University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Master of Science, Data Science	Spring semester, 2023 Open access
Writer: Dag Hermann Valvik	……Dag Hermann Valvik……. (Writer's signature)
Faculty supervisor: Vinay Jayarama Setty External supervisor(s):	
Thesis title: Human-Guided Phasic Policy Gradient in Minecraft: Exploring Deep Reinforcement Learning with Human Preferences in Complex Environments	
Credits (ECTS): 30	
Key words: Machine Learning, Imitation Learning, VPT Minecraft, PPG, PPO, HPPG, AI Human Preferences, Reinforcement Learning Behavior Cloning, ML, Artificial Intelligence	Pages: 47 + enclosure: 60 Stavanger, 14.06.2023 Date/year

Front page for master thesis
Faculty of Science and Technology
Decision made by the Dean October 30th 2009

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Human-Guided Phasic Policy Gradient in Minecraft: Exploring Deep Reinforcement Learning with Human Preferences in Complex Environments

Master's Thesis in Computer Science
by

Dag Hermann Valvik

Internal Supervisors

Vinay Jayarama Setty

June 14, 2023

"The question of whether machines can think is about as relevant as the question of whether submarines can swim."

Edsger Dijkstra

Abstract

This study presents a novel approach to enhancing the performance of artificial agents in complex environments like Minecraft, where traditional reward-based learning strategies can be challenging to apply. To improve the efficacy and efficiency of fine-tuning a foundation model for complex tasks, we propose the Human-Guided Phasic Policy Gradient (HPPG) algorithm, which combines human preference learning with the Phasic Policy Gradient technique. Our key contributions include validating the use of behavioral cloning to improve agent performance and introducing the HPPG algorithm, which employs a reward predictor network to estimate rewards based on human preferences. We further explore the challenges associated with the HPPG algorithm and propose strategies to mitigate its limitations. Through our experiments, we demonstrate significant improvements in the agent's performance when executing complex tasks in Minecraft, laying the groundwork for future developments in reinforcement learning algorithms for complex, real-world tasks without defined rewards. Our findings contribute to the broader goal of bridging the gap between artificial agents and human-like intelligence.

Acknowledgements

My deepest thanks go to my supervisor, Vinay Jayarama Setty, for his unwavering support and enthusiasm. His guidance has been essential in shaping my research and writing; this thesis is a testament to his dedication. In addition, I wish to express my deepest gratitude to my loving fiancé, whose unwavering support and understanding have been my rock throughout this journey. Her patience, encouragement, and love have been a constant source of strength. This achievement would not have been possible without them.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background and Motivation

Artificial intelligence (AI) has made remarkable advancements in the field of deep reinforcement learning (DRL), achieving superhuman performance in games like Go, chess, and Dota 2 [1–3]. AI systems such as AlphaGo and AlphaZero have demonstrated unprecedented achievements, surpassing human experts in strategic gameplay. These milestones have highlighted DRL algorithms' potential and ability to master complex tasks.

However, these achievements have come at a cost. The success of these AI systems heavily relies on extensive data sampling and significant computational resources, limiting their accessibility and practical applicability. Furthermore, these AI systems have primarily excelled in games with well-defined reward functions, making them less suitable for scenarios where explicit rewards are not readily available or easily defined.

This limitation becomes particularly apparent in environments like Minecraft, where the game's objective is open-ended, and players define their own goals. Minecraft's survival mode, for instance, lacks a specific reward function except for the survival duration. Players have the freedom to explore and interact with the game world, making each session unique. Developing AI systems that can effectively learn and perform tasks in Minecraft using a human interface presents a significant challenge, requiring approaches that are sample-efficient and adaptable to environments without explicit rewards.

Recent advancements in the field have shown promise in addressing these challenges. Baker et al. [4] demonstrated that Video Pretraining (VPT) and Inverse Dynamics Model (IDM) have the ability to leverage large-scale unlabeled video data to train models for complex tasks in Minecraft. By pretraining on videos and utilizing the IDM to label

unlabeled videos, they expanded the training dataset and achieved impressive results, enabling agents to perform on par with human players in some tasks with sparse rewards. Their work serves as a testament to the effectiveness of this approach in overcoming the limitations of explicit reward functions.

However, a goal remains to train AI systems that can perform tasks in Minecraft without relying solely on explicit reward structures and achieve performance comparable to that of human players. One promising approach is reinforcement learning from human preferences, where humans provide evaluations and select optimal task executions. This approach leverages human judgment to guide learning and improve task performance.

In this paper, we aim to investigate the utilization of human feedback and sample-efficient algorithms, specifically the Phasic Policy Gradient (PPG), to train agents for lifelike tasks in Minecraft. Our approach involves fine-tuning the VPT foundation model using imitation learning and reinforcement learning with human preferences. By combining the strengths of human-guided learning and sample-efficient algorithms, we aim to enhance an agent's performance in Minecraft and enable it to execute complex tasks with human-like proficiency.

1.2 Objectives

This study is guided by the following research questions (RQs) to achieve our objectives:

- **RQ1:** How can we enhance sample efficiency in fine-tuning a foundation model for executing complex tasks in Minecraft using imitation learning?

  Objective: Given the challenges of expensive sampling in real-world domains and complex games like Minecraft, our objective is to develop an imitation learning algorithm that can effectively leverage limited data and computational resources to improve the agent's performance.

- **RQ2:** How can human feedback be incorporated to fine-tune a model in the absence of explicit reward functions, thereby enhancing the agent's task performance in environments like Minecraft?

  Objective: Our aim is to incorporate human guidance into the learning process, using human preferences to refine the agent's behavior for better task completion.

- **RQ3:** What are the potential limitations of the proposed method that combines imitation learning with human-guided learning and how can we evaluate its effectiveness and efficiency in complex environments like Minecraft?

  Objective: Our aim is to understand and mitigate the limitations of our proposed method and establish a robust evaluation framework to quantify its impact on the agent's performance in complex tasks.

By addressing these research questions, we aim to advance the field of reinforcement learning in complex and challenging environments. Our objectives include developing effective algorithms for sample-efficient fine-tuning in Minecraft, utilizing human feedback to enhance agent performance, and ultimately enabling the agent to exhibit lifelike behavior in complex tasks. These objectives contribute to the broader goal of bridging the gap between artificial agents and human-like intelligence in challenging real-world domains.

1.3 Approach and Contributions

In this thesis, we propose a novel algorithm called Human-Guided Phasic Policy Gradient (HPPG) which integrates human preference learning with Phasic Policy Gradient (PPG), a technique known for its sample efficiency and stability. This hybrid algorithm aims to enhance agent performance in environments where rewards are not easily defined or are difficult to obtain, such as in Minecraft.

We made the following key contributions:

- **Contribution to RQ1:** We confirm the efficacy of behavioral cloning (BC) in enhancing agent performance, even in complex tasks within Minecraft. By leveraging human demonstrations, the agent can mimic desired behavior, improving task execution, and making human feedback more efficient.
- **Contribution to RQ2:** We introduce HPPG, a novel algorithm that employs a reward predictor network to estimate rewards based on human preferences. The reward predictor network plays a vital role in guiding the agent's learning process by providing reward signals in the absence of explicit environmental rewards.
- **Contribution to RQ3:** We explore the challenges associated with implementing the HPPG algorithm and propose strategies to mitigate its limitations. Our findings emphasize the need for continual refinement of the reward predictor network to enhance the agent's learning capabilities.

- **Summary of Results:** We demonstrate, through experimentation, that our method of fine-tuning a foundation model with behavioral cloning effectively increases task performance. We present a new HPPG algorithm that integrates human preference learning with PPG. It highlights the crucial role of the reward predictor network and the need for more research. Our results lay a foundation for human-guided reinforcement learning algorithms in challenging environments without rewards.

1.4 Outline

This thesis is organized as follows:

- Chapter 2 provides an overview of the MineRL project and its integration with Minecraft and OpenAI Gym. It explores previous research in the areas of learning from human preferences. In addition, it covers sample-efficient algorithms like Proximal Policy Optimization and Phasic Policy Gradient, and the foundation model used in this thesis.
- Chapter 3 outlines the approach used in this thesis, including the process of fine-tuning with behavior cloning, the selection of optimal hyperparameters, and the proposed Human-Guided Phasic Policy Gradient method, explaining its various components in detail.
- Chapter 4 presents the experimental setup and the corresponding results. It offers a comprehensive analysis of these results, highlighting the successes, failures, and limitations of the approach.
- Chapter 5 concludes the thesis by summarizing the main findings and addressing the research questions posed throughout the study. Finally, it proposes some future directions of research.

Chapter 2

Related Work

This chapter introduces MineRL, a Python3 library that serves as a valuable resource for bridging Minecraft and artificial intelligence research. The chapter also provides an overview of existing research and studies in the areas of sample-efficient algorithms, reinforcement learning in Minecraft, and learning from human preferences. It offers insights into the potential of video games as experimental platforms for AI development, with a particular focus on Minecraft's open-ended sandbox world.

2.1 MineRL

The potential of video games as experimental platforms for reinforcement learning and AI development cannot be overstated. One such gaming environment, Minecraft, is especially noteworthy, as its open-ended, sandbox-style world lends itself perfectly to sophisticated testing and learning scenarios [5]. Recognizing this, the team behind the project MineRL developed an invaluable resource that bridges the gap between Minecraft and artificial intelligence research [6].

While MineRL provides a specific interface and dataset for reinforcement learning research using Minecraft, it's not the only project to recognize the potential of this gaming environment. Microsoft's Project Malmo also uses Minecraft as a platform for AI research. Project Malmo, in particular, was designed to support a wide range of fundamental AI research, enabling previously infeasible or inconvenient experiments within a game environment [7]. This diversity in research platforms illustrates the versatility and utility of Minecraft in the field of AI development.

MineRL [6], a robust Python3 library, serves as the link between AI research and Minecraft by providing an interface that integrates Minecraft with the OpenAI Gym [8], a toolkit

for developing and comparing reinforcement learning algorithms. This integration allows developers and researchers to interact with Minecraft as an experimental environment, unlocking many opportunities for machine learning experiments and reinforcement learning.

Notably, MineRL goes beyond providing just a mere interface. The project encompasses a suite of environments within Minecraft that researchers can use for varied learning and testing scenarios. In reinforcement learning, an "environment" refers to the domain or the 'world', which could be real or simulated, where an agent interacts, makes decisions, and learns from the consequences of its actions. In this context, Minecraft serves as the simulated environment, offering a suite of diverse yet uniform domains for varied learning and testing scenarios. Though diverse, these environments maintain uniformity in observation and action space, facilitating easy inter-experiment comparison and data consistency.

