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

Vinicius G. Goecks[1], Nicholas Waytowich[1,2], David Watkins-Valls[2] and Bharat Prakash[3]

[1]DEVCOM Army Research Laboratory, Aberdeen Proving Ground, Maryland, USA
[2]Columbia University, New York City, New York, USA
[3]University 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, combined with an estimated odometry map, become a powerful state-machine designed to utilize human knowledge in a natural hierarchical paradigm. 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 2021 Neural Information Processing Systems (NeurIPS) MineRL Benchmark for Agents that Solve Almost-Lifelike Tasks (BASALT) competition[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,

In A. Martin, K. Hinkelmann, H.-G. Fill, A. Gerber, D. Lenat, R. Stolle, F. van Harmelen (Eds.), Proceedings of the AAAI 2022 Spring Symposium on Machine Learning and Knowledge Engineering for Hybrid Intelligence (AAAI-MAKE 2022), Stanford University, Palo Alto, California, USA, March 21–23, 2022.

vinicius.goecks@gmail.com (V. G. Goecks); nicholas.r.waytowich.civ@mail.mil (N. Waytowich); davidwatkins@cs.columbia.edu (D. Watkins-Valls); bhp1@umbc.edu (B. Prakash)
https://vggoecks.com/ (V. G. Goecks); https://davidwatkinsvalls.com/ (D. Watkins-Valls)

CEUR Workshop Proceedings (CEUR-WS.org)

[1]Official competition webpage: https://www.aicrowd.com/challenges/neurips-2021-minerl-basalt-competition.

and defining one from scratch can be quite challenging. Therefore, 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 no reward function.

The MineRL BASALT competition tasks do not contain any reward functions for the four tasks. 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]. When learning from human feedback, this 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. Subtask selection is controlled by an engineered state-machine, which relies on estimated agent odometry and the outputs of a learned state classifier. We also use the competition-provided human demonstration dataset to train a navigation policy subtask via imitation learning to replicate how humans traverse the environment.

In this paper, we give a detailed overview of our approach and perform an ablation study to investigate how well our hybrid intelligence approach works compared to 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 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. For example, deep reinforcement learning algorithms learning directly from raw pixels [7, 8] and algorithms that automatically decompose tasks in hierarchies with different time scales [9, 10]. This often includes massive parallelization and distributed computation [11, 12] to fulfill the data requirement of these algorithms. Other works train robotic agents to navigate through their environment using RGB-D (visible spectrum imagery, plus depth) information to determine optimal discrete steps to navigate to a visual goal [13]. 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 [14, 15]. A learning agent can be trained based on human demonstrations of a task [16, 17, 3]. Agents can learn from suboptimal demonstrations [18], end goals [19], or directly from successful examples instead

of a reward function [20]. Human operators can augment the human demonstrations with online interventions [21, 22, 23, 24] or offline labeling [25, 26] while still maintaining success. Agents can learn the reward or cost function used by the demonstrator [27, 28] through sparse interactions in the form of evaluative feedback [29, 30, 31] or human preferences given a pair of trajectories [32]. Additionally, agents can learn from natural language-defined goals [33]. Finally, agents can learn from combining human data with reinforcement learning [34, 35, 36]. However, 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 augment 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, giving the task designer flexibility to define tasks with virtually any level of complexity. The release of Malmo [1], a platform that enabled AI 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.

The Minecraft environment also served as a platform to collect large human demonstration datasets such as the *MineRL-v0* dataset [37] and experiment with large scale imitation learning algorithms [38] as a world generator for realistic terrain rendering [39], a sample-efficient reinforcement learning competition environment 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 [40].

3. Problem Formulation

The 2021 NeurIPS MineRL BASALT competition, "Learning from Human Feedback in Minecraft", challenged participants to come up with creative solutions to solve four different tasks in Minecraft [40] using the "MineRL: Towards AI in Minecraft"[2] simulator [37]. 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 competition[3] 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.

[2]MineRL webpage: https://minerl.io/.
[3]MineRL 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 of 40 to 80 human demonstrations for each task, not all completing the task, and the starter codebase to train a behavior cloning baseline. Additionally, the training time for all four tasks together was limited to four days and participants were allowed to collect up to 10 hours of additional human-in-the-loop feedback.

