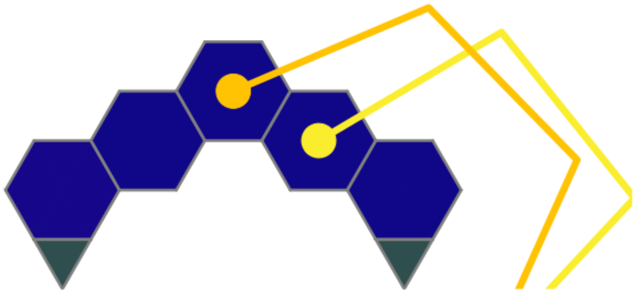
École Polytechnique Fédérale de Lausanne

Semester Project

Reinforcement Learning from Human Feedback for the Construction of Spanning Structures



Author
Sabri El Amrani

Supervisors
Maryam Kamgarpour
Andreas Schlaginhaufen
Anna Maddux

Academic Year 2023 - 2024

Abstract

This semester project addresses the challenge of building a structure connecting two supports across a gap, using reinforcement learning from human preferences. This approach involves learning a reward predictor from human feedback between pairs of demonstrations of the construction task. After presenting the algorithm used to train the agent with human feedback, the report begins by experimentally validating the methodology. A comparison between the effectiveness of two reward models follows: one based on a linear combination of handcrafted features and another on a convolutional neural network. Subsequently, the report assesses the impact of a query selection strategy based on the disagreement among an ensemble of reward predictors. The report concludes with tests comparing an agent trained with a reward derived from human preferences with a benchmark forward reinforcement learning agent, demonstrating the promise of the proposed reward shaping strategy.

Contents

1 Introduction

1.1 Motivation

The present project continues the work on learning to build self-supporting structures in simulation by [1], which showed the potential of reinforcement learning (RL) in the achievement of this complex task shown in Figure 1. A serious challenge they faced and left open for further investigation is the problem of reward shaping. Solely rewarding the eventual success or failure of their task doesn't provide sufficient feedback for the agent to effectively learn. Instead, they resorted to the manual design and tuning of a simple reward function. This procedure has two major drawbacks, however. First, the tuning process is time-consuming. Second, there is a risk that the reward is not well aligned with the actual objective: the agent maximizing its rewards might be diverted from its actual objective [2].

To address the issue of reward shaping, several approaches have been tried in the literature. Some of these methods, such as inverse reinforcement learning [3] or imitation learning [4], [5] learn a reward from human demonstrations. These demonstrations can, however, be complex. Building a self-supporting arch spanning a large gap can be a difficult task for an untrained human, especially in simulation. It is indeed hard to grasp the effect of friction and properly assess the stability of the structure for simulated blocks.

Instead, we therefore opted for reinforcement learning from human preferences (RLHF) [2], [6], [7]. In this framework, the RLHF algorithm selects pairs of demonstrations of the agent performing the task and submits them to the human's review. The reward function is then learned from the human's preferences.

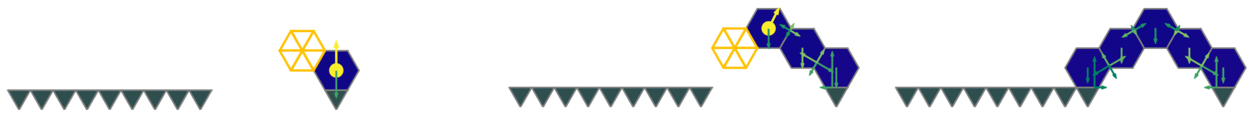


Figure 1: Demonstration of the successful construction of a structure spanning a gap of size 5 in the simulated environment used throughout this project.

1.2 Related Work

Several recent works on RLHF, including [2], [8], show the promise of learning from human preferences for a range of tasks such as simulated robotics and Atari games. In [2], humans were queried on a range of simulated robotic tasks implemented in MuJoCo [9]. Their experiments showed that a feedforward neural network reward model trained with human feedback could outperform standard RL in certain tasks when human raters were given hints such as preferring episodes where the Ant robot is "standing upright". The same researchers also queried humans on Atari tasks such as Breakout or Pong [10]. For these tasks, reward models based on convolutional neural networks failed to beat standard RL agents trained with the real rewards of the Atari games, except in the game of Enduro. In this endurance racing video game, in which the purpose is to pass a certain number of cars to stay in the race, humans proved better at identifying driving behaviors that are more likely to pass a car, thus outperforming standard RL.

Past work on RLHF has focused on learning rewards that can be expressed as a linear combination of handcrafted features [11], [12]. In this semester project, as a first step, we consider such a linear reward class. On the one hand, we consider this class of rewards since the underlying reward in the work of [1] belongs to this class, allowing us to compare our results. On the other hand, the coefficients of each feature are easily interpretable and solid theoretical foundations [12] exist. Recently, however, several works including [2], [8], [13] use neural networks to parametrize their reward model. In particular, [2] use a convolutional neural network (CNN) architecture to model their reward for Atari games. The CNN is particularly well-suited to the task of extracting hierarchical and spatial features from data [14], enabling it to more effectively learn complex patterns and relationships in an environment with a grid structure, such as the Atari games, than a linear combination of features. The bridge-building application that we are tackling takes place in a simulated 2D grid in which state encoding with CNNs has proved effective for policy networks [1]. As a next step we therefore switch to the more complex class of rewards parameterized by CNNs in the hope of better capturing human preferences We then compare the performance of linear and CNN-based reward models.

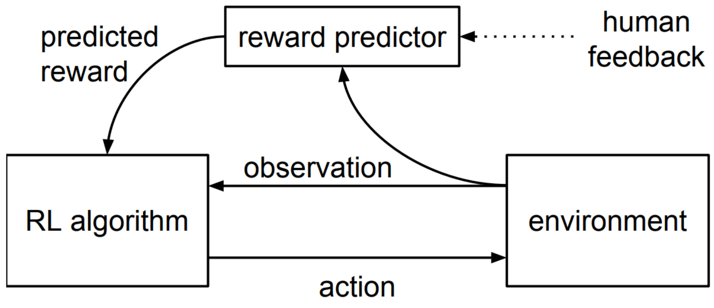


Figure 2: Schematic illustation of the RLHF algorithm from [2]: a reward predictor is trained with human feedback while a policy learns to optimize the predicted reward.

Several works on RLHF (theoretical and implementation-based), such as [2], [12], have proposed algorithms for reward learning from human preferences. In particular, [2] proposes an algorithm in which a reward model is trained from human preferences while an RL policy is trained to optimize the reward being learned. Their algorithm is illustrated in Figure 2. This semester project builds on their RLHF algorithm and connects it with the task of building self-supporting bridges that [1] tackled with forward RL. In [2]'s work, humans are asked to compare the performance of the agent on two segments of demonstrations of the task, to efficiently make use of human time. As the bridge-building task that this semester project tackles only takes a few seconds to complete, we queried comparisons of complete demonstrations rather than segments. Also, we modified the algorithm to take into account the fact that we deal with finite state spaces, whereas [2] only addressed continuous state spaces.