The observation space offered by MineRL consists of RGB images with a resolution of 360x640x3. This image data serves as the primary input for the reinforcement learning model, with the RGB components representing the pixel-wise color information captured from the Minecraft environment.

Meanwhile, the action space (see Table 2.1) encapsulates a broad array of Minecraft-specific actions. These actions can be understood as the output of the series of instructions that the model can issue within Minecraft. It is essential to clarify that 'Discrete' and 'Box' are terms used to define specific types of action spaces in the context of OpenAI Gym [8], offering options for discrete and continuous actions.

However, MineRL's most significant offering is arguably its extensive imitation learning dataset. It is one of the world's largest, boasting over 60 million frames of recorded human gameplay. This extensive corpus of data allows reinforcement learning models to learn from and build upon human players' strategies and techniques, fast-tracking their learning process and potentially developing more sophisticated AI behaviors [9].

Key	Action Space
ESC	Discrete(2)
attack	Discrete(2)
back	Discrete(2)
camera	Box(low=-180.0, high=180.0, shape=(2,))
drop	Discrete(2)
forward	Discrete(2)
hotbar.1	Discrete(2)
hotbar.2	Discrete(2)
hotbar.3	Discrete(2)
hotbar.4	Discrete(2)
hotbar.5	Discrete(2)
hotbar.6	Discrete(2)
hotbar.7	Discrete(2)
hotbar.8	Discrete(2)
hotbar.9	Discrete(2)
inventory	Discrete(2)
jump	Discrete(2)
left	Discrete(2)
pickItem	Discrete(2)
right	Discrete(2)
sneak	Discrete(2)
sprint	Discrete(2)
swapHands	Discrete(2)
use	Discrete(2)

Table 2.1: The MineRL action space. The left column shows the keys in the action space dictionary, while the right shows the corresponding Gym [8] action spaces

2.2 BASTALT 2022 Dataset

The BASALT (Benchmark for Agents that Solve Almost-Lifelike Task) 2022 Dataset, the primary data source used in this study, represents a significant contribution to the field of reinforcement learning. OpenAI provided this dataset of demonstrations for the MineRL BASALT 2022 competition, encompassing over 600 GB of labeled data. In this thesis, labeled data refers to videos with ground truth labels of what actions were taken in each frame unless stated otherwise. The dataset involves four different lifelike tasks,

providing diverse scenarios that AI models can learn from [10]. See Table 2.2 for basic dataset statistics like the size of each dataset, number of videos, and environment name.

Size	#Videos	Name
146G	1399	MineRLBasaltBuildVillageHouse-v0
165G	2833	MineRLBasaltCreateVillageAnimalPen-v0
165G	5466	MineRLBasaltFindCave-v0
175G	4230	MineRLBasaltMakeWaterfall-v0

Table 2.2:]
Statistics for the BASTALT (Benchmark for Agents that Solve Almost-Lifelike Task) 2022 Dataset

Each task has a corresponding MineRL environment with different starting inventories and goals. The common denominator is that all four environments provide no rewards and end when the Player or Agent presses "ESC" or after a certain amount of steps. Where a step corresponds to moving forward one frame, which is done at 20hz in MineRL. The goal is to produce agents that real humans judge to solve the different tasks effectively. This information is taken from the MineRL Basalt Environment documentation [11].

1. **MineRLBasaltFindCave-v0**
    - Objective: After spawning in a plains biome, explore and find a cave. When inside a cave, end the episode by setting the "ESC" action to 1. Do not dig down from the surface to find a cave.
    - Max Episode Steps: 3600
    - Starting Inventory: Empty
2. **MineRLBasaltCreateVillageAnimalPen-v0**
    - Objective: After spawning in a village, build an animal pen next to one of the houses in a village. Use your fence posts to build one animal pen that contains at least two of the same animal. (You are only allowed to pen chickens, cows, pigs, or sheep.) There should be at least one gate that allows players to enter and exit easily. The animal pen should not contain more than one type of animal. (You may kill any extra types of animals that accidentally got into the pen.) Do not harm villagers or existing village structures in the process. Send 1 for "ESC" key to end the episode.
    - Max Episode Steps: 6000
    - Starting Inventory:

Item	**Quantity**
carrot	1
oak_fence	64
oak_fence_gate	64
wheat	1
wheat_seeds	1

3. **MineRLBasaltMakeWaterfall-v0**

   - Objective: After spawning in an extreme hills biome, use your waterbucket to make a beautiful waterfall. Then take an aesthetic "picture" of it by moving to a good location, positioning player's camera to have a nice view of the waterfall, and ending the episode by setting "ESC" action to 1.
   - Max Episode Steps: 6000
   - Starting Inventory:

Item	**Quantity**
cobblestone	20
stone_pickaxe	1
stone_shovel	1
water_bucket	1

4. **MineRLBasaltBuildVillageHouse-v0**

   - Objective: Build a house in the style of the village without damaging the village. It should be in an appropriate location (e.g. next to the path through the village). Then, give a brief tour of the house (i.e. spin around slowly such that all of the walls and the roof are visible). Finally, end the episode by setting the "ESC" action to 1.
   - Tip: Different materials are used in each biome-specific village (plains, savannah, taiga, desert).
   - Max Episode Steps: 14400
   - Starting Inventory:

Item	Quantity
acacia_log	64
black_dye	64
blue_dye	64
brown_dye	64
cactus	64
cobblestone	64
cobweb	64
dirt	64
flower_pot	64
glass_pane	64
grass_block	64
green_dye	64
jungle_log	64
lily_pad	64
magenta_dye	64
oak_log	64
orange_dye	64
pink_dye	64
red_dye	64
sand	64
spruce_log	64
stone	64
stone_bricks	64
torch	64
vines	64
white_dye	64
yellow_dye	64

2.3 Video PreTraining (VPT)

Baker et al. [4] introduced the VPT foundation models used throughout this thesis, including the Inverse Dynamics Model (IDM) and the Video Pre Trained (VPT) Foundation Model [10]. These models come in various sizes: 71M, 248M, and 500M trainable parameters. The IDM model can incorporate both past and future events, while the VPT Foundation Model is strictly causal, predicting future events based solely on past data.

The foundation model was trained through a two-step process. Firstly, a small amount of labeled data was collected from contractors to train the IDM. This model was tasked

to minimize the negative log-likelihood of an action at a given timestep, considering a trajectory of T observations. Subsequently, this model was utilized to label online videos, thereby amplifying the training data for the VPT foundation model from an initial 2k video hours to a substantial 70k video hours after filtering for clean video segments.

The architecture of these models primarily mirrors each other, with the IDM integrating an initial non-causal convolution and four non-causal residual transformer blocks to predict actions at each frame. In contrast, the VPT Foundation Model omits the non-causal convolution and enforces causal masking on the residual transformer layers, introducing Transformer-XL-style training for causality [4].

The architecture of these models primarily mirrors each other, with the IDM integrating an initial non-causal convolution and four non-causal residual transformer blocks to predict actions at each frame. In contrast, the VPT Foundation Model omits the non-causal convolution and enforces causal masking on the residual transformer layers. This approach introduces Transformer-XL-style training for causality [4]. Transformer-XL, a variant of the traditional transformer model, handles long-term dependencies in sequences effectively by using a recurrence mechanism and a novel positional encoding scheme. This allows for an extended context by reusing hidden states from previous segments, enabling learning dependencies over a longer context than a fixed-length window [12].

Subsequent research demonstrated fine-tuning of the VPT foundation model for the early game using behavioral cloning (BC) on datasets specific to early game development [4]. This approach significantly boosted crafting capabilities compared to the zero-shot VPT foundation model. Following this approach, this thesis also fine-tunes the VPT model using BC on task-specific labeled data before any reinforcement learning, given its demonstrated efficiency.

To illustrate the potency of RL fine-tuning, Baker et al. [4] opted for the goal of crafting a diamond pickaxe within ten minutes in a fresh Minecraft survival world. Using sparse environmental rewards and a Phasic Policy Gradient [13] algorithm (PPG), the agents accomplished this task, which typically requires over twenty minutes for a human player. However, this thesis diverges in focus, concentrating on RL fine-tuning without explicit environmental rewards, instead leveraging human feedback to fine-tune the VPT foundation model.

Despite their differences in size, all models displayed impressive zero-shot performance, though larger models outperformed smaller ones during fine-tuning. Due to computational constraints, this thesis opts to employ the smallest VPT model (71M parameters).

2.4 Proximal Policy Optimization

Proximal Policy Optimization (PPO) was introduced by Schulman et al. [14] as a more efficient and stable alternative to traditional policy gradient methods such as Trust Region Policy Optimization (TRPO) [15]. Both PPO and TRPO build upon seminal works in deep reinforcement learning such as Q-Learning, Double Q-Learning [16], and Deep Q-Learning applied to the Atari games [5]. PPO addresses the challenges of sample complexity and sensitivity to hyperparameters, which were prevalent in earlier policy optimization algorithms.

In the context of continuous control tasks, methods like Deep Deterministic Policy Gradient (DDPG) [17] helped pave the way for the development of actor-critic methods, which subsequently influenced the development of PPO. At the heart of these actor-critic methods lie two components: the actor, which decides the actions based on the current policy, and the critic, which evaluates the quality of these actions. The actor improves its policy based on the feedback from the critic, and the critic refines its value estimates based on the reward signals from the environment.

The primary goal of PPO is to optimize the policy while maintaining a balance between exploration and exploitation, which is achieved by constraining the policy updates to be within a trust region. Introduced as a refined alternative to prior policy gradient methods, PPO emerged as a solution for providing efficiency and stability in reinforcement learning (RL) tasks [14]. Unlike the TRPO [15] algorithm, PPO successfully addresses the intricacies of sample complexity and sensitivity to hyperparameters, constraints frequently observed in traditional policy optimization algorithms.

Building on the principles of natural policy gradient [18] and TRPO [15], PPO stands as an on-policy algorithm. The unique value proposition of PPO lies in introducing a surrogate objective function. This function restricts the magnitude of policy updates, penalizing unduly significant changes. This surrogate objective function is defined as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\big[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\big]. \tag{2.1}$$

Here, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio of the new to the old policy, $\hat{A}_t$ is the estimated advantage function, and ϵ represents the hyperparameter that determines permissible policy changes.

The utility of this surrogate function is to facilitate a balance in PPO, allowing large policy updates while circumventing excessively substantial alterations, which may cause instability in the learning process.

