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

by Goecks, Vinicius G. Waytowich , Nicholas, Watkins, David, Prakash, Bharat

AAAI 2022 SPRING SYMPOSIUM

Sirine Younsi, 1st year Master's Student, Kaneko Lab

Ref: https://arxiv.org/abs/2112.03482

SUMMARY

Reinforcement learning has had a lot of breakthroughs in game AI (Dota / Go)

Mainly achieving super-human performance

What about **human-like performance**?

Real-world Tasks

Instinctive

Have no reward signal

- How to evaluate whether a behavior is human like or not?

- How to solve a problem that has no task specification?
- How to overcome the complexity of the real world? (Massive amount of data)

Open-world game
Human gameplay imitation +
Solution engineering

- Overcome RL limitations
- Achieve faster and better results

CONTENTS

CONTENTS

MINECRAFT

Environment

- Open-world : procedurally generated 3D world
- Instinctive task hierarchies
- Sparse rewards

Interactions

- Free navigation
- Interaction with a variety of fauna and flora.
- Mining, collecting, and searching for resources.
- Designing and building complex objects.

Figure 1. Game of Minefcraft

MINERL

Large-scale dataset of human gameplay (MineRL-v0 Dataset)

Set of Minecraft environments

MINERL Challenge (started 2019)

Overcome Deep RL limitations

HOW?

Leverage imitation Learning

Sample inefficiency (e.g, AlphaGoZero played 4.9 million games)

Specifying Tasks / Sparse Reward

Figure 2. Minecraft Tasks addressed by MineRL

CONTENTS

MINERL CHALLENGE 3rd EDITION: BASALT COMPETITION 2021

Mission:

Develop agents with human-like behavior capable of solving the following tasks

Time:

4 days

Dataset:

40 to 80 human demonstrations for each task

Evaluation method:

Human feedback (No reward function)

Find Cave

Make Waterfall

Create Animal Pen

Build Village House

CONTENTS

MINERL DATASET

60 Million State-Action pair * 4 Versions

- **2 Resolutions (64 × 64 / 192 × 256)**
- **2 Textures (default Minecraft / simplified)**

4 Task families:

- Navigation
- Tree Chopping
- Obtain Item
- Survival

State :

- RGB video frame of the player's point-of-view
- player inventory
- item collection events
- distances to objectives
- player attributes (health, level, achievements)
- current GUI details

Action

- keyboard presses on the client.
- The change in view caused by mouse movement.
- All player GUI click and interaction events
- Chat messages sent,
- Other actions such as item crafting.

END-TO-END MACHINE LEARNING

Algorithms that learn purely from data, with minimal bias or constraints added by human designers

Example:

Deep reinforcement learning algorithms directly playing video games from pixel inputs

HUMAN-IN-THE-LOOP MACHINE LEARNING

Algorithms that learn from human feedback

Example:

Agent trained based on human demonstrations of the task (successful and failed examples)

LIMITATIONS

- Limited human demonstration dataset (40-80 demonstrations is not enough)
- Data with low quality
- Large-observation space
- Limited computing time