4. Methods

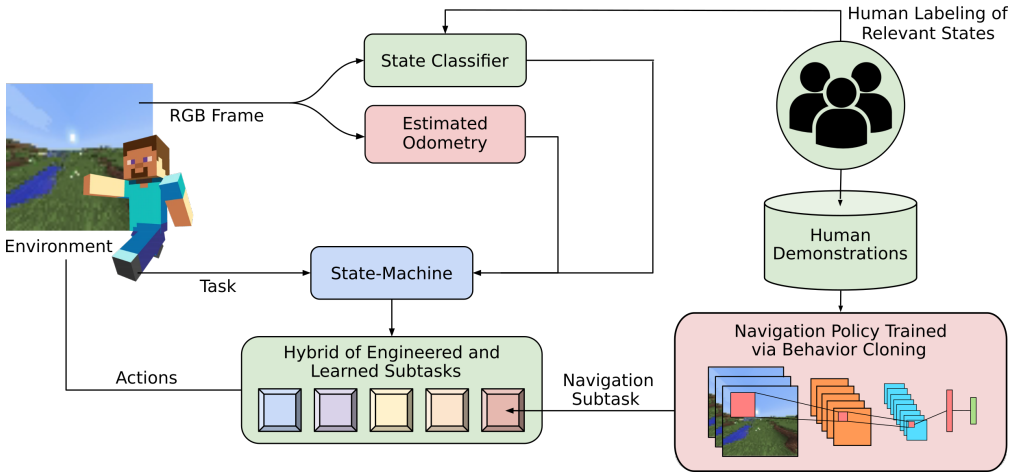


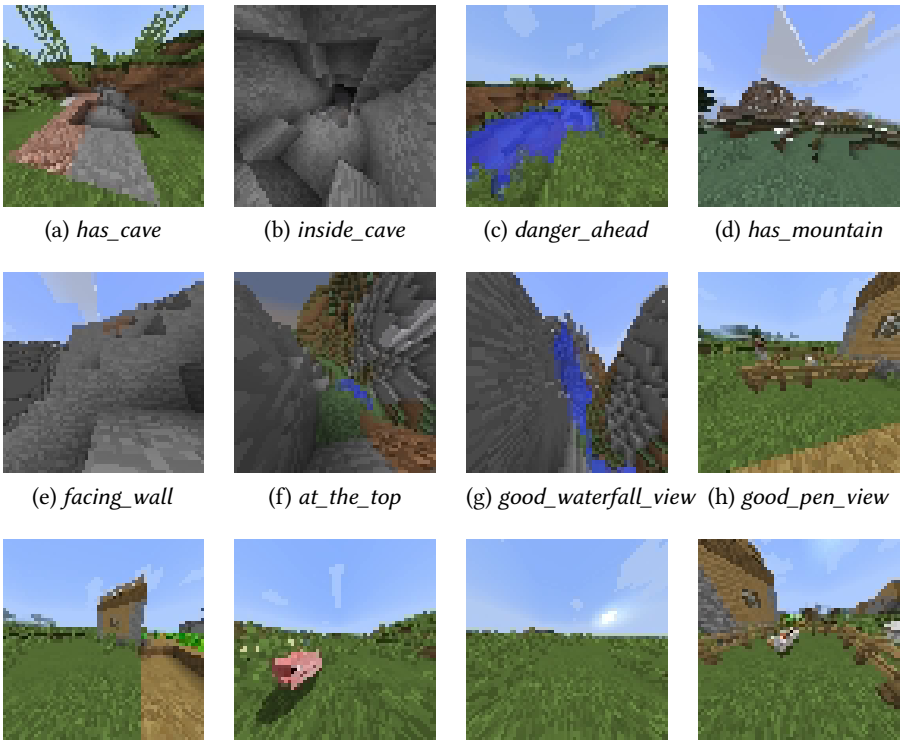
Figure 1: Diagram illustrating our approach. Using data from the available human demonstration dataset, humans provide additional binary labels to image frames to be used to train a state classifier that can detect relevant features in the environment such as caves and mountains. The available human 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 given by the competition organizers 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, because 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 controls which one should be executed at every time-step; second, we engineered solutions for the more challenging 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 and enables the execution of the more complex engineered subtasks.

4.1. State Classification



(a) *has_cave* (b) *inside_cave* (c) *danger_ahead* (d) *has_mountain*

(e) *facing_wall* (f) *at_the_top* (g) *good_waterfall_view* (h) *good_pen_view*

(i) *good_house_view* (j) *has_animals* (k) *has_open_space* (l) *animals_inside_pen*

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.

Our approach relies on a state machine that changes the high-level goals depending on the

task to be solved. Without having information about the environment's voxel data, we opted to use the visual RGB information from the simulator to determine the agent's current state. 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 *You Only Look Once* (YOLO) [43]. Multiple labels can be present on the same image as there were cases with multiple objects or scenes of interest at the same time in the field of view of the agent. These labels are used by the state-machine to decide which subtask should be followed at any time-step.

There are 13 possible labels for an 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 (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 see water 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 (flat area to build a small house or pen) (7.33 % of the labels).
- *animals_inside_pen*: agent is inside the pen after luring all animals and has them in view (0.81 % of the labels).

The possible labels were defined by a human designer with knowledge of the relevant states to be identified and given to the state-machine to solve all tasks. These labels were also designed to be relevant to all tasks to ease data collection and labelling efforts. For example, the "has_open_space" label identifies flat areas that are ideal to build pens or houses for both *CreateVillageAnimalPen* and *BuildVillageHouse* tasks. Unknown and other non-relevant states were attached the label "*none*" to indicate that no important states were in view.

To train this system, we labeled $81,888$ images using a custom graphical user interface (GUI), as showed in Appendix A. Once the data was labeled, 80% of images were used for training, 10% were used for validation, and 10% for testing. The model is a convolutional classifier with a $64 \times 64 \times 3$ input and 13×1 output. Our autoencoder is modeled after the Deep TAMER (Training Agents Manually via Evaluative Reinforcement) [31] model. The problem of training with an uneven number of labels for each class was mitigated by implementing a weighted sampling scheme that sampled more often classes with lower representation with probability:

$$P(x_i) = 1 - \frac{N_i}{M}, \tag{1}$$

where $P(x_i)$ is the probability of sampling class i that contains N_i number of labels out of the total M labels for all classes.

4.2. Estimated Odometry

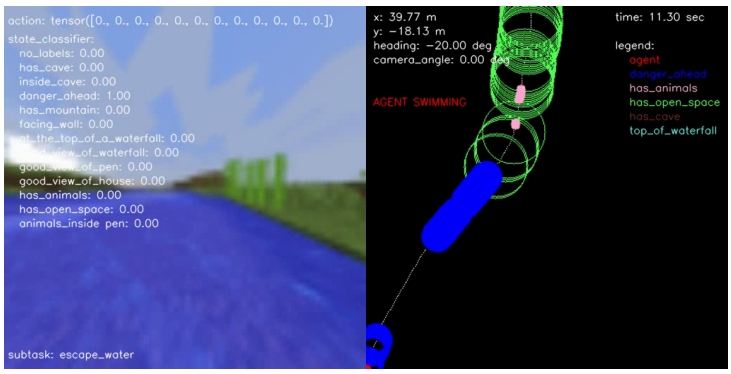


Figure 3: Example of odometry map (right frame) generated in real time from the agent's actions as it traverses the environment. Besides the agent's pose, the classified states from image data (left frame) also have their locations tagged in the map to be used for specific subtasks. For example, when the agent finishes building the pen, it uses the location of previously seen animals to attempt to navigate and lure them to the pen.

Some of the engineered subtasks required basic localization of the agent and relevant states of the environment. For example, the agent needs to know the location of previously seen animals to guide them to a pen in the *CreateVillageAnimalPen* task. However, under the rules of the competition, we were not allowed to use any additional information from the simulator besides the current view of the agent and the player's inventory. Which means there is no information about the ground truth location of the agent, camera pose, or any explicit terrain information available.

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 simulator. It is known that the simulator runs at 20 frames per second, which means there is a 0.05 second interval between each frame. According to the Minecraft Wiki[4], 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. If we assume the agent is operating in a flat world, starting at position $(0, 0)$ in map coordinates facing north, when the agent executes a move forward action its position is moved 0.216 meters north to position $(0, 0.216)$. The agent does not have acceleration in MineRL and their velocity is immediately updated upon keypress. 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 θ is simply updated by accumulating the horizontal camera angles commanded by the agent. More generally, this odometric estimation assumes the agent follows point-mass kinematics:

$$\dot{x} = V cos(\theta)$$

[4]Minecraft Wiki - Walking: https://minecraft.fandom.com/wiki/Walking.

$$\dot{y} = V sin(\theta),$$

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

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 so they can 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 a 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 in solving the proposed four tasks is that most required the agent to have certain levels of perception capabilities, memory, and reasoning over long-term dependencies in a hierarchical manner. For example, the *CreateVillageAnimalPen* task required the agent to first build a pen nearby an existing village, which requires identifying what a village is, then indicating a good location to build a pen such as a flat terrain. Once the pen was built, the agent had to search for at least two of the same animal type in the nearby vicinity using 64×64 resolution images as input. Return animals to the pen required coordination to combine different blocks and place them adjacently to each other in a closed shape. After the animals were found, the agent had to lure them with the specific food type they eat, walk them back to the pen the agent initially built, leave the pen, lock the animals inside, then take a picture of the pen with the animals inside.

