Combining Learning from Human Feedback and Knowledge Engineering to Solve Hierarchical Tasks in Minecraft

Vinicius G. Goecks¹, Nicholas Waytowich¹², David Watkins² and Bharat Prakash³

¹Army Research Laboratory, Aberdeen Proving Ground, Maryland, USA
²Columbia University, New York City, New York, USA
³University of Maryland, Baltimore, Maryland, USA

Abstract

Real-world tasks of interest are generally poorly defined by human-readable descriptions and have no pre-defined reward signals unless it is defined by a human designer. Conversely, data-driven algorithms are often designed to solve a specific, narrowly defined, task with performance metrics that drives the agent’s learning. In this work, we present the solution that won first place and was awarded the most human-like agent in the 2021 NeurIPS Competition MineRL BASALT Challenge: Learning from Human Feedback in Minecraft, which challenged participants to use human data to solve four tasks defined only by a natural language description and no reward function. Our approach uses the available human demonstration data to train an imitation learning policy for navigation and additional human feedback to train an image classifier. These modules, together with an estimated odometry map, are then combined into a state-machine designed based on human knowledge of the tasks that breaks them down in a natural hierarchy and controls which macro behavior the learning agent should follow at any instant. We compare this hybrid intelligence approach to both end-to-end machine learning and pure engineered solutions, which are then judged by human evaluators. Codebase is available at https://github.com/viniciusguigo/kairos_minerl_basalt.

Keywords

Hybrid Intelligence, Human-in-the-Loop Learning, Machine Learning, Artificial Intelligence, Knowledge Engineering, Minecraft

1. Introduction

In this paper, we present the solution that won first place and was awarded the most human-like agent in the MineRL Benchmark for Agents that Solve Almost-Lifelike Tasks, the 2021 Neural Information Processing Systems (NeurIPS) MineRL BASALT competition, “Learning from Human Feedback in Minecraft”¹. Most artificial intelligence (AI) and reinforcement learning (RL) challenges involve solving tasks that have a reward function to optimize over. Real-world
tasks, however, do not automatically come with a reward function, and defining one from scratch can be quite challenging. Because of this, teaching AI agents to solve complex tasks and learn difficult behaviors without any reward function remains a major challenge for modern AI research. The MineRL BASALT competition is aimed to address this challenge by developing AI agents that can solve complex, almost-lifelike, tasks in the challenging Minecraft environment [1] using only human feedback data and without access to any reward function.

Since the MineRL BASALT competition tasks do not contain any reward function, we propose a human-centered machine learning approach instead of using traditional RL algorithms [2]. However, learning complex tasks with high-dimensional state-spaces (i.e. from images) using only end-to-end machine learning algorithms requires large amounts of high-quality data [3, 4], which in this case translates to large amounts of either human-collected or human-labeled data. To circumvent this data requirement, we opted to combine machine learning with knowledge engineering, also known as hybrid intelligence [5, 6]. Our approach uses human knowledge of the task to break them down in a natural hierarchy of subtasks that are controlled by an engineered state-machine, estimated odometry, and the outputs of a learned state classifier that can detect relevant features in the environment. The available human demonstration dataset is also used to train a navigation policy via imitation learning to replicate how humans traverse the environment that is also incorporated in the engineered state-machine.

In this paper, we give a detailed overview of our approach, as well as perform an ablation study on how well our hybrid intelligence approach works compared using either learning from human demonstrations or engineered solutions alone. Our two main contributions are:

• An architecture that combines engineered knowledge of the tasks to be solved together with machine learning modules in order to solve complex hierarchical tasks in Minecraft.
• Empirical results on how hybrid intelligence compares to both end-to-end machine learning and pure engineered approaches when solving complex, real-world-like tasks, as judged by real human evaluators.

2. Background and Related Work

End-to-end machine learning: in this work we use the term “end-to-end machine learning” for algorithms that learn purely from data, with minimal bias or constraints added by human designers, besides the ones that are already inherently built-in the learning algorithm. Such examples include deep reinforcement learning algorithms directly playing video games from pixel inputs [7, 8] or with algorithms that automatically decomposes tasks that involve long-term credit assignment or memorization in hierarchies with different time scales [9, 10]. This often includes massive parallelization and distributed computation [11, 12] in order to fulfill the data requirement of these algorithms. Despite all advances made in this field, these techniques have not been demonstrated to scale to tasks with the complexity presented in the MineRL BASALT competition.

Human-in-the-loop machine learning: Learning from human feedback can take different forms depending on how human interaction is used in the learning loop [13, 14]. A learning agent can be trained based on human demonstrations of the task [15, 16, 3] (including learning from suboptimal demonstrations [17], end goals [18], or directly from successful examples instead
of a reward function [19]), augmenting the human demonstrations with online interventions [20, 21, 22, 23] or offline labeling [24, 25], learning the reward or cost function from used by the demonstrator [26, 27], through sparse interactions in the form of evaluative feedback [28, 29, 30], human preferences given a pair of trajectories [31], natural language-define goals [32], and even combining human data with reinforcement learning [33, 34, 35]. Similar to end-to-end machine learning approaches previously discussed, these techniques have not scaled to tasks with the complexity presented in the MineRL BASALT competition.

**The Minecraft learning environment:** Minecraft is a procedurally-generated, 3D open-world game, where the agent observes the environment from a first-person perspective, collects resources, modifies the environment’s terrain, crafts tools that modifies the agent’s capabilities, and possibly interacts with other agents in the same environment. Given the open-world nature of this environment, there are no predefined tasks or built-in reward signals, which gives the task designer flexibility to define tasks with virtually any level of complexity. The release of Malmo [1], a platform that enabled artificial intelligence experimentation in the game of Minecraft, gave researchers the capability to develop learning agents to solve tasks similar or analogous to the ones seen in the real world. Additionally, the Minecraft environment also served as a platform to collect large human demonstrations datasets such as the MineRL-v0 dataset [36], a dataset of human demonstrations consisting of over 60 million automatically annotated state-action pairs across a variety of related tasks in Minecraft, experimentation with large scale imitation learning algorithms [37], as a world generator for realistic terrain rendering [38], a sample-efficient reinforcement learning competition using human priors (MineRL DIAMOND challenge) [2], and now as a platform for a competition on solving human-judged tasks defined by a human-readable description and no pre-defined reward function, the MineRL BASALT competition [39].