For RLHF to be applicable to real-world application such as the construction of a self-supporting spanning structure, sample efficiency is essential [2], [13]. Namely, the purpose of RLHF is to learn an effective reward function with minimimal human feedback. In a first step, we randomly sample pairs of demonstrations of two robots building a spanning structure, and query a human on which demonstration they prefer. We then compare this naive approach with a querying strategy based on *disagreement* [2], [15]. In this approach, an ensemble of reward models is maintained and a human is queried on their preference for pairs of trajectories for which the preference predictions have the highest variance among the ensemble of rewards.

1.3 Contribution

The main contributions of this semester project are:

- A maintainable and modular code for reinforcement learning from human feedback on the application of constructing a self-supporting spanning structure,
- An experimental study of learning to build a bridge with RLHF, benchmarked against forward Soft Actor-Critic RL,
- An experimental comparison of RLHF with two reward models for the task of building a bridge: linear with handcrafted features and CNN-based,
- An experimental assessment of the impact of disagreement-based querying strategies on the speed of convergence of RLHF,
- An experimental study of the effect of real human feedback, compared with synthetic feedback derived from an underlying handcrafted reward.

2 Background

The purpose of this semester project is to complement the work by [1] on building self-supporting bridges using reinforcement learning with reward shaping from human feedback. Before diving into RLHF, the present section therefore summarizes the key points in the work by [1] that are required to understand the contributions of the present project. It starts with a theoretical background on the regularized MDP framework used throughout this work and a brief presentation of the forward RL algorithm used. Subsequently, the task, state space, action space, transition matrix and reward of the bridge-building application are defined.

2.1 Regularized Markov Decision Process

We consider a regularized Markov decision process (MDP) defined by the tuple $(S, A, p, r, \gamma, \alpha)$, where S and A denote a finite state and action space, respectively, $p : S \times S \times A \to [0, 1]$ is a state transition probability distribution with $p(s_{t+1}|s_t, a_t)$ describing the probability of transitioning to state s_{t+1} when taking action a_t in state s_t, $r : S \times A \to \mathbb{R}$ is a reward, $\gamma \in (0, 1)$ is a discount factor, and α is the temperature of the regularized MDP.

The goal of entropy-regularized reinforcement learning is to find a Markov policy $\pi : S \to \Delta_A$, where Δ_A is the probability simplex over A, that maximizes the following expected entropy-regularized discounted cumulative reward $J(\pi)$:

$$J(\pi) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha H(\pi(\cdot|s_t)) \right) \right], \tag{1}$$

where $\mathbb{E}_\pi$ denotes the expectation over the trajectory distribution induced by the policy π, and $H(\pi(\cdot|s))$ denotes the Shannon entropy of the policy $\pi(\cdot|s)$, defined as

$$H(\pi) = -\sum_{a \in A} \pi(a|s) \log(\pi(a|s)). \tag{2}$$

The effect of entropy is to regularize the policy towards a uniform distribution and therefore to enforce some exploration.

2.2 Reinforcement Learning Algorithm

For standard reinforcement learning, which is used as a baseline and is also part of the RLHF algorithm (see Subsection § 3.2), the Soft Actor-Critic (SAC) [16] method is used.

SAC is an actor-critic method based on the maximization of an entropy-regularized discounted return (1). The purpose of the entropy bonus is to boost exploration. The SAC alternates between a soft policy evaluation and a soft policy improvement step.

As shown by [1], SAC performs well on the task of building a bridge, which this project addresses. In addition, the algorithm is well-suited for the reinforcement learning from human preferences framework, as demonstrated by [8]. As the focus of this semester project is on RLHF, we will from hereon consider SAC as a black-box forward RL algorithm.

2.3 Application: Building a Bridge

2.3.1 Task Description

For the task of building a bridge we consider the discretized environment depicted in Figure 3. In particular, we assume that the environment is a simulated 2D grid subject to gravity, in which the elementary shapes are equilateral triangles.

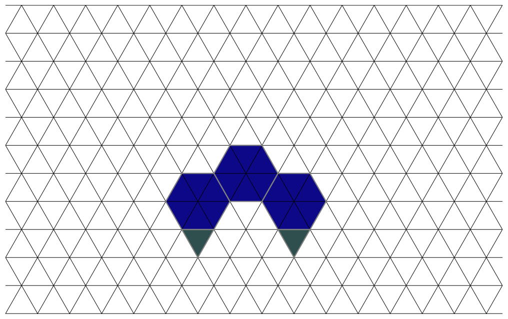


Figure 3: Illustration of the grid structure of the simulated environment used in this work, adapted with permission from [1].

In this environment, two simulated robots are tasked with building a spanning structure connecting two supports. We model the supports as downward-pointing triangles (the grey triangles in Figure 3) referred to as *grounds*. By self-supporting, we mean that the structure holds with friction and compression forces only.

To connect the two supports, the two robots are assumed to be centrally controlled. They alternately place and hold one hexagon-shaped block (represented in dark blue in Figure 3) at the time. The task is complete when the bridge connects the supports and holds without the assistance of the two robots as illustrated in Figure 3, where an arch connects the two supports. Note that for a successful completion of the construction, the bridge should not collapse at any point during the process. To add complexity to the task, several gap sizes separating the two supports have been tested.

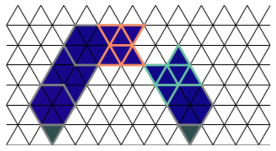
2.3.2 State Space

The state s_t of the environment at time t contains the following information about both the grounds and the supports of the spanning structure:

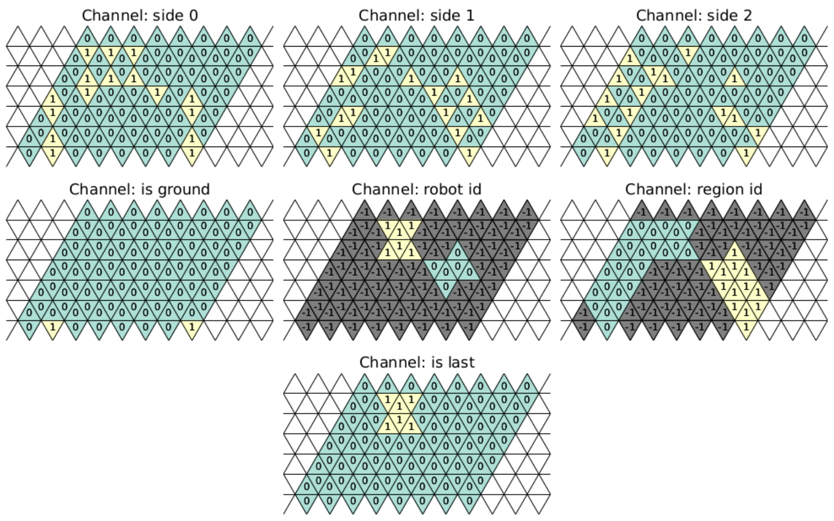
- The block type: triangle/hexagon,
- The position of the block in the grid,
- An indicator which signals whether the block is being held by a robot.