Algorithm 2.1 PPO, Actor-Critic Style

for iteration = 1, 2, . . . **do**
  for actor = 1, 2, . . ., N **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

2.5 Phasic Policy Gradient (PPG)

Reinforcement learning (RL) methods, particularly actor-critic algorithms, have shown promising results in various domains [4, 5, 13, 14, 19–22]. A key challenge in these methods is to balance the competing objectives of the policy and value functions. Cobbe et al. [13] introduced Phasic Policy Gradient (PPG), a new method that addresses this challenge by separating policy and value function training into distinct phases.

In the actor-critic framework, sharing parameters between the policy and value function networks presents distinct advantages. Notably, the optimization of each objective can benefit from the features learned by the other. However, this shared approach also introduces challenges, such as balancing the competing objectives of the policy and value function. PPG addresses this issue by decoupling the training of the policy and value function objectives. This reduces interference between the two objectives and can improve sample efficiency, as the optimization of the value function typically tolerates higher levels of sample reuse than policy optimization.

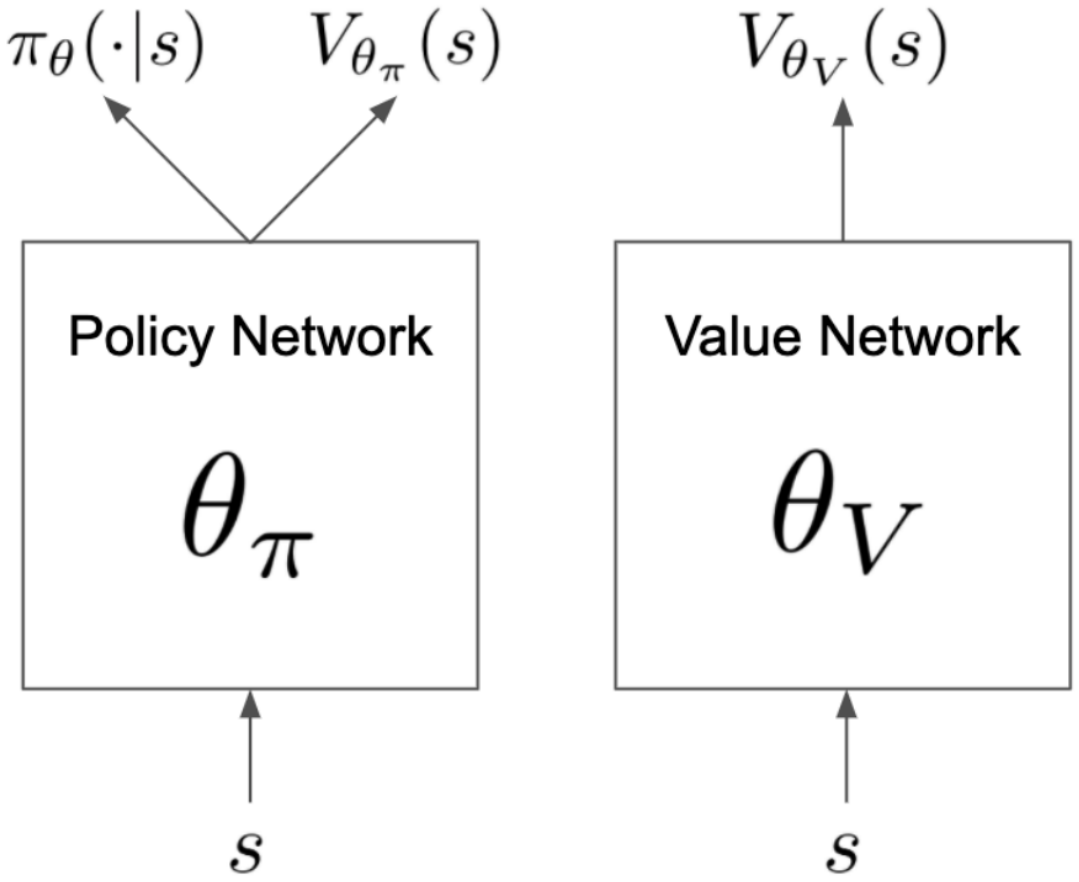

Figure 2.1: Phasic Policy Gradient (PPG) uses disjoint policy and value networks to reduce interference between objectives. The policy network includes an auxiliary value head. Figure taken from the Cobbe et al. [13]

The *policy phase* in PPG is similar to Proximal Policy Optimization (PPO) [14], a widely adopted RL algorithm known for its stability and performance. During the policy phase the policy network is optimized using the clipped surragate objective and the value network is trained by optimizing

$$L_{\text{value}} = \hat{\mathbb{E}}t\left[\frac{1}{2}\left(V\theta(s_t) - \hat{V}^t_{\text{targ}}\right)^2\right] \tag{2.2}$$

where $\hat{V}$targ represents the value function targets. Both $\hat{A}$ and $\hat{V}$targ are computed using Generalized Advantage Estimation (GAE) [23].

However, the key difference in PPG is the introduction of periodic auxiliary phases, where features from the value function are distilled into the policy network, improving future policy phases.

In the *auxiliary phase*, the policy network is optimized with a joint objective that incorporates an arbitrary auxiliary loss and a behavioral cloning loss:

$$L_{\text{joint}} = L_{\text{aux}} + \beta_{\text{clone}} \cdot \hat{\mathbb{E}}t \left[\text{KL}[\pi\theta_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right] \tag{2.3}$$

where $\pi_{\theta_{\text{old}}}$ represents the policy right before the auxiliary phase begins. The term inside the expectation, $\text{KL}[\pi\theta_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]$, represents the Kullback-Leibler (KL) divergence between the old policy ($\pi_{\theta_{\text{old}}}$) and the new policy ($\pi_\theta$), given the state s_t. The KL divergence is used to quantify how much the policy has changed during the update. A smaller KL divergence indicates that the updated policy is not drastically different from the old policy, which is desirable to maintain stability during learning.

The hyperparameter β_{clone} controls the trade-off between the auxiliary objective and preserving the original policy. The auxiliary objective, denoted as L_{aux}, can be any desired auxiliary objective. It can be seen as the phase where features from the value function are distilled into the policy network. Specifically, we define L_{aux} as:

$$L_{\text{aux}} = \frac{1}{2} \cdot \hat{\mathbb{E}}t \left[\left(V\theta^\pi(s_t) - \hat{V}^t_{\text{targ}} \right)^2 \right] \tag{2.4}$$

where V^π_θ represents an auxiliary value head of the policy network, as depicted in Figure 2.1.

Algorithm 2.2 Phasic Policy Gradient (PPG)

for phase = 1, 2, ... **do**
  Initialize empty buffer B
  for iteration = 1, 2, ..., N_π **do**
    Perform rollouts under current policy π
    Compute value function target $\hat{V}^t_{\text{targ}}$ for each state s_t
    for epoch = 1, 2, ..., E_π **do**
      Optimize $L_{\text{clip}} + SS[\pi]$ w.r.t. θ_π
    end for
    for epoch = 1, 2, ..., E_V **do**
      Optimize L_{value} w.r.t. θ_V
    end for
    Add all $(s_t, \hat{V}^t_{\text{targ}})$ to B
  end for
  Compute and store current policy $\pi_{\theta_{\text{old}}}(\cdot|s_t)$ for all states s_t in B
  for epoch = 1, 2, ..., E_{aux} **do**
    Optimize L_{joint} w.r.t. θ_π, on all data in B
    Optimize L_{value} w.r.t. θ_V, on all data in B
  end for
end for

PPG has demonstrated improved sample efficiency compared to a PPO baseline across various tasks and environments [14]. The algorithm mitigates the interference between the policy and value function objectives while still sharing representations, and optimizing each with the appropriate level of sample reuse.

2.6 Reinforcement Learning from Human Preferences

Learning from human preferences can be inefficient in reinforcement learning (RL) systems, as they require extensive experience, and human interactions are expensive. To address this issue, researchers have explored ways to decrease the feedback required, resulting in several orders of magnitude decrease. One such approach is proposed by Christiano et al. [19], who fit a reward function from human feedback and then optimize that function in deep RL. Their algorithms focus on fitting a reward function to human preferences while training a policy to optimize the predicted reward function.

Christiano et al. [19] demonstrate the effectiveness of their approach in the Arcade Learning Environment [24] using Atari games and the physics simulator MuJoCo [22] for

robotics tasks. They show that a small amount of feedback is sufficient to learn RL tasks without a reward from the environment, using only the predicted reward. They do this by using a human's preference between trajectory segments. A trajectory segment is a sequence of observations and actions, $\sigma = ((o_0, a_0), (o_1, a_1), \ldots, (o_{k-1}, a_{k-1})) \in (O \times A)^k$. For example, $\sigma_1 \succ \sigma_2$ indicates the human preferred trajectory segment σ_1 to trajectory segment σ_2.

To interpret a reward function estimate $\hat{r}$ as a preference-predictor, Christiano et al. [19] assume that the human's probability of preferring a segment σ_i depends exponentially on the value of the latent reward summed over the length of the clip:

$$\hat{P}(\sigma_1 \succ \sigma_2) = \frac{\exp \sum_t \hat{r}(o_1^t, a_1^t)}{\exp \sum_t \hat{r}(o_1^t, a_1^t) + \exp \sum_t \hat{r}(o_2^t, a_2^t)}. \tag{2.5}$$

They choose $\hat{r}$ to minimize the cross-entropy loss between these predictions and the actual human labels:

$$\text{loss}(\hat{r}) = - \sum_{(\sigma_1, \sigma_2, \mu) \in D} \mu(1) \log \hat{P}(\sigma_1 \succ \sigma_2) + \mu(2) \log \hat{P}(\sigma_2 \succ \sigma_1). \tag{2.6}$$

This approach follows the Bradley-Terry model for estimating score functions from pairwise preferences [25]. It can be understood as equating rewards with a preference ranking scale analogous to the famous Elo [26] ranking system developed for chess. Their algorithm incorporates several modifications to this basic approach, which early experiments discovered helpful, such as fitting an ensemble of predictors, using a fraction of the data as a validation set for each predictor, and assuming a 10% chance that the human responds uniformly at random.

The overall process comprises three primary components. In the first component, the policy π interacts with the environment to generate a set of trajectories, during which it receives predicted rewards from the predictor network. These predicted rewards are then utilized to update the policy using Advantage Actor-Critic (A2C) [20]. The second component involves querying a human for preferences on pairs of segments derived from the trajectories produced by the first component. In the third component, these preferences are employed to train the predictor network for improved future reward predictions. Though these components are distinct, it's crucial to note that they operate asynchronously. This setup allows the policy π to continually update itself as preferences are gathered and used to train the predictor network.