Not adequate to solve the BASALT competition tasks because of their complexity

CONTENTS

HYBRID INTELLIGENCE

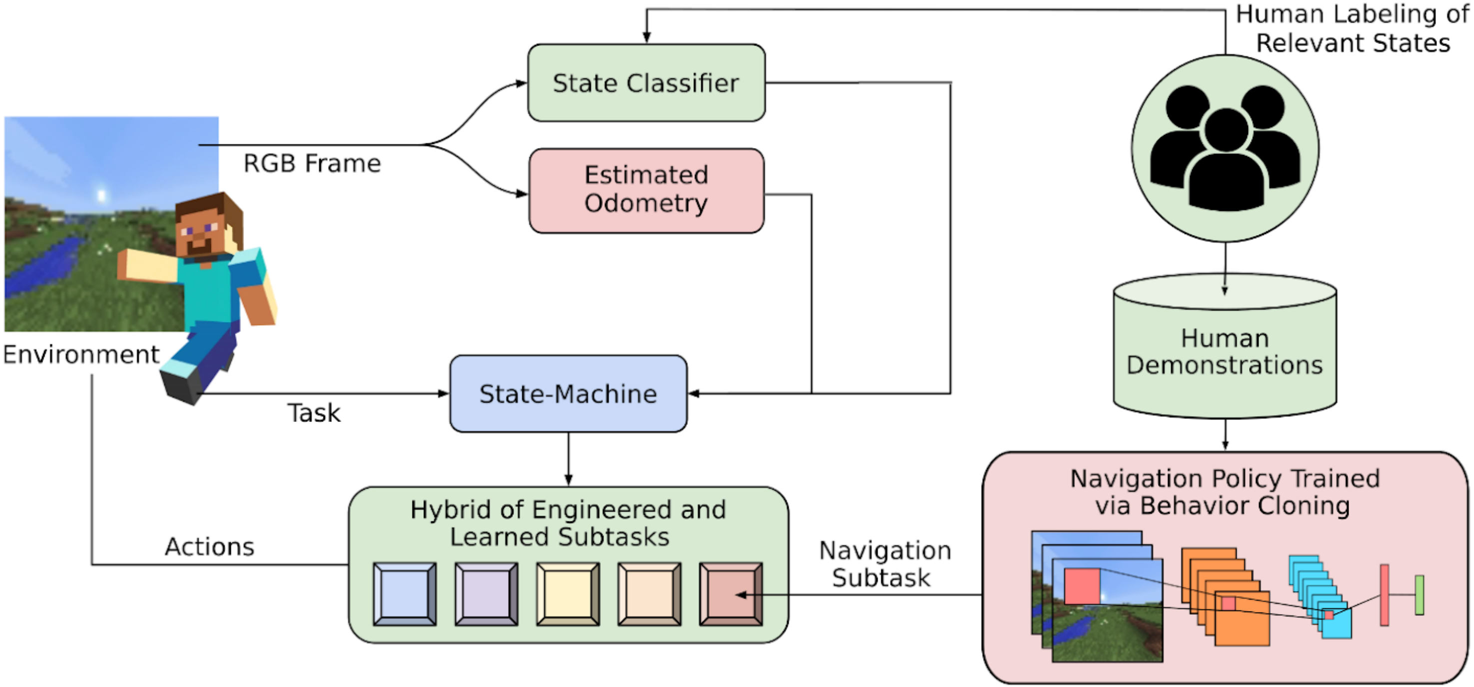
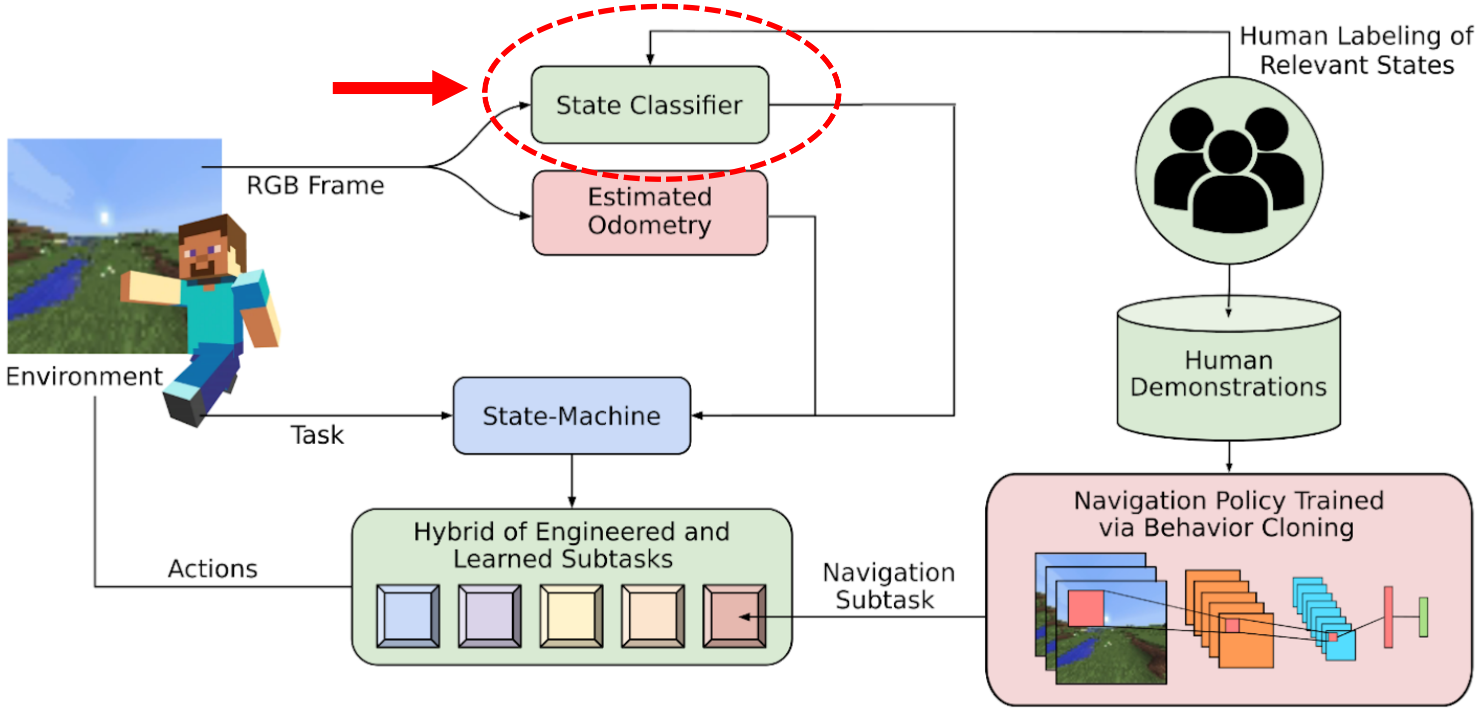


Figure 3. proposed solution: Hybrid Intelligence, Goeks et al. (2021)

STATE CLASSIFICATION

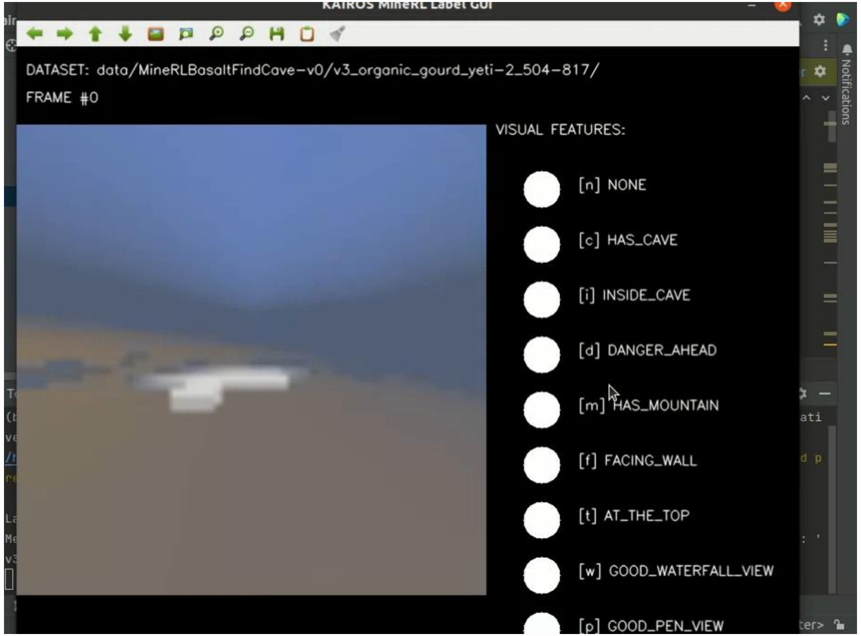


STATE CLASSIFICATION

Label data using human feedback – 2nd Step:

labelDataGUI.py:

Humans were assigned to label image frames from the collected human demonstration data.