Given the combinatorial complexity of encoding the state described above, a convolutional neural network (CNN) is used to encode the state. The CNN's input consists of seven channels, depicted in Figure 4, which identify:

- The sides of the block,
- Whether the blocks are grounds,
- The ID of the robot holding the blocks,
- An identifier of the region the block belongs to, i.e., whether the block is connected to the left or the right support,
- A last channel to identify the last block, against which new blocks have to be placed.



(a) Human readable version



(b) Computer version

Figure 4: Illustration of the seven channels given as input to the CNN state encoder, taken from [1]. Note that the hourglass-shaped blocks shown in the picture were not used in this project.

2.3.3 Action Space and Transition Matrix

To build the bridge the agent can take the following action:

- Drop the block currently being held (if any), then place and hold the following hexagonal block against the block that was placed last.

Relative positioning is used to describe where the new block should be placed. Thus, choosing an action means choosing against which side of the existing structure to place the new block rather than giving its absolute coordinates.

The transition matrix is then defined as either adding a block to the structure or going to the terminal state. An episode terminates when one of the following events occurs:

- The structure collapses,
- Blocks collide, i.e. whether some blocks intersect each other,

- The two supports are successfully connected by a stable structure.

2.3.4 Reward

Fundamentally, the objective of the present task is to connect the two supports with a stable structure. However, merely rewarding the final success results in a too sparse reward which is difficult to optimize. To mitigate the problem of a too sparse reward, [1] designed a reward function $r : S \times A \to \mathbb{R}$ which is a linear combination of six features $\phi^i : S \times A \to \mathbb{R}$ with $i = 1, .., 6$. Their reward can be expressed as

$$r(s_t, a_t) = \boldsymbol{w}^T \boldsymbol{\phi}(s_t, a_t), \tag{3}$$

where $\boldsymbol{w} \in \mathbb{R}^6$ is a vector of coefficients and $\boldsymbol{\phi}(s_t, a_t) \in \mathbb{R}^6$ is the vector of features for state s_t and action a_t. In particular, rather than hand-tuning the vector of weights in this project, we aim to learn it from human feedback.

The six features correspond to the following:

- **Action**: a binary flag set to true every time an action is taken,
- **Closer**: a binary flag set to true if the latest action reduced the distance between the two sides of the bridge,
- **Success**: a binary flag set to true if the two supports are successfully connected by a stable structure,
- **Failure**: a binary flag set to true if a collapse or collision occurs,
- **Number of Sides**: the number of new interfaces created by adding a new block,
- **Number of Opposing Sides**: the number of opposing sides created by placing a new block.

3 Method

Now that the necessary background on the bridge-building task and its theoretical foundations have been introduced, this section presents the methodology employed to apply RLHF to the construction of self-supporting spanning structures. The section starts with a formal definition of the problem of learning rewards from human preferences. The RLHF algorithm adapted from [2] and its practical implementation for this project are subsequently detailed in the remaining subsections.

3.1 Problem Definition

As explained in Subection § 2.1, in the traditional reinforcement learning framework, we consider an agent that, at each time step t, makes an observation of its state $s_t \in S$, takes an action $a_t \in A$, and receives a reward $r(s_t, a_t) \in \mathbb{R}$. The agent's goal is to maximize its expected regularized discounted reward (1).

In reinforcement learning from human preferences, we consider that, instead of a reward signal, the agent receives feedback from a human. This feedback is expressed as a preference between a pair of trajectories σ, where a trajectory is defined as a sequence of state-action pairs $\sigma = ((s_0, a_0), \ldots, (s_{k-1}, a_{k-1})) \in S^k \times A^k$ made when the agent plays an episode in its environment. We write $\sigma_1 \succ \sigma_2$ to denote that the human expressed a preference for trajectory σ_1 over trajectory σ_2. The objective of our agent is to generate trajectories which are preferred by the human while minimizing the number of human queries made.

In practice, preferences over a pair of trajectories can be produced in two fashions:

- Quantitatively: We assume that preferences are generated by an underlying reward function $r : S \times A \to \mathbb{R}$ if

$$((s_0^1, a_0^1), \ldots, (s_{k-1}^1, a_{k-1}^1)) \succ ((s_0^2, a_0^2), \ldots, (s_{k-1}^2, a_{k-1}^2)) \tag{4}$$

  whenever

$$r(s_0^1, a_0^1) + \ldots + r(s_{k-1}^1, a_{k-1}^1) > r(s_0^2, a_0^2) + \ldots + r(s_{k-1}^2, a_{k-1}^2) \tag{5}$$

  Normally, the agent receiving preferences from a reward function should obtain high rewards under this r. From now on, preferences generated this way will be referred to as *synthetic feedback*. This form of feedback is interesting to validate our methodology,

- Qualitatively: A human expresses its preference between two trajectories which are shown to them in the form of a short video. This setting, in the absence of underlying reward function r, is the most interesting in practice.

From hereon, the human or synthetic feedback will be referred to as the *feedback-givers*.

3.2 Reinforcement Learning from Human Feedback: the Algorithm

The algorithm for reinforcement learning from human feedback (RLHF) is shown in Figure 5. During the RLHF process, a policy π and a reward estimate $\hat{r} : S \times A \to \mathbb{R}$ are updated according to the following steps, which are iteratively executed after randomly initializing the reward:

1. Train the policy π with standard reinforcement learning using the trained estimate $\hat{r}$ as a reward signal,
2. Generate a set of trajectories $\{\sigma_0, \ldots, \sigma_{K-1}\}$ with the policy π and regroup these trajectories by pairs (the strategies for pairing up will be discussed in Subection § 3.4),
3. Query the human (in the human feedback setting, query the underlying reward r in the synthetic feedback setting) to obtain preferences for each trajectory pair generated at the previous point. Store the resulting comparisons in a replay buffer R storing elements of the form $(\sigma_1, \sigma_2, \mu)$, where μ is a distribution over $\{1, 2\}$ indicating the human/synthetic preference,
4. Update the reward model $\hat{r}$ with supervised learning to fit the last N comparisons collected in R.

These steps are iterated over sequentially for a certain number of *rounds*. The four steps are discussed in more detail in the following subsections.