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 [44, 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 time-step. The subtask that performs task-specific navigation is learned from the provided human demonstration dataset. For example, searching for the best spot to place a waterfall in the *MakeWaterfall* task requires navigation. Subtasks with little demonstration data available are engineered in combination with the learned state classifier. Throwing a snowball while inside the cave to signal the end of the episode can be engineered using human demonstration data.

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 allowing 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 substasks 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, that combines both learned and engineered modules. The learned modules are the navigation subtask policy (learns how to navigate using the human demonstration data provided by the competition) and the state classifier (learns how to identify relevant states using additional human-labeled data). The engineered modules are the multiple subtasks, hand-designed to solve subtasks that were not able to be learned from data. These engineered modules are the estimated odometry and the state-machine, which uses the output of the state classifier and engineered task structure to select which subtask should be followed at each time-step.
- **Engineered:** almost identical to the Hybrid agent described above, however, the navigation subtask policy that was learned from human demonstrations is now replaced by a hand-designed module that randomly selects movement and camera commands to explore 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 other learned or engineered module, which includes the state classifier, the estimated odometry, and the state-machine.
- **Human:** human-generated trajectories provided by the competition. They are neither guaranteed to solve the task nor solve it optimally because they depend on the level of expertise of each human controlling the agent.

To collect human evaluations for each of the four baselines in a head-to-head comparison we set up a web application[5] similar to how the teams were evaluated during the official MineRL BASALT competition, as seen in Appendix C. In our case, each participant was asked to see two videos of different agents performing the same task then to 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 participants were given three possible answers: “Agent 1”, “Agent 2”, or “None”.

Our database had videos of all four types of agents (Behavior Cloning, Engineered, Hybrid, and Human) performing all four tasks (FindCave, MakeWaterfall, CreateVillageAnimalPen, and BuildVillageHouse). There were 10 videos of each agent type solving all four tasks, for a total of 160 videos in the database. Task, agent type, and videos were uniformly sampled from the database at each time a new evaluation form was generated and presented to the human evaluator.

We collected a total of 268 evaluation forms (pairwise comparison where a human evaluator judged which agent was the best, fastest, and more human-like performing the tasks) from 7 different human evaluators.

[5]Custom MineRL BASALT evaluation webpage: https://kairosminerl.herokuapp.com/.

Table 1
Summary of the *TrueSkill*™ [45] 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			
		Behavior Cloning	Engineered	Hybrid	Human
All Tasks Combined	Best Performer	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

All agent-generated videos were scaled from the original 64×64 image resolution returned by the environment to 512×512 image resolution in an attempt to make the videos clearer for the human evaluators. The videos of the "Human" agent type were randomly selected from the video demonstrations provided by the MineRL BASALT competition and scaled to 512×512 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.

5. Results and Discussion

Each combination of condition (behavior cloning, engineered, hybrid, human) and performance metric (best performer, fastest performer, most human-like performer) is treated as a separate participant of a one-versus-one competition where skill rating is computed using the *TrueSkill*™[6] Bayesian ranking system [45]. In this Bayesian ranking system, the skill of each participant is characterized by a Gaussian distribution with a mean value μ, representing the average skill of a participant and standard deviation σ 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 [45], similar to how scores were computed in the official 2021 NeurIPS MineRL BASALT competition. We used the open-source *TrueSkill* Python package[7].

The final mean and standard deviation of the *TrueSkill*™ scores computed for each performance metric and agent type are shown in Table 1. The 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")

[6] Microsoft's *TrueSkill*™ Ranking System: https://www.microsoft.com/en-us/research/project/trueskill-ranking-system

[7] *TrueSkill* Python package: https://github.com/sublee/trueskill and https://trueskill.org/.

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.

When looking at the "Fastest Performer" metric, our "Hybrid" agent outperforms both "Engineered" and "Behavior Cloning" baselines, respectively, scoring 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 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 allowed the agent to escape bodies of water and other obstacles around the environment. Plots showing how the *TrueSkill*™ 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, excluding 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, 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; the hybrid approach scores 53.2% higher than the human players. This large margin comes from human players taking more time to find the best spot to place the waterfall and signal the end of the episode when compared to the engineered subtasks. Plots showing all results for each individual pairwise comparison are shown in Appendix E.

We now consider qualtitative evaluation of our agents. When solving the *FindCave* task[8], the agent spawns in the plains biome and uses the learned navigation policy to search for caves while avoiding water while simultaneously building the map of its environment. Once the agent finds the cave, it throws the snowball to signal the end of the episode. In the *MakeWaterfall* task[9], the hybrid agent spawns in a mountainous area, uses the learned navigation policy to climb the mountains, detects a good location to build the waterfall, builds it, then moves to the picture location using engineered subtasks, and throws the snowball to signal the end of the episode. For the *CreateVillageAnimalPen* task[10], the agent uses the learned navigation policy and the state classifier to search for an open location to build a pen, builds the pen using an engineered building subtask that repeats the actions taken by the human demonstrators, uses the state classifier and odometry map to go to previously seen animal locations, and then attempts to lure them back to the pen and throws the snowball to signal the end of the episode.

[8]Sample trajectory of hybrid agent solving the *FindCave* task: https://youtu.be/MR8q3Xre_XY.

[9]Sample trajectory of hybrid agent solving the *MakeWaterfall* task: https://youtu.be/eXp1urKXIPQ.

[10]Sample trajectory of hybrid agent solving the *CreateVillageAnimalPen* task: https://youtu.be/b8xDMxEZmAE.