**3. Problem Formulation**

The 2021 Neural Information Processing Systems (NeurIPS) MineRL BASALT competition, “Learning from Human Feedback in Minecraft”², challenged participants to come up with creative solutions to solve four different tasks in Minecraft [39] using the “MineRL: Towards AI in Minecraft”³ simulator [40]. These tasks aimed to mimic real-world tasks, being defined only by a human-readable description and no reward signal returned by the environment. The official task descriptions for the MineRL BASALT competition⁴ were the following:

- **FindCave:** The agent should search for a cave, and terminate the episode when it is inside one.
- **MakeWaterfall:** After spawning in a mountainous area, the agent should build a beautiful waterfall and then reposition itself to take a scenic picture of the same waterfall.
- **CreateVillageAnimalPen:** After spawning in a village, the agent should build an animal pen containing two of the same kind of animal next to one of the houses in a village.

---

²Official competition webpage: https://www.aicrowd.com/challenges/neurips-2021-minerl-basalt-competition.
³MineRL webpage: https://minerl.io/.
⁴MineRL BASALT documentation: https://minerl.io/basalt/.
• **BuildVillageHouse**: Using items in its starting inventory, the agent should build a new house in the style of the village, in an appropriate location (e.g. next to the path through the village), without harming the village in the process.

The competition organizers also provided each participant team with a dataset with 40 to 80 human demonstrations for each task, not all of them completing the task, and the starter codebase to train a behavior cloning agent, to evaluate the agent, and to make the solution submission. Additionally, the training for all four tasks together was limited to four days of computing time, and participants were allowed to collect up to 10 hours of additional human-in-the-loop feedback.

4. Methods

**Figure 1.** Diagram illustrating our approach. Using data from the available human demonstration dataset, humans provide additional binary labels to image frames to be used to train a state classifier that can detect relevant features in the environment such as caves and mountains. The available human demonstration dataset is also used to train a navigation policy via imitation learning to replicate how humans traverse the environment. A separate odometry module estimates the current agent’s position and heading solely based on the action taken by the end. During test time, the agent uses the learned state classifier to provide useful information to an engineered state-machine that controls which subtask the agent should execute at every time step.

Since no reward signal was available from the environment and compute time was limited, direct deep reinforcement learning approaches were not feasible [7, 41, 8]. With the limited human demonstration dataset, end-to-end behavior cloning also did not result in high-performing policies, given that imitation learning requires large amounts of high-quality data [3, 4]. We also attempted to solve the tasks using adversarial imitation learning approaches such as Generative Adversarial Imitation Learning (GAIL) [42], however, the large-observation space and limited compute time also made this approach infeasible.
Hence, to solve the four tasks of the MineRL BASALT competition, we opted to combine machine learning with knowledge engineering, also known as hybrid intelligence [5, 6]. As seen in the main diagram of our approach shown in Figure 1, the machine learning part of our method is seen in two different modules: first, we learn a state classifier using additional human feedback to identify relevant states in the environment; second, we learn a navigation subtask separately for each task via imitation learning using the human demonstration dataset provided by the competition. The knowledge engineering part is seen in three different modules: first, given the relevant states classified by the machine learning model and knowledge of the tasks, we designed a state-machine that defines a hierarchy of subtasks and control which one should be executed at every time step; second, we engineered the more complex subtasks that we were not able to learn directly from data; and third, we engineered an estimated odometry module that provides additional information to the state-machine about the agent and relevant states’ locations, enabling the execution of the more complex engineered subtasks.

4.1. State Classification

Our approach relies on a state machine that changes the strategy depending on the task and subtask. Without having information about the environment’s voxel data, we opted to use the visual RGB information from the simulator to determine what state the agent was currently in. Due to the low resolution of the simulator of $64 \times 64 \times 3$, we decided to use a classifier that labels the whole image rather than parts of the image, such as YOLO [43]. Each label can be present on a given image as there were cases where there were multiple labels at the same time.

The goal of labeling the provided human dataset is to train a classifier that will take the RGB frame as input and output a label for that frame (state classifier), which will later be used to guide the agent’s subtasks. There are 13 possible labels for a RGB frame, as illustrated in Figure 2 and described below:

- **none**: frame contains no relevant states (54.47 % of the labels).
- **has_cave**: agent is looking at a cave (1.39 % of the labels).
- **inside_cave**: agent is inside a cave (1.29 % of the labels).
- **danger_ahead**: agent is looking at a large body of water or monster (3.83 % of the labels).
- **has_mountain**: agent has a complete view of a mountain (usually, from far away) (4.38 % of the labels).
- **facing_wall**: agent is facing a wall that cannot be traversed by jumping only (4.55 % of the labels).
- **at_the_top**: agent is at the top of a mountain and looking at a cliff (3.97 % of the labels).
- **good_waterfall_view**: agent has framed a waterfall in view (3.16 % of the labels).
- **good_pen_view**: agent has framed a pen with animals in view (4.12 % of the labels).
- **good_house_view**: agent has framed a house in view (2.58 % of the labels).
- **has_animals**: frame contains animals (pig, horse, cow, sheep, or chicken) (9.38 % of the labels).
- **has_open_space**: agent is looking at an open-space of about 6x6 blocks with no small cliffs or obstacles (imagine you are looking for a flat area to build a small house or pen) (7.33 % of the labels).
Figure 2. Illustration of positive classes of states classified using additional human feedback. Humans were given image frames from previously collected human demonstration data and were assigned to give binary labels for each of the illustrated 12 states, plus a null case when no relevant states were identified.