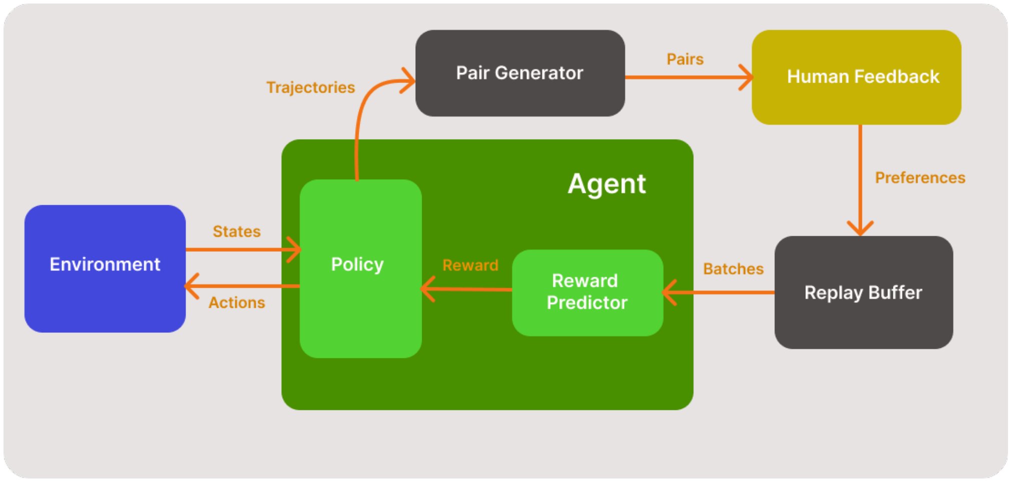


Figure 5: Schematic illustration of the RLHF algorithm used in this work. A policy is optimized using reward estimates from a reward predictor. This reward predictor is trained on batches of human preferences between pairs of trajectories generated by the policy being learned.

3.3 Policy Improvement

As explained in Subsection § 3.2, the steps of the RLHF algorithm are run sequentially. This means that during the policy training step, the reward estimate $\hat{r}$ remains constant and can be used as a reward signal for running standard reinforcement learning. The most conclusive algorithm for the bridge application according to [1], Soft Actor-Critic, was used for this policy improvement step. Besides, this RL algorithm has already been successfully applied by others [8] in the RLHF framework. The same hyperparameters as [1] were used for the policy and value networks (see Annex A.2 for the detailed hyperparameters). Research by [2] has empirically shown that the hyperparameters of policy networks trained with traditional RL are effective in RLHF too, despite the non-stationarity of the reward estimate $\hat{r}$.

3.4 Trajectory Generation and Query Selection

As explained in step 2 of the RLHF Algorithm (§3.2), the first step to generate comparisons for querying our feedback-giver is to generate a number K of trajectories $\{\sigma_0, \ldots, \sigma_{K-1}\}$. Two pairing strategies were tested in this project to generate pairs from this set: *random* and *disagreement*-based sampling.

The *random* sampling strategy simply consists in uniformly sampling pairs without replacement from the set of trajectories. This naive approach has nonetheless proved effective for other researchers [2].

The *disagreement*-based sampling is a strategy to query the feedback-giver more efficiently, i.e. reduce the number of queries for the RLHF algorithm to converge. The sample complexity of RLHF is indeed one of its major drawbacks [7], [8]. With this strategy, an ensemble of L reward predictors $(\hat{r}_0, \ldots, \hat{r}_{L-1})$ is maintained. These reward predictors are independently initialized (see Section § 3.6 for more details about the initialization) and trained. The queries are then generated with the following procedure:

1. Similarly to the *random* sampling strategy, uniformly sample $f \cdot K/2$ pairs without replacement from a set of generated trajectories, where $f \in \mathbb{R}^+$ is an oversampling coefficient,

2. The reward predictors $(\hat{r}_0, \ldots, \hat{r}_{L-1})$ each return their preference for all the trajectory pairs previously generated, resulting in a set of elements of the form $(\sigma_1, \sigma_2, (\hat{\mu}_0, ..., \hat{\mu}_{L-1}))$, where $\hat{\mu}_i$ is an estimate of the human preference μ over the trajectory pair (σ_1, σ_2) generated by $\hat{r}_i$ (see Subsection § 3.5 for more details about the preference signal μ),

3. For each trajectory pair, the variance of the preferences $(\hat{\mu}_0, ..., \hat{\mu}_{L-1})$ is then computed: this is what we will hereon call the *disagreement*,

4. The $f \cdot K/2$ trajectory pairs are then sorted in decreasing order of *disagreement*. The top $K/2$ pairs are then returned: they are the most efficient queries according to the *disagreement* metric.

The reasoning behind this *disagreement*-based sampling is pretty intuitive: it is more efficient to query the human for trajectories on which the predictions made by the reward estimates diverge the most.

3.5 Preference Elicitation

The trajectory pairs generated at the previous step are then compared by the feedback-giver, as illustrated in Figure 6. This feedback produces a set of comparisons of the form $(\sigma_1, \sigma_2, \mu)$, which are stored in a first-in, first-out (FIFO) replay buffer R of maximum size N. The value of the preferences μ are set the following way:

- If the human prefers the first trajectory $\mu = 1$,
- If they prefer the second $\mu = 0$,
- If both trajectories look as good $\mu = 0.5$,
- If the feedback-giver doesn't know which trajectory they prefer, the preference is not included in the buffer.

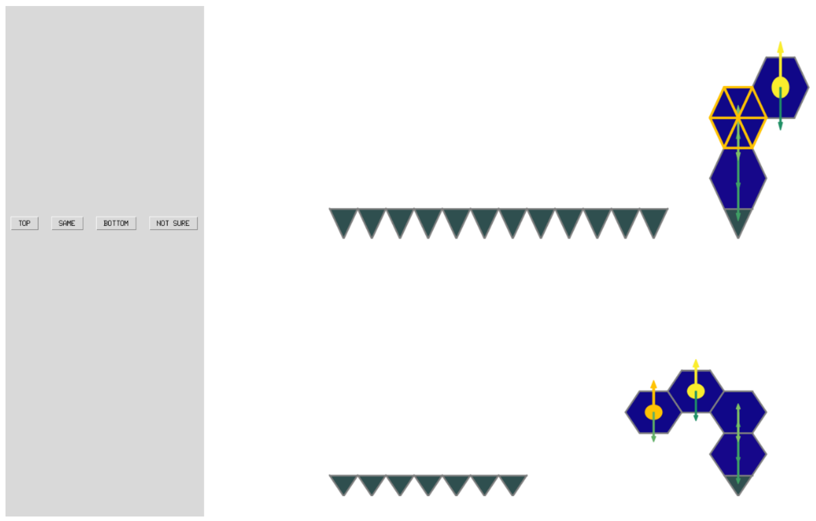


Figure 6: Illustration of the interface used for querying human feedback on a pair of trajectories. The feedback provider can choose between four options to express their preference: Top, Same, Bottom, and Not Sure.

3.6 Fitting the Reward Function

The reward function estimate $\hat{r} : S \times A \to \mathbb{R}$ can now be updated from the human preferences collected at the previous point. This reward estimate $\hat{r}$ can be seen as a predictor of the feedback-giver's preference.

According to the Bradley-Terry model [17], human preference can be modeled by the following expression:

$$\hat{P}(\sigma^1 \succ \sigma^2) = \frac{\exp \sum_t \gamma^t \ \hat{r}(s_t^1, a_t^1)}{\exp \sum_t \gamma^t \ \hat{r}(s_t^1, a_t^1) + \exp \sum_t \gamma^t \ \hat{r}(s_t^2, a_t^2)}, \tag{6}$$

where $\hat{P}(\sigma^1 \succ \sigma^2)$ is the probability that the human will prefer trajectory 1 over the other according to our preference model, and γ is the discount factor of the MDP (see Subsection § 2.1).