However, while human feedback recently has been used to optimize deep learning models, it has only been applied to simple environments [19, 27, 28]. By contrast, the challenging

Minecraft environment [6] makes learning particularly difficult due to using a human interface, much larger observation and action spaces, and increased complexity because of the procedurally generated Minecraft world.

The incorporation of human preferences to fine-tune a language model for summarization and style continuation tasks was examined by Ziegler et al. [21], which represents a step towards the application of these techniques in more real-world contexts compared to Christiano et al. [19]. We continue to argue that Minecraft is an even more difficult domain, and like Ziegler et al. [21], we leverage a pre-trained model and fine-tune it using human judgment and sample effective algorithms.

Chapter 3

Approach

This chapter provides an overview of the Human-Guided Phasic Policy Gradient algorithm and its application in training an agent capable of performing complex tasks in Minecraft's interactive game environment. The chapter outlines the methodology used to combine human preference learning and the Phasic Policy Gradient algorithm to achieve efficient training. The primary objective is to train an agent proficient in the human-like task of building a village house in Minecraft, utilizing a combination of behavioral cloning and reinforcement learning techniques.

3.1 Overview

This work introduces the Human-Guided Phasic Policy Gradient (HPPG). This novel algorithm synergizes human preference learning and Phasic Policy Gradient, aiming to efficiently train an agent capable of performing complex tasks in rich environments. We test our algorithm in Minecraft's challenging and interactive game. We use the MineRL project gym environment to fine-tune a Video Pre-Trained (VPT) foundation model through behavioral cloning (BC) using labeled expert demonstrations. We then apply our reinforcement learning (RL) algorithm for further refinement. Our primary objective is to train an agent proficient in the human-like task of building a village house in Minecraft using a combination of BC and RL.

Due to computational constraints, we focus on this single task out of the four available tasks in the accompanying datasets, intending to complete the training using a mid-grade consumer GPU. We employ imitation learning in the form of BC to initially fine-tune the model for the task, using the *BuildVillageHouse* dataset and random search to identify the most promising hyperparameters. Upon completing the behavioral cloning stage, we

collect human preferences using the current model. Furthermore, the collected preferences are used to pre-train the Reward Predictor for more accurate initial rewards when RL fine-tuning. Finally, the model is further fine-tuned following the HPPG algorithm leveraging the predicted rewards from the pre-trained reward predictor.

In contrast to traditional reinforcement learning, where the environment supplies a reward signal, we assume the presence of a human overseer capable of expressing preferences between *trajectory segments*. A trajectory segment is a sequence of observations and predicted rewards, $\sigma = ((o_0, \hat{r}_0), (o_1, \hat{r}_1), ..., (o_{k-1}, \hat{r}_{k-1})) \in (O \mathrm{x} \hat{R})^k$. We write $\sigma_1 \succ \sigma_2$ to indicate that the human preferred trajectory segment σ_1 to trajectory segment σ_2. In Figure 3.1, an overview of the architecture is illustrated. The HPPG algorithm operates as follows:

- The agents explore the environment under the current policy, gathering memories (trajectories).
- The reward predictor network predicts a surrogate reward $\hat{r}$, supplanting the environment's rewards.
- Following the rollout, the policy and disjoint value network are trained using Proximal Policy Optimization (PPO).
- The trajectories $\{\tau^1, \ldots, \tau^i\}$ are divided into fixed length segments $\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ and added to segment list.
- We select random pairs of segments (σ_1, σ_2) and query a human for preference.
- The preference is stored as a triple $(\sigma_1, \sigma_2, \mu)$ in the preference queue.
- We update reward predictor on preferences to give better future rewards.
- We optimize both the L_{joint} and L_{value} losses during auxiliary phase.

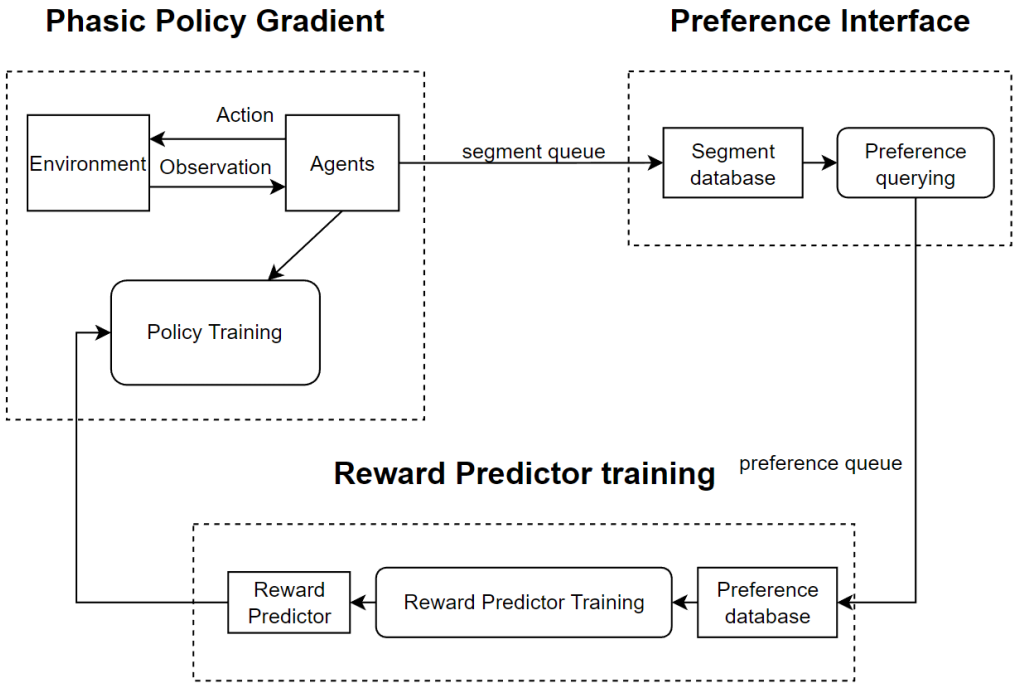

Figure 3.1: Architecture of the Human-Guided Phasic Policy Gradient algorithm, highlighting the interactions between the environment, agents, reward predictor network, and the human overseer.

3.2 Fine-tuning with Behavioral Cloning

Fine-tuning with standard behavioral cloning provides a better starting point or initialization for RL fine-tuning [4]. Furthermore, when fine-tuning for a specific task when using human feedback, it could improve feedback efficiency [28]. One could look at it as priming the network for the given task. Unfortunately, early experiments using the hyperparameters provided in Baker et al. [4] did not provide excellent results for our task, as seen in the smoothed training loss from the training run in Figure 3.2. The training loss indicated that the model never learned to perform our tasks. This was confirmed by evaluating the agent by observation, which seemed to have forgotten almost all primary navigation. The agent displayed erratic behavior by moving the camera seemingly at random, dropping items from the inventory, and placing blocks at every timestep.

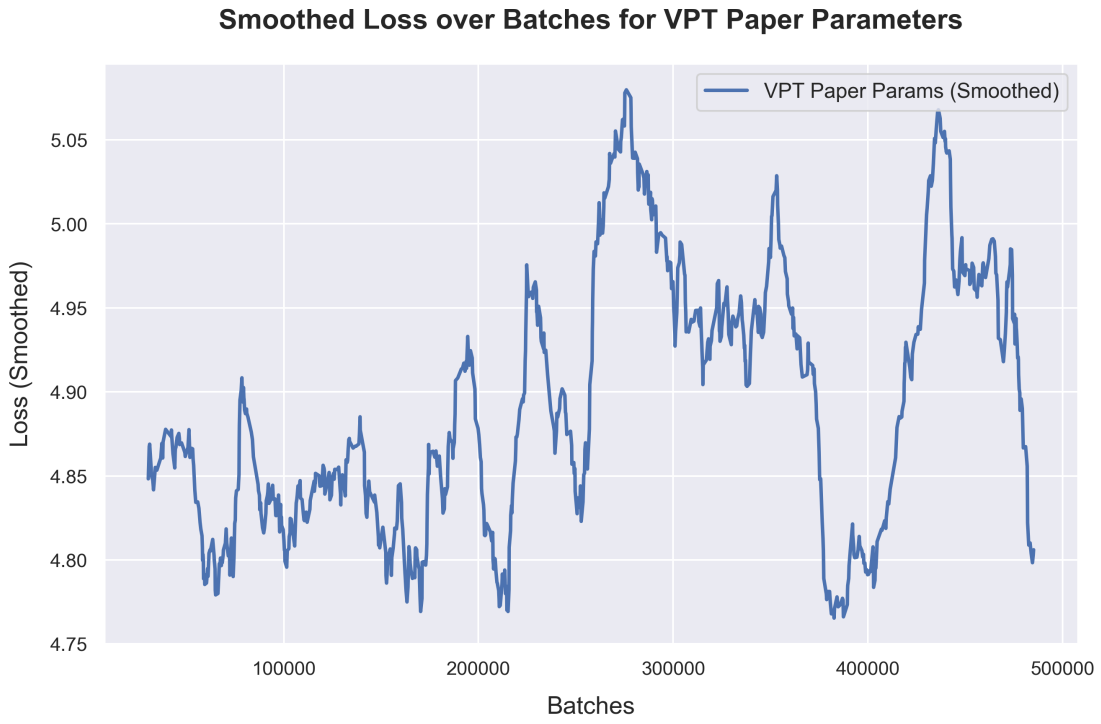

Figure 3.2: Training loss on the BuildVillageHouse dataset, using hyperparameters for behavioral cloning from the OpenAI VPT paper [4]. The hyperparameters used were epochs 2, batch size 16, learning rate 0.000181, and weight decay 0.039428 with Adam optimizer Kingma and Ba [29]

We then split the datasets into training and validation sets and randomly searched for the most promising hyperparameters. In addition to the standard behavioral cloning loss, we introduce a Kullback-Leibler [30] (KL) divergence and gradient clipping to control eventual exploding gradients and catastrophic forgetting. The foundation model was trained on enormous amounts of data for general Minecraft gameplay, which includes gathering logs, ore, and crafting necessary for the early game in a Minecraft survival world. In contrast, in our tasks, the player spawns with the required items to complete the task in the inventory. We aim to make minor updates to the policy and slowly converge towards a policy where the agent places items from the inventory sequentially but still knows how to navigate the environment. We fine-tune the foundation model by minimizing the negative log-likelihood of actions predicted by the model based on the observations using the datasets described in chapter 2 and their ground truth labels. For a particular trajectory of length T, we minimize the objective function, which combines the negative log-likelihood and the Kullback-Leibler divergence between the action distributions of the fine-tuned model and the original model, weighted by the hyperparameter λ. The objective function is given as:

$$\min_{\theta} \sum_{t=1}^{T} (-\log \pi_\theta(a_t|o_1, \ldots, o_t) + \lambda \cdot \mathrm{KL}(\pi_\theta(a_t|o_1, \ldots, o_t)||\pi_{\text{original}}(a_t|o_1, \ldots, o_t)))$$

.

