from the RGB pixels of the agent POV state (640x360), and the policy network outputting probabilities for the Minecraft action state. Since we had access to

[1] [2206.11795] Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos (arxiv.org)
[2] https://www.aicrowd.com/challenges/neurips-2022-minerl-basalt-competition
[3] https://github.com/openai/Video-Pre-Training

supercomputers with commercial-grade NVIDIA A100 GPUs, we modified the script to utilize multiple GPUs through Pytorch's Distributed Data-Parallel module and compute gradients for entire batches simultaneously instead of using gradient accumulation recommended for single GPUs. After these optimizations, our training time improved significantly and was much faster than the VPT repository's model. Despite this, our fine-tuned models for the four tasks provided by the dataset were not great.

The existing tasks might have been too complex to fine-tune effectively since the supercomputer's session quota limited our training time and we could only download about 100 videos for each task before running out of space. We decided to create our own data for an easier task to get around this issue. At first, we tried using the Inverse Dynamics Model provided in the VPT repository[2], which uses a model to automatically label videos, facilitating the creation of new training data. However, the results were unreliable, as the model mislabeled a "Left-Click Hold" action as repeated "Left-Click Press" actions, which fails when the player tries to chop a tree. Ultimately, we created our own recording script that directly captured the MineRL gameplay render and saved the keyboard and mouse inputs into a JSON file. Using this data, we got successful results to use as a base for reinforcement learning (RL).



The final step in the process was the actual RL training for the agent. Here, we implemented a custom reinforcement learning model using PPO (Proximal Policy Optimization) and Stable-Baseline-3 since we hoped it would perform better in an environment with discrete and continuous actions. The script would take in the agent's screen in the render, specifically the "Obtain diamond shovel" since it was the only one with built-in rewards and predict an action to perform. Unlike behavioral cloning, the weights would be updated based on the accumulation of rewards for its specific task. Training the agent with no weights proved unsuccessful, as the actions were too random; although we wanted to use our behavioral cloning weights, this proved impossible due to compatibility issues between the model and our weights. Ideally, the proposed weights after RL fine-tuning would achieve performance similar to those in the VPT repository, in which we could see the agent excelling at complicated tasks, such as obtaining diamonds.

Computing Resources

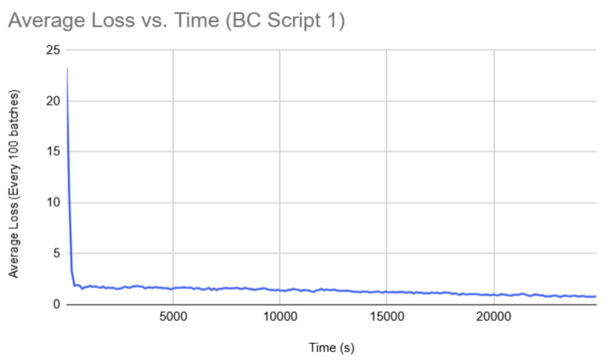
Regarding resource management, we utilized both supercomputers, Perlmutter and Delta. Initially, we started with Perlmutter, but we had a lot of trouble getting MineRL to render on Perlmutter. This was due to GLFW errors (OpenGL was not properly configured) and the inability to resolve them since we didn't have root access to download the required packages. Even on the NoMachine client, the environment couldn't render it. On Delta however, the agent was able to be tested, as OpenGL and Nvidia drivers were properly configured. For the training of the first two scripts, you will see below that only required one GPU. This is because we used

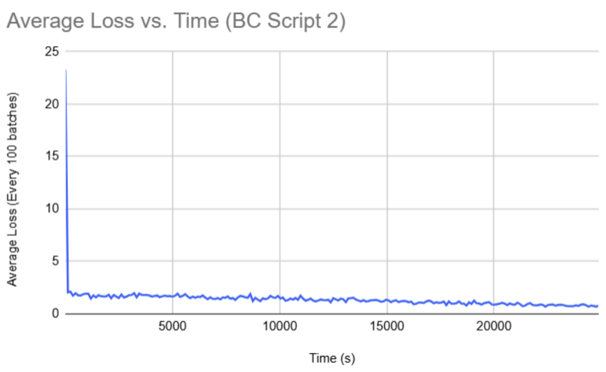
Desktop sessions in Delta, which essentially created a Remote Desktop environment for us to use the MineRL environment. Eventually, when it came to testing multinode training, our salloc commands wouldn't work, and so we had to switch back to Perlmutter to train headless to get weights more efficiently with DDP. Even then, we could only train for around an hour without waiting in long queues for more hours of multinode training. Eventually, we used SCP to transfer the weights back to Delta for rendering and testing. Overall, we used many hours on Delta for most of the testing and training and some on Perlmutter for DDP training.

Results

The first two scripts we used for training, the VPT's behavioral cloning[4] and our custom implementation optimized for one GPU, were run on 100 videos, around 100 epochs, and a batch size of 64. We also kept these hyperparameters the same for all training:

Learning_Rate = 0.0001
Weight_Decay = 0.0001
Max_Grad_Norm = 5.0
Loss_Report_Rate = 100