- **animals_inside_pen**: agent is inside the pen after luring all animals and has them in view (0.81% of the labels).

To train this system we labeled 81,888 images each corresponding to labels from all tasks using a custom graphical user interface (GUI), as showed in Appendix A. 80% of images were used for training, 10% were used for validation, and 10% was used for testing. The model is a convolutional classifier with a $64 \times 64 \times 3$ input and $13 \times 1$ output. It is modeled after the encoder from the autoencoder from DeepTamer [44].

### 4.2. Estimated Odometry

To engineer some of the needed subtasks, for example, to return to a location where animals were previously seen after building a pen, our method required basic localization from the agent
Figure 3: Example of odometry map (right frame) generated in real time from the agent’s actions as it traverses the environment. Relevant states classified from image data (left frame) also have their locations tagged in the map and used for specific subtasks, for example, to return to a location where animals were previously seen after building a pen.

With respect to the environment. However, under the rules of the competition, we were not allowed to use any additional information from the simulator besides the image data the agent has access to and inventory information. That means no information about the ground truth location of the agent, camera pose, or any explicit terrain information.

Given these constraints, we opted to implement a custom odometry method that took into consideration only the actions from the agent and basic characteristics of the Minecraft world and the simulator. It is known that the simulator runs at 20 frames per second, which means that there is a 0.05 second interval between each frame. According to the Minecraft Wiki\(^5\), walking speed is approximately 4.317 m/s, 5.612 m/s while sprinting, or 7.127 m/s when sprinting and jumping at the same time, which translates to approximately 0.216, 0.281, or 0.356 meters per frame when walking, sprinting, or sprinting and jumping, respectively. In this case, under the assumption that the world is flat and that the agent starts at the \((0, 0)\) \(x\) and \(y\) map coordinates, when the agents commands a discrete action to move forward its position is updated by 0.216 meters. The time it takes to accelerate to defined terminal speeds was not taken into consideration during the estimated odometry. Another limitation is that we were not able to reliably detect when the agent is stuck behind an obstacle, which causes the estimated location to drift even though the agent is not moving in the simulator.

Since the agent already commands camera angles in degrees, the heading angle \(\theta\) is simply updated by accumulating the horizontal camera angles commanded by the agent. More generally, this estimated odometry assumes the agent follows point-mass kinematics:

\[
\begin{align*}
\dot{x} &= V \cos(\theta) \\
\dot{y} &= V \sin(\theta),
\end{align*}
\]

where \(V\) is the velocity of the agent, which takes into consideration if the agent is walking, sprinting, or sprinting and jumping.

\(^5\)Minecraft Wiki - Walking: https://minecraft.fandom.com/wiki/Walking.
Using this estimated odometry and the learned state classifier, it is possible to attach a coordinate to each classified state and map important features of the environment to be used later by different subtasks and the state-machine. For example, it is possible to keep track of where the agent found water, caves, animals, and areas of open space that can be used to build a pen or house. Figure 3 shows a sample of the resulting map overlaid with the classified states’ location and current odometry readings.

4.3. Learning and Engineering Subtasks and the State-Machine

One of the main complexities of solving the proposed four tasks is that most of them required from the agent certain levels of perception capabilities, memory, and reasoning over long-term dependencies in a hierarchical manner. For example, the \textit{CreateVillageAnimalPen} task required the agent to first build a pen nearby an existing village (requires identifying what a village is, then identifying a good location to build a pen such as a flat terrain), then, once the pen was built (required coordination to combine different blocks and place them adjacently to each other in a shape the resembles a pen, usually a square or rectangle, with an access door that can be closed or open in order to trap animals), search for at least two of the the same animal in the nearby vicinity (requires being able to navigate around the terrain, avoiding obstacles and dangerous terrain such as lava or holes, then being able to identify animals and their species from $64 \times 64$ image data), lure them with the specific food type they eat (requires being able to associate pixels to the correct food types), walk them back to the pen the agent initially built (requires similar navigation capabilities to avoid obstacles and dangerous terrain, in addition to remembering where the pen was built), escape the pen (requires knowing how to identify where the main gate of the pen is, then navigate towards it while surrounded by animals already inside the pen), then lock the animals inside (requires identifying the main gate and using the action close the gate, which is used only once throughout the complete trajectory).

Reasoning over these long-term dependencies in hierarchical tasks is one of the main challenges of end-to-end learning-based approaches [10]. Conversely, reactive policies such as the one required to navigate with certain boundaries and avoid obstacles have been learned directly from demonstration data or agent-generated trajectories [45, 3]. In this work, we use human knowledge of the tasks to decompose these complex tasks in multiple subtasks, which are either reactive policies learned from data or directly engineered, and a state-machine that selects the most appropriate one to be followed at every instant. Specifically, the subtask that performs task-specific navigation as, for example, searching for caves in the \textit{FindCave} task or the best spot to place a waterfall in the \textit{MakeWaterfall} task, is learned from the provided human demonstration dataset, while subtasks with little demonstration data available, such as throwing a snowball while inside the cave to signal the end of the episode (single data point in thousands of samples), are engineered in combination with the learned state classifier.

Once the complex tasks are decomposed into multiple small subtasks, we engineered a state-machine in combination with the learned state classifier to select the best subtask to be followed at every time step. Each of these engineered subtasks was implemented by a human designer who hard-coded a sequence of actions to be taken using the same interface available to the agent. In addition to these subtasks, the human designer also implemented a safety-critical subtask that allows the agent to escape a body of water whenever the state classifier detects
that the agent is swimming. Appendix B describes in detail the sequence of subtasks followed by the state-machine for each task.