By minimizing this objective function, we aim to improve the model's performance on the task while maintaining its generalization capability by constraining the divergence from the original model. The model trained with the hyperparameters is provided in Table 3.1

Hyperparameter	**Value**
Learning Rate	1×10^{-6}
Weight Decay	0.01
Batch Size	32
KL Loss Weight(λ)	0.5
Max Grad Norm	5.0
Epochs	1

Table 3.1: Hyperparameters used for fine-tuning with behavioral cloning, found using multiple runs of random search on the BuildVillageHouse dataset.

The random search was run twice with different hyperparameter spaces. First, a small run of 2000 batches for each iteration was completed with a broad range of learning rates. From this initial run, we could identify that learning rates of 1e-3 and 1e-4 were too large, confirming the suspicion that the hyperparameters during the training run (Figure 3.2) were not optimal for the BuildVillageHouse task. After removing the mentioned learning rates from the hyperparameter space, a second random search with a length of 5000 batches and 15 iterations found that the hyperparameters in Table 3.1 had the best validation loss.

In addition to the hyperparameters, we used the *ReduceLROnPlateau*[1] learning rate scheduler with a factor of 0.1 and patience of 50, along with the Adam optimizer [29]. The scheduler is stepped every 100 batches. This implies that if the training loss has plateaued over 5000 batches ($50 \cdot 100$), the learning rate will be reduced by a factor of 0.1. The ReduceLROnPlateau was also initialized with a minimum learning rate of 1e-7, meaning the learning rate can never go below this level. As the training is only run for one epoch (iteration over the training dataset once), validation is skipped.

The training losses for our runs with the optimal hyperparameters are shown below in Figure 3.3. We performed three training runs, one run with an LR scheduler, then one without a scheduler. Finally, we performed one training run with an LR scheduler where

[1] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

all parameters were frozen except for the π head and the last layer. One of the runs was stopped early by mistake. Interestingly, it was this model that showed the most promise.

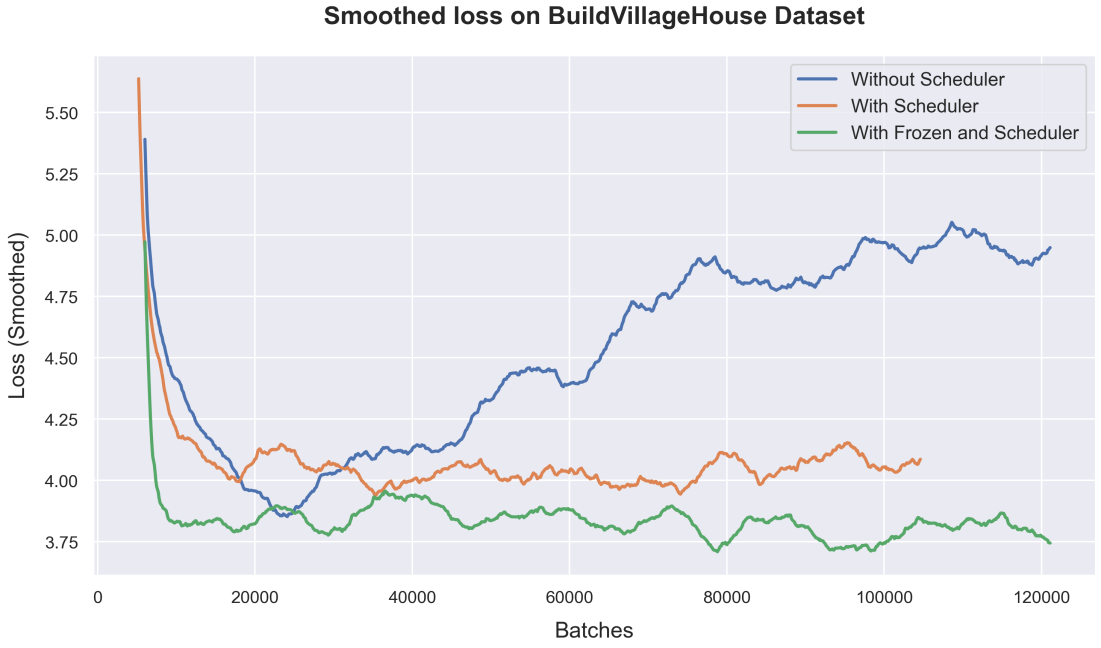

Figure 3.3: Behaviour cloning fine-tuning loss for the 71M parameter foundation model on the BuildVillageHouse dataset, using hyperparameters in Table 3.1 found using random search. All three runs use the same hyperparameters and Dataset, which is the train split of the BuildVillageHouse dataset Table 2.2, which consists of 80% of the data, randomly selected. The runs with scheduler used the ReduceLROnPlateau scheduler with factor 0.1 and patience 50, stepped every 100 batches.

3.3 Fine-tuning with Reinforcement Learning

This section explains how we extend the Phasic Policy Gradient algorithm with human preferences, which we call Human-guided Phasic Policy Gradient. We start by giving an overview of the algorithm and some general implementation details related to the algorithm and the VPT foundation model for Minecraft. Finally, we provide details on the preference interface in subsection 3.3.2 and the reward predictor network in subsection 3.3.3, which are both crucial parts of the system.

3.3.1 Human-Guided Phasic Policy Gradient

The HPPG algorithm comprises three phases: the policy phase, the preference phase, and the auxiliary phase. The policy phase, commonly called the Proximal Policy Optimization (PPO) phase, is analogous to the PPO algorithm and focuses on optimizing the policy and value networks. A human overseer evaluates trajectory segments in the preference

phase, updating the reward predictor network accordingly. Lastly, the auxiliary phase leverages all data in the auxiliary buffer to update the networks, typically for multiple epochs. Algorithm 3.1 aims to incorporate a human-in-the-loop approach into the original PPG algorithm by introducing an additional phase for human preferences.

Algorithm 3.1 Human-Guided Phasic Policy Gradient (HPPG)

Initialize preference queue Q_P
Initialize segment queue Q_S
Initialize Preference Interface with Q_P and Q_S
Initialize Reward Predictor with Q_P
for phase $= 1, 2, \ldots$ **do**
  Initialize empty buffer B_{mem}
  Initialize empty buffer B_{aux}
  for iteration $= 1, 2, \ldots, N_\pi$ **do**
    Perform rollouts under current policy π and add all memories to B_{mem}
    Compute value function target $\hat{V}_{\text{targ}}$ for each state s_t
    Add all memories to B_{mem}
    for epoch $= 1, 2, \ldots, E_\pi$ **do**
      Optimize $L_{\text{clip}} + \text{SS}[\pi]$ w.r.t. θ_π
    end for
    for epoch $= 1, 2, \ldots, E_V$ **do**
      Optimize L_{value} w.r.t. θ_V
    end for
    Add all $(s_t, \hat{V}_{\text{targ}})$ to B
  end for
  for memory in B_{mem} **do**
    Create fixed-length segments from memory
    Add segments to Q_S
  end for
  for segment pairs in Q_S **do**
    Query human for preference on random segment pairs
  end for
  Train the reward predictor on Q_P
  Compute and store current policy $\pi_{\theta_{\text{old}}}(\cdot|s_t)$ for all states s_t in B
  for epoch $= 1, 2, \ldots, E_{\text{aux}}$ **do**
    Optimize L_{joint} w.r.t. θ_π, on all data in B
    Optimize L_{value} w.r.t. θ_V, on all data in B
  end for
end for

To optimize memory efficiency, we employ named tuples for Memory and AuxiliaryMemory. The Memory tuple represents a single timestep t in an episode, whereas the AuxiliaryMemory tuple encompasses values for an entire episode. For the buffers B_{mem} and B_{aux}, we utilize a deque, a Python list optimized for data access at its endpoints. The buffers can be thought of as lists of episodes. Moreover, the segment queue is implemented as a standard list, enabling easy random segment sampling, while the preference queue is also a deque.

Like the PPG algorithm, the HPPG algorithm employs a disjoint value network that we call the critic, in contrast to other actor-critic algorithms where the actor and critic are part of the same network. The absence of shared parameters between networks allows for enhanced sample efficiency by reusing data more frequently. In our implementation using the VPT models, we initialize two instances of the model, one as the actor and the other as the critic. Both instances have π and value heads; however, we freeze some parameters in the critic instance since it is solely used for value predictions and optimized during the PPO phase. Only the π head is optimized during the PPO phase for the actor instance.

We first resize the environment observation during rollouts to a 128x128 RGB image as input to the VPT model. Subsequently, we act in the environment based on the current observation, which is used to calculate log probabilities with the actor and value predictions with the critic. The reward predictor then estimates a reward for the observation, and all values are added to a memory appended to a buffer for each timestep, t, in an episode. Upon completing an episode (either due to the agent dying or reaching the maximum environment timesteps), we calculate all values for the last observation and add the episode to a buffer.

Early experimentation with the algorithm showed signs of catastrophic forgetting after just 1-2 iterations, which we assumed could be because the reward predictor gave bad rewards in the beginning. We implement a pre-training method to prevent rewards from not aligning with human preferences at the beginning of training. The pre-training method collects memories, queries preferences, and trains the reward predictor before RL fine-tuning using its predicted rewards. To introduce more variation to the segments and control how many segment pairs are presented in each phase, we introduce a hyperparameter n_pairs which controls how many pairs of segments are sampled from the segment database. This allows us to control the time spent querying the human overseer instead of using all the segments available, which increases for each episode collected during rollout.

Policy/PPO Phase

During the policy phase, each episode's Generalized Advantage Estimation (GAE) is calculated, and observations, returns, and log probabilities are added to the auxiliary buffer. Next, a data loader is initialized with these memories, maintaining the sequential structure of the episode to reconstruct the hidden states during training. This sequential data form mini-batches looped through to calculate new action log probabilities, entropy, and value predictions.

One of the significant challenges encountered in this phase is the reconstruction of hidden states, which limits the full exploitation of matrix multiplication capabilities in the PyTorch library. Each observation and action in the mini-batch must be iterated to calculate log probabilities and new values while updating the hidden state for each iteration. This constraint slows down the computation process. One solution to this batch issue is saving the initial hidden states during rollout equal to the minibatch size, then reconstructing the hidden state to work with the minibatch size during the policy phase. However, it is uncertain how necessary this step is as Baker et al. [4] does not provide implementation details, and it might be that reconstruction could be disregarded completely.

