Exploring the Synergy between Inverse Reinforcement Learning and Reinforcement Learning From Human Feedback for Query Reduction

Ana Bătrîneanu

Supervisors: Luciano Cavalcante Siebert, Antonio Mone

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2022

Name of the student: Ana Bătrîneanu
Final project course: CSE3000 Research Project
Thesis committee: Luciano Cavalcante Siebert, Antonio Mone, Wendelin Böhmer

Abstract

Reinforcement Learning is a powerful tool for problems that require sequential-decision-making. However, it often faces challenges due to the extensive need for reward engineering. Reinforcement Learning from Human Feedback (RLHF) and Inverse Reinforcement Learning (IRL) hold the promise of learning a reward function without manual encoding. While RLHF uses feedback to estimate a reward function, IRL learns from demonstrations, examples provided by a teacher. In practice, both approaches have their advantages and disadvantages. IRL typically learns faster, provided that demonstrations are correct and sufficiently diverse. However, obtaining optimal demonstrations is inherently hard, since a teacher may not cover all possibilities, and their examples might fail to demonstrate the behaviour intended. Interactive feedback is believed to be easier to provide than demonstrations. However, RLHF suffers from the curse of dimensionality and the learner's random behavior at early learning trials. It also requires a large number of evaluative feedbacks, queries to a human labeler. We propose a learning framework in which these two approaches would potentially benefit from one another, with the purpose of investigating whether we can reduce the number of queries RLHF needs. Furthermore, we use Adversarial IRL (AIRL) and RLHF with preference comparisons. We examine our approach in two experimental studies. Our results indicate that combining AIRL with RLHF yields promising outcomes, but the effectiveness highly depends on the nature and number of demonstrations, and the specifics of the environment.

Keywords: *Reinforcement Learning from Human Feedback, Adversarial Inverse Reinforcement Learning, Preference Comparisons, Learning from demonstrations and feedback, Proximal Policy Optimization*

1 Introduction

Reinforcement Learning (RL) is a machine learning technique which focuses on goal-directed learning from interaction with an environment. In RL, an agent traditionally navigates through an environment and attempts to make optimal actions or decisions through a process of trial and error, being guided by manually defined reward signals (Kaufmann et al. 2023). Although RL algorithms assume the existence of a reward function which the agent tries to maximize, in complex applications, it may be impossible to hard-code one. For example, suppose that we want to train a robot to assist humans in a household environment. In this case, it is unclear how to design a suitable reward function, since it must account for the dynamic nature of the environment, the diverse range of tasks the robot might encounter, and the varying preferences and expectations of different users. As a consequence, teaching by demonstrations and interactive learning have become a powerful replacement for manual coding and behavior tuning (Ezzeddine et al. 2018). Interactive learning techniques can be categorized into two major clusters: Learning from Feedbacks (LfF) and Learning from Demonstrations (LfD) (Ezzeddine et al. 2018).

LfF assumes the existence of a trainer which iteratively evaluates the agent's behavior and provides feedback in different formats (binary comparisons, trajectory rankings, numeric rewards, etc.) in order to improve the agent's policy (state-action mapping) (Ezzeddine et al. 2018). Reinforcement Learning from Human Feedback (RLHF) is the RL technique in which an initially unknown reward function is learned through human feedback. The learned reward function is further on used to optimize the agent's policy. The trainer is a critical component, referred to as human-in-the-loop (HITL). This approach overcomes the limitations of traditional methods to define a reward function. Also, benefits for agent alignment (the agent's learning goals are more closely aligned with human values, promoting ethically sound and socially responsible AI systems) have been claimed to arise from this approach by (Kaufmann et al. 2023).

In contrast, in LfD the agent tries to learn its policy by observing the trainer demonstrations (Ezzeddine et al. 2018). LfD techniques can be further classified into two main categories: "Direct imitation learning[1]" and "Apprenticeship learning". This paper focuses on the latter, which is also referred to as Inverse Reinforcement Learning (IRL). These methods aim to infer the underlying goals and motivations of the expert. In IRL, the agent learns a reward function by observing demonstrations provided by a teacher.

IRL ((Abbeel and Ng 2004), (Ratliff, Bagnell, and Zinkevich 2006), (Ziebart et al. 2008)) and RLHF ((Christiano et al. 2023), (Lee et al. 2023)) have been extensively studied on their own. There is also some research around combining the two methods in order to benefit from the complementary properties of IRL and RLHF in realistic situations ((Ezzeddine et al. 2018), (Argall, Browning, and Veloso 2007), (Nicolescu and Mataric 2003)). This integration between the two can be effective to overcome some challenges in the learning process, such as the correspondence problem[2] between the trainer and the agent, and to make up for the unseen situations that are not covered in the demonstrations (Ezzeddine et al. 2018). Moreover, the mentioned papers investigate the synergy between IRL and RLHF, by either using supervised learning approaches to directly learn the policy, or by investigating how human feedback can overcome the challenge of non-optimal demonstrations. However, none of them focuses on investigating whether the number of queries from the agent to a trainer is reduced when integrating IRL in the RLHF process. This gap creates an opportunity for our research.

This paper aims to combine the two RL methods and to particularly investigate: *"To what extent can IRL complement RLHF to reduce the number of queries RLHF needs?"*.

The rest of the report is structured as follows: Section 2 aims to provide background information by recalling the basic settings of RL, IRL and RLHF algorithms. Section 3 describes the methodology used in this research, while section 4 outlines the experiments conducted in order to answer the main research question. Section 5 goes over the results achieved through the experiments and analyses them, while

[1] Direct imitation learning comprises those methods that use supervised learning algorithms to directly imitate the expert's policy.

[2] Correspondence problem due to some differences in physical embodiment and perception of the trainer and the learner (Ezzeddine et al. 2018).

Fig. 1: The basic RL setting

section 6 includes a comprehensive discussion of the results achieved. Section 7 addresses the topic of responsible research within our study, followed by section 8 which outlines the conclusions and future research directions.