Table 2
Summary of the *TrueSkill*™ [45] 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	TrueSkill Rating			
		Behavior Cloning	**Engineered**	**Hybrid**	**Human**
FindCave	Best Performer	24.32 ± 1.27	24.29 ± 1.21	25.14 ± 1.19	32.90 ± 1.52
	Fastest Performer	24.65 ± 1.27	24.16 ± 1.21	24.79 ± 1.19	32.75 ± 1.54
	More Human-like Behavior	21.53 ± 1.70	26.61 ± 1.43	28.25 ± 1.51	38.95 ± 1.96
MakeWaterfall	Best Performer	15.16 ± 2.10	23.16 ± 1.60	26.53 ± 1.39	24.39 ± 1.62
	Fastest Performer	14.67 ± 2.26	28.95 ± 1.74	28.88 ± 1.46	18.85 ± 2.02
	More Human-like Behavior	21.27 ± 1.98	24.51 ± 1.52	26.91 ± 1.35	26.48 ± 1.61
CreateVillage AnimalPen	Best Performer	21.87 ± 1.94	23.56 ± 1.38	26.49 ± 1.48	33.89 ± 1.73
	Fastest Performer	18.62 ± 2.27	27.00 ± 1.32	29.93 ± 1.50	28.59 ± 1.53
	More Human-like Behavior	21.54 ± 2.29	25.53 ± 1.57	27.99 ± 1.68	40.60 ± 2.44
BuildVillage House	Best Performer	19.83 ± 1.92	25.81 ± 1.66	23.81 ± 1.55	39.05 ± 2.53
	Fastest Performer	19.75 ± 1.97	27.58 ± 1.54	26.76 ± 1.35	33.24 ± 1.67
	More Human-like Behavior	16.04 ± 2.19	27.42 ± 1.72	24.61 ± 1.72	39.61 ± 2.46

Finally, when solving the *BuildVillageHouse* task[11], our hybrid agent spawns nearby a village and uses the learned navigation policy and the state classifier to search for an open location to build a house, builds a house using an engineered building subtask that repeats the actions taken by the human demonstrators, tours the house, and throws the snowball to signal the end of the episode. Each of the described subtasks 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 NeurIPS MineRL BASALT competition, "Learning from Human Feedback

[11]Sample trajectory of hybrid agent solving the *BuildVillageHouse* task: https://youtu.be/_uKO-ZqBMWQ.

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 12 performance metrics computed, even outperforming human players in 4 of them. Our results also showed that incorporating machine learning modules for navigation as opposed to engineering navigation policies led to higher scores in 8 out of 12 performance metrics.

Overall, we demonstrated 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: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16, AAAI Press, 2016, p. 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, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 2146–2153.

[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/d14220ee66aeec73c49038385428ec4c-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] D. Watkins-Valls, J. Xu, N. Waytowich, P. Allen, Learning your way without map or compass: Panoramic target driven visual navigation, 2020, pp. 5816–5823. doi:10.1109/IROS45743.2020.9341511.
[14] 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.
[15] V. G. Goecks, Human-in-the-loop methods for data-driven and reinforcement learning systems, arXiv preprint arXiv:2008.13221 (2020).
[16] 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.
[17] 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.
[18] 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.
[19] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, Learning manipulation trajectories using recurrent neural networks, CoRR abs/1603.03833 (2016). arXiv:1603.03833.
[20] B. Eysenbach, S. Levine, R. Salakhutdinov, Replacing rewards with examples: Example-based policy search via recursive classification, arXiv preprint arXiv:2103.12656 (2021).
[21] B. Akgun, M. Cakmak, K. Jiang, A. L. Thomaz, Keyframe-based learning from demon-

stration, International Journal of Social Robotics 4 (2012) 343–355. doi:10.1007/s12369-012-0160-0.

[22] 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:10.1145/2157689.2157815.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] A. Y. Ng, S. J. Russell, Algorithms for inverse reinforcement learning, in: Proceedings of the Seventeenth International Conference on Machine Learning, ICML '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, p. 663–670.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.
[33] L. Zhou, K. Small, Inverse reinforcement learning with natural language goals, Proceedings of the AAAI Conference on Artificial Intelligence 35 (2021) 11116–11124. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17326.
[34] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine, Learning complex dexterous manipulation with deep reinforcement learning and demonstrations, in: Proceedings of Robotics: Science and Systems (RSS), 2018.
[35] 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.
[36] S. Reddy, A. D. Dragan, S. Levine, SQIL: imitation learning via reinforcement learning with sparse rewards, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020. URL: https://openreview.net/forum?id=S1xKd24twB.
[37] 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.
[38] A. Amiranashvili, N. Dorka, W. Burgard, V. Koltun, T. Brox, Scaling imitation learning in Minecraft, arXiv preprint arXiv:2007.02701 (2020).
[39] Z. Hao, A. Mallya, S. Belongie, M.-Y. Liu, GANcraft: Unsupervised 3D neural rendering of Minecraft worlds, in: ICCV, 2021.
[40] 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).
[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] 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.
[45] 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

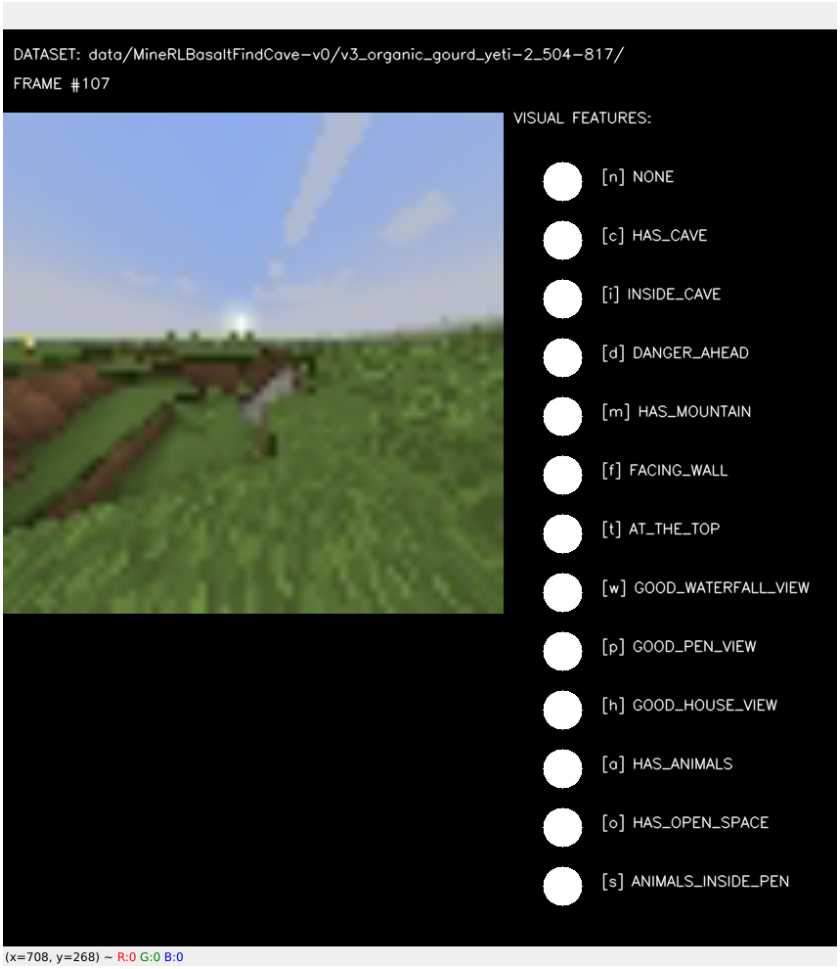


Figure 4: Custom GUI to relabel human dataset provided by the competition to train a classifier to identify relevant states for the state-machine.