The reward estimate $\hat{r}$ is learned by minimizing the cross-entropy loss between the model's predictions and actual human preferences, sampled uniformly in batches from the replay buffer R:

$$\text{loss}(\hat{r}) = - \sum_{(\sigma^1, \sigma^2, \mu) \in R} \mu(1) \log \hat{P}[\sigma^1 \succ \sigma^2] + \mu(2) \log \hat{P}[\sigma^2 \succ \sigma^1]. \tag{7}$$

The reward fitting is performed using the Nadam Optimization Algorithm [18].

Two reward models were implemented and tested in this project: a *linear* and a *convolutional neural network (CNN)* model.

In the *linear* model, the reward function estimate $\hat{r}$ is expressed as a linear combination of the set of 6 features $\phi^i : S \times A \to \mathbb{R}, i = 1, .., 6$ handcrafted by [1] and presented in Subsection § 2.3.4:

$$\hat{r}(s_t, a_t) = \boldsymbol{w}^T \boldsymbol{\phi}(s_t, a_t), \tag{8}$$

where $\boldsymbol{w} \in \mathbb{R}^6$ is a vector of coefficients (the parameters to be learned) and $\boldsymbol{\phi}(s_t, a_t) \in \mathbb{R}^6$ is the vector of features for state s_t and action a_t. The coefficients $\boldsymbol{w}$ are randomly initialized using values sampled from a Gaussian distribution with zero mean and unit standard deviation, a standard procedure in machine learning [2]. At every training step, the vector $\boldsymbol{w}$ is normalized to have zero mean and the same Euclidean norm as the handcrafted reward of [1]. As explained by [2], this is a typical processing step as the reward learning problem is underdetermined. Also, we chose to make the learned reward have the same norm as in [1] to speed up hyperparameter tuning. Indeed, the performance of the Soft Actor-Critic algorithm (see Subection § 2.2) is particularly sensitive to the hyperparameter α, the entropy bonus, which dictates the relative weight of the reward and entropy terms in the return of Equation (1) [16]. To effectively reuse the hyperparameter values of [1], particularly their learning rate for α, it is therefore beneficial to make the coefficients $\boldsymbol{w}$ of the reward predictor $\hat{r}$ have the same norm.

The *CNN*-based model, on the other hand, is of a similar format to the policy and value networks of the SAC algorithm used for forward RL (see Subection § 2.2). It uses the same CNN-based encoder architecture (independently trained, however) and is also followed by a few fully connected layers (see Annex A.2 for the detailed hyperparameters). The main difference is that there is only one final output: the reward estimate for this particular configuration of the environment.

4 Experimental Results

The present part shows and discusses the experimental results obtained when applying the method developed in the previous section. It starts with a brief description of the experimental set-up. Next, the dynamics of learning with RLHF using a linear reward model trained with synthetic feedback are studied. The purpose of this first experimental study is to validate the methodology used. Subsequently, the performance of RLHF with synthetic feedback using a linear reward model (with and without disagreement for optimizing query selection) and a CNN reward model is benchmarked against forward RL with the handcrafted reward found by [1]. Finally, the results of RLHF with synthetic feedback are compared with RLHF using real human feedback.

4.1 Set-up

All the experiments that follow were carried out in a grid of size 15×15, with the objective of building a bridge crossing gaps of sizes 1 to 7. The friction coefficient employed is of 0.7, which approximately amounts to sticking the hexagonal building blocks together with mortar in the real world [1]. The work by [1] found that the tasks in this medium-sized environment and with this high friction coefficient are simple enough for forward RL, although gaps of sizes 6 and 7 start to be challenging. Figure 7 illustrates the successful construction of a bridge spanning a gap of size 7 in this 15×15 grid environment. As the main purpose of this project was to validate the methodology for RLHF described in the previous Section (§ 3), we chose to start by conducting tests in this environment of medium-low difficulty. Refer to Annex A.2 for detailed specifications of the environment used for the following experiments.

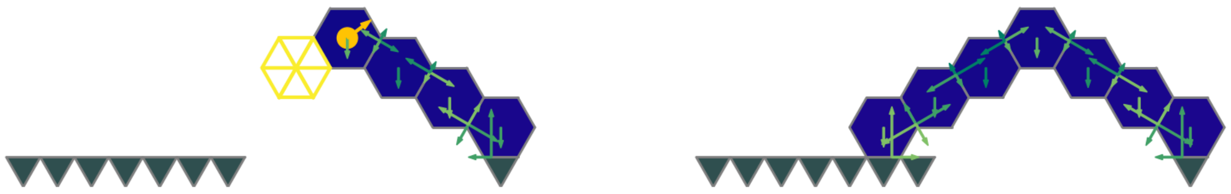


Figure 7: Successful and most efficient construction of a bridge spanning a gap of size 7.

4.2 Dynamics of Learning with a Linear Reward Model

4.2.1 The learned reward model converges towards the reward underlying the synthetic preferences.

In this first experiment, the RLHF algorithm was run with a linear reward model, as defined in Subsection § 3.6, and random query selection. This first RLHF training used synthetic human feedback, i.e. preferences derived from an underlying reward model. For the underlying reward, we used a linear model with the coefficients found empirically by [1]. The demeaned values of these coefficients are given in Table 1.

The initial coefficients of the reward model to be learned, also in Table 1 and initialized as described in Subsection § 3.6, were normalized to have a Euclidean norm of 5.40 (the same norm as the underlying reward model). The coefficients learned by running the RLHF algorithm over 20 rounds (see Annex A.2 for the detailed hyperparameters of the run) are given in Table 1.

discuss

	Action	Closer	Success	Failure	Sides	Opposing sides
RLHF linear, initial coefficients	-2.80	3.46	0.51	-1.37	1.22	-1.01
RLHF linear, after learning	-0.84	0.74	3.44	-2.80	-0.18	-0.361
Underlying reward [1]	-0.74	-0.14	4.46	-2.54	-0.49	-0.54

Table 1: Coefficients of the underlying feedback-giving reward model taken from [1] and the linear reward model being learned with synthetic feedback (at initialization and after learning). Note how the values of the learning reward predictor converged towards the underlying reward.

One can see that overall the values of the learned coefficients got fairly close to those of the underlying reward model that gave synthetic feedback. In particular, the sign of all the coefficients except that associated with the feature "Closer" are the same as those of the underlying reward. Given the high initial value of this particular coefficient (3.46, compared with -0.14 for the underlying reward), it is, however, no surprise that this coefficient is further off than most others. For the feature "Success", the coefficient is probably still a bit off because successes in the first few RLHF rounds were pretty rare. This coefficient stagnated for some time before finally increasing from round 10 onward.

Note that 20 rounds were not enough for the values of the coefficients to converge: some coefficients were clearly still on an upward or downward trend when the training finished. Unfortunately, only one GPU was available to run all the experimental results for this semester project. The length of the trainings run was adapted accordingly: the aim of the present results is therefore to highlight learning trends, rather than to study values at convergence.