2 Background

In this section we focus on describing the most important concepts of RL, RLHF and IRL.

2.1 Reinforcement Learning Terminology

In RL, an agent interacts with an environment by choosing actions based on its current state, receives rewards, and follows a policy to maximize a long-term cumulative reward throughout episodes. (Sutton and Barto 2018) provide a comprehensive overview of RL.

The basic interaction setup can be visualized in Fig. 1. The agent chooses an action at time t given its current state and based on its current policy (π_{s_t}). Afterwards, the agent moves to the next state s_{t+1} in a deterministic or stochastic manner, and receives the immediate reward r_{t+1}. Consequently, the interaction loop is restarted. In this setting, the reward function is known beforehand. The RL agent aims at learning a policy that maximizes the expected return. The policy is not fixed; it is iteratively improved through a process called policy iteration. In policy iteration, the agent alternates between evaluating how good the current policy is (policy evaluation) and making the policy better (policy improvement / optimization).

2.2 Markov Decision Process

RL is a perfect fit for problems that require sequential-decision-making (a series of decisions that all affect one another). Therefore, the learning environment in RL is formalized as a Markov decision process (MDP). An MDP is an extension of a Markov Chain, through the addition of actions and rewards. In an MDP, an agent iteratively observes its current state, takes an action that causes the transition to a new state, and finally receives a reward that depends on the action's effectiveness (Kaufmann et al. 2023).

The canonical MDP is defined as a tuple (***S***, ***A***, $P^a_{s,s'}$, **R**, γ), where:

- ***S*** is a finite set of states. A particular state at time t is denoted using S_t.
- ***A*** is a finite set of actions. A particular action at time t is denoted using A_t.
- $P^a_{s,s'}$, transition function = $P(s' \mid s, a)$ (the probability to move to state s' by taking action a, given the current state s).
- **R** : $S \times A \to \mathbb{R}$ represents the reward function.
- $\gamma \in [0, 1]$ represents the discount factor (a constant quantifying the importance of short-term and long-term rewards).

We call the value R(s, a) an instantaneous reward, the immediate feedback received by performing action a in state s. In an MDP, an H-step trajectory (where H is often referred to as *horizon*) is a sequence of state-action pairs, ending in a terminal state. It is represented as $\tau = (s_0, a_0, s_1, a_1, ..., s_H)$. We define a segment $\sigma = (s_{t_0}, a_{t_0}, s_{t_0+1}, a_{t_0+1}, ..., s_{H'})$ as a continuous sequence of steps within a larger trajectory, where $t_0 \geq 0$ and $H' \leq H$.

A trajectory τ's return value is represented by the following formula:

$$R(\tau) = \sum_{h=0}^{H-1} \gamma^h R(s_h, a_h) \tag{1}$$

(1) is the cumulative discounted sum of rewards along the τ trajectory. Note that **R** can represent two different values, depending on its signature: return value of τ trajectory or the value of the reward function given a state and an action.

2.3 Proximal policy optimization algorithm

Proximal Policy Optimization (PPO) is an on-policy algorithm. In short, on-policy learning algorithms evaluate and improve the same policy which is being used to select actions.

PPO is part of the policy-gradient family of methods, and more specifically, it belongs to the actor-critic family. Those methods are able to handle continuous actions with ease. Policy gradient methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm. Standard algorithms belonging to this family perform one gradient update per data sample. PPO differentiates itself by having a novel objective function (Clipped Surrogate Objective function) that enables multiple epochs of minibatch updates. The Clipped Surrogate Objective is a replacement for the policy gradient objective, and is designed to improve training stability by limiting the change you make to your policy at each time step (Schulman et al. 2017).

For our research, we have chosen to use PPO for both RLHF and IRL algorithms, because of its scalability (to large models and parallel implementations), data efficiency and robustness (it is successful on a variety of problems without hyperparameter tuning) (Schulman et al. 2017).

2.4 Reinforcement Learning From Human Feedback with Preference Comparisons

A generic RLHF algorithm consists of repeating two phases (Kaufmann et al. 2023):

1. Reward Learning
   - Generate queries to ask the oracle (human labeler).
   - Train a reward function approximator with the answers provided by the oracle.
2. RL training

Algorithm 1: Generic RLHF Algorithm in an Actor-Critic Scheme

Initialize parameters θ (policy), ϕ (critic), and ψ (reward)
Initialize replay buffer β with randomly-generated trajectories
Initialize database *D* of triples (σ^1, σ^2, μ), where σ^1 and σ^2 are two segments and μ is a distribution over $\{1, 2\}$ indicating which segment the oracle preferred
for i = 1, ..., *N* **do**
    // Reward learning
    Generate queries from β
    Update *D* with answers to queries from the oracle
    Update ψ using *D* (e.g., to minimize Eq. 3)
    // RL training
    Update β with new trajectories generated with π_θ
    Update θ (actor) using β and R_ψ
    Update ϕ (critic) using β and R_ψ
end for

- Running a deep RL algorithm using the currently trained reward function approximator.

This algorithm is summarized in Algorithm 1 (Kaufmann et al. 2023). If the oracle marks the segments σ^1 and σ^2 as equally preferable, then μ is uniform. Finally, if they are marked as incomparable, then the comparison is not included in the database (Christiano et al. 2023). The algorithm assumes an off-policy actor-critic[3] scheme for the RL training phase. However, other RL policy learning approaches can be used, such as PPO (on-policy algorithm), which we will also use in our research. For PPO, only the recently generated transitions are used for training.

In this paper we have chosen to focus on RLHF with preference comparisons as feedback type, a commonly used approach. In addition, (Christiano et al. 2023) claim in their paper to have found with predicting comparisons better performance than with predicting scores for continuous control tasks. Throughout our research, we will use the term "queries" and "preference comparisons" interchangeably to indicate the number of interactions, or the amount of feedback, between the agent and the human labeler.