GUI label demonstration

STATE CLASSIFICATION

Label data using human feedback – 3rd Step:

RGB images (low resolution 64x64x3) → State Classifier → **State label**

- none
- has_cave
- inside_cave :
- danger_ahead
- has_mountain
- facing_wall
- at_the_top
- good_waterfall_view
- good_pen_view
- good_house_view
- has_animals
- has_open_space
- animals_inside_pen

STATE CLASSIFICATION

Label data using human feedback – 1st Step:

dataProcessing.py:

Actions and images are extracted from recorded videos and saved as NumPy arrays

actions.npy

```
0000 0000 0000 0000 0000 0000 0000 0000
3000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
6e00 0000 6f00 0000 6e00 0000 6500 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
3000 0000 0000 0000 0000 0000 0000 0000
```

Figure 3. actions NumPy array extracted from one demonstration video

Convert NumPy arrays to PNG images

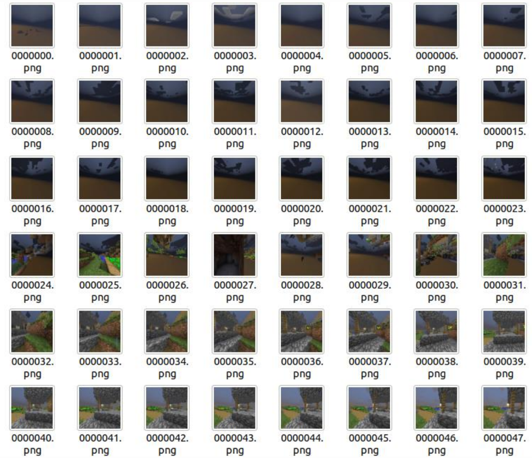


Figure 4. sequence of PNG images extracted from video

STATE CLASSIFICATION

Label data using human feedback – 4th Step:

compileLabels.py:

Group images and labels in two NumPy files

```
Images labeled: 81888 images
  Labels for class 0: 44605 (54.471 %)
  Labels for class 1: 1138 (1.390 %)
  Labels for class 2: 1055 (1.288 %)
  Labels for class 3: 3135 (3.828 %)
  Labels for class 4: 3591 (4.385 %)
  Labels for class 5: 3730 (4.555 %)
  Labels for class 6: 3253 (3.972 %)
  Labels for class 7: 2587 (3.159 %)
  Labels for class 8: 3373 (4.119 %)
  Labels for class 9: 2112 (2.579 %)
  Labels for class 10: 7685 (9.385 %)
  Labels for class 11: 6004 (7.332 %)
  Labels for class 12: 664 (0.811 %)
```

Figure 5. Amount of data for each state label/class

STATE CLASSIFICATION

Label data using human feedback – 5th Step:

StateClassifier.py:

Used a convolutional classifier inspired from Deep Tamer

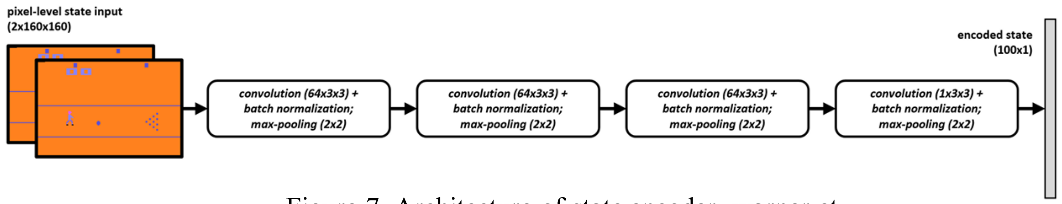


Figure 7. Architecture of state encoder, warner et al. 2018

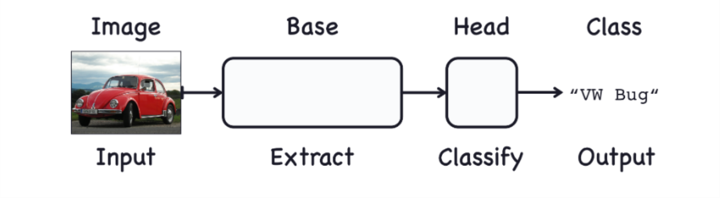
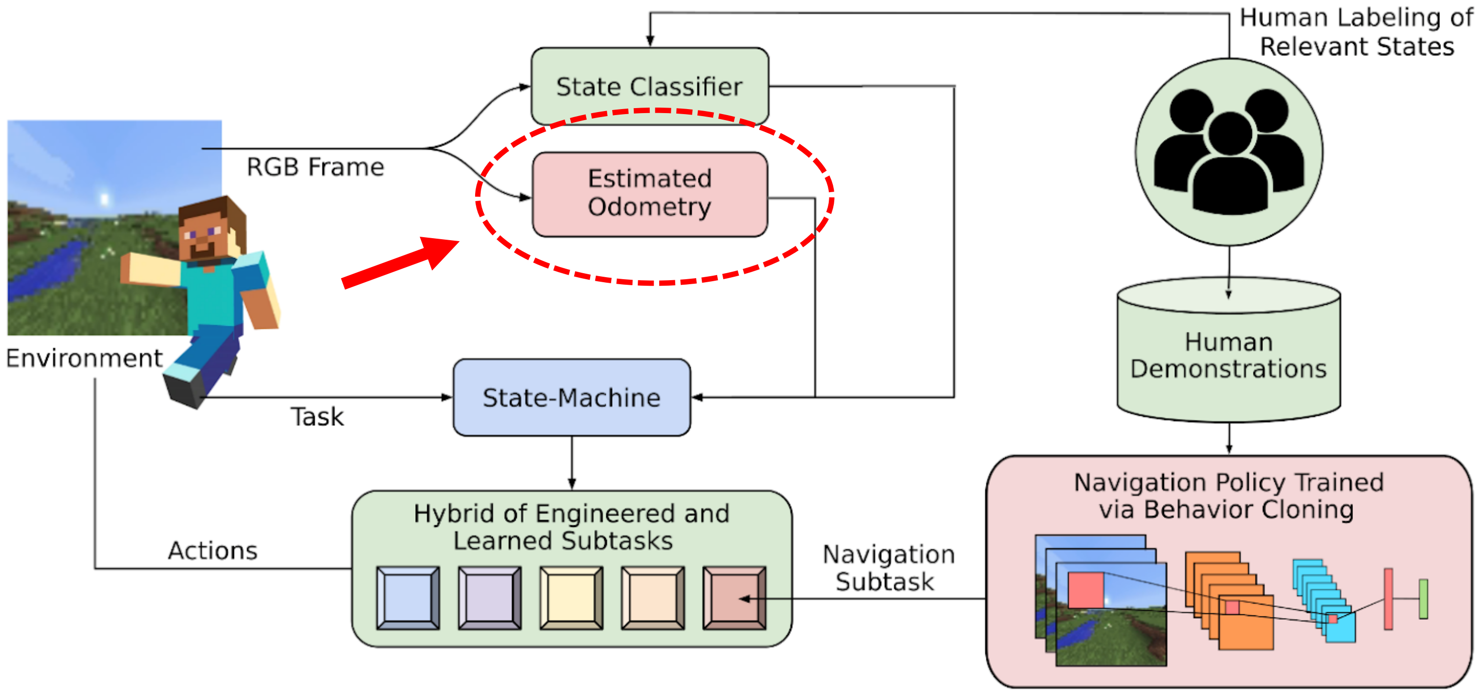