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, indicating 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 will istill need to press the keyboard key to advance to the

next frame (any key for a label that was already selected clicking).

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 the waterfall is built, keep moving forward to move away from it;
    5. Once the 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

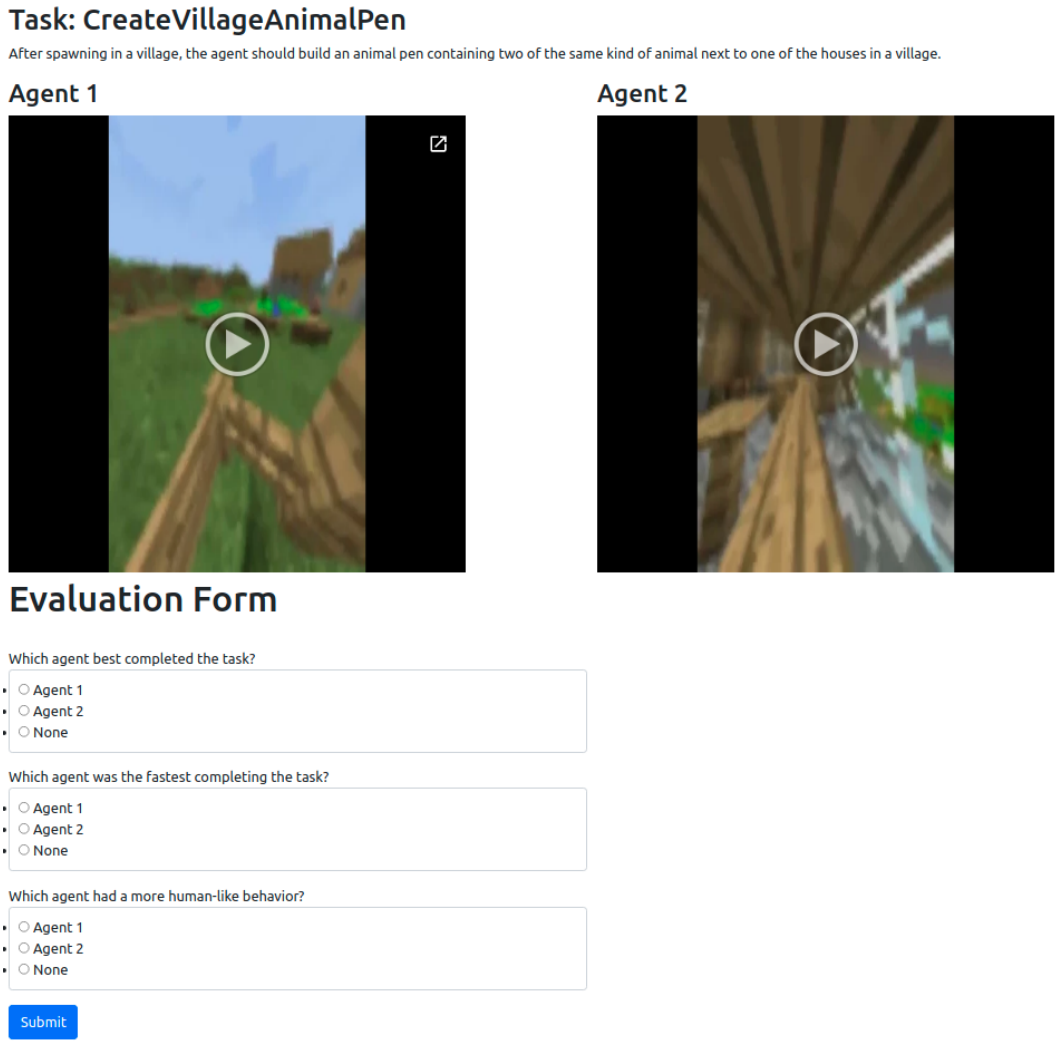


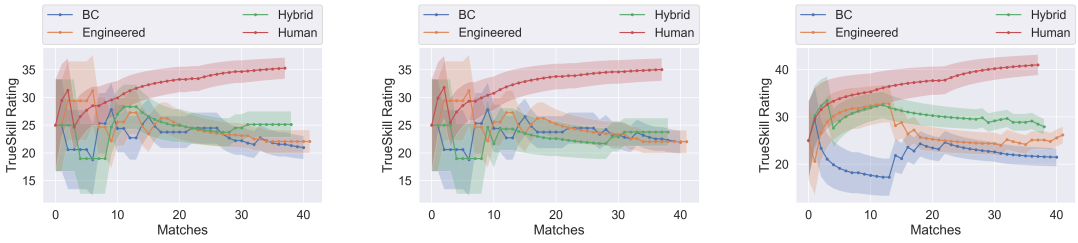
Figure 5: Web evaluation form used to collect additional human evaluation data to evaluate the multiple agent conditions presented in paper.

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.

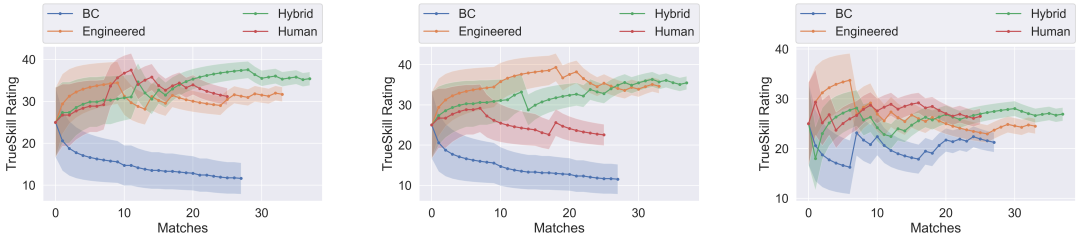
D. *TrueSkill*™ Score per Match

Figures 6, 7, 8, and 9 show the evolution of the *TrueSkill*™ scores 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.