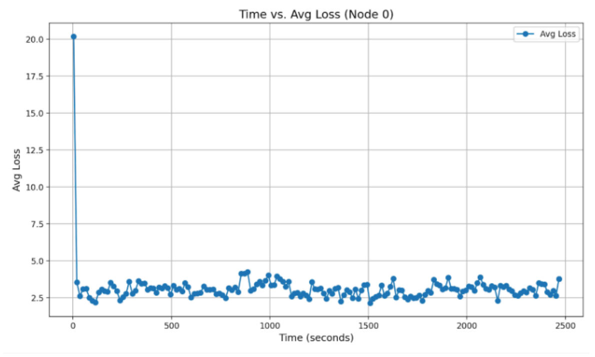


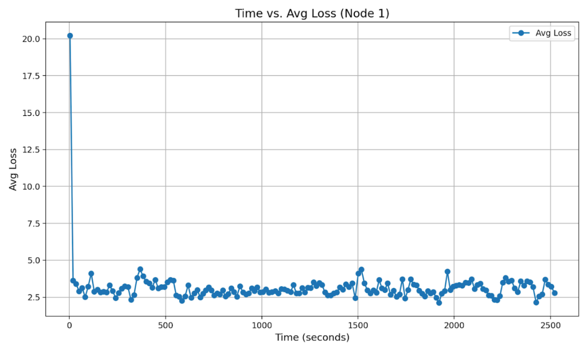
Behavioral cloning script from VPT repository (left) versus our modified script (right)

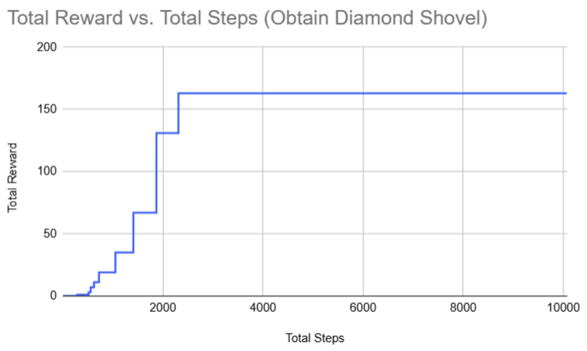
In the figures above, the training loss over time is included to show how closely the agent mimics human action. The loss over time seems similar between the two graphs, but overall, the second script was slightly faster, completing 440 more batches. The resulting weights performed similarly, however.

Our third script, which utilized multiple GPUs, had a faster training rate, but we could only use it for one hour of allocated time, so the resulting weights were around the same success rate. The first two scripts completed about 20000 batches in around seven hours, and the third script completed around 16400 batches on 8 GPUs split between 2 nodes in an hour. Although it is faster, we were limited by the time allocated to train on multiple processors when using interactive queue to get the quickest results to test hyperparameters. Regarding testing, we were getting very low success for random seed "FindCave" weights about 1 in 30, and decent success for "ChopTree" 4 in 10. Since this is only behavioral cloning right now, the rates make sense as RL would further refine them, and the easier task had a better time reaching its goal.

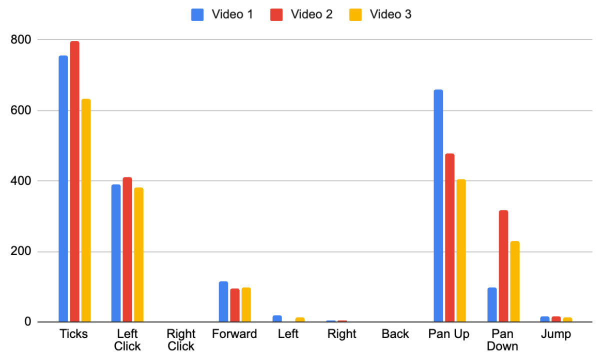
4 https://github.com/openai/Video-Pre-Training?tab=readme-ov-file#using-behavioural-cloning-to-fine-tune-the-models







We generated this reward graph when testing the RL weights from the repository. In the beginning, the smaller increments are for mining wood, crafting sticks, and making a pickaxe. As the agent gets closer to finding diamonds, the more significant tasks of finding iron are highly rewarded.



Above is a histogram of the JSON data from three of our recorded videos. The “Ticks” for each video represents the length of the video in frames. For all three videos, the “Left Click” and “Pan Up” actions are very apparent, which makes sense since the player would cut down trees by clicking the left mouse button and looking up to chop the logs higher on the tree. However, due to this disparity, the agent doesn’t know when other movements are necessary, resulting in a biased agent. This bias in the data was frustrating to see appear during evaluation, but it demonstrates the importance of variety in training data, especially for small datasets.

Reflection

Even though we did not have to start from scratch, we faced several obstacles along the way. Before beginning to train, there was a tedious process of setting up the Python and Java environments with the dependencies required to use MineRL. Additionally, we had many difficulties setting up the environment to properly show the MineRL Render and many issues with the correct X Display Server and Nvidia Graphics Drivers. Training and testing the agent on a rendered environment took hours, and we could not even ensure the agent's actions made sense without visualizing them since the loss wasn't a great indicator of success. However, through all of this, we learned the value of the OpenAI gym environment, which allowed us to connect the output of our model to Minecraft action states. This also extends outside of Minecraft, as the power of pre-built connectivity between unique environments (for example, Atari 2600 Games, Lunar Lander Simulator, etc.) still saves quite a bit of setup time[5]. It allowed us to focus on what model to use instead of struggling with the interface.

Takeaways and Future Goals

Compared to traditional reinforcement learning problems, rewards in Minecraft are sparse, making it difficult for a random policy to learn anything. Through behavioral cloning, we can develop baseline weights to aid in training the agent. Before this, we need a considerable amount of labeled demonstration data, which can come in the form of datasets online, auto-labeled data through inverse dynamics, or human recording. After hours of trial and error, we successfully created proper weights in tasks like "ChopTree" and "FindCave" with behavioral cloning. In the future, we hope to make a more efficient agent by properly fine-tuning pre-trained weights with reinforcement learning.

Contributions

We all had an equal part when working on the project. Regarding training, we spent around the same hours fine-tuning the weights to have a more efficient time testing and figuring out what hyperparameters worked the best. In terms of creating the script for recording and testing of RL, we collaborated and made adjustments together, pushing adjustments to our repository.

[5] https://gymnasium.farama.org/