Figure 6. convolutional classifier
source: https://www.kaggle.com/code/ryanholbrook/the-convolutional-classifier/tutorial

```
Number of classes: 13
data_images: (81888, 3, 64, 64)
data_labels: (81888, 13)
x_train: (65511, 3, 64, 64)
y_train: (65511, 13)
x_test: (8186, 3, 64, 64)
y_test: (8186, 13)
```

Figure 8. Train and Test dataset

ODOMETRY MAP



ODOMETRY MAP

Problem:

Absence of basic localization

Solution: Estimated Odometry:

Attach a coordinate to each classified state

Initial Values and Assumptions

- Player starts at position (0,0)
- Agent's Velocity:
  - Walking: 4 . 317 m/s.
  - Sprinting: 5 . 612 m/s
  - Sprinting + Jumping: 7 . 127 m/s
- Heading angle θ :
  - =>Follows point-mass kinematics:
  - $\dot{x} = V \cos(\theta)$
  - $\dot{y} = V \sin(\theta)$

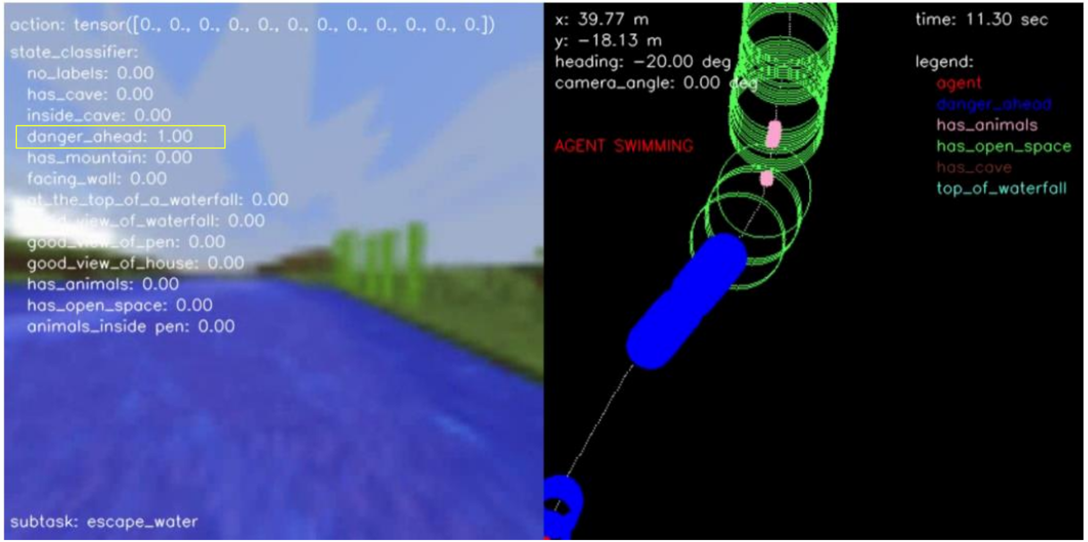
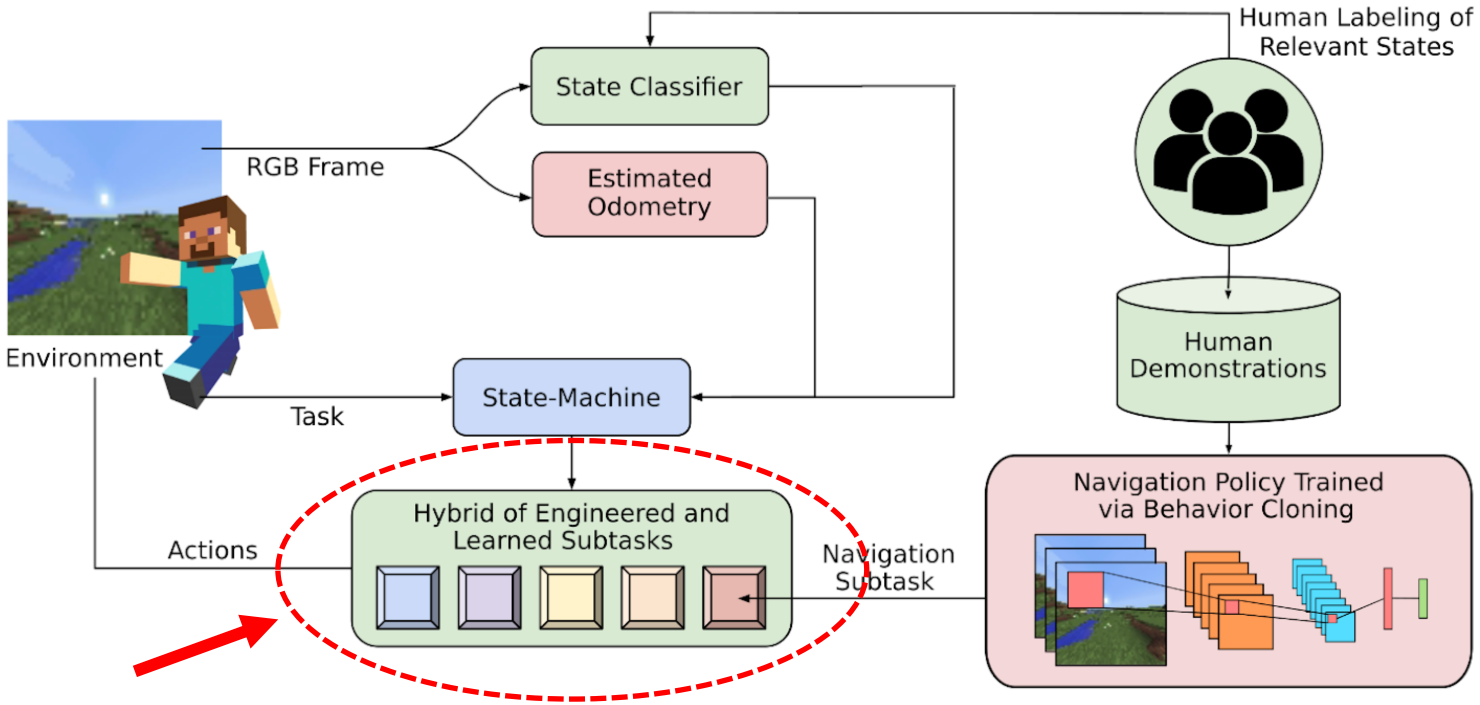


Figure 9. Odometry map generated in real time, Goeks et al. (2021)

LEARNING AND ENGINEERING SUBTASKS



LEARNING AND ENGINEERING SUBTASKS

The 4 tasks to solve are too complex

Need for Perception, Memory, Reasoning

- Decompose each task to subtasks using human knowledge of the task
- Use **hybrid Intelligence**:

  Some subtasks are learned from human data (e.g., Navigation).

  Some subtasks are directly engineered (e.g., Throw a snowball at the end of an episode).

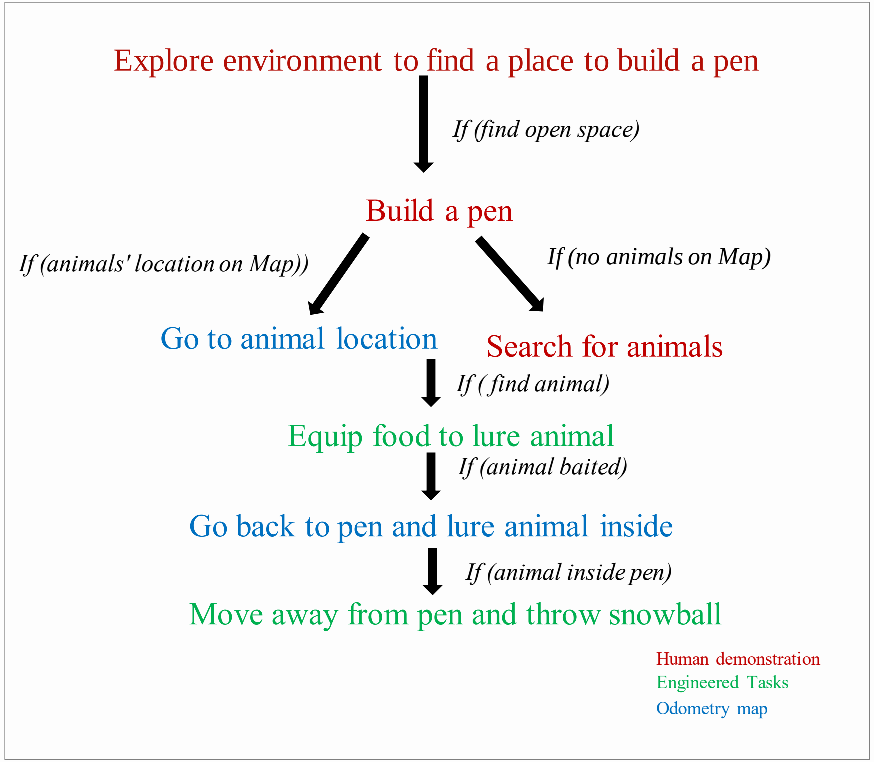


Figure 10. Subtasks of the Task CreateVillageAnimalPen

LEARNING AND ENGINEERING SUBTASKS

Engineered subtask

```
def subtask_end_episode(self):
    # reset actions
    action = th.zeros(12)

    # throw snowball
    if self.task == "BUILD_HOUSE":
        action[4] = 22 # equip it
    else:
        action[4] = 8 # equip it
    action[11] = 1 # throw it

    return action
```

Figure 11. end_episode subtask (in package Kairos_minerl)

Equip agent with snowball from Inventory

4 is the label of action “equip“ in the list of defined actions

8 and 22 are the positions of the snowball item in the agentrs inventory

```
self.action_str_to_int = {
    "attack": 0,
    "back": 1,
    "camera_up_down": 2,
    "camera_right_left": 3,
    "equip": 4,
    "forward": 5,
    "jump": 6,
    "left": 7,
    "right": 8,
    "sneak": 9,
    "sprint": 10,
    "use": 11,
}
```