4.4. Evaluation Methods

In this work we compared four different approaches to solve the four tasks proposed in the Minecraft competition:

- **Hybrid**: the main proposed agent in this work. It leverages the human demonstration data provided by the competition to learn how to navigate, together with the learned state classifier from additional human feedback to identify relevant states, coordinate by an engineered state-machine and subtasks module to solve specific parts of the tasks that were not able to be learned from data.
- **Engineered**: almost identical to the Hybrid agent described above, however, the navigation subtask that was learned from data is not replaced by an engineered module that randomly explores the environment.
- **Behavior Cloning**: end-to-end imitation learning agent that learns solely from the human demonstration data provided during the competition. This agent does not use any engineered knowledge, which includes the state classifier, state-machine, and odometry.
- **Human**: human demonstrated trajectories provided by the competition. This agent also does not use any engineered knowledge since it is fully controlled by a human player.

We set up a web application\(^6\), as seen in Appendix C, to collect human evaluations for each of the four baselines in a head-to-head comparison, similar to how the teams were evaluated during the official MineRL BASALT competition. In our case, each participant was asked to see two videos of different agents performing the same task then answer three questions:

1. Which agent best completed the task?
2. Which agent was the fastest completing the task?
3. Which agent had a more human-like behavior?

For each question, the participant was given three possible answers: “Agent 1”, “Agent 2”, or “None”. Task, agent type, and video of the agent performing the task were uniformly sampled at each time the participants were presented with the evaluation form. Our database had all four tasks (FindCave, MakeWaterfall, CreateVillageAnimalPen, and BuildVillageHouse), four agent types (Behavior Cloning, Engineered, Hybrid, and Human), and 10 videos of agent performance for each condition, for a total of 160 videos. All agent-generated videos were scaled from the original $64 \times 64$ image resolution returned by the environment to $512 \times 512$ image resolution in an attempt to make the videos more clear to the human evaluators. The videos for the "Human" agent type were randomly selected from the dataset provided by the MineRL BASALT competition and also scaled to $512 \times 512$ image resolution to match the agent-generated videos. All videos were generated and saved at 20 frames per second to match the sampling rate of the Minecraft simulator used by both agents and humans.

\(^6\)Custom MineRL BASALT evaluation webpage: https://kairosminerl.herokuapp.com/.
Table 1. Summary of the TrueSkill™[46] scores with mean and standard deviation computed from human evaluations separately for each performance metric and agent type averaged out over all tasks. Scores were computed after collecting 268 evaluations from 7 different human evaluators.

| Task                          | Performance Metric      | TrueSkill Rating |
|-------------------------------|-------------------------|------------------|
|                               | Best Performer          | Behavior Cloning | Engineered | Hybrid   | Human    |
| All Tasks Combined            |                         | 20.30 ± 1.81     | 24.21 ± 1.46 | 25.49 ± 1.40 | 32.56 ± 1.85 |
| Fastest Performer            |                         | 19.42 ± 1.94     | 26.92 ± 1.45 | 27.59 ± 1.38 | 28.36 ± 1.69  |
| More Human-like Behavior     |                         | 20.09 ± 2.04     | 26.02 ± 1.56 | 26.94 ± 1.57 | 36.41 ± 2.12  |

5. Results and Discussion

Each combination of condition (behavior cloning, engineered, hybrid, human) and performance metric (best performer, fastest performer, and the one that presented more human-like behavior) is treated as a separate participant of a one-versus-one competition where skill rating is computed using the TrueSkill™ Bayesian ranking system [46]. In this Bayesian ranking system, the skill of each participant is characterized by a Gaussian distribution with a mean value \( \mu \) representing the average skill of a participant and standard deviation \( \sigma \) representing the degree of uncertainty in the participant’s skill. There are three possible outcomes after each comparison: the first agent wins the comparison, the second agent the comparison, or there is a draw (human evaluator selects "None" when asked which participant performed better in a given metric). Given this outcome, the TrueSkill™ ranking system updates the belief distribution of each participant using Bayes’ Theorem [46]. A similar ranking system is also used in the official 2021 NeurIPS MineRL BASALT competition.

The final mean and standard deviation of TrueSkill™ scores computed for each performance metric and agent type, averaged out for all tasks are shown in Table 1. The current scores were computed after collecting 268 evaluations from 7 different human evaluators. Our main proposed “Hybrid” agent, which combines engineered and learned modules, outperforms both pure hand-designed (“Engineered”) and pure learned (“Behavior Cloning”) agents in the “Best Performer” category, achieving 5.3% and 25.6% higher mean skill rating when compared to the “Engineered” and “Behavior Cloning” baselines, respectively. However, when compared to the “Human” scores, our main proposed agent achieves 21.7% lower mean skill rating, illustrating that even our best approach is still not able to outperform a human player with respect to best performing the task. Interestingly, when looking at the “Fastest Performer” metric, our “Hybrid” agent outperforms both “Engineered” and “Behavior Cloning” baselines, respectively, it scores only 2.7% lower than the human players. As expected, in the “More Human-like Behavior” performance metric the “Human” baseline wins by a large margin, however, the “Hybrid” still
Table 2.
Summary of the TrueSkill\textsuperscript{TM}[46] scores with mean and standard deviation computed from human evaluations separately for each performance metric, agent type, and task. Scores were computed after collecting 268 evaluations from 7 different human evaluators.