4.2.2 The average return under the feedback-giving reward model increases during the RLHF training.

During the RLHF training rounds, we also tracked the evolution of the average discounted return under the feedback-giving reward model obtained by the agent being trained with the RLHF reward. The empirical average discounted return $\hat{J}(\pi)$ over n episodes is given by the following formula:

$$\hat{J}(\pi) = \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{\overset{\swarrow T_i ?}{T}} \gamma^t r(s_{i,t}, a_{i,t}), \tag{9}$$

where T is a final time step.

As the purpose of RLHF is to learn a reward that matches the preferences of the synthetic feedback given, this return should increase over the course of the training. Figure 8 confirms this trend and thus seems to validate our methodology: the average return steadily increases towards 5. Table 2 shows that this value is close enough to the average return of 6 obtained by an agent trained directly with forward RL using the feedback-giving reward, as in [1].

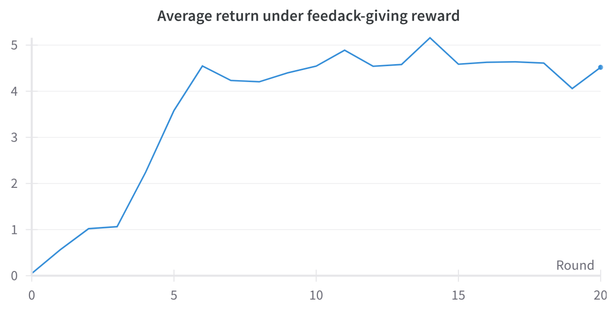


Figure 8: Evolution of the average return (9) under the feedback-giving reward from [1] during the RLHF training with linear reward model and synthetic feedback.

4.2.3 An agent trained with the learned reward performs almost as well as one trained with forward RL.

Finally, Table 2 summarizes the success rates in the bridge-building task for policies trained using forward RL with the feedback-giving reward and the learned reward. Both models appear to perform equally well, achieving a 100% success rate across all gap sizes when acting greedily based on the learned policy. The training curves in Figure 9 show, however, that under the softmax strategy used in training, the agent trained with the feedback-giving reward performs manifestly better, especially for the harder task of building a bridge crossing gaps of sizes 6 and 7.

A closer look at some sample constructions (see Figure 10) shows that a too high value for the coefficient associated with the feature “Closer” (see Table 1) might be at fault. Rewarding actions that reduce the distance with the opposing support seem to be beneficial in the first 1000 rounds of the policy training, as Figure 9b shows, but seem to hurt the learned policy in the long run. Indeed, the success rate under the feedback-giving reward converges to 90%, compared with only around 60% for the learned reward.

4.3 Study of Enhancement Techniques

The present subsection experimentally studies the effect of two improvements to our current RLHF algorithm suggested in the literature: optimizing query selection using disagreement (see Subsection § 3.4) and replacing the linear reward model by a convolutional neural network

?

	Average return	Success rate
Forward RL	6.03	1.0
RLHF Linear	6.17	1.0
RLHF Disagreement	6.05	1.0
RLHF CNN	-1.32	0.0
RLHF Human*	5.00	1.0

Table 2: Average success rates and returns under the benchmark reward from [1] over 100 episodes (mean over the 7 gap sizes) for the forward RL agents trained with the various rewards learned in this work and the benchmark reward found by [1].
* the RLHF training was interrupted after 11 rounds out of 20 due to technical issues. The forward RL training with the learned reward was carried out over as many episodes as the other agents, however.

(see Subsection § 3.6). Both tests of this subsection are carried out using synthetic feedback from the reward handcrafted by [1].

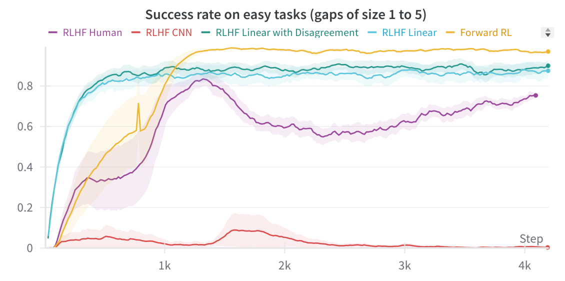
4.3.1 Effect of Disagreement on Learning

For the RLHF algortihm with disagreement-based sampling, pairs of trajectories were oversampled by a factor of 2 (see Subsection § 3.4) compared with the random sampling strategy of the previous point. Only half of the generated trajectories (those for which disagreement was the highest) were then used to query the synthetic feedback-giver. Disagreement was computed as the variance of the preferences predicted by an ensemble of 3 linear reward models trained independently.

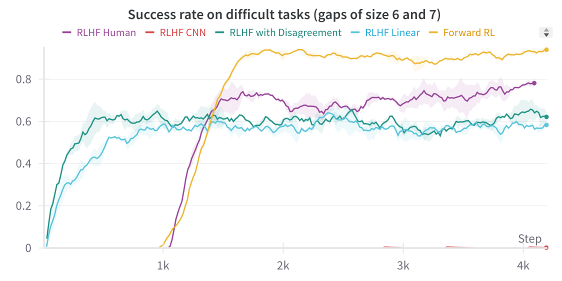
Table 3 shows that the coefficients learned also seemed to converge toward the feedback-giving reward. The values are slightly more off, however, than with the random sampling strategy. In particular, the coefficient rewarding “Success” is substantially lower than that learned without optimizing queries using disagreement.

Similarly, Figure 9 shows that the evolution of the success rates during policy training with the learned reward follows practically the same evolution as when not optimizing query selection.

On the other hand, querying with disagreement significantly lengthened trainings. Running the RLHF algorithm took 2 days and 13 hours when sampling queries randomly, compared with 3 days and 13 hours using disagreement-based sampling. Given the small benefits of the method in practice, which corroborate the findings by [2], we therefore chose to run the following experiments without optimizing queries.



(a) Evolution of the success rates for gaps of sizes 1 to 5.



(b) Evolution of the success rates for gaps of sizes 6 and 7.

Figure 9: Evolution of the success rates when running forward reinforcement learning with the benchmark handcrafted reward found by [1] and the rewards learned in the various RLHF trials of this section. The hyperparameters of the runs can be found in Annex A.2. The shaded areas correspond to standard errors.

	Action	Closer	Success	Failure	Sides	Opposing sides
RL [1]	-0.74	-0.14	4.46	-2.54	-0.49	-0.54
RLHF Linear	-0.84	0.74	3.44	-2.80	-0.18	-0.361
RLHF Disagreement	-0.40	0.51	2.28	-2.22	-0.10	0.05
RLHF Human*	0.41	0.76	3.07	-3.78	-1.25	0.78

Table 3: Coefficients of the linear reward models learned with RLHF in this work's experiments, compared with those of the handcrafted benchmark reward found by [1].
* this RLHF training was interrupted after 11 rounds out of 20 due to technical issues.