Figure 12. list of labeled Actions

LEARNING AND ENGINEERING SUBTASKS

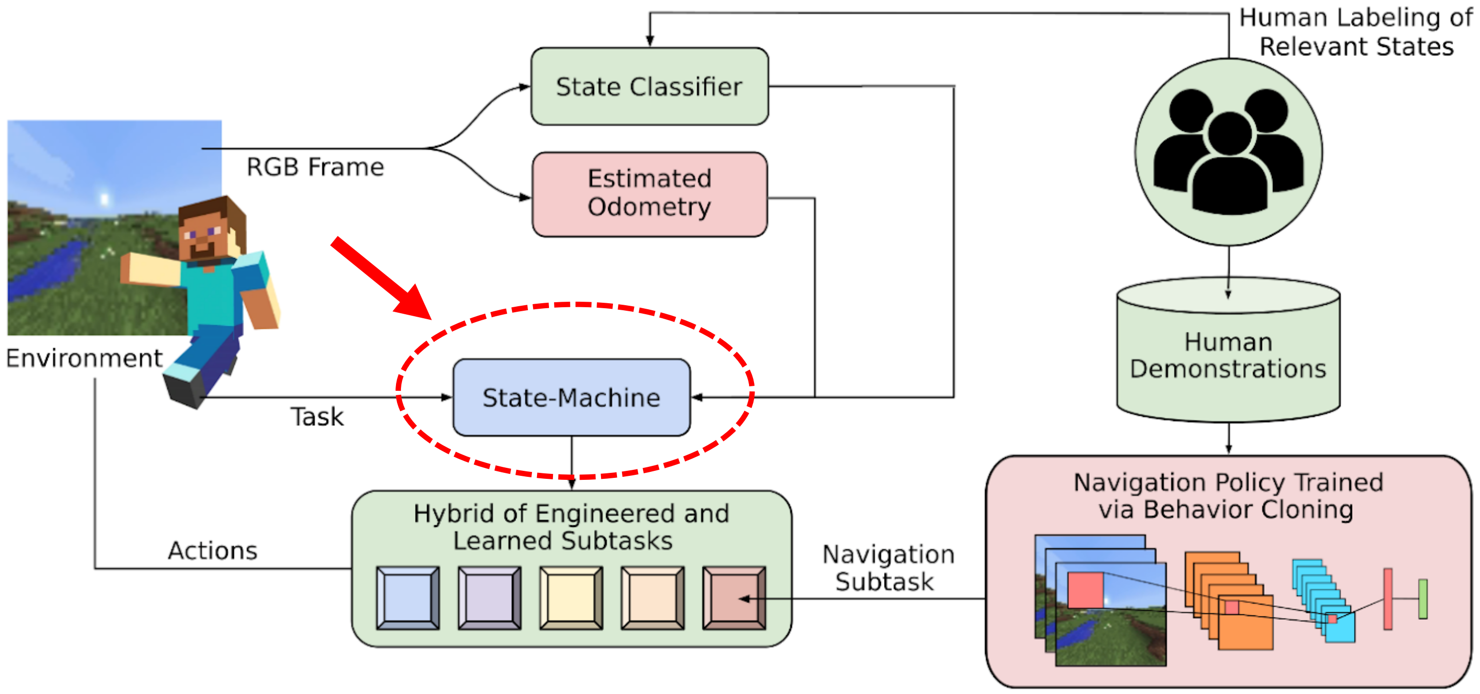
learned subtask

behaviorCloner.py

```
Initializing Standard BC model!
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 128, 64, 64]           3,584
              ReLU-2         [-1, 128, 64, 64]               0
       BatchNorm2d-3         [-1, 128, 64, 64]             256
         MaxPool2d-4         [-1, 128, 32, 32]               0
            Conv2d-5          [-1, 64, 32, 32]          73,792
              ReLU-6          [-1, 64, 32, 32]               0
       BatchNorm2d-7          [-1, 64, 32, 32]             128
         MaxPool2d-8          [-1, 64, 16, 16]               0
            Conv2d-9           [-1, 3, 16, 16]           1,731
             ReLU-10           [-1, 3, 16, 16]               0
      BatchNorm2d-11           [-1, 3, 16, 16]               6
          Dropout-12                 [-1, 768]               0
           Linear-13                 [-1, 256]         196,864
             ReLU-14                 [-1, 256]               0
          Dropout-15                 [-1, 256]               0
           Linear-16                 [-1, 256]          65,792
             ReLU-17                 [-1, 256]               0
           Linear-18                  [-1, 13]           3,341
----------------------------------------------------------------
```

Figure 13. Autoencoder Architecture

STATE MACHINE



STATE MACHINE

Example: Task MakeWaterfall

search for location in the mountain to make a waterfall

↓

build additional blocks to add height to the waterfall

↓

Place waterfall using bucket

↓

Move forward from waterfall

↓

Turn around and throw snowball

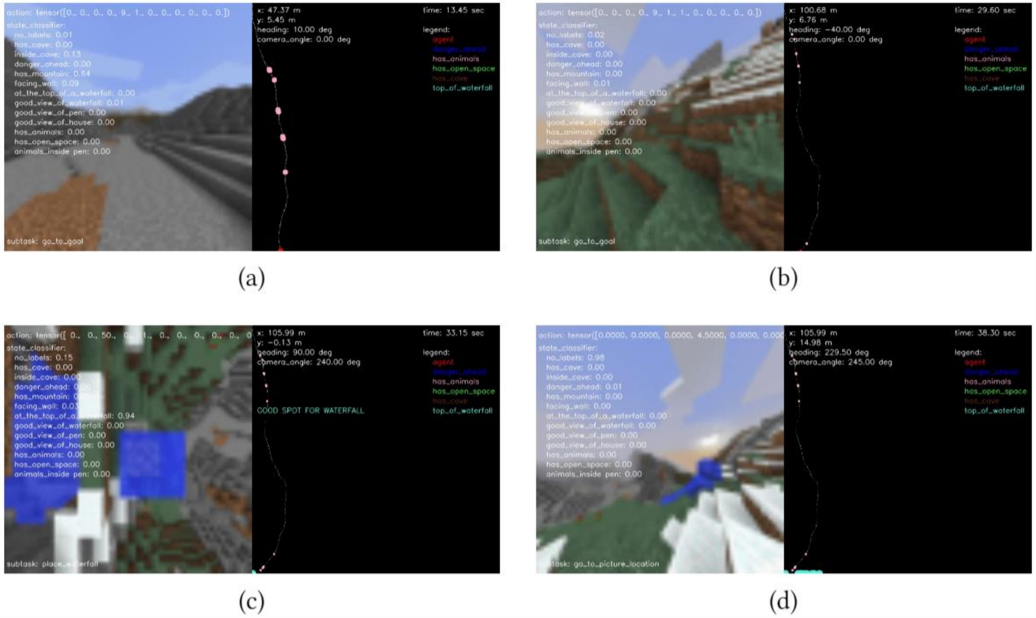


Figure 14. Sequence of frames of the hybrid Agent solving MakeWaterfall task

STATE MACHINE

stateMachine.py

```
"BuildWaterfall": [
    {'trigger': 'detected_top_of_mountain', 'source': 'find_spot_to_build_waterfall', 'dest': 'build_waterfall'},
    {'trigger': 'finished_waterfall', 'source': 'build_waterfall', 'dest': 'go_to_picture_location'},
    {'trigger': 'detected_good_waterfall_view', 'source': 'go_to_picture_location', 'dest': 'looking_at_waterfall'},
],
```