The preference comparisons algorithm learns a reward function from preferences between pairs of trajectories. The comparisons are modeled as being generated from a Bradley-Terry (or Boltzmann rational) model (Eq. 2), where the probability of preferring trajectory A (σ^1) over B (σ^2) is proportional to the exponential of the difference between the return of trajectory A minus B. In other words, the difference in returns forms a logit for a binary classification problem, and accordingly the reward function is trained using a cross-entropy loss (or equivalently using the maximum likelihood principle) to predict the preference comparison ((Christiano et al. 2023), (Gleave et al. 2022)). Eq. 3 illustrates the cross-entropy loss between the predictions and the actual human labels.

$$\hat{P}[\sigma^1 \succ \sigma^2] = \frac{exp \sum \hat{r}(o_t^1, a_t^1)}{exp \sum \hat{r}(o_t^1, a_t^1) + exp \sum \hat{r}(o_t^2, a_t^2)} \quad (2)$$

where $\hat{r}$ = reward function estimate

$$loss(\hat{r}) = - \sum_{(\sigma^1, \sigma^2, \mu) \in D} \mu(1) log\hat{P}[\sigma^1 \succ \sigma^2] + \mu(2) log\hat{P}[\sigma^2 \succ \sigma^1] \quad (3)$$

2.5 Adversarial Inverse Reinforcement Learning

IRL algorithms seek to infer a reward function given a set of demonstrations ($D = \{\tau_1, ..., \tau_N\}$) of behavior in the environment. Demonstrations are usually assumed to be provided by an expert (human teacher), and come in the form of trajectories composed of state-action pairs. Another assumption is that the expert is following an optimal policy. However, in recent work, these assumptions have been relaxed (Adams, Cody, and Beling 2022).

Adversarial Inverse Reinforcement Learning (AIRL) is a practical and scalable IRL algorithm based on an *adversarial reward learning formulation*. AIRL uses a generative adversarial network (GAN) approach (Goodfellow et al. 2020), where a discriminator (esentially acting as a reward model) and a generator (policy optimization agent) are trained simultaneously.

The main idea behind the AIRL algorithm is the following: The discriminator model utilizes both expert demonstrations (which remain constant throughout the training) and the latest agent's trajectories as input (generated by the policy being learned, the generator). Its aim is to distinguish between the two sets of demonstrations. This process implicitly constructs a reward function. The policy, on the other hand, aims to produce state-action pairs that the discriminator cannot distinguish from the expert's and is updated based on the reward signal inferred from the discriminator. The entire process is iterated to refine both the policy and the discriminator.

For our research, we have chosen to focus on the AIRL algorithm. Other options included the Maximum Entropy IRL algorithm (Ziebart et al. 2008), which models behavior by choosing the probability distribution over trajectories that maximizes entropy while aligning with the observed feature expectations. We opted for AIRL due to its ability to recover reward functions that are robust to changes in dynamics, enabling us to learn policies even under significant variation in the environment seen during training (Fu, Luo, and Levine 2018). The authors of the paper mention that rewards recovered by AIRL generalize better than those produced by previous methods, and they demonstrate that AIRL performs well even in high-dimensional control tasks.

3 Methodology

This section highlights the methodology used in investigating our main research question. We first discuss related work to our approach. Subsequently, we introduce our proposed learning framework which combines IRL and RLHF.

[3]The actor-critic algorithm is a type of RL algorithm that combines aspects of both policy-based methods (Actor) and value-based methods (Critic)

3.1 Related work

The authors of the following paper (Ezzeddine et al. 2018) developed a methodology for learning from non-optimal demonstrations and human feedback by combining IRL and RLHF. Their methodology involved three phases: first, an agent learns from a set of non-optimal demonstrations using a modified version of the Maximum Likelihood IRL (MLIRL) algorithm to derive an initial policy; second, the agent's behavior is improved through an interactive learning phase where human evaluative feedback is incorporated; and third, the improved policy and initial demonstrations are combined in an extended IRL algorithm to refine the reward function further. The feedback mechanism they used involves the trainer providing explicit negative feedback through a keyboard in response to incorrect actions. Positive feedback is assumed when no negative feedback is given, reducing the burden on the trainer. This interactive phase continues until the trainer is satisfied with the agent's behavior. Moreover, their methodology is tested on 3 environments: a grid-world navigation task, a highway car driving, and a robotic case study. Finally, their paper demonstrates that their proposed approach successfully learns the desired behaviors from non-optimal demonstrations combined with human evaluative feedback.

Our approach, as we will describe in the next section, is inspired and based on the methodology implemented by (Ezzeddine et al. 2018), and it is a simpler version of their framework. We also use a different IRL algorithm, and a different procedure of providing feedback.

3.2 AIRLHF framework

The suggested framework in this paper is outlined in Fig. 2. We will call AIRLHF the combination between the AIRL algorithm and the RLHF algorithm. The algorithm consists of three phases, respectively:

- **Phase 1**: An AIRL agent learns a reward function from the given expert demonstrations and derives a preliminary policy.
- **Phase 2**: A RLHF agent receives R_{AIRL}, π_{AIRL}, learned during Phase 1, as initializations (thus, becoming an AIRLHF agent). The agent's reward function is improved, guided by the human trainer who provides feedback through preference comparisons. Once the robust reward function is learned, it is used to re-learn the policy of the environment and to optimize it (π_{AIRLHF}).
- **Phase 3**: The improved and final reward function (R_{AIRLHF}) and the final policy derived (π_{AIRLHF}) in the precedent phase, are evaluated. The performance of the AIRLHF agent is compared to the performance of an individual RLHF agent.

To restate, RLHF suffers from random behavior at early learning trials, especially when learning starts from scratch. As a consequence, it needs a large number of evaluative feedbacks and has a slow convergence rate to the desired policy. Hence, by initializing RLHF with a reward and policy learned from AIRL, we make sure RLHF's learning does not start from scratch, with the intention to accelerate its learning process and lower the number of queries it needs.

