MACHINE LEARNING Science and Technology

PAPER

Actively learning costly reward functions for reinforcement learning

André Eberhard[1], Houssam Metni[1,2,5], Georg Fahland[3], Alexander Stroh[3] and Pascal Friederich[1,4,*]

[1] Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany
[2] Université de Strasbourg, 4 rue Blaise Pascal, 67081 Strasbourg, France
[3] Institute of Fluid Mechanics, Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany
[4] Institute of Nanotechnology, Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany
[5] Current address: Institute of Nanotechnology, Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany.
[*] Author to whom any correspondence should be addressed.

E-mail: pascal.friederich@kit.edu

Keywords: reinforcement learning, optimization, natural science, machine learning

Supplementary material for this article is available online

OPEN ACCESS

RECEIVED
15 November 2023

REVISED
12 February 2024

ACCEPTED FOR PUBLICATION
14 March 2024

PUBLISHED
26 March 2024

Abstract
Transfer of recent advances in deep reinforcement learning to real-world applications is hindered by high data demands and thus low efficiency and scalability. Through independent improvements of components such as replay buffers or more stable learning algorithms, and through massively distributed systems, training time could be reduced from several days to several hours for standard benchmark tasks. However, while rewards in simulated environments are well-defined and easy to compute, reward evaluation becomes the bottleneck in many real-world environments, e.g. in molecular optimization tasks, where computationally demanding simulations or even experiments are required to evaluate states and to quantify rewards. When ground-truth evaluations become orders of magnitude more expensive than in research scenarios, direct transfer of recent advances would require massive amounts of scale, just for evaluating rewards rather than training the models. We propose to alleviate this problem by replacing costly ground-truth rewards with rewards modeled by neural networks, counteracting non-stationarity of state and reward distributions during training with an active learning component. We demonstrate that using our proposed method, it is possible to train agents in complex real-world environments orders of magnitudes faster than would be possible when using ground-truth rewards. By enabling the application of RL methods to new domains, we show that we can find interesting and non-trivial solutions to real-world optimization problems in chemistry, materials science and engineering. We demonstrate speed-up factors of 50–3000 when applying our approach to challenges of molecular design and airfoil optimization.

1. Introduction

Reinforcement learning (RL) techniques have achieved impressive results in a wide range of applications such as robotics [44], games [56, 74, 84] or natural sciences [55, 94]. This success is the result of improvements along multiple independent branches of RL research such as an improved understanding of rewards in difficult environments [1, 17, 70, 89], more sample-efficient training via experience replay [4, 45, 51, 71] or more effective sampling via active learning [12, 20, 22], more powerful algorithms [28, 32, 50, 56, 82] and more efficient and scalable implementations [21, 35, 36] of established techniques. These extensions were primarily developed and benchmarked in simulated environments such as *OpenAI Gym* [13], *MuJoCo* [81] or the *Deepmind Control Suite* [80], among many others. In these scenarios, research is primarily centered around improving sample efficiency, e.g. the *Atari100k* benchmark [91], exploration techniques [14], or better representation learning in very high-dimensional, visual environments [15, 46, 90]. In all these environments, the underlying Markov decision process (MDP) is well-designed with meaningful and computationally cheap rewards, such that agents can be trained for millions or even billions [6] of steps.

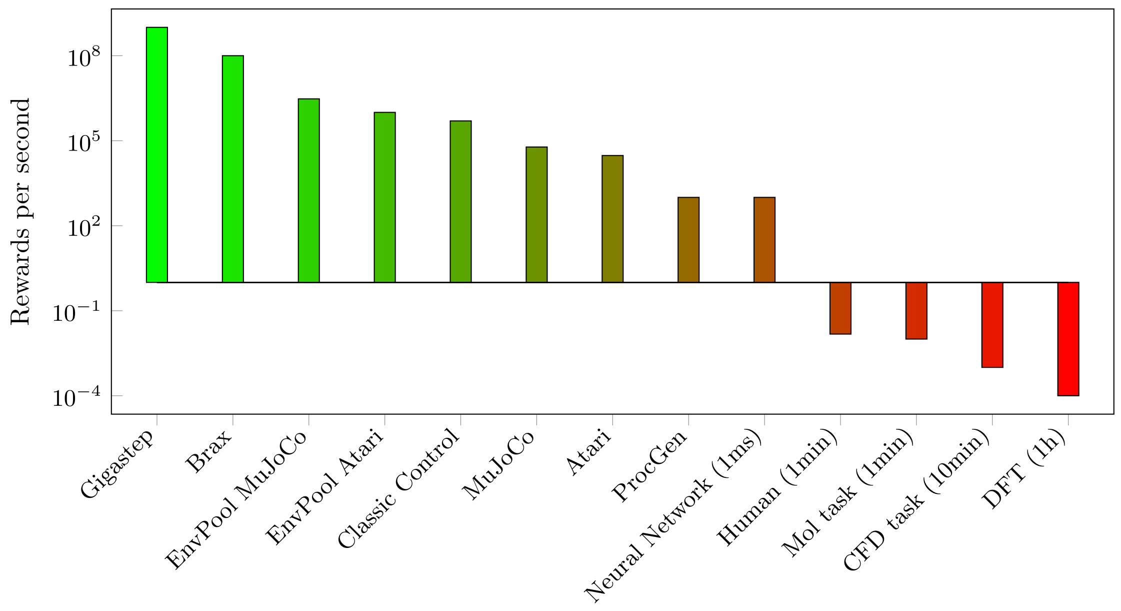

Figure 1. Processing speeds of rewards for different environments or simulators. Numbers for *Gigastep* are from [48]. Numbers for Brax are from [26]. Numbers for *Classic Control*, *Atari* and *MuJoCo* and their *EnvPool* variants are from [86]. