Combining Learning From Human Feedback And Knowledge Engineering To Solve Hierarchical Tasks In Minecraft

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 \sepHuman-in-the-Loop Learning \sepMachine Learning \sepArtificial Intelligence \sepKnowledge Engineering \sepMinecraft

\conference
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.

[email=vinicius.goecks@gmail.com, url=https://vggoecks.com/,] [email=nicholas.r.waytowich.civ@mail.mil,] [email=davidwatkins@cs.columbia.edu, url=https://davidwatkinsvalls.com/,] [email=bhp1@umbc.edu,]

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"111Official competition webpage: https://www.aicrowd.com/challenges/neurips-2021-minerl-basalt-competition.. 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"[222]Official competition webpage: https://www.aicrowd.com/challenges/neurips-2021-minerl-basalt-competition., challenged participants to come up with creative solutions to solve four different tasks in Minecraft [39] using the "MineRL: Towards AI in Minecraft"[333]MineRL webpage: https://minerl.io/. 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 competition[444]MineRL BASALT documentation:

https://minerl.io/basalt/. 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.

- 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

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

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 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 64×64×36464364\times 64\times 364 × 64 × 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 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 (imagine you are looking for a 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).

To train this system we labeled 81,8888188881,88881 , 888 images each corresponding to labels from all tasks using a custom graphical user interface (GUI), as showed in Appendix A. 80%percent8080\%80 % of images were used for training, 10%percent1010\%10 % were used for validation, and 10%percent1010\%10 % was used for testing. The model is a convolutional classifier with a 64×64×36464364\times 64\times 364 × 64 × 3 input and 13×113113\times 113 × 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 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 20202020 frames per second, which means that there is a 0.050.050.050.05 second interval between each frame. According to the Minecraft Wiki555Minecraft Wiki - Walking: https://minecraft.fandom.com/wiki/Walking., walking speed is approximately 4.3174.3174.3174.317 m/s, 5.6125.6125.6125.612 m/s while sprinting, or 7.1277.1277.1277.127 m/s when sprinting and jumping at the same time, which translates to approximately 0.2160.2160.2160.216, 0.2810.2810.2810.281, or 0.3560.3560.3560.356 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)00(0,0)(0 , 0) xxitalic_x and yyitalic_y map coordinates, when the agents commands a discrete action to move forward its position is updated by 0.2160.2160.2160.216 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 \thetaitalic_ 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:

x\displaystyle\dotxover start_ARG italic_x end_ARG =Vcos()absent\displaystyle=Vcos(\theta)= italic_V italic_c italic_o italic_s (italic_)

y\displaystyle\dotyover start_ARG italic_y end_ARG =Vsin(),absent\displaystyle=Vsin(\theta),= italic_V italic_s italic_i italic_n (italic_) , where VVitalic_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 learn 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 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 64×64646464\times 6464 × 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 remember 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 FindCave task or the best spot to place a waterfall in the 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. Minecraft java 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. 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 application666Custom MineRL BASALT evaluation webpage: https://kairosminerl.herokuapp.com/., 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 10101010 videos of agent performance for each condition, for a total of 160160160160 videos. All agent-generated videos were scaled from the original 64×64646464\times 6464 × 64 image resolution returned by the environment to 512×512512512\times 512512 × 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 512×512512512\times 512512 × 512 image resolution to match the agent-generated videos. All videos were generated and saved at 20202020 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, 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 TrueSkillTM777Microsoft's TrueSkillTM Ranking System: https://www.microsoft.com/en-us/research/project/trueskill-ranking-system Bayesian ranking system [46]. In this Bayesian ranking system, the skill of each participant is characterized by a Gaussian distribution with a mean value \muitalic_ representing the average skill of a participant and standard deviation \sigmaitalic_ 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 TrueSkillTM 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 TrueSkillTM 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%percent5.35.3\%5.3 % and 25.6%percent25.625.6\%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%percent21.721.7\%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%percent2.72.7\%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, 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 TrueSkillTM 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%percent66.666.6\%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%percent33.333.3\%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%percent53.253.2\%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 task888Sample trajectory of hybrid agent solving the FindCave task: https://youtu.be/MR8q3Xre_XY., 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 task999Sample trajectory of hybrid agent solving the MakeWaterfall task: https://youtu.be/eXp1urKXIPQ., 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 task101010Sample trajectory of hybrid agent solving the CreateVillageAnimalPen task: https://youtu.be/b8xDMxEZmAE., 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 task111111Sample trajectory of hybrid agent solving the BuildVillageHouse task: https://youtu.be/_uKO-ZqBMWQ., 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.

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 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.

Acknowledgements.
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.