To calculate the policy loss, we first define a few key variables. We start with the new and old policy ratios, denoted as $\mathrm{r}[\pi]$. This ratio is calculated by exponentiating the difference between the new and old log probabilities:

$$\mathrm{r}[\pi] = \exp(\log p(\theta_\pi) - \log p(\theta_{\pi_{\text{old}}})) \tag{3.1}$$

Next, we define the advantage, denoted as $\hat{A}$, and value function targets, denoted as $\hat{V}^{\text{targ}}$. Both $\hat{A}$ and $\hat{V}^{\text{targ}}$ are computed with GAE [23]. Using these variables, we define two surrogate losses, $\mathrm{L}^{pol}_{\text{unclip}}$ and $\mathrm{L}^{pol}_{\text{clipped}}$:

$$\mathrm{L}^{\pi_\theta}_{\text{unclip}} = \mathrm{r}[\pi] \cdot \hat{A} \tag{3.2}$$

$$\mathrm{L}^{\pi_\theta}_{\text{clip}} = \mathrm{clip}(\mathrm{r}[\pi], 1-\epsilon, 1+\epsilon) \cdot \hat{A} \tag{3.3}$$

Where ϵ is a hyperparameter determining the allowed policy change, the clipped surrogate objective helps prevent overly large policy updates.

Finally, the policy loss is calculated as the negative expectation of the minimum of the two surrogate losses, subtracted by the entropy bonus S of the policy scaled by a hyperparameter β_s, and the Kullback-Leibler divergence from the original policy scaled by a hyperparameter λ:

$$L^{\pi_\theta} = -\min(\mathrm{L}^{\pi_\theta}_{\text{unclip}}, \mathrm{L}^{\pi_\theta}_{\text{clip}}) - \beta_s \cdot \mathrm{S}[\pi] - \lambda \cdot \mathrm{KL}(p(\theta_{\text{old}}) || p(\theta)) \tag{3.4}$$

Note that while the original PPO loss L_{CLIP} is similar to $\text{L}_{\text{clipped}}$, in the policy loss calculation, we take the minimum of the two surrogate losses, which can be seen as a more conservative update rule.

In the calculation of the value loss for the critic network, we first establish the notion of clipped values. The clipped value, denoted by $\theta_{V_{\text{clipped}}}$, is computed by taking the old value estimate $\theta_{V_{\text{old}}}$ and adding a clipped difference between the current value estimate θ_V and $\theta_{V_{\text{old}}}$. The clipping is performed within a specified range defined by the variable 'clip'. This operation ensures that the updated value estimate does not deviate too much from the old estimate. This step is mathematically represented as follows:

$$\theta_{V_{\text{clipped}}} = \theta_{V_{\text{old}}} + \text{clip} \cdot (\theta_V - \theta_{V_{\text{old}}}) \tag{3.5}$$

With the clipped values computed, we proceed to calculate two types of value losses, denoted by L_{clipped} and $L_{\text{unclipped}}$ respectively. The first type, L_{clipped}, is computed as the squared difference between the clipped value estimate $\theta_{V_{\text{clipped}}}$ and the GAE:

$$L^{val}_{\text{clipped}} = (\theta_{V_{\text{clipped}}} - \hat{V}^{\text{targ}})^2 \tag{3.6}$$

The second type of value loss, $L_{\text{unclipped}}$, is the squared difference between the current value estimate θ_V and the GAE:

$$L^{val}_{\text{unclipped}} = (\theta_V - \hat{V}^{\text{targ}})^2 \tag{3.7}$$

Finally, the total value loss L_{value} is computed as the maximum of L_{clipped} and $L_{\text{unclipped}}$. This step ensures that the total value loss captures the worst-case scenario between the two types of losses, promoting more robust learning:

$$L^{\text{value}} = \max(L^{val}_{\text{clipped}}, L^{val}_{\text{unclipped}}) \tag{3.8}$$

This method of calculating the value loss for the critic network facilitates more stable updates, helping to mitigate the high variance often associated with policy gradient methods.

Interestingly, challenges with the hidden states initially made it impossible to back-propagate both the actor and critic networks during this phase. However, this issue was resolved by detaching the reconstructed hidden states. In alignment with the PPG

algorithm's standard setting for the policy phase, the policy phase is executed for one epoch.

Preference Phase

During the preference phase, the algorithm solicits preferences from a human overseer for fixed-length segments and trains the reward predictor accordingly. This process aims to better align the predictor rewards with human preferences before the subsequent rollout. Detailed descriptions of the preference interface and reward predictor can be found in subsection 3.3.2 and subsection 3.3.3, respectively.

Auxiliary Phase

In the auxiliary phase, we optimize both the L_{joint} and L_{value} losses, utilizing all the data stored in the auxiliary buffer. We generate new value predictions during this phase using the current policy θ_π and the disjoint value network θ_{V}.

The joint loss L_{joint} is determined by applying the clipped value loss function (refer to Equation 3.8) to the value predictions obtained under the current policy θ_π. This loss calculation also considers the Kullback-Leibler divergence from the policy saved before the auxiliary phase.

On the other hand, the value loss L_{value} is updated further by employing the clipped value loss function on the values predicted under the current disjoint value network θ_{V}.

The auxiliary phase is usually run for several epochs in contrast to the policy phase for a standard PPG implementation [4, 13]. We decided to follow this and choose 5 epochs based on this information. Conceptually, this phase can be considered the process of distilling the value function into the policy network, improving the policy's ability to approximate the optimal strategy. Furthermore, the hyperparameters chosen were based on a best-guess approach based on related works and experiences with the model during behavior cloning. The learning rate was set to the standard learning rate of 2e-5 for PPO/PPG algorithms when using a KL divergence loss [4].

Hyperparameter	Value
Epochs	1
Auxiliary Epochs	5
Minibatch Size	32
Max Gradient Norm	5
Learning Rate	2×10^{-6}
Betas	(0.9, 0.999)
Gamma	0.99
Lambda	0.95
PPO Clip	0.2
Value Clip	0.2
Beta S	0.01
Iterations	100
Wake cycles pr. Preference	1
Wake cycles pr. Auxiliary	2
Segment Length	100
Rollouts	4

Table 3.2: Hyperparameters for the HG-PPG Algorithm.

3.3.2 Preference Interface

The preference interface allows the human overseer to evaluate and compare trajectory segments efficiently. The interface displays two randomly selected trajectory segments as short, looped videos side by side, offering a clear visualization of the agent's actions in different situations.

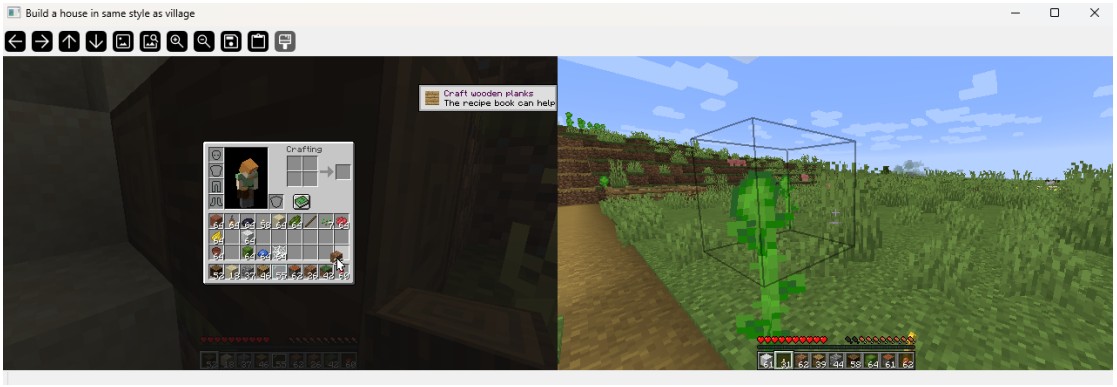

Figure 3.4: The preference window displaying two segments (σ^1, σ^2) side by side. Each game window is in the dimensions of the environment, 360x640. The task is described in natural language in the title of the window. When the user presses a key, the window will disappear, and the options for selecting a segment will appear in the command-line interface. Finally, after a preference is given, the next segment pair in the segment queue will appear until there are no more segments or max segment pairs are reached.

Upon pressing any key, the video playback window closes, and the human overseer is given four choices to express their preference in the command line interface (CLI): left video, right video, equal, or incomparable. These preferences are recorded in the preference queue as triples (σ^1, σ^2, μ), where σ^1 and σ^2 denote the left and right segments, and μ is the distribution over $\{1, 2\}$ representing the preferred segment.

When the overseer selects one segment as preferable, μ allocates all its mass to that choice. If both segments are considered equally preferable, μ is set to a uniform distribution. In cases where the human deems the segments incomparable, the segment pair is not included in the preference queue. Therefore, selecting a segment pair as incomparable is discouraged.

The segment pairs are selected randomly from the segment queue, ensuring a diverse set of samples for the human overseer to evaluate and allowing for the efficient incorporation of human preferences into the learning process.

3.3.3 Reward Predictor Network

The Reward Predictor Network is responsible for estimating rewards for observations from human preferences. The network architecture comprises four Convolutional layers, with Batch Normalization and Dropout layers inserted between each convolutional layer to improve stability and reduce overfitting.

The Reward Predictor Network is implemented as two separate classes, the `RewardPredictorCore` class and the `RewardPredictorNetwork` class. The `RewardPredictorCore` class is responsible for defining the architecture of the network,

including the feature extraction and classification stages. The feature extractor consists of convolutional layers, batch normalization, dropout layers, and ReLU activation functions. The classifier is composed of linear layers and ReLU activation functions. The network takes input in the shape of (3, 256, 256), representing the RGB channels and spatial dimensions of the input image, which are twice the dimensions as the input to the VPT model after resizing the observation from the environment.

The `RewardPredictorNetwork` class wraps the `RewardPredictorCore` and includes additional functionality for training, saving, and loading the model. The network employs the Adam optimizer as it requires little to no hyperparameter-tuning with a standard learning rate of 0.001 [29]. The `train_step` method calculates the probability that one segment is preferred over the other by comparing the sum of rewards (as shown in Equation 3.10 and Equation 3.11), computes the loss using cross-entropy Equation 3.9, and performs backpropagation to update the model parameters. Finally, the `train` method iteratively applies the `train_step` method on the preference queue until it is empty or the maximum number of iterations is reached.