(a) Best Performer (b) Fastest Performer (c) More Human-like Behavior

Figure 6: *TrueSkill*™[45] scores computed from human evaluations separately for each performance metric and for each agent type performing the *FindCave* task.

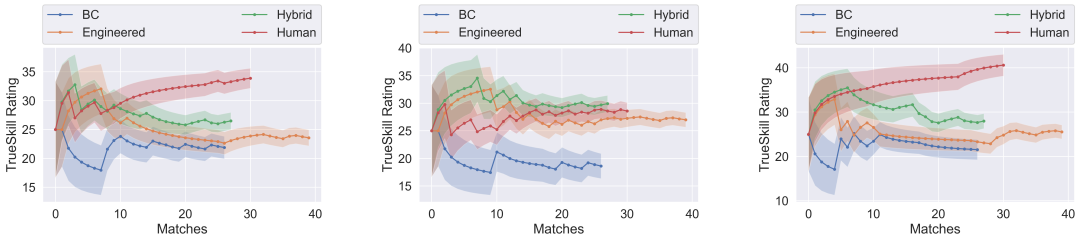


(a) Best Performer (b) Fastest Performer (c) More Human-like Behavior

Figure 7: *TrueSkill*™[45] scores computed from human evaluations separately for each performance metric and for each agent type performing the *MakeWaterfall* task.

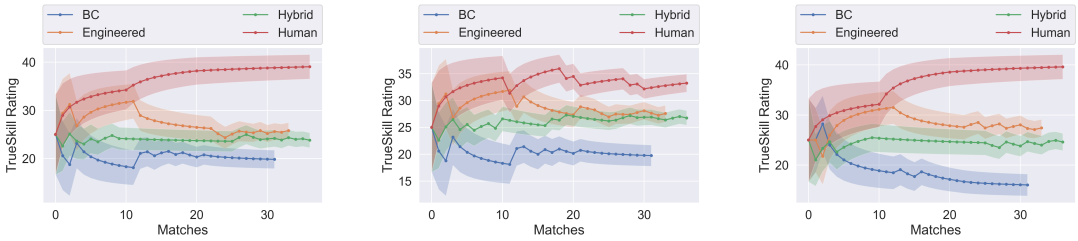
E. Pairwise Comparison per Performance Metric and Task

Figures 10, 11, 12, and 13 show bar plots with the individual pairwise comparisons compiled from the human evaluations for the *FindCave*, *MakeWaterfall*, *CreateVillageAnimalPen*, and



(a) Best Performer (b) Fastest Performer (c) More Human-like Behavior

Figure 8: *TrueSkill*™[45] scores computed from human evaluations separately for each performance metric and for each agent type performing the *CreateVillageAnimalPen* task.

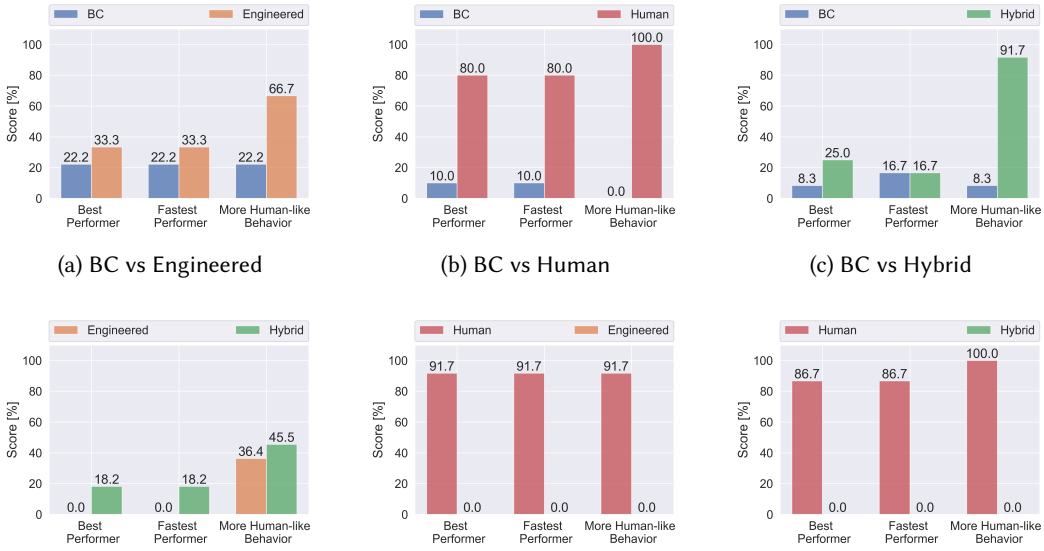


(a) Best Performer (b) Fastest Performer (c) More Human-like Behavior

Figure 9: *TrueSkill*™[45] scores computed from human evaluations separately for each performance metric and for each agent type performing the *BuildVillageHouse* task.

BuildVillageHouse tasks, respectively. Each bar represents the percentage of the time a given condition was selected 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 analyzing Figure 10(a), we can see that 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 the 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, 11, 12, and 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" agent, where the proposed hybrid method outperforms the fully engineered approach in all tasks except the *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 judged to better solve the task 70% of the time, to solve it faster 90% of the time, and even present a more human-like behavior 60% of the time. The "Hybrid" agent performing better can be attributed to the fact that the human players were not always able or willing to solve the task as described



(a) BC vs Engineered (b) BC vs Human (c) BC vs Hybrid

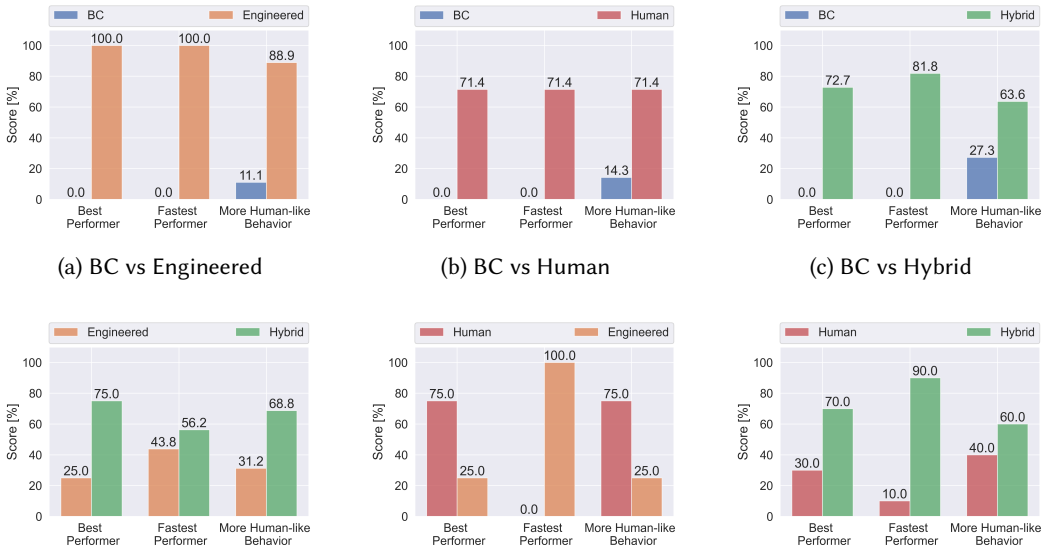
(d) Engineered vs Hybrid (e) Human vs Engineered (f) Human vs Hybrid

Figure 10: 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 *FindCave* task.

in the prompt.