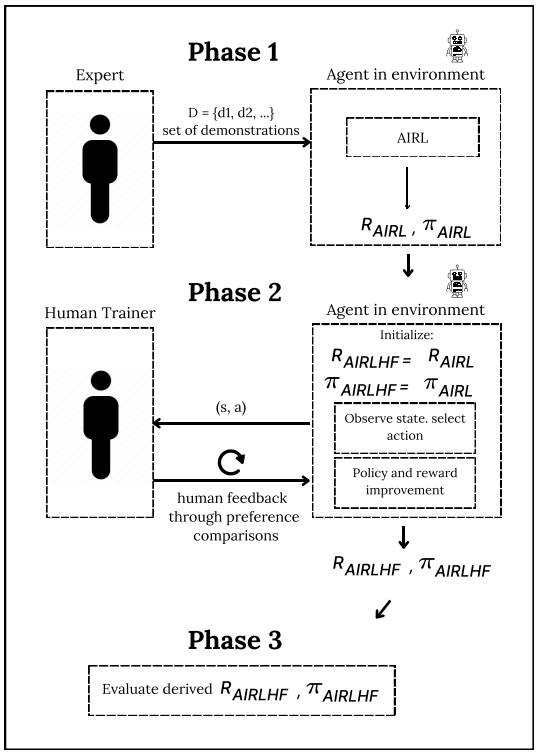


Fig. 2: AIRLHF Algorithm

IRL algorithms are influenced by the number of demonstrations they receive. In order to investigate to what extent IRL can complement RLHF, we believe it is sensible to also vary, alongside the number of queries RLHF needs, the number of demonstrations our AIRL algorithm will learn from. To do so, we will investigate two scenarios for our AIRL agent, respectively:

1. AIRL agent learns from a varying number of *optimal, expert demonstrations*.
2. AIRL agent learns from a varying number of *suboptimal demonstrations*.

In real-world applications, the conditions of sufficiently diverse and highly optimal demonstrations are hard to meet. Moreover, as described in the section 3.1, the focus of the work of (Ezzeddine et al. 2018) is also on enhancing learning by overcoming non-optimality in demonstrations. In their work, they also mention the fact that most IRL papers use the traditional assumption of the IRL that demonstrations are optimal or near to the optimal. As a consequence, very few papers consider suboptimality in demonstrations, and more research is needed in this area. Therefore, we deem it is important to also investigate the impact an IRL algorithm with manually defined suboptimality can have upon a RLHF algorithm.

3.3 Synthetic feedback and demonstrations

Due to the limited time of our research, we have only made use of feedback (queries) and demonstrations generated syn-

thetically. That is, no real-human data was collected and included in our research. However, using synthetic data ensures consistency and reproducibility in our experiments and provides a clear benchmark for evaluating our RLHF and AIRLHF implementations. In the future, our research can be extended with human-generated data.

4 Experimental setup

In this section we will present our experimental setup. In Appendix A we outline the technologies and some more implementation details we have used in order to set up our experiments.

4.1 Experimental Setup

We evaluate the approach presented in this paper through two case-studies: two classic control environments, **CartPole-v0** (see Appendix B) and **Pendulum-v1** (see Appendix B). These environments are commonly used to benchmark RL algorithms due to their simplicity and the control challenge they present. We deemed these environments to be good starting points in the area of this research, and we believe our results on them will be useful for future directions.

For each environment, we have settled the same experimental setup, which can be depicted in Fig.3. We introduced suboptimality into our demonstrations by adding a controlled level of randomness. For this research, we focused on settings where there was a 10% chance of taking a random action and a 20% chance of switching between the expert and random actions. Due to the limited time of our research, we impose a threshold on the number of demonstrations we will experiment with, equal to the number of comparisons of the baseline RLHF agent.

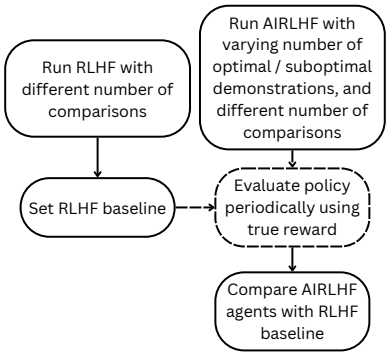


Fig. 3: Experimental setup

CartPole-v0 Environment. The objective in this environment is to balance a pole on a moving cart by applying forces to the left or right. For every step taken, there is a +1 reward. The CartPole problem is considered to be solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials (Brockman et al. 2016).

Pendulum-v1 Environment. The goal in this environment is to swing a pendulum to keep it upright by applying torque. The reward is designed so that a better performance corresponds to a reward closer to 0.

Evaluation. For each of the figures that will follow in Section 5, we have used an identical process of evaluation: while the policy is learned, we evaluate it periodically on the true reward, for a number of times. At each evaluation point, the agent's performance is tested over 5 episodes with a separate test environment. We refer to a test environment as an environment that does not have the reward learned through RLHF, but the true reward. The mean reward across these episodes is calculated and logged against the corresponding time step. At the end of the process, we evaluate the final derived policy on the true reward, over 100 episodes. We report these results in Table 1, which includes useful information with regards to the reward mean, standard deviation and 95% confidence interval achieved.

Note. Deep RL seems fairly brittle with respect to random seeds in a lot of common use cases (Islam et al. 2017). Random seeds can give a weaker / stronger impression on the performance than the reality confirms. Therefore, it is important to test out different seeds in our environments. As a consequence and as (Islam et al. 2017) indicates, we will average many trials using different random seeds in our experiments.