| Task          | Performance Metric | Best Performer | Fastest Performer | More Human-like Behavior |
|---------------|--------------------|----------------|-------------------|--------------------------|
| FindCave      |                    | 24.32 ± 1.27   | 24.65 ± 1.27      | 21.53 ± 1.70             |
|               |                    | 24.29 ± 1.21   | 24.16 ± 1.21      | 26.61 ± 1.43             |
|               |                    | 25.14 ± 1.19   | 24.79 ± 1.19      | 28.25 ± 1.51             |
|               |                    | 32.90 ± 1.52   | 32.75 ± 1.54      | 38.95 ± 1.96             |
| MakeWaterfall |                    | 15.16 ± 2.10   | 14.67 ± 2.26      | 21.27 ± 1.98             |
|               |                    | 23.16 ± 1.60   | 28.95 ± 1.74      | 24.51 ± 1.52             |
|               |                    | 26.53 ± 1.39   | 28.88 ± 1.46      | 26.91 ± 1.35             |
|               |                    | 24.39 ± 1.62   | 18.85 ± 2.02      | 26.48 ± 1.61             |
| CreateVillage |                    | 21.87 ± 1.94   | 18.62 ± 2.27      | 21.54 ± 2.29             |
| AnimalPen     |                    | 23.56 ± 1.38   | 27.00 ± 1.32      | 25.53 ± 1.57             |
| BuildVillage  |                    | 26.49 ± 1.48   | 29.93 ± 1.50      | 27.99 ± 1.68             |
| House         |                    | 33.89 ± 1.73   | 28.59 ± 1.53      | 40.60 ± 2.44             |
|               |                    | 21.93 ± 1.92   | 19.75 ± 1.97      | 19.83 ± 1.92             |
|               |                    | 25.81 ± 1.66   | 27.58 ± 1.54      | 25.81 ± 1.66             |
|               |                    | 23.81 ± 1.55   | 26.76 ± 1.35      | 23.81 ± 1.55             |
|               |                    | 39.05 ± 2.53   | 33.24 ± 1.67      | 39.05 ± 2.53             |
|               |                    | 16.04 ± 2.19   | 27.42 ± 1.72      | 24.61 ± 1.72             |
|               |                    | 27.42 ± 1.72   | 39.61 ± 2.46      | 27.42 ± 1.72             |

outperforms all other baselines, including the “Behavior Cloning” agent, which is purely learned from human data. We attribute this to the fact that the pure learned agent did not make use of the safety-critical engineered subtask, which, for example, allowed the agent to escape a body of water, and was sometimes stuck in obstacles when navigating around the environment. Plots showing how the TrueSkill\textsuperscript{TM} scores evolved after each match (one-to-one comparison between different agent types) are shown in Appendix D.

Table 2 breaks down the results presented in Table 1 for each separate task. Similar to what was discussed for Table 1, not considering the “Human” baseline, the “Hybrid” approach outperforms both “Behavior Cloning” and “Engineered” baselines in terms of mean skill rating in 8 out of the 12 performance metrics, or in 66.6% of the comparisons. Similarly, hybrid intelligence approaches, which include both “Hybrid” and “Engineered” baselines, outperform
the pure learning “Behavior Cloning” approach in all 12 performance metrics when not taking into account the “Human” baseline. The “Hybrid” approach only outperforms the “Human” baseline in 4 out of the 12 performance metrics, or in 33.3% of the comparisons. Particularly for the MakeWaterfall task, the proposed hybrid approach outperforms human players for all performance metrics. The largest margin observed is for the “Fastest Performer” metric, where the hybrid approach scores 53.2% higher than the human players. The main reason for this largest margin is that human players normally take more time to find the best spot to place the waterfall and take the final picture to signal the end of the episode, as opposed to the engineered subtasks that rely only on signals from the state classifier and the engineered subtasks to directly move away from the waterfall and take the final picture. Plots showing all results for each individual pairwise comparison are shown in Appendix E.

In terms of qualitative results, we present complete video trajectories of the proposed hybrid agent solving each of the four tasks: FindCave, MakeWaterfall, CreateVillageAnimalPen, and BuildVillageHouse. Each video shows the image frames received by the agent (left panel) overlaid with the actions taken (top), the output of the state classifier (center), and the subtask currently being followed (bottom). The right panel shows the estimated odometry map overlaid with the location of the relevant states identified by the state classifier.

As seen in videos provided in the footnotes, when solving the FindCave task, the agent spawns in the plains biome and uses the learned navigation policy to search for caves while avoiding water, at the same it builds the map. Once the agent finds the cave, it throws the snowball to signal the end of the episode. In the MakeWaterfall task, the hybrid agent spawns in a mountainous area and uses the learned navigation policy to climb mountains, detect a good location to build a waterfall, then builds it and move to the picture location using engineered subtasks, then throws the snowball to signal the end of the episode. For the CreateVillageAnimalPen task, the agent uses the learned navigation policy and state classifier to search for an open location to build a pen, builds it using an engineered building subtask that clones action taken by the human demonstrators, uses the state classifier and odometry map to go to previously seen animal locations, then attempts to lure them back to the pen and throws the snowball to signal the end of the episode. Finally, when solving the BuildVillageHouse task, our hybrid agent spawns nearby a village and uses the learned navigation policy and state classifier to search for an open location to build a house, building it using an engineered building subtask that clones action taken by the human demonstrators, tours the house and throws the snowball to signal the end of the episode. Each of the described videos are shown in Appendix F as a sequence of frames.

6. Conclusions

In this paper, we presented the solution that won first place and was awarded the most human-like agent in the 2021 Neural Information Processing Systems (NeurIPS) MineRL BASALT

---

6Sample trajectory of hybrid agent solving the FindCave task: https://youtu.be/MR8q3Xre_XY.
7Sample trajectory of hybrid agent solving the MakeWaterfall task: https://youtu.be/eXp1urKXIPQ.
8Sample trajectory of hybrid agent solving the CreateVillageAnimalPen task: https://youtu.be/b8xDmXeEZmAE.
9Sample trajectory of hybrid agent solving the BuildVillageHouse task: https://youtu.be/_uKO-ZqBMWQ.
competition, “Learning from Human Feedback in Minecraft”. Our approach used the available human demonstration data and additional human feedback to train machine learning modules that were combined with engineered ones to solve hierarchical tasks in Minecraft.