4.3.2 Linear vs CNN Reward Model

Next, taking inspiration from [2] among other works on Deep RLHF, we trained a CNN reward to model synthetic preferences, while randomly sampling queries. The average return under the feedback-giving reward (see Table 2) and the evolution of the success rates during forward RL training with the learned reward (see Figure 9) show, however, that the CNNs failed to match the performance of linear reward models. It is quite likely that the complex CNN we used (see Subsection 3.6) required many more episodes to converge. Given the duration of a complete run of the RLHF algorithm with this deep learning model (5 days and 8 hours on a GPU), we chose to leave further exploration of the effect of switching to CNN reward models for future work.

4.4 Human Feedback

Finally, we ran an RLHF training with linear reward model using real human feedback, while randomly sampling queries. Table 3 shows the reward coefficients learned. Interestingly, these

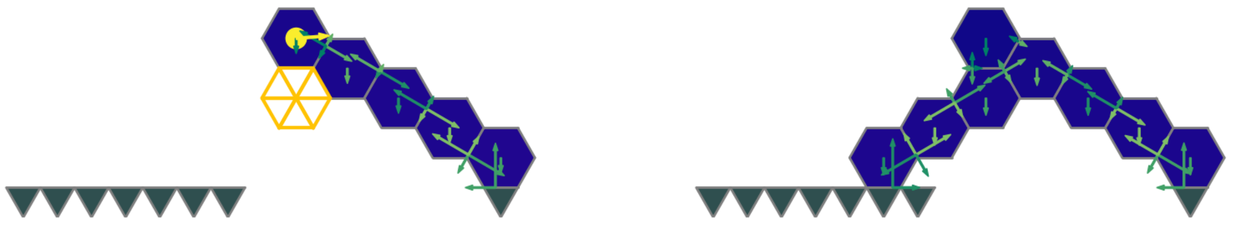


Figure 10: Rewarding actions that reduce the distance with the opposing side can result in constructions as the one shown here. The superfluous block on top of the structure was probably placed because it took the bridge closer to the other side, but it is of no use for the successful construction of a stable structure.

coefficients are quite similar to the empirical values found by [1]: successes are generously rewarded, whereas failures are heavily penalized (in fact, they are penalized even more). The main difference lies in the coefficients associated with the features “Action” and “Opposing sides”. The learned reward gives a slight bonus for those features, rather than penalizing them. It is quite likely that the early interruption of the RLHF algorithm is partly at fault. Taking more actions and creating opposing sides implies building more complex structures, which the human feedback-giver clearly preferred in the early phases of training, when most attempted constructions failed at the first or second block (see Figure 11). As the training progressed, however, both coefficients were clearly on a downward trend, getting closer to the values found empirically by [1].

Interestingly, Figure 9 shows that a policy trained with the reward derived from human feedback slightly outperforms agents trained with synthetic feedback on more difficult tasks. Excessively rewarding the features “Action” and “Opposing sides” clearly hurts the agent’s performance on simpler tasks, however. The complex structures these features encourage seem to be detrimental to the construction of a bridge crossing small gaps. This RLHF agent was not able to match the benchmark foward RL agent by [1] either, but the evolution of the reward coefficiens during training suggests that better performance might have been achieved if the RLHF algorithm had not been interrupted halfway. Overall, this first result using real human feedback is encouraging: rather than hand-tuning coefficients, deducing a reward model from human preferences seems to show some promise.

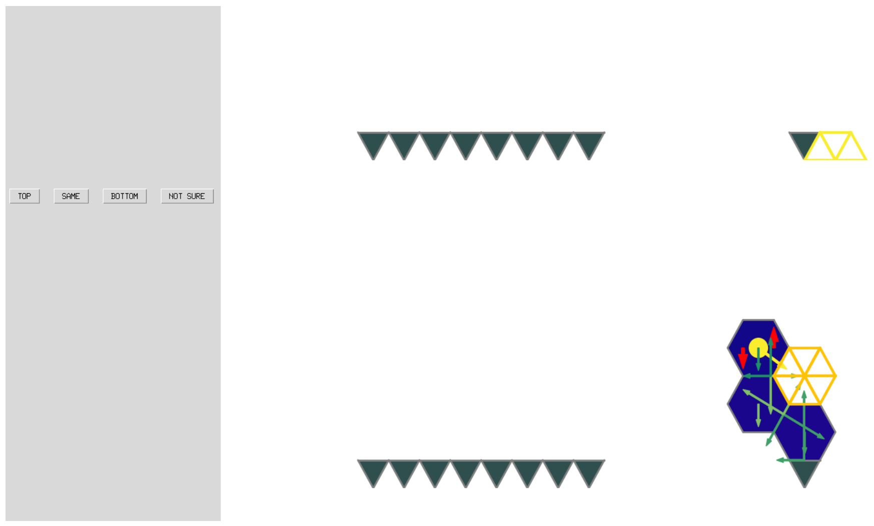


Figure 11: In the first rounds of the RLHF training, most demonstrations failed very early, as shown on top of this illustration. The human feedback provider therefore tended to favor relatively complex structures like the one below, even if several blocks were placed in nonsensical positions.

5 Conclusion

In conclusion, the successful completion of learning a reward from human preferences was achieved in the context of building a bridge with blocks in simulation. An experimental examination of the learning dynamics, employing a linear reward model and synthetic feedback, revealed the convergence of the learned reward predictor towards the underlying feedback-giving reward. Furthermore, a policy trained with this learned reward demonstrated performance nearly equivalent to that of a forward RL agent trained with the true underlying reward.

Moving forward, an experimental study investigating the impact of optimized query selection using the disagreement among an ensemble of independently trained reward models indicated minimal benefits to the training process. While the training duration increased, the rate of convergence was very similar to that of RLHF with random query selection.

Subsequent tests revealed the suboptimal performance of an RLHF agent trained with a CNN reward model compared to its linear RLHF counterparts. The complex network architecture probably requires more episodes to converge and achieve performance levels comparable to the simpler linear models, particularly on tasks of medium-low difficulty.

Finally, evaluations demonstrated that an RLHF agent trained with human feedback slightly surpasses RLHF agents trained with synthetic feedback on difficult tasks, although it does not match the performance of our benchmark forward RL agent. These first results are still encouraging and clearly show the potential of RLHF for efficient reward shaping, hopefully eliminating the need for manual fine-tuning of rewards in the future.

Please note that due to limited GPU capacity, most experimental tests were conducted only once. Consequently, both the results and the explanations provided above should be interpreted with caution. Ideally, these experiments should be replicated with different seeds to validate their repeatability.

6 Outlook

6.1 Challenges

The positive outcomes of this semester project have also highlighted a major challenge in RLHF. The training process is extended, with interruptions occurring at the end of each round, awaiting human input. This approach was time-consuming and impractical, especially when dealing with complex reward models like CNNs, potentially requiring a continuous human presence for days. To address this issue in the future, a possible solution could involve enabling feedback with a smartphone. This would allow training to proceed without waiting for the feedback provider to return to their office, speeding up the whole process.