5 Results and Analysis

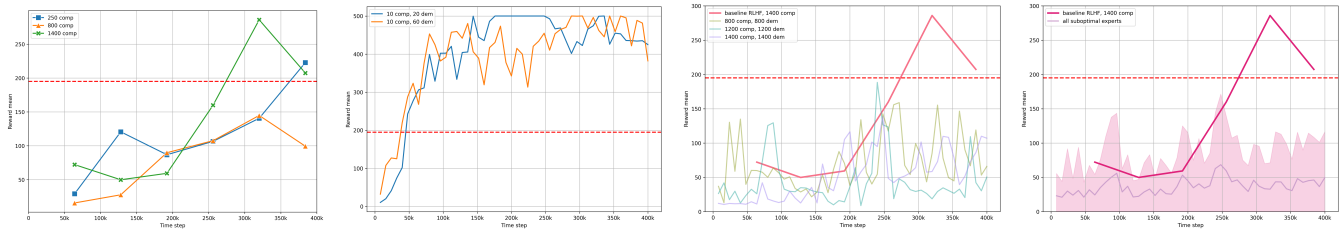
The aim of this section is to present and analyze the results gathered from the experiments presented in the previous chapter. We will first discuss our results on the CartPole-v0 environment, followed by the Pendulum-v1 environment.

5.1 Results on CartPole-v0

RLHF baseline. The benchmarks achieved on RLHF can be depicted in Fig. 4a, which highlights the evaluation of 3 RLHF learners, with a varied number of comparisons: 250, 800 and 1400 (numbers inspired from (Islam et al. 2017)). Given our collected results, we arrive at the conclusion that our RLHF agents with 250 comparisons and 800 comparisons perform similarly, but do not manage to solve the CartPole-v0 problem, as their reward mean does not go past 195.0. However, our **RLHF agent with 1400 comparisons** managed to solve the environment and was chosen as the baseline.

AIRLHF with optimal demonstrations. We will first illustrate the results gathered from *AIRL with optimal demonstrations + RLHF*, which can be depicted in Fig. 4b. Analyzing the results gathered, we see how we can set the amount of queries as low as 10, and still achieve outstanding numbers. In both scenarios, the agents successfully achieve a reward mean close to 450. Therefore, both agents solve the CartPole problem. Comparing these results to our RLHF baseline of 1400 comparisons, we conclude that we can decrease the number of queries **from 1400 to 10** (approximately 93% decrease).

AIRLHF with suboptimal demonstrations. Subsequently, we analyze the results gathered from *AIRL with*



(a) RLHF agents with varying comparisons.

(b) AIRLHF agents with different comparisons and optimal demonstrations.

(c) AIRLHF agents with different comparisons and suboptimal demonstrations, compared to the RLHF baseline agent.

(d) A comparison between our selected RLHF baseline agent and the mean-curve of all suboptimal AIRLHF agents.

Fig. 4: Performance on **CartPole-v0** as measured by the true mean reward over time. Each learner (curve) represents the mean of 5 runs with different seeds (34, 43, 52, 61, and 70). For (a), evaluation was done 6 times with 8 parallel environments. For (b), (c) and (d), evaluation was done 400 times with 8 parallel environments. The horizontal dotted line at 195.0 reward mean represents the baseline of solving this environment.

suboptimal demonstrations + RLHF. Fig. 4c outlines the top 3 AIRLHF agents with the highest reward means achieved, against our selected baseline RLHF agent, and Fig. 4d provides a closer look between the baseline agent and a mean of all the suboptimal experts we tested. More than half of the suboptimal AIRLHF agents we experimented with were far from the performance achieved by the RLHF baseline. The ones that came closer are only the top 3 aforementioned. However, we get the idea that in this scenario, we **cannot reduce the number of queries**, since the suboptimal agents perform worse, or barely similar to our baseline agent.

5.2 Results on Pendulum-v1

RLHF baseline. We first outline the benchmarks set by running RLHF on Pendulum-v1. Fig. 5a describes the evaluation of 3 RLHF learners on the Pendulum-v1 environment with a different number of comparisons: 250, 800 and 1400.

Pendulum-v1 is an unsolved environment, which means it does not have a specified reward threshold at which it is considered solved, like CartPole-v0 has. However, some research papers that aim to benchmark RL algorithms do mention that a possible ”solved score” for Pendulum-v1 would be near -140. They explain it is a number taken from current leader boards in RL communities (Oller, Glasmachers, and Cuccu 2020). We will not judge on such a limit, as in general, Pendulum-v1 still remains an unsolved environment. Our chosen RLHF baseline agent, the one with 1400 comparisons, does exceed -140, more specifically it attains a value of -131.8. We have chosen the 1400 comparisons RLHF agent since it attains the highest reward mean and an adequate standard deviation (a bit lower than the other two agents we had).

AIRLHF with optimal demonstrations Results can be visualized in 5b. We observe that the AIRLHF agents do not exceed the RLHF baseline. We can only conclude that the best AIRLHF agent (the one who learned from 1400 comparisons and 1400 demonstrations) performs barely similar to our selected baseline. We **cannot reduce the number of queries** to achieve the same, or better, results.

AIRLHF with suboptimal demonstrations Results are shown in Fig. 5c. Interestingly, our experiments reveal that our AIRLHF agents trained with suboptimal demonstrations, which included controlled randomness, performed better on average compared to those trained with optimal demonstrations. Comparing to the RLHF baseline, the best AIRLHF agents managed to perform very similar to it. However, we **cannot make a reduction** in the number of queries, since the performance of the agents is very similar, but does not exceed the baseline.

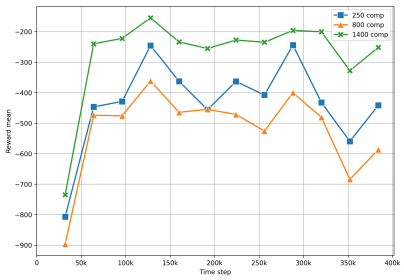
6 Discussion

In this section we will further interpret and discuss the results we achieved. In addition, we will highlight certain limitations of our approach.

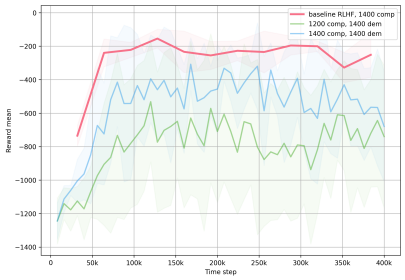
To begin with, Table 2 reiterates the conclusions reached from our results. Table 3 revises our judgement, for each environment, on the performance of the best AIRLHF agents in comparison to the selected RLHF baseline agent.