The proposed method was compared to both end-to-end machine learning and pure engineered solutions by collecting human evaluations that judged agents in head-to-head matches to answer which agent best solved the task, which agent was the fastest, and which one had the most human-like behavior. These human evaluations were converted to a skill rating score for each question, similar to how players are ranked in multiplayer online games.

After collecting 268 human evaluations, we showed that hybrid intelligence approaches outperformed end-to-end machine learning approaches in all of the 12 performance metrics computed, even outperforming human players in 4 of them. Our ablation studies also showed that incorporating machine learning modules for navigation as opposed to engineering navigation policies led to the best results in 8 out of 12 performance metrics.

Overall, we showed that hybrid intelligence approaches are viable solutions to solve hierarchical tasks when the subcomponents of the task are understood by human experts and limited human feedback data is available.

Acknowledgments

Research was sponsored by the Army Research Laboratory and was accomplished partly under Cooperative Agreement Number W911NF-20-2-0114. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] M. Johnson, K. Hofmann, T. Hutton, D. Bignell, The malmo platform for artificial intelligence experimentation., in: IJCAI, Citeseer, 2016, pp. 4246–4247.
[2] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, et al., The MineRL competition on sample efficient reinforcement learning using human priors, NeurIPS Competition Track (2019).
[3] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, K. Zieba, ‘End to End Learning for Self-Driving Cars’, CoRR abs/1604.07316 (2016). arXiv:1604.07316.
[4] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, S. Levine, ‘Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation’, CoRR abs/1703.02018 (2017). arXiv:1703.02018.
[5] E. Kamar, ‘Directions in hybrid intelligence: Complementing ai systems with human intelligence.’, in: IJCAI, 2016, pp. 4070–4073.
[6] D. Dellermann, P. Ebel, M. Söllner, J. M. Leimeister, ‘Hybrid intelligence’, Business & Information Systems Engineering 61 (2019) 637–643.
[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. A. Riedmiller, Playing Atari with Deep Reinforcement Learning, CoRR abs/1312.5602 (2013). arXiv:1312.5602.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, K. Kavukcuoglu, Asynchronous Methods for Deep Reinforcement Learning, in: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML’16, JMLR.org, 2016, p. 1928–1937. doi:10.5555/3045390.3045594.

[9] P. Dayan, G. E. Hinton, Feudal reinforcement learning, in: S. Hanson, J. Cowan, C. Giles (Eds.), Advances in Neural Information Processing Systems, volume 5, Morgan-Kaufmann, 1993. URL: https://proceedings.neurips.cc/paper/1992/file/d14220ee6e6aee73c49038385428ec4c-Paper.pdf.

[10] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, K. Kavukcuoglu, Feudal networks for hierarchical reinforcement learning, in: International Conference on Machine Learning, PMLR, 2017, pp. 3540–3549.

[11] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, K. Kavukcuoglu, IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures, volume 80 of Proceedings of Machine Learning Research, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, pp. 1407–1416.

[12] N. Rudin, D. Hoeller, P. Reist, M. Hutter, Learning to walk in minutes using massively parallel deep reinforcement learning, arXiv preprint arXiv:2109.11978 (2021).

[13] N. R. Waytowich, V. G. Goecks, V. J. Lawhern, Cycle-of-Learning for Autonomous Systems from Human Interaction, CoRR abs/1808.09572 (2018). arXiv:1808.09572.

[14] V. G. Goecks, Human-in-the-loop methods for data-driven and reinforcement learning systems, arXiv preprint arXiv:2008.13221 (2020).

[15] D. A. Pomerleau, ALVINN: An Autonomous Land Vehicle in a Neural Network, in: Advances in Neural Information Processing Systems 1, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989, p. 305–313. doi:10.5555/89851.89891.

[16] B. D. Argall, S. Chernova, M. Veloso, B. Browning, A Survey of Robot Learning from Demonstration, Robot. Auton. Syst. 57 (2009) 469–483. doi:10.1016/j.robot.2008.10.024.

[17] D. Brown, W. Goo, P. Nagarajan, S. Niekum, Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations, in: International conference on machine learning, PMLR, 2019, pp. 783–792.

[18] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, Learning Manipulation Trajectories Using Recurrent Neural Networks, CoRR abs/1603.03833 (2016). arXiv:1603.03833.

[19] B. Eysenbach, S. Levine, R. Salakhutdinov, Replacing rewards with examples: Example-based policy search via recursive classification, arXiv preprint arXiv:2103.12656 (2021).

[20] B. Akgun, M. Cakmak, K. Jiang, A. L. Thomaz, Keyframe-based Learning from Demonstration, International Journal of Social Robotics 4 (2012) 343–355. doi:10.1007/s12369-012-0160-0.

[21] B. Akgun, M. Cakmak, J. W. Yoo, A. L. Thomaz, Trajectories and Keyframes for Kinesthetic Teaching: A Human-Robot Interaction Perspective, in: 2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 2012, pp. 391–398. doi:10.1145/2157689.
[22] W. Saunders, G. Sastry, A. Stuhlmüller, O. Evans, Trial without Error: Towards Safe Reinforcement Learning via Human Intervention, in: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS ’18, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2018, p. 2067–2069. doi:10.5555/3237383.3238074.

[23] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, N. R. Waytowich, Efficiently Combining Human Demonstrations and Interventions for Safe Training of Autonomous Systems in Real-Time, in: Proceedings of the AAAI Conference on Artificial Intelligence, AAAI Press, 2019, pp. 2462–2470. doi:10.1609/aaai.v33i01.33012462.

[24] S. Ross, G. Gordon, D. Bagnell, A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, volume 15 of Proceedings of Machine Learning Research, JMLR Workshop and Conference Proceedings, Fort Lauderdale, FL, USA, 2011, pp. 627–635.