6.2 Next Steps

While the present research has highlighted the potential of reward shaping from human preferences, there are areas where future improvements could be beneficial.

Firstly, the optimization of query selection strategies deserves further exploration. It would be valuable to reevaluate the use of disagreement-based sampling by testing it with a larger ensemble of reward models, perhaps 10 instead of the original 3. Additionally, various other strategies mentioned in the literature, such as information directed reward learning as discussed in [19] or using the disagreement among an ensemble of reward models as an intrinsic reward for the policy optimization step of the RLHF algorithm as in [13], should be considered and compared against the sampling approaches employed in this study.

Secondly, additional tests with CNN-based reward models should be conducted. Insights from the literature on deep Reinforcement Learning with Human Feedback (RLHF), as referenced in [2], [8], suggest that prolonged training periods are essential for optimal performance when using deep learning reward models. It is plausible that linear reward models may not capture enough complexity to address more challenging tasks. For example, [1] showed that building a bridge across a gap of size 7 with a lower friction coefficient of 0.5 is challenging for a linear reward model. The agent needs to understand the need for building small vertical towers to increase friction with the supports before proceeding with the construction of the bridge, as illustrated in Figure 12. Repeating longer experiments with CNN rewards guided by human expertise may outperform the manually crafted linear reward model proposed by [1].

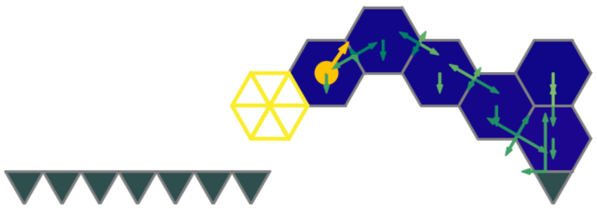


Figure 12: When the friction coefficient between the blocks is reduced, small vertical towers need to be built at the supports before constructing the bridge, as shown here (the lower the friction coefficient, the higher this tower should be). Otherwise, the structure collapses because it slips on the support.

References

[1] G. Vallat, J. Wang, A. Maddux, M. Kamgarpour, and S. Parascho, "Reinforcement learning for scaffold-free construction of spanning structures," in *Proceedings of the 8th ACM Symposium on Computational Fabrication*, 2023, pp. 1–12.

[2] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, vol. 30, 2017.

[3] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International conference on machine learning*, PMLR, 2016, pp. 49–58.

[4] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[5] B. C. Stadie, P. Abbeel, and I. Sutskever, "Third-person imitation learning," *arXiv preprint arXiv:1703.01703*, 2017.

[6] R. Akrour, M. Schoenauer, and M. Sebag, "Preference-based policy learning," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I 11*, Springer, 2011, pp. 12–27.

[7] C. Wirth and J. Fürnkranz, "Preference-based reinforcement learning: A preliminary survey," in *Proceedings of the ECML/PKDD-13 Workshop on Reinforcement Learning from Generalized Feedback: Beyond Numeric Rewards*, Citeseer, 2013.

[8] K. Lee, L. Smith, and P. Abbeel, "Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training," *arXiv preprint arXiv:2106.05091*, 2021.

[9] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2012, pp. 5026–5033.

[10] M. Towers, J. K. Terry, A. Kwiatkowski, *et al.*, *Gymnasium*, Mar. 2023. DOI: `10.5281/zenodo.8127026`. [Online]. Available: `https://zenodo.org/record/8127025` (visited on 07/08/2023).

[11] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia, *Active preference-based learning of reward functions*. 2017.

[12] B. Zhu, J. Jiao, and M. I. Jordan, "Principled reinforcement learning with human feedback from pairwise or K-wise comparisons," *arXiv preprint arXiv:2301.11270*, 2023.

[13] X. Liang, K. Shu, K. Lee, and P. Abbeel, "Reward uncertainty for exploration in preference-based reinforcement learning," *arXiv preprint arXiv:2205.12401*, 2022.

[14] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[15] C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters, "Active reward learning.," in *Robotics: Science and systems*, vol. 98, 2014.

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

[17] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952.

[18] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[19] D. Lindner, M. Turchetta, S. Tschiatschek, K. Ciosek, and A. Krause, "Information directed reward learning for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3850–3862, 2021.

[20] A. Gleave, M. Taufeeque, J. Rocamonde, *et al.*, *Imitation: Clean imitation learning implementations*, arXiv:2211.11972v1 [cs.LG], 2022. arXiv: `2211.11972 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2211.11972`.

A Annexes

A.1 Code Architecture

The full RLHF code[1], which was built as a complement to the modules written by [1], similarly follows an object-oriented approach. The following main modules, which take inspiration from the code of the *imitation* library [20], were added with the permission of [1]:

- Gym module: this module written by [1], which combines the RL policy and the environment, was modified to fit the other RLHF modules,
- Pair Generator: this module implements the sampling of pairs of trajectories, either randomly or using the disagreement among an ensemble of reward models,
- Preference Gatherer: this modules includes classes for synthetic and human feedback querying,
- Reward trainer: this module implements the various reward models used in this project, as well as a trainer class to facilitate the adjustment of trainings,
- Preference Comparison: this is the main class coordinating the modules above to run the RLHF algorithm.

Note that the modular and hierarchical structure with classes used in this code enables a user to independently modify the components above by inheriting the provided template abstract classes.

A.2 Hyperparameters

Grid size	15×15
Blocks available	hexagons
Gap range	1 to 7
Gap position	random gap
Friction coefficient	0.7
Robot max torque	0
Robot max force	1000
Max blocks	15

Table 4: Hyperparameters of the set-up of all the experimental results of Section § 4.

[1]the code is available on the following GitHub repository: https://github.com/Sabri2001/SycamoreProject

Dueling Q table	Yes
Fully connected layers	3
Number of neurons in the fully connected layer	64
Convolution layers	4
Kernel size	3
Number of internal channels (CNN)	64
Convolution stride	1
Last block only	Yes
Gamma	0.9
Batch size	512
Learning rate	0.0001
Target entropy	0.5
Tau	0.0005
Weight decay	0.0001
Initial alpha	1
Learning rate alpha	0.001
Lower bound on V	-2
Optimizer	NAdam
Number of training episodes	20000
Length of replay buffer	1000000

Table 5: Hyperparameters of the agent's state encoder (for both the actor and the critic network) and the SAC learning algorithm used for forward RL policy training.

Fully connected layers (CNN)	3
Number of neurons in the fully connected layer (CNN)	64
Convolution layers (CNN)	4
Kernel size (CNN)	3
Number of internal channels (CNN)	64
Convolution stride (CNN)	1
Learning rate (Linear)	0.0001
Learning rate (CNN)	0.00003
Weight decay (CNN)	0.0001
Batch size	32
Length of replay buffer	100
Optimizer	NAdam
Number of rounds	20
Policy training episodes per round	1000
Reward training episodes per round	1000
Number of queries	400
Query schedule	hyperbolic
Reward models in ensemble (Disagreement)	3

Table 6: Hyperparameters of the reward models (both linear and CNN) and the RLHF learning algorithm.