Optimal demonstrations. With regards to the CartPole-v0 environment, we expected to see AIRLHF agents outperforming the RLHF baseline. CartPole-v0 is a simple environment, has discrete actions and straightforward dynamics, and a relatively simple task. While training was done, we analyzed the trend AIRL agents had (see Appendix C). Having optimal demonstrations, AIRL quickly converged to a reward mean of almost 500. Therefore, we believe that AIRL found a good initialization for the reward and policy, which was later passed to RLHF.

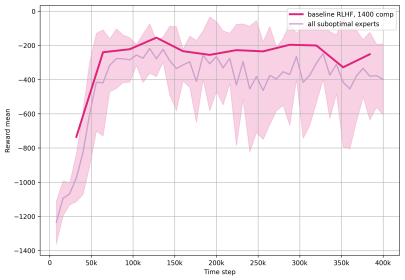
On the other hand, on the Pendulum-v1 environment, the results were surprising. Given that Pendulum-v1 is bit more complex than CartPole-v0, has continuous state and action spaces, we hypothesize the following: optimal demonstrations might have led to overfitting and reduced exploration. We suspect that AIRL did not find a good enough initialization with the number of timesteps (400k) and demonstrations we provided. We suspect that increasing the number of optimal demonstrations above 1400 (our threshold) would provide better results. Also, in Appendix D we included a figure illustrating the training trend of the AIRL agents.



(a) RLHF agents with varying comparisons.



(b) AIRLHF agents with different comparisons and optimal demonstrations.



(c) A comparison between our selected RLHF baseline agent and the mean-curve of all suboptimal AIRLHF agents.

Fig. 5: Performance on **Pendulum-v1** as measured by the true mean reward over time. Each learner (curve) represents the mean of 5 runs with different seeds (34, 43, 52, 61, and 70). For (a), evaluation was done 12 times with 4 parallel environments. For (b) and (c), evaluation was done 200 times with 4 parallel environments.

Table 1: Results on **CartPole-v0** and **Pendulum-v1** when evaluating the derived policy on the true reward. Mean and std are the average of 5 runs with seeds (34, 43, 52, 61, and 70). Evaluation was done over 100 episodes.

	Experiment			Metrics		
Environment	Agent	Comparisons	Demonstrations	Mean	Std	CI 95%
CartPole-v0						
	RLHF	250	-	171.28	±1.36	171.28 ± 1.19
	RLHF	800	-	162.09	±8.56	162.09 ± 7.50
	RLHF	1400	-	207.57	±1.39	207.57 ± 1.21
	AIRLHF optimal	10	20	424.94	±0.54	424.94 ± 0.47
	AIRLHF optimal	10	60	395.93	±17.44	395.93 ± 15.28
	AIRLHF suboptimal	250	150	10.59	±6.48	10.59 ± 5.67
	AIRLHF suboptimal	250	500	37.92	±2.85	37.92 ± 2.50
	AIRLHF suboptimal	500	150	32.01	±1.87	32.01 ± 1.64
	AIRLHF suboptimal	500	500	24.71	±3.70	24.71 ± 3.24
	AIRLHF suboptimal	800	800	92.12	±7.59	92.12 ± 6.65
	AIRLHF suboptimal	1200	1200	49.48	±2.75	49.48 ± 2.41
	AIRLHF suboptimal	1400	1400	108.49	±3.19	108.49 ± 2.80
Pendulum-v1						
	RLHF	250	-	-448.97	±329.40	-448.97 ± 288.73
	RLHF	800	-	-650.36	±386.49	-650.36 ± 338.77
	RLHF	1400	-	-317.70	±323.37	-317.70 ± 283.44
	AIRLHF optimal	1200	1400	-662.30	±429.89	-662.30 ± 376.81
	AIRLHF optimal	1400	1400	-557.99	±461.25	-557.99 ± 404.30
	AIRLHF suboptimal	800	1000	-663.30	±494.36	-663.30 ± 433.32
	AIRLHF suboptimal	800	1400	-387.42	±386.27	-387.42 ± 338.58
	AIRLHF suboptimal	1000	1400	-441.73	±426.76	-441.73 ± 374.07
	AIRLHF suboptimal	1200	1400	-332.04	±359.14	-332.04 ± 314.80

Table 2: Main conclusions derived from the AIRLHF framework, when comparing the selected RLHF baseline with the benchmarks achieved by the AIRLHF agents, with regards to decreasing the number of queries.

	AIRL optimal demonstrations + RLHF	AIRL suboptimal demonstrations + RLHF
CartPole-v0	≈ 93% decrease	no decrease
Pendulum-v1	no decrease	no decrease

Table 3: Our judgements on the performance of the AIRLHF agents when compared to the selected RLHF agent, for each environment.

	AIRL optimal demonstrations + RLHF	AIRL suboptimal demonstrations + RLHF
CartPole-v0	out-performing	worse / barely similar
Pendulum-v1	worse / barely similar	very similar

Suboptimal demonstrations. With regards to the CartPole-v0 environment, we did not expect to see that introducing a relatively small amount of noise into the demonstrations would disturb the learning process in such a large amount. However, we believe that increasing the

number of suboptimal demonstrations would improve the performance of the AIRLHF agents. However, this should be investigated in future extensions of our work, by also varying the amount of randomness introduced and reporting the resulted behaviors.

Moreover, on the Pendulum-v1 environment, the results were, again, interesting. We suspect that the variability and diversity introduced by the suboptimal demonstrations enhanced the robustness and generalizability of the learned policies. As a consequence, we believe this could have led to a better exploration of the state space and possibly provided richer learning signals. We believe that AIRL was able to handle the suboptimality in the demonstration data and provided a good enough initialization to RLHF. We also believe that a larger amount of demonstrations would surpass the baseline, allowing us to reduce the number of queries.