[25] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, M. Hebert, Learning Monocular Reactive UAV Control in Cluttered Natural Environments, in: 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 1765–1772. doi:10.1109/ICRA.2013.6630809.

[26] A. Ng, S. Russell, Algorithms for Inverse Reinforcement Learning, Proceedings of the Seventeenth International Conference on Machine Learning 0 (2000) 663–670. doi:10.2460/ajvr.67.2.323. arXiv:arXiv:1011.1669v3.

[27] C. Finn, S. Levine, P. Abbeel, Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization, in: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML’16, JMLR.org, 2016, p. 49–58. doi:10.5555/3045390.3045397.

[28] W. B. Knox, P. Stone, Interactively Shaping Agents via Human Reinforcement: The TAMER Framework, in: Proceedings of the Fifth International Conference on Knowledge Capture, K-CAP ’09, Association for Computing Machinery, New York, NY, USA, 2009, p. 9–16. doi:10.1145/1597735.1597738.

[29] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor, M. L. Littman, Interactive Learning from Policy-Dependent Human Feedback, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML’17, JMLR.org, 2017, p. 2285–2294, doi:10.5555/3305890.3305917.

[30] G. Warnell, N. R. Waytowich, V. Lawhern, P. Stone, Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces, in: S. A. McIlraith, K. Q. Weinberger (Eds.), Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), AAAI Press, 2018, pp. 1545–1554.

[31] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, D. Amodei, Deep Reinforcement Learning from Human Preferences, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 4302–4310. doi:10.5555/3294996.3295184.

[32] L. Zhou, K. Small, Inverse reinforcement learning with natural language goals, ArXiv abs (2008).

[33] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine, Learning complex dexterous manipulation with deep reinforcement learning and demonstrations,
[34] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, N. R. Waytowich, Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments, in: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS ’20, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2020, p. 465–473. doi:10.5555/3398761.3398819.

[35] S. Reddy, A. D. Dragan, S. Levine, Sql: Imitation learning via reinforcement learning with sparse rewards, arXiv preprint arXiv:1905.11108 (2019).

[36] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, R. Salakhutdinov, Minerl: A large-scale dataset of minecraft demonstrations, arXiv preprint arXiv:1907.13440 (2019).

[37] A. Amiranashvili, N. Dorka, W. Burgard, V. Koltun, T. Brox, Scaling imitation learning in minecraft, arXiv preprint arXiv:2007.02701 (2020).

[38] Z. Hao, A. Mallya, S. Belongie, M.-Y. Liu, Gancraft: Unsupervised 3d neural rendering of minecraft worlds, arXiv preprint arXiv:2104.07659 (2021).

[39] R. Shah, C. Wild, S. H. Wang, N. Alex, B. Houghton, W. Guss, S. Mohanty, A. Kanervisto, S. Milani, N. Topin, P. Abbeel, S. Russell, A. Dragan, NeurIPS 2021 competition proposal: The MineRL BASALT competition on learning from human feedback, NeurIPS Competition Track (2021).

[40] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, R. Salakhutdinov, MineRL: A large-scale dataset of Minecraft demonstrations, Twenty-Eighth International Joint Conference on Artificial Intelligence (2019). URL: http://minerl.io.

[41] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous Control with Deep Reinforcement Learning, CoRR abs/1509.02971 (2016).

[42] J. Ho, S. Ermon, Generative adversarial imitation learning, Advances in neural information processing systems 29 (2016) 4565–4573.

[43] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779–788. doi:10.1109/CVPR.2016.91.

[44] G. Warnell, N. R. Waytowich, V. Lawhern, P. Stone, Deep TAMER: interactive agent shaping in high-dimensional state spaces, CoRR abs/1709.10163 (2017). URL: http://arxiv.org/abs/1709.10163. arXiv:1709.10163.

[45] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al., A machine learning approach to visual perception of forest trails for mobile robots, IEEE Robotics and Automation Letters 1 (2015) 661–667.

[46] R. Herbrich, T. Minka, T. Graepel, Trueskill™: A bayesian skill rating system, in: Proceedings of the 19th international conference on neural information processing systems, 2006, pp. 569–576.
Appendix

A. State Classifier Labeling GUI

The labeling process of relevant states to the state-machine uses both mouse clicks and keyboard presses and takes place in a custom GUI, as seen in Figure 4. On the top left of the GUI, users can double-check which dataset and frame number they are labeling. Below that, the GUI displays the RGB frame to be labeled (center left) and the options for labels (center right, same labels for all tasks). To label a frame, the user can simply press the keyboard key corresponding to the desired label (shown in brackets, for example, [c] for has\_cave), or click in the white circles in front of the label (which will then turn green, displaying that the label was selected). Frames will automatically advance when a key is pressed. If the users only use the mouse to select the labels, they still need to press the keyboard key to advance to the next
B. State-Machine Definition for each Task

The sequence of subtasks used by the state-machine for each task is defined as follows:

- **FindCave**:
  1. Use navigation policy to traverse the environment and search for caves;
  2. If the state classifier detects the agent is inside a cave, throw a snowball to signal that the task was completed (end of episode).

- **MakeWaterfall**:
  1. Use navigation policy to traverse the environment and search for a place in the mountains to make a waterfall;
  2. If the state classifier detects the agent is at the top of a mountain, build additional blocks to give additional height to the waterfall;
  3. Once additional blocks are built, look down and place the waterfall by equipping and using the bucket item filled with water;
  4. After waterfall is built, keep moving forward to move away from the waterfall;
  5. Once agent has moved away from the waterfall, turn around and throw a snowball to signal that a picture was taken and the task was completed (end of episode).