F. Samples of Hybrid Agent Solving the Tasks

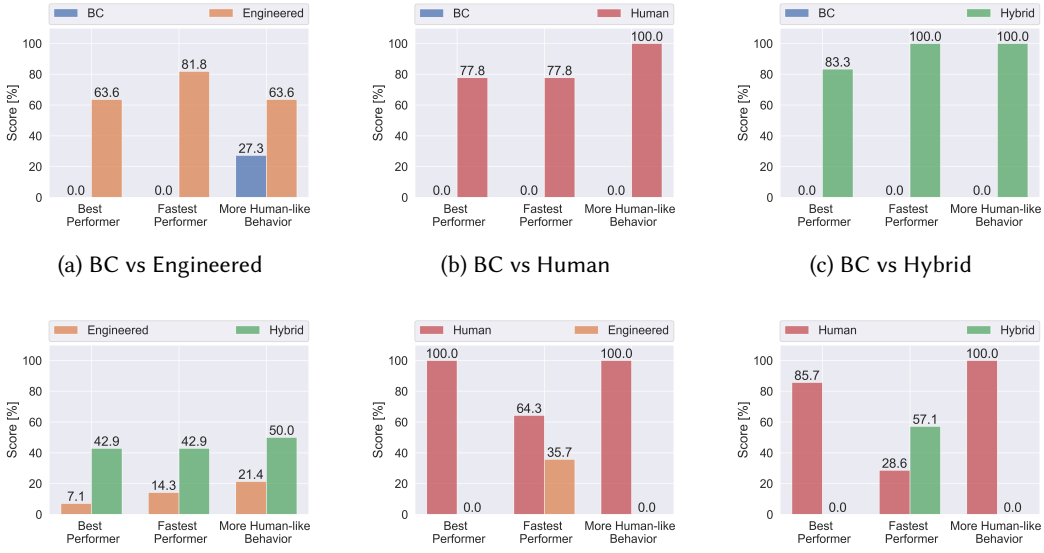
In terms of qualitative results, Figures 14, 15, 16, and 17 show a sample episode illustrated by a sequence of frames of our hybrid agent solving the *FindCave*, *MakeWaterfall*, *CreateVillageAnimalPen*, and *BuildVillageHouse* tasks, respectively. Each figure shows the image frames received by the 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.



(a) BC vs Engineered (b) BC vs Human (c) BC vs Hybrid

(d) Engineered vs Hybrid (e) Human vs Engineered (f) Human vs Hybrid

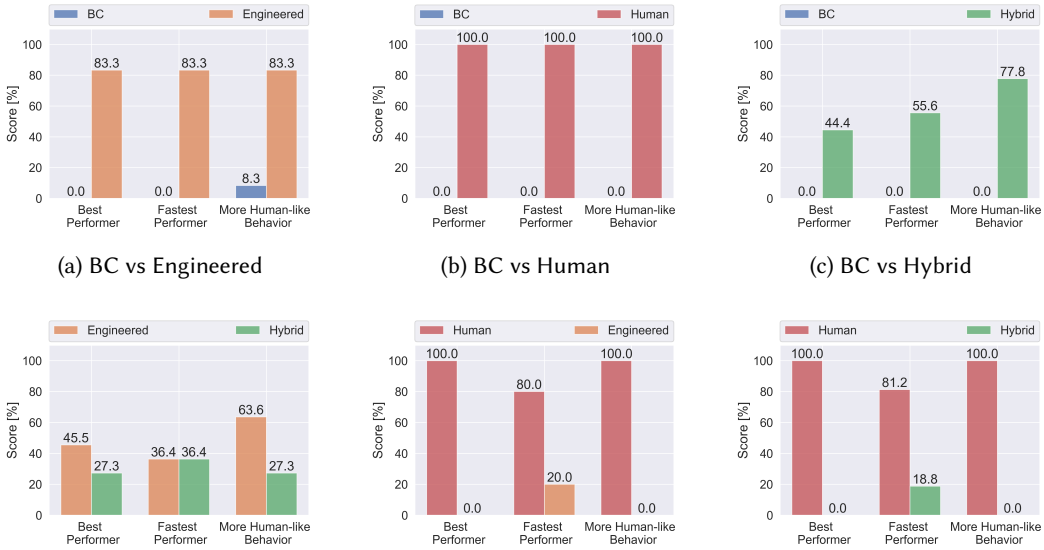
Figure 11: Pairwise comparison displaying the normalized scores computed from human evaluations separately for each performance metric on all possible head-to-head comparisons for all agent type performing the *MakeWaterfall* task.



(a) BC vs Engineered (b) BC vs Human (c) BC vs Hybrid

(d) Engineered vs Hybrid (e) Human vs Engineered (f) Human vs Hybrid

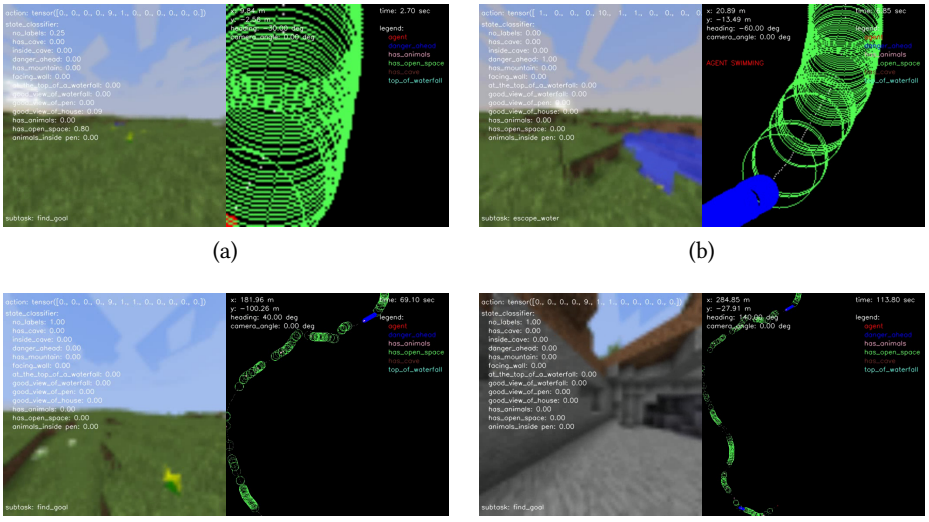
Figure 12: Pairwise comparison displaying the normalized scores computed from human evaluations separately for each performance metric on all possible head-to-head comparisons for all agent type performing the *CreateVillageAnimalPen* task.