As previously defined, the notation $\sigma_1 \succ \sigma_2$ indicates that segment σ_1 is preferred over segment σ_2. The loss function for the Reward Predictor Network is the cross-entropy loss, defined as:

$$loss(\hat{r}) = -\left(\mu(1)\log\hat{P}(\sigma_1 \succ \sigma_2) + \mu(2)\log\hat{P}(\sigma_2 \succ \sigma_1)\right), \tag{3.9}$$

where $\mu(1)$ and $\mu(2)$ represent the human preference distribution over the two segments, and $\hat{P}(\sigma_1 \succ \sigma_2)$ and $\hat{P}(\sigma_2 \succ \sigma_1)$ are the predicted probabilities for the two segments, calculated as:

$$\hat{P}(\sigma_1 \succ \sigma_2) = \frac{\exp\sum\hat{r}(\sigma_t^1)}{\exp\sum\hat{r}(\sigma_t^1) + \exp\sum\hat{r}(\sigma_t^2)}, \tag{3.10}$$

$$\hat{P}(\sigma_2 \succ \sigma_1) = \frac{\exp\sum\hat{r}(\sigma_t^2)}{\exp\sum\hat{r}(\sigma_t^1) + \exp\sum\hat{r}(\sigma_t^2)}, \tag{3.11}$$

After gathering the initial preferences, we pre-train the network for 200 epochs with around 1000 preferences. Figure 3.5 shows the loss over epochs in the pre-training of the network. Previous works had shown to get results on par with using true rewards after just 5-900 preference queries during training [19], so we chose to pre-train with 1000 preferences in hopes of remedying the catastrophic forgetting experienced.

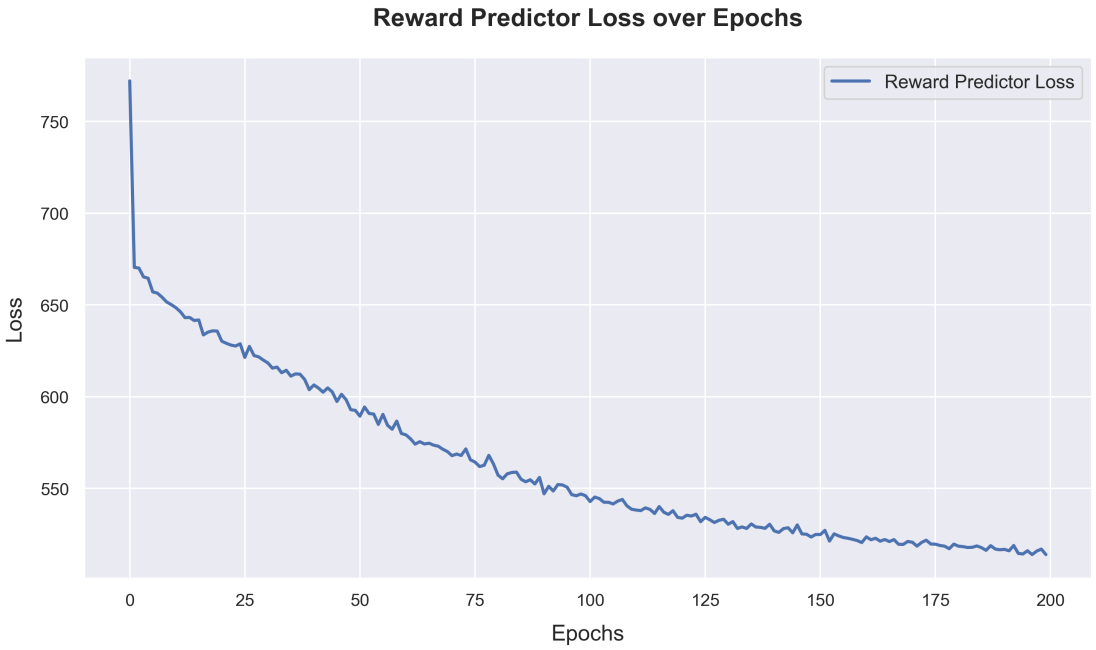

Figure 3.5: The loss of the Reward Predictor network over 200 epochs using 1000 initially gathered preferences with a learning rate of 0.001 and the Adam Optimizer.

Chapter 4

Experimental Evaluation

This chapter will present the experimental setup, as well as the experimental results. Finally, the results are discussed in the analysis section.

4.1 Experimental Setup

The experimental protocol followed in this study revolves around leveraging Behavior Cloning (BC) and Reinforcement Learning (RL) methods with human preferences to fine-tune a foundation model for performing house construction tasks in Minecraft. We implemented our algorithms in PyTorch [31] and Python [32]. We interface with the MineRL BASALT environment [11] through the OpenAI Gym [8]. Because of computing constraints, we fine-tune the most miniature model with 71M parameters as the foundation model, ruling out the 248M and 500M models [4].

We evaluate the models on the `BuildvillageHouse-v0` BASALT environment outlined in chapter 2 and the corresponding dataset described in Table 2.2. The dataset consists of expert demonstrations by contractors completing the task of building a village house through videos and ground truth labels. Each episode may span multiple videos, as the maximum length for a video is 5 minutes, while the length of an episode is 12 minutes (14400 timesteps). In order to establish the train and validation datasets, we organized the videos and labels according to their episode ID and selected a random 20% of episodes for the validation dataset, while the remaining episodes were assigned to the training dataset.

The dataset provides valuable ground truth data that captures the expertise of human contractors in building a village house. These expert demonstrations serve as a reference for training and evaluating the models. In addition, the ground truth labels provide

information on the correct sequences of actions and placements of blocks to complete the task successfully.

This task is considered complex due to several challenges, including the sheer size of the observation and action space. In addition, the agent must stay near the village to complete the task, navigate potential hazards such as water bodies and aggressive golems, choose suitable materials to match the village biome and avoid confusion from other village houses. An example of a human build house can be seen below in Figure 4.1.

Figure 4.1: An example of a human-constructed house. The image is taken from a video in the BuildVillageHouse Dataset

For the qualitative evaluation of the models after fine-tuning, we assess their capacity to generate task performances that resemble those of human beings. This evaluation involves visually inspecting the model's behavior during house construction tasks and comparing it to the behavior observed in expert demonstrations. However, for the Reward Predictor, a qualitative evaluation is not possible.

Quantitative evaluation of the models is challenging when an observable environmental reward is unavailable. Instead, we rely on alternative quantitative measures for building a house, such as the number of blocks placed during an episode. By comparing the number of blocks placed after fine-tuning with BC to the number before fine-tuning, we can assess the impact of the fine-tuning process. Additionally, we evaluate the reward predictor by predicting rewards on episodes of expert demonstrations and comparing them to rewards predicted for agents exploring the environment.

The experimental setup included an annotation experiment to gather initial preferences for training the reward predictor network. The human overseers were given instructions

and guidelines to assess their preferences for different segments. These preferences were collected after fine-tuning with BC and served as ground truth for training the reward predictor network.

Further details of the instructions provided to the human overseers and the annotation experiment can be found in Appendix A.

4.2 Experimental Results

Post BC fine-tuning, the model showed a proclivity for placing blocks sequentially indicative of house construction. Unlike the foundation model, which primarily focused on material collection and crafting, the fine-tuned model demonstrated a range of behaviors necessary for house construction. Notably, the model showed competence in clearing land plots, increased block-placing frequency, and remained near the village, suggesting its understanding of the task location. Additionally, the model often employed a human-like building technique involving jumping and placing a block underneath itself. Compared to the foundation model, the fine-tuned model performs the task better after assessing the agent in the environment for several episodes and during preference collection.

We showed 10 videos of the foundation model and the post BC fine-tuned model to a non-expert human to evaluate the performance. The experiment was performed by first showing a video of a human performing the task of building a house, taken from the dataset for the evaluator to get idea of the task. Then, the evaluator was shown a video of the foundation model, followed by a video of the fine-tuned model and asked to write down if the second video performed the task better or worse than the first video. This was done for a total of 10 rounds and we found that the fine-tuned model won all ten rounds using an unbiased non-expert evaluator.

However, there is still a long way to go to perform at a human level. For example, the model lacks an understanding of different materials, often leading to the model getting stuck by placing cobwebs that limit the agent's movement when standing inside them. Once the agent is stuck inside a cobweb, it often fails to recover. It also fails to understand that water is a hazard, sometimes leading to its death by drowning while building underwater. The model can also be seen moving to another part of the village to start a new building project during an entire episode. Another seen "failure" is the model constructing a wall around itself, effectively making it unable to move unless it starts breaking the walls down again (which it sometimes does).

After incorporating Behavior Cloning (BC) during the fine-tuning process, the reward predictor network was trained using approximately 1000 preferences (σ^1, σ^2, μ) over 200 epochs. The evaluation of the network involved predicting rewards on expert demonstrations. Our underlying assumption was that a well-performing reward predictor network would assign higher rewards to expert demonstrations compared to an agent operating at a sub-human level.

However, when we employed the reward predictor network to predict rewards based on observations from the agent after BC fine-tuning and expert demonstrations, we observed a strong bias toward assigning negative rewards, seen in Figure 4.2. This outcome further reinforced our initial observation during early RL fine-tuning experiments, indicating that the predicted rewards did not align with human preferences or correspond to the "true" reward.

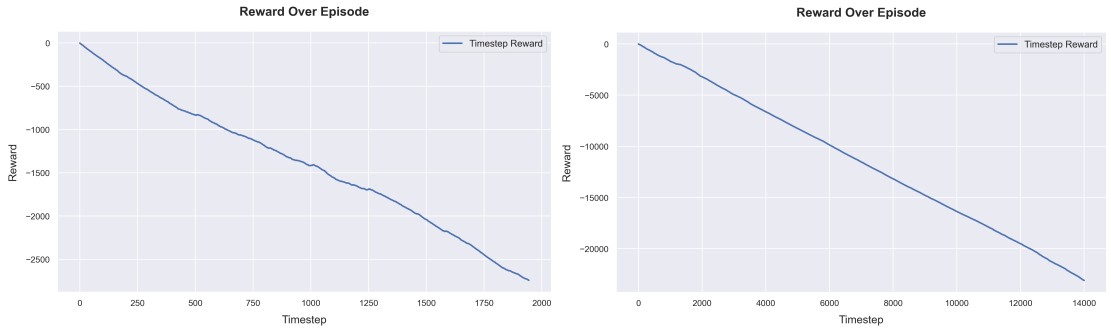

Figure 4.2: These figures show the predicted reward over an episode for observations from an expert demonstration and the agent after fine-tuning with behavioral cloning. **(left)** The predicted reward over an episode for an expert demonstration. **(right)** The predicted reward over an episode of the agent after fine-tuning with behavioral cloning. The Reward Predictor Network is seen to be heavily biased toward predicting negative rewards after training on 1000 preferences for 200 epochs.