7 Responsible Research

While conducting our research, no human data was used for either IRL or RLHF. Although human behavior usually plays a huge part in IRL and RLHF through the demonstrations and feedback the agents learn from, it also gives rise to ethical implications with regards to how the human data was selected, stored and used. On a further note, gathering human data is a long-term process. Thus, in the limited time allocated for our research, as described in the section 3.3, we opted for synthetic demonstrations and synthetic queries. Using synthetic feedback and demonstrations ensures a high level of consistency and reproducibility in our experiments. Synthetic data, based on predefined algorithms and reward structures, eliminates the variability and biases that human-generated data might introduce. This consistency allows us to focus on the performance of the RL algorithms without the added complexity of human error or subjective judgments. Furthermore, synthetic data provides a clear and controlled benchmark for evaluating the effectiveness of our RLHF and AIRLHF implementations. By using synthetic preferences and demonstrations, we can establish a baseline performance that can be compared to results obtained with human-generated data in future extensions of our work. This step is crucial for validating our methodology and ensuring that any improvements or changes in performance metrics can be attributed to the algorithms themselves rather than variations in data quality or sources.

In order to promote transparency in research and deployment of RL systems, the complete set-up of our research has been made public on Github [4]. In deep RL systems, reproducibility can be affected by extrinsic factors (e.g. hyperparameters or codebases) and intrinsic factors (e.g. effects of random seeds or environment properties) (Henderson et al. 2019). To overcome this matter, we ensured the reproducibility and extensibility of our research by reporting all hyperparameters used, implementation details, experimental-setup, always providing the seeds we used and by writing clear, coherent and commented code that can be easily scaled to new experiments, new algorithms, and new environments. Moreover, we include a transparent review of our methodology, the AIRLHF framework, and instructions on how to use it.

[4] Github repository

8 Conclusions and Future Work

This paper presented an approach of combining AIRL with RLHF with preference comparisons to investigate whether the number of queries can be reduced. Starting from the idea that RLHF suffers from random behavior at early learning stages and most of the times needs a large number of evaluative feedbacks to converge to a desired policy, we integrated AIRL in the learning process with the purpose of overcoming the aforementioned challenges. More specifically, we designed a framework in which AIRL would find a reward and policy, which would later be passed as input to RLHF. Furthermore, we believed RLHF would benefit from these initializations, and would proceed to optimize the reward and derive a final policy. We also deemed necessary to investigate our approach with both optimal and suboptimal demonstrations, since in practice, perfect demonstrations are hard to obtain. We conducted our research on two environments. For each, we set as baseline the best performing RLHF agent, and compared it with agents of our framework (AIRLHF agents). To achieve suboptimality, we introduced controlled randomness, in a proportion of $\approx$10%. From the evaluation results, we have concluded that when using suboptimality, a large number of demonstrations needs to be provided. This number would typically need to equal or potentially exceed the number of comparisons used by the baseline RLHF agent, in order to be able to decrease the number of queries used by the AIRLHF agents. When having optimal demonstrations, we concluded that the environment's dynamics and specifics greatly influence the outcome of our framework. We derived the conclusion that in some environments, a larger number of optimal demonstrations and execution time is needed in order to provide a good initialization to RLHF. In other environments, a relatively small number of optimal demonstrations is enough to provide a good starting point to RLHF.

While implementing our approach we also found important points for future research. Firstly, we only use synthetic data. We believe a combination of human and synthetic data could improve our results. Secondly, we only use random queries. This means that we generate random trajectories. An alternative approach is to use an active selection of queries (e.g. pick trajectories which have a great variance of rewards between each other). This approach would eliminate redundant queries, and potentially lower their number. Thirdly, we use a standard exploration factor in our environments. A future direction could be to increase the exploration and investigate whether it impacts the number of queries. In addition, our work can be extended with other IRL algorithms, such as Maximum Entropy IRL, and other techniques of generating feedback. Lastly, other possible combinations between IRL and RLHF can be investigated.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.;

Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Abbeel, P.; and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on machine learning*, 1. ACM.

Adams, S.; Cody, T.; and Beling, P. A. 2022. A survey of inverse reinforcement learning. *Artificial Intelligence Review*, 55(6): 4307–4346.

Argall, B.; Browning, B.; and Veloso, M. 2007. Learning by demonstration with critique from a human teacher. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, HRI '07, 57–64. New York, NY, USA: Association for Computing Machinery. ISBN 9781595936172.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5): 834–846.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. .

Christiano, P.; Leike, J.; Brown, T. B.; Martic, M.; Legg, S.; and Amodei, D. 2023. Deep reinforcement learning from human preferences. arXiv:1706.03741.

Ezzeddine, A.; Mourad, N.; Araabi, B. N.; and Ahmadabadi, M. N. 2018. Combination of learning from non-optimal demonstrations and feedbacks using inverse reinforcement learning and Bayesian policy improvement. *Expert Systems with Applications*, 112: 331–341.

Fu, J.; Luo, K.; and Levine, S. 2018. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. arXiv:1710.11248.

Gleave, A.; Freire, P.; Wang, S.; and Toyer, S. 2020. seals: Suite of Environments for Algorithms that Learn Specifications. https://github.com/HumanCompatibleAI/seals.

Gleave, A.; Taufeeque, M.; Rocamonde, J.; Jenner, E.; Wang, S. H.; Toyer, S.; Ernestus, M.; Belrose, N.; Emmons, S.; and Russell, S. 2022. imitation: Clean Imitation Learning Implementations. arXiv:2211.11972v1 [cs.LG]. arXiv:2211.11972.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2020. Generative adversarial networks. *Commun. ACM*, 63(11): 139–144.

Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2019. Deep Reinforcement Learning that Matters. arXiv:1709.06560.

Islam, R.; Henderson, P.; Gomrokchi, M.; and Precup, D. 2017. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *CoRR*, abs/1708.04133.

Kaufmann, T.; Weng, P.; Bengs, V.; and Hüllermeier, E. 2023. A Survey of Reinforcement Learning from Human Feedback. *arXiv*, arXiv:2312.14925.

Lee, H.; Phatale, S.; Mansoor, H.; Mesnard, T.; Ferret, J.; Lu, K.; Bishop, C.; Hall, E.; Carbune, V.; Rastogi, A.; and Prakash, S. 2023. RLAIF: Scaling Reinforcement Learning from Human Feedback with AI Feedback. arXiv:2309.00267.

Nicolescu, M. N.; and Mataric, M. J. 2003. Natural methods for robot task learning: instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems*, 241–248. ACM.

Oller, D.; Glasmachers, T.; and Cuccu, G. 2020. Analyzing Reinforcement Learning Benchmarks with Random Weight Guessing. arXiv:2004.07707.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.

Ratliff, N. D.; Bagnell, J. A.; and Zinkevich, M. A. 2006. Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, 729–736. New York, NY, USA: Association for Computing Machinery. ISBN 1595933832.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Towers, M.; Terry, J. K.; Kwiatkowski, A.; Balis, J. U.; Cola, G. d.; Deleu, T.; Goulão, M.; Kallinteris, A.; KG, A.; Krimmel, M.; Perez-Vicente, R.; Pierré, A.; Schulhoff, S.; Tai, J. J.; Shen, A. T. J.; and Younis, O. G. 2023. Gymnasium.

Ziebart, B.; Maas, A.; Bagnell, J.; and Dey, A. 2008. Maximum Entropy Inverse Reinforcement Learning. 1433–1438.

A RL Libraries and Technologies

For the purposes of our research, we have mainly used the following libraries and technologies to build our framework and run our experiments:

- Gym (newer and maintained version: Gymnasium (Towers et al. 2023)) Library: Gym is a standard API for reinforcement learning, and has a diverse collection of reference environments.
- Imitation Library (Gleave et al. 2022): Imitation provides clean implementations of reward learning and training algorithms (including AIRL and RLHF with preference comparisons). Moreover, it uses Modular PyTorch implementations of the aforementioned algorithms.

- Stable Baselines 3 (SB3) Library (Raffin et al. 2021): SB3 contains a set of reliable implementations of RL algorithms in PyTorch. It is also compatible with the Imitation Library.
- seals Library(Gleave et al. 2020): seals is a toolkit for evaluating specification learning algorithms, such as reward or imitation learning. The environments are compatible with Gym. This library also includes multiple renovated Gym environments, designed specifically for reward learning (this involves removing any side-channel sources of reward information such as episode boundaries, etc.).
- TensorBoard (Abadi et al. 2015): As TensorFlow's visualization toolkit, it provides the visualization and tooling needed for machine learning experimentation. SB3 also provides an integration with TensorBoard.
- Hugging Face: Open source, provides trained models of agents with tuned hyperparameters.

We have chosen those RL libraries in particular due to their user-friendly APIs, modularity, scalability and popularity in our research area. We have specifically used *HumanCompatibleAI/ppo-seals-CartPole-v0* and *HumanCompatibleAI/ppo-Pendulum-v1* as experts from Hugging Face to generate optimal demonstrations.

In addition, all of our experiments were conducted on a laptop with Intel Core i7 processor and NVDIA RTX 3050 Ti.

B CartPole Environment

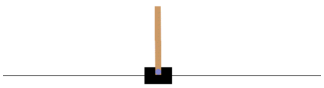


Fig. 6: CartPole-v0

The CartPole environment is a Classic Control environment with a discrete action space, corresponding to the version of the cartpole problem described in the paper (Barto, Sutton, and Anderson 1983). A pole is attached to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart (Towers et al. 2023). Since the goal is to keep the pole upright for as much as possible, there is a reward +1 for every step taken, including the termination step.

We chose to use the renovated CartPole environment from the Seals library (Gleave et al. 2020), as opposed to using it directly from the Gymnasium library (Towers et al. 2023). The renovated environment is particularly useful because it has a fixed episode length (500 steps), helping us in removing possible biases towards shorter or longer episode boundaries.

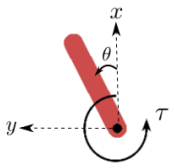


Fig. 7: Pendulum-v1

The Pendulum environment is a Classic Control environment, where the goal is to swing a pendulum to keep it upright. The pendulum starts in a random position, and the agent must apply torque to maintain an upright position, minimizing the angle and angular velocity while keeping the control effort (torque) small. The state space consists of the pendulum's angle and angular velocity, and the action space is a continuous range of torques. The reward is designed so that a better performance corresponds to a reward closer to 0, indicating minimal deviation from the upright position with minimal effort.

We chose to use the Pendulum-v1 Environment from the Gymnasium Library directly since it already had a fixed episode length, and the Seals library does not yet have a renovated environment for it.

C AIRL, optimal demonstrations, training on CartPole-v0

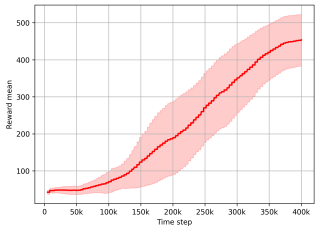


Fig. 8: Trend observed while training AIRL with optimal demonstrations. Figure represents the mean of 10 AIRL learners, with the following seeds: 0, 9, 34, 43, 52, 61, 70, 79, 88, 97.

D AIRL, optimal demonstrations, training on Pendulum-v1

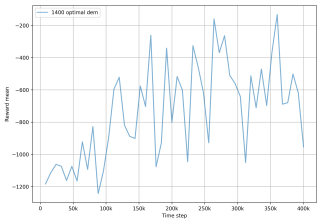


Fig. 9: Trend observed while training AIRL with optimal demonstrations. Figure represents the mean of 5 AIRL learners, with the following seeds: 34, 43, 52, 61, 70.