(a) BC vs Engineered (b) BC vs Human (c) BC vs Hybrid

(d) Engineered vs Hybrid (e) Human vs Engineered (f) Human vs Hybrid

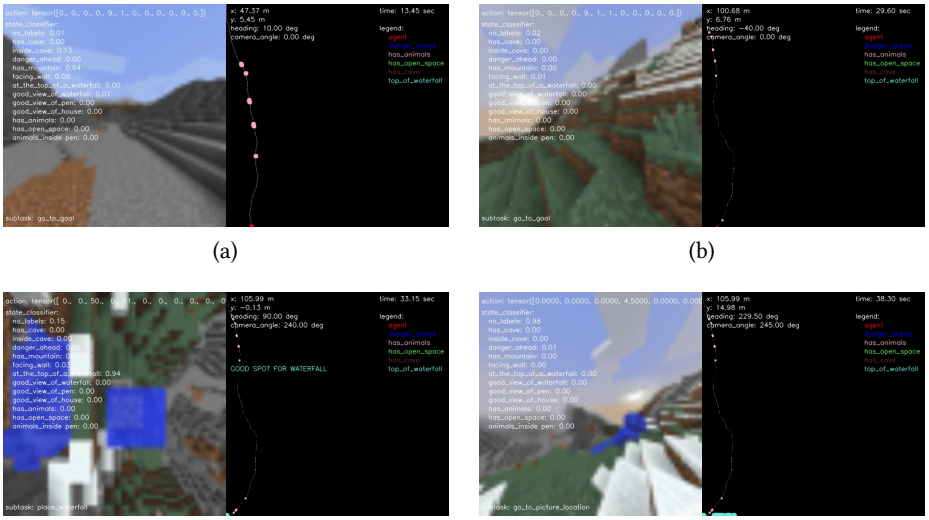
Figure 13: Pairwise comparison displaying the normalized scores computed from human evaluations separately for each performance metric on all possible head-to-head comparisons for all agent type performing the *BuildVillageHouse* task.



(a) (b)

(c) (d)

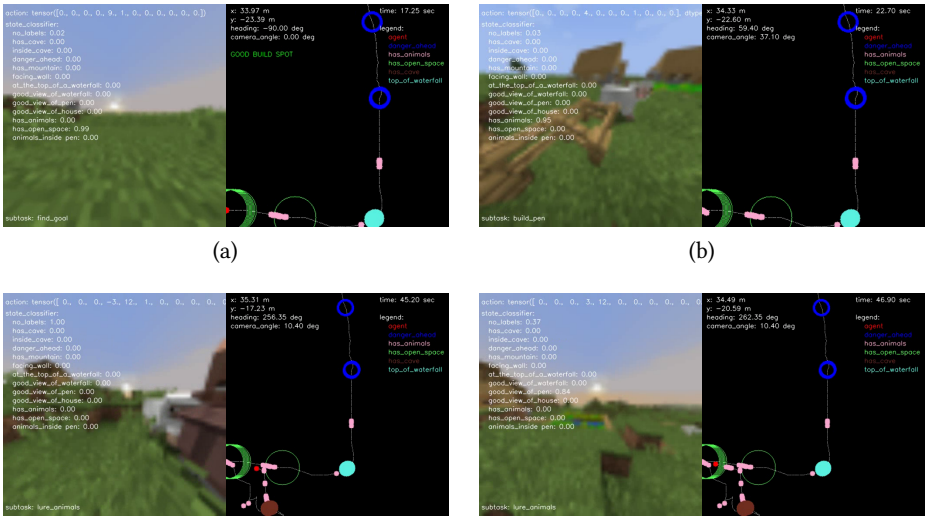
Figure 14: Sequence of frames of our hybrid agent solving the *FindCave* task (complete video available at https://youtu.be/MR8q3Xre_XY).



(a) (b)

(c) (d)

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



(a) (b)

(c) (d)

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

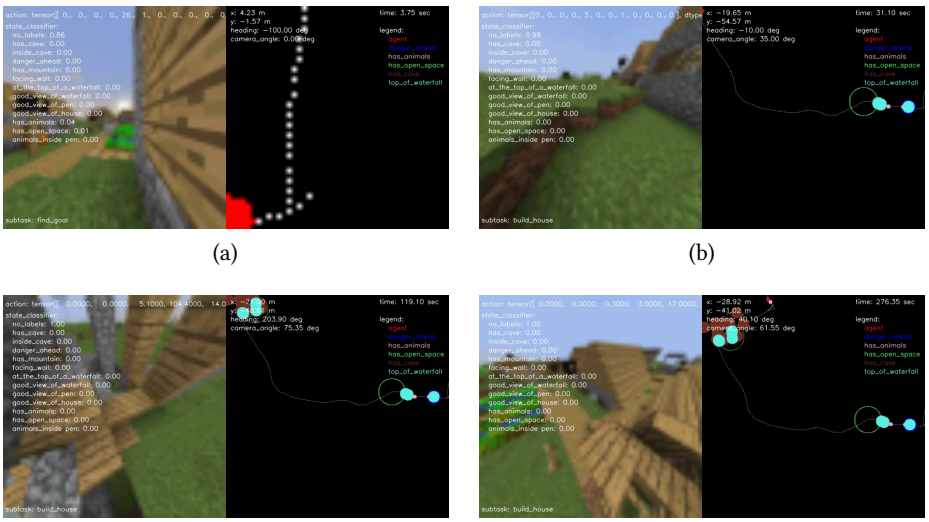


Figure 17: Sequence of frames of our hybrid agent solving the *BuildVillageHouse* task (complete video available at https://youtu.be/_uKO-ZqBMWQ).