While gathering preferences for pre-training, the overseer encountered challenges in selecting one segment over the other due to their comparable performance. This sentiment is reflected in the statistical analysis of the preferences, as shown in Table 4.1, where nearly 70% of the preferences were labeled as having no preference (equal).

Preference Type	**Count**	**Percentage (%)**
One Segment Preferred	263	26.3
No Preference	698	69.8
Incomparable	39	3.9
Total	1000	100

Table 4.1: Statistics showing the count and percentages of the preferences collected during gathering initial preferences used to pre-train the Reward Predictor Network.

Consequently, the results from Reinforcement Learning (RL) fine-tuning did not yield the desired outcomes, as the model exhibited signs of catastrophic forgetting despite implementing measures to mitigate this issue. Following a few policy updates, the model completely lost the skills it had acquired during the BC fine-tuning phase. Instead of performing the intended house construction task, the model displayed a peculiar behavior characterized by continuously staring at the sky or ground, rotating the camera, and jumping without any apparent purpose or direction.

4.3 Analysis

The presented results shed light on the achievements and shortcomings of our approach in developing a Minecraft house-building model through a combination of Behavior Cloning (BC) and Reinforcement Learning (RL). Furthermore, these findings have implications for the objectives outlined in the introduction of this thesis, and they provide insights into the success and failure cases observed during the evaluation.

Regarding BC fine-tuning, the results demonstrate promising progress in acquiring the necessary skills for the house-building task. While the model did not reach human-level proficiency, its performance surpassed the foundation model's, indicating a positive impact of BC fine-tuning despite the limitations imposed by the available data and model size. This outcome aligns with our objective of improving the model's performance through BC.

In contrast, the outcomes of RL fine-tuning revealed significant challenges and limitations. Catastrophic forgetting during RL fine-tuning can be a setback, suggesting that the model lost previously acquired skills. However, it is crucial to note that the primary reason for this result lies in the inability of the reward predictor network to assign meaningful rewards. Minecraft's inherent complexity and expansive state space make it difficult for the reward predictor network to identify human-preferred observations accurately. This limitation calls for further exploration of alternative methods, such as inferring a reward function from expert demonstrations, to address the issue of reward assignment.

The shortcomings of the reward predictor network may stem from the relatively simplistic structure of its architecture. Despite past research demonstrating the effectiveness of similar designs in less complex environments, the intricacy of capturing human preferences within Minecraft might necessitate more advanced network configurations. Furthermore, the scope of preferences utilized in this study was relatively narrow, a factor that may have impacted the network's performance. Therefore, future explorations should consider a

more substantial dataset of preferences and experiment with diverse network architectures to optimize the system's performance.

Despite the challenges encountered during RL fine-tuning, this thesis provides valuable insights into the limitations of this approach. The complex nature of the Minecraft task and environment poses significant hurdles to effectively incorporating human preferences into the reward function. Nevertheless, these findings pave the way for future research to explore alternative techniques and address the shortcomings observed in this study.

It is crucial to acknowledge that the effectiveness of the proposed human-guided phasic policy gradient algorithm is contingent upon its components, with the reward predictor network identified as the weakest link. The biases observed in the reward predictor network, leaning towards assigning negative rewards, limit the comprehensive evaluation of the algorithm. This limitation underscores the importance of rectifying and improving the reward prediction mechanism in future research endeavors.

Chapter 5

Conclusions

This study aimed to address two primary research questions involving enhancing sample efficiency in a complex task environment using imitation learning and human feedback to improve agent performance in an environment without observable rewards. Through our experiments, we have gained valuable insights in these research directions.

Our approach utilizing behavioral cloning demonstrated several promising results. Our agent displayed behaviors indicative of a rudimentary understanding of the task and showed competence in activities necessary for house construction. Notably, a non-expert human preferred its performance to the foundation model, suggesting that the imitation learning process yielded substantial improvements.

Despite these achievements, our model is still far from reaching human-level performance. Furthermore, we have identified several areas where the model's understanding needs improvement, such as material discrimination, recovery from certain predicaments, and overall task efficiency. To answer research question 1, using a KL divergence loss function when fine-tuning with behavior cloning gives decent results even with limited data and compute.

We suggested extending the Phasic Policy Gradient algorithm with a stage for human feedback. The proposed approach is constituent upon its components and is only as effective as its weakest link. One of the main challenges encountered in this study was the bias of the reward predictor network towards negative rewards. This outcome may suggest that the complexity of the network architecture needed to be revised to handle the expansive state space of Minecraft effectively. Despite these limitations, our work contributes valuable insights to the existing body of knowledge and opens up several avenues for future research. Furthermore, the approach and observations from this

study offer a foundation to develop more sophisticated models capable of handling the complexity of Minecraft or similar environments.

While the agent's performance is not yet at the human level, our results signify that enhancing an agent's performance in complex tasks is possible using a modest dataset and constrained computational resources. This study brings us closer to realizing more efficient and competent AI agents in real-world domains and complex exploration games and offers exciting opportunities for future research.

5.1 Future Directions

For future work in rewardless environments like Minecraft, we encourage exploring more sophisticated network designs for the reward predictor and increasing the number of preferences. In our work, we used a network architecture and amount of preferences similar to what previous works used in simpler environments without success.

Despite the results of the proposed Human-guided Phasic Policy Gradient algorithm, we suggest future researchers optimize the efficiency of the preference phase. For example, we suggest creating an online framework for preference solicitation that could be run asynchronously from the algorithm. The preference phase in the algorithm could then train the reward predictor network and not slow down the training by collecting preferences.

An alternative avenue for future exploration involves pre-training the reward predictor using the available dataset of expert demonstrations. This approach could yield a reward that aligns more closely with the "true" reward, significantly reducing the number of preferences needed during the preference phase. In addition, this strategy could lead to a more efficient overall system by diminishing the need for human interaction, making it a promising solution to the challenges encountered in this study.

Future researchers with more computational resources are also encouraged to use the larger Foundation Models for fine-tuning as these gave better results in previous works when fine-tuning.

Appendix A

Instructions Provided to Human Overseers

A.1 BuildVillageHouse

Giving Feedback

You will be repeatedly presented with two video clips of an AI controlling a Minecraft character. Your task is to evaluate the clips and select the one where better things happen. Base your decision solely on what you observe in the clips.

Task Description

The objective is to build a house in the style of the village without causing any damage to the village structures. The house should be in an appropriate location, preferably next to the path through the village. To assess the performance of the AI, pay attention to the following aspects in the clips:

What constitutes good and bad behavior

1. **House Construction:** Evaluate the quality of the house being built. Consider factors such as completeness, accuracy, and adherence to the village style. Keep in mind that any house construction is better than no construction at all.
2. **Block Placement:** Assess the AI's block placement strategy. Look for patterns or techniques that indicate competence in house construction. Placing blocks in a sequential and organized manner, such as in a row or on top of each other, is preferable to placing blocks far away from each other.

3. **Village Impact:** Determine if the AI's actions have any negative impact on the existing village structures. Look for any signs of damage or disruption caused during the construction process. Avoiding accidental destruction or interference with village structures is important.

4. **Location Selection:** Evaluate if the AI builds the house in an appropriate location. Check if it is positioned next to the path through the village, following the established village layout. The AI should avoid building the house too far away from the village or in a location that disrupts the village structure.

5. **Aesthetics:** Consider the overall aesthetics of the house. If the performance in other aspects is similar, the visually pleasing design may serve as a tiebreaker. While not the primary focus, aesthetics can provide an additional criterion for evaluation.

Instructions for giving feedback:

1. Carefully watch both video clips.

2. Compare the content and actions in the clips based on the aspects mentioned above.

3. Select the number that corresponds to your preference:

   - Press 1 if the clip to the left shows better performance.
   - Press 2 if the clip to the right shows better performance.
   - Press 3 if both clips appear to be the same or if you have no preference.
   - Press 4 if you find it difficult to understand or observe the content of the clips.

4. Please note the following:

   - Base your evaluation solely on the visual information presented in the clips.
   - Do not consider any additional knowledge or context beyond what is shown.
   - If both clips look the same to you or you cannot determine a preference, select option 3 for equal.
   - If you find it challenging to understand or observe the content of the clips, select option 4 for incomparable.

We appreciate your careful evaluation and feedback. Your input will help us assess the performance of the AI in the given task accurately.

Appendix B

Instructions to Compile and Run System

The purpose of this appendix is to provide instructions on how to install, compile, and run the Human-Guided Phasic Policy Gradient system developed in this research, hosted on the Github repository found at https://github.com/DagValvik/Human-Guided-Phasic-Policy-Gradient-in-Minecraft.

B.1 Installation and Execution Instructions

Detailed instructions for installing the necessary software and dependencies, setting up the system, and running the code are provided in the README file in the GitHub repository. Please follow the instructions in the README for a step-by-step guide on how to install and execute the system.

B.2 Source Code/Class Structure

The source code is structured into several files and directories for easy readability and maintainability.

- `code/` - Directory containing the main behavior cloning and HPPG code.
- `data/` - Directory holding the data. You should download the BASALT dataset and place your foundation model files here.
- `scripts/` - Directory containing utility scripts for downloading and splitting data.

Each Python file contains relevant comments to explain what each part of the code does.

Bibliography

[1] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi: 10.1038/nature16961.

[2] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[3] OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

[4] Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos, 2022. URL https://arxiv.org/abs/2206.11795.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[6] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations, 2019.

[7] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 4246–4247. AAAI Press, 2016. ISBN 9781577357704.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[9] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011.

[10] OpenAI. Video-pre-training. https://github.com/openai/Video-Pre-Training, 2022. Accessed: 2023-06-12.

[11] MineRL. Minerl basalt environments, 2020. URL https://minerl.readthedocs.io/en/latest/environments/basalt.html. Accessed: 2023-04-06.

[12] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

[13] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2020–2027. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/cobbe21a.html.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[15] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[16] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[17] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[18] Sham M Kakade. A natural policy gradient. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

[19] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[20] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/mniha16.html.

[21] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences, 2020.

[22] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[23] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[24] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013. doi: 10.1613/jair.3912. URL https://doi.org/10.161%2Fjair.3912.

[25] Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. ISSN 00063444. URL http://www.jstor.org/stable/2334029.

[26] Arpad Elo. *The Rating of Chessplayers, Past and Present*. Arco Pub., 1978.

[27] Wanqi Xue, Bo An, Shuicheng Yan, and Zhongwen Xu. Reinforcement learning from diverse human preferences, 2023.

[28] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari, 2018.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[30] James M. Joyce. *Kullback-Leibler Divergence*, pages 720–722. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_327. URL https://doi.org/10.1007/978-3-642-04898-2_327.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[32] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.