- **CreateVillageAnimalPen**:
  1. Use navigation policy to traverse the environment and search for a place to build a pen;
  2. If the state classifier detects an open-space, build the pen. The subtask to build the pen directly repeats the actions taken by a human while building the pen, as observed in provided demonstration dataset;
  3. Once the pen is built, use the estimated odometry map to navigate to the closest animal location. If no animals were seen before, use navigation policy to traverse the environment and search for animals;
  4. At the closest animal location, equip food to attract attention of the animals and lure them;
  5. Using the estimated odometry map, move back to where the pen was built while animals are following the agent;
  6. Once inside the pen together with the animals, move away from pen, turn around and throw a snowball to signal that the task was completed (end of episode).

- **BuildVillageHouse**:
  1. Use navigation policy to traverse the environment and search for a place to build a house;
  2. If the state classifier detects an open-space, build the house. The subtask to build the house directly repeats the actions taken by a human while building the house, as observed in provided demonstration dataset;
  3. Once the house is built, move away from it, turn around and throw a snowball to signal that the task was completed (end of episode).
C. Human Evaluation Interface

**Task: Create Village Animal Pen**

After spawning in a village, the agent should build an animal pen containing two of the same kind of animal next to one of the houses in a village.

**Figure 5.** Web evaluation form used to collect additional human evaluation data to evaluate the multiple agent conditions presented in paper.

Figure 5 shows a sample of the web evaluation form available at https://kairosminerl.herokuapp.com/ that was used to collect human evaluations for each of the four baselines in a head-to-head comparison, similar to how the teams were evaluated during the official MineRL BASALT competition. Each participant was asked to see two videos of different agents performing the same task then answer three questions with respect to the agent’s performance.
D. TrueSkill™ Score per Match

Figures 6, 7, 8, and 9 show the evolution of the TrueSkill™ score after each match (one-to-one comparison between different agent types) for each performance metric when the agents are solving the FindCave, MakeWaterfall, CreateVillageAnimalPen, and BuildVillageHouse tasks, respectively. The bold line represents the mean estimated skill rating and shaded area the standard deviation of the estimation.

![Figure 6](image1.png)  
(a) Best Performer  
(b) Fastest Performer  
(c) More Human-like Behavior

**Figure 6.** TrueSkill™[46] score computed from human evaluations separately for each performance metric and for each agent type performing the FindCave task.

![Figure 7](image2.png)  
(a) Best Performer  
(b) Fastest Performer  
(c) More Human-like Behavior

**Figure 7.** TrueSkill™[46] score computed from human evaluations separately for each performance metric and for each agent type performing the MakeWaterfall task.

E. Pairwise Comparison per Performance Metric and Task

Figures 10, 11, 12, and 13 show barplots with the individual pairwise comparisons compiled from the human evaluations for the FindCave, MakeWaterfall, CreateVillageAnimalPen, and BuildVillageHouse tasks, respectively. Each bar represents the percentage of time a given condition was select as a winner for each performance metric by the human evaluator when they were presented with a video of the agent performance solving the task for each analysed condition. For example, when analysing Figure 10(a), we can see that when the human evaluator was presented with a video of the "Behavior Cloning" agent and another from the "Engineered" agent, they
selected the “Engineered” agent as the best performer 33.3% and the “Behavior Cloning” agent 22.2% of time. The remaining accounts for the “None” answer to the questionnaire selected when none of the agents were judged to have solved the task.

When directly comparing the “Behavior Cloning” baseline to the main proposed “Hybrid” method for all tasks, as shown in Figures 10 to 13 (c) plots, we observe that the proposed hybrid intelligence agent always match or outperforms the pure learned baseline. This is similar to the case we compare the “Engineered” agent to the “Hybrid”, where the proposed hybrid method outperforms the fully engineered approach in all tasks with the exception of BuildVillageHouse task, as seen in Figure 13. The human players always outperform the hybrid agent with exception to the MakeWaterfall task, where the “Hybrid” agent is judge to better solve the task 70% of time, to solve it faster 90% of the time, and even present a more human-like behavior 60% of the time. This can be attributed to the fact that the human players were not always able or willing to solve the task as described in the prompt.

F. Samples of Hybrid Agent Solving the Tasks

In terms of qualitative results, Figures 14, 15, 16, and 17 show a sample episode illustrated by a sequence of frames of our hybrid agent solving the FindCave, MakeWaterfall, CreateVillageAni-
Pairwise comparison displaying the normalized scores computed from human evaluations separately for each performance metric on all possible head-to-head comparisons for all agent types performing the FindCave task.

malPen, and BuildVillageHouse tasks, respectively. Each figure shows the image frames received by the agent (left panel) overlaid with the actions taken (top), output of the state classifier (center), and the subtask currently being followed (bottom). The right panel shows the estimated odometry map overlaid with the location of the relevant states identified by the state classifier. Link to the videos are provided in the figure captions.
Figure 11: Pairwise comparison displaying the normalized scores computed from human evaluations separately for each performance metric on all possible head-to-head comparisons for all agent type performing the MakeWaterfall task.

Figure 12: Pairwise comparison displaying the normalized scores computed from human evaluations separately for each performance metric on all possible head-to-head comparisons for all agent type performing the CreateVillageAnimalPen task.
Figure 13.: Pairwise comparison displaying the normalized scores computed from human evaluations separately for each performance metric on all possible head-to-head comparisons for all agent type performing the *BuildVillageHouse* task.

Figure 14.: Sequence of frames of our hybrid agent solving the *FindCave* task (complete video available at https://youtu.be/MR8q3Xre_XY).
Figure 15.: Sequence of frames of our hybrid agent solving the MakeWaterfall task (complete video available at https://youtu.be/eXp1urKXIPQ).

Figure 16.: Sequence of frames of our hybrid agent solving the CreateVillageAnimalPen task (complete video available at https://youtu.be/b8xDMxEZmAE).
Figure 17: Sequence of frames of our hybrid agent solving the *BuildVillageHouse* task (complete video available at https://youtu.be/_uKO-ZqBMWQ).