Figure 15. State machine for task build waterfall code

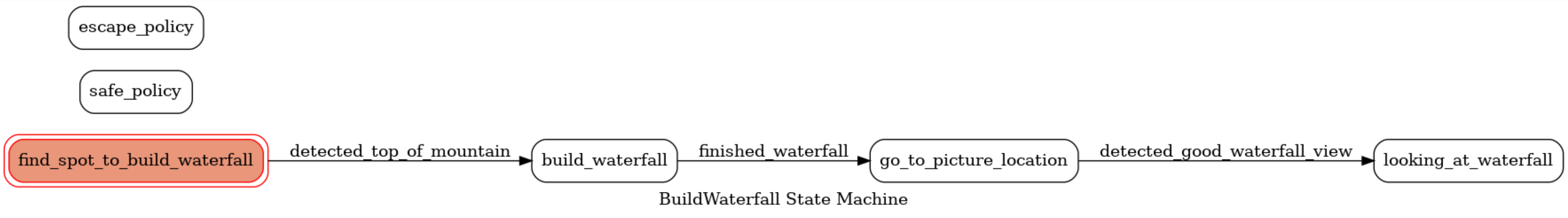


Figure 16. State machine plotted for task build waterfall

STATE MACHINE

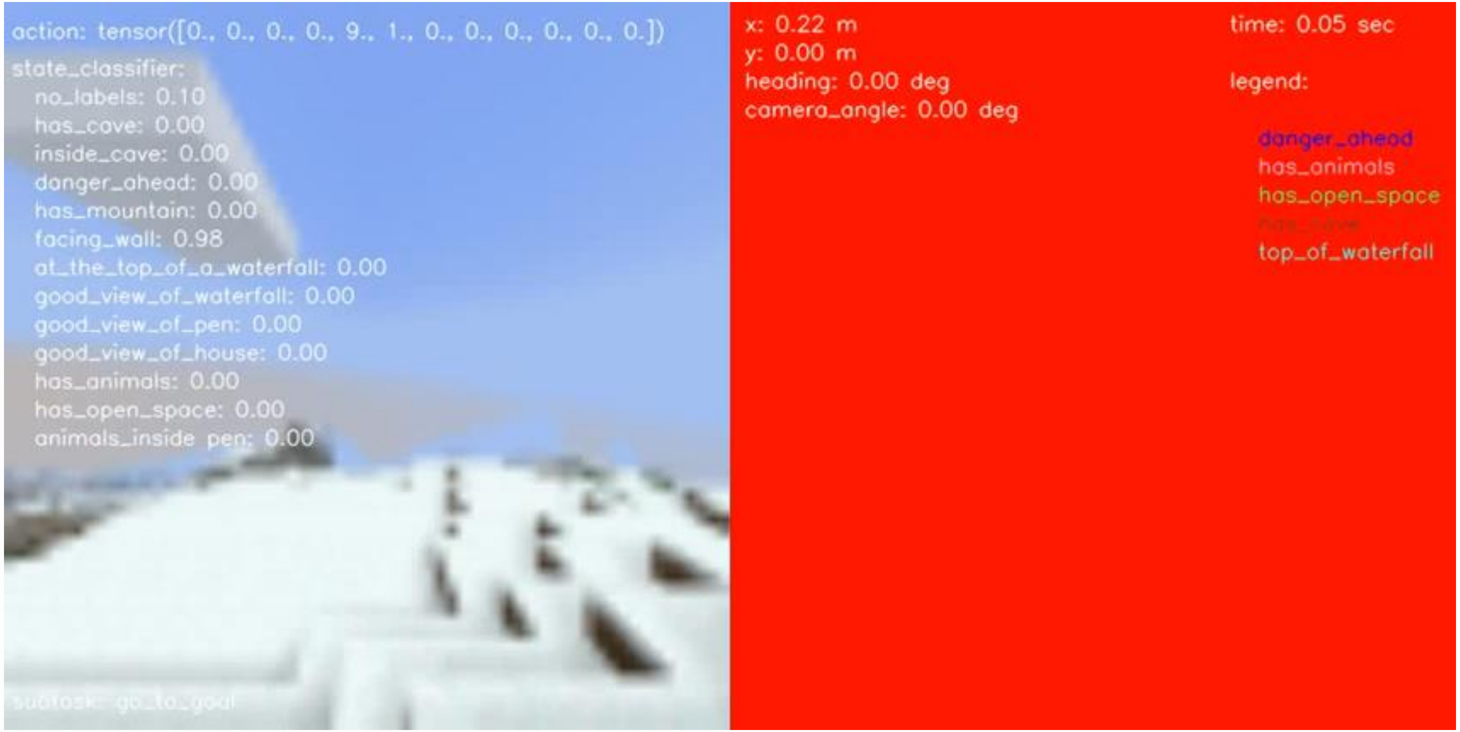


Figure 17. MakeWaterfall task demonstration

CONTENTS

EVALUATION METHOD

4 types of agents were compared head-to-head by 7 different human evaluators:

- **Hybrid:** proposed solution
- **Engineered:** only navigation task was learned from human data and rest engineered)
- **Behavior cloning (BC):** all tasks learned from human data
- **Human:** human demonstration from the provided dataset

Metrics:

- Best performer
- Fastest performer
- More human-like behavior

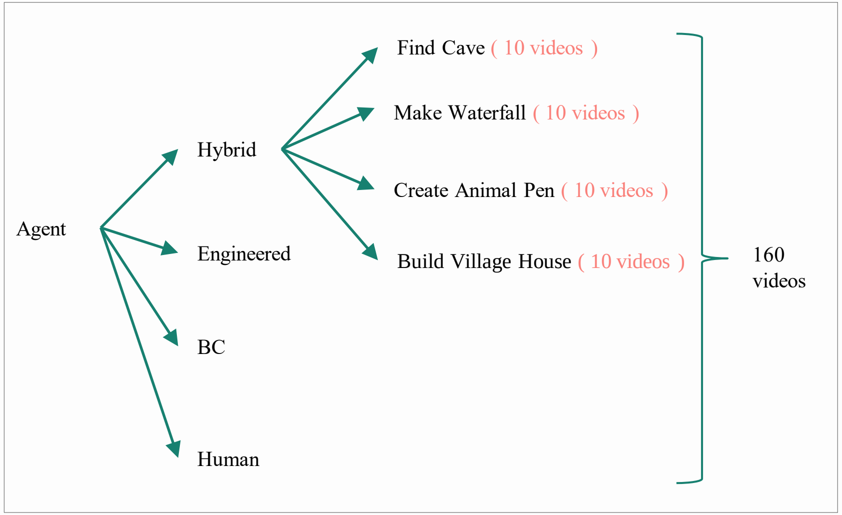


Figure 18. Evaluation data structure for the human evaluators

EVALUATION INTERFACE

Possible outcomes:

- Agent 1 wins
- Agent 2 wins
- None (Draw when none of the agents achieved the task)

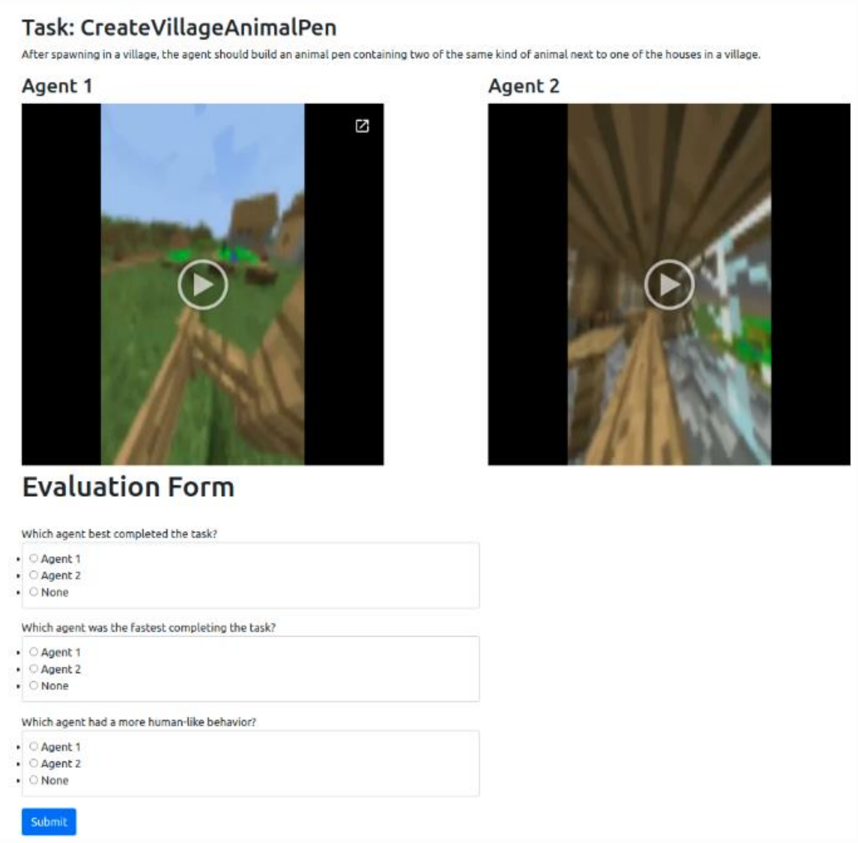
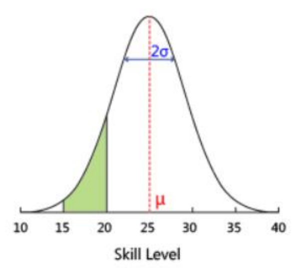


Figure 19. Evaluation Interface, Goeks et al. (2021)

TrueSkill TM SCORE



Each participant/Agent type is characterized by a Gaussian distribution that has:

Mean value μ (Average skill of a participant)

Standard deviation μ (Degree of uncertainty in the participant's skill)

TrueSkill Rate calculation: R= μ - 3*μ

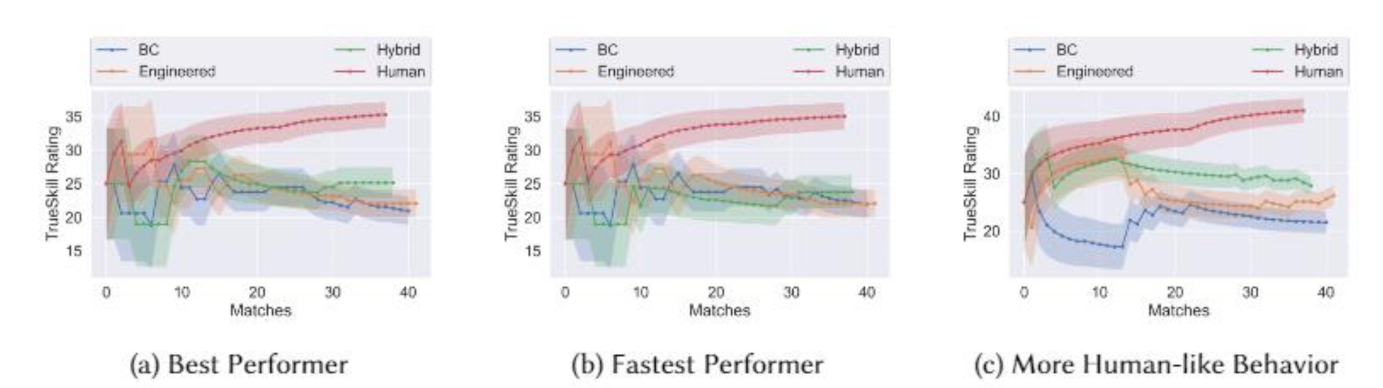


Figure 20. Evolution of TrueSkill TM score for each performance metric and for each agent type performing the FindCave task., Goeks et al. (2021)

AVERAGED PERFORMANCE

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

Figure 21. Averaged score of all tasks for each performance metric and agent type, Goeks et al. (2021)

TrueSkill TM SCORE

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

Figure 22. Score of each task for each performance metric and agent type, Goeks et al. (2021)

CONTENTS

The proposed solution:

- Outperformed end-to-end machine Learning algorithm ✓
- Outperformed purely engineered solution ✓
- Used less data ✓
- Used less computing time ✓

- Still not as good as a human player especially for the more human-like behavior metric ✗

THANK YOU FOR YOUR ATTENTION

Sirine Younsi

Q&A

BIBLIOGRAPHY

Goecks , Vinicius G. & Waytowich, Nicholas & Watkins, David & Prakash, Bharat. (2021). Combining Learning from Human Feedback and Knowledge Engineering to Solve Hierarchical Tasks in Minecraft

Warnell, Garrett & Waytowich, Nicholas & Lawhern, Vernon & Stone, Peter. (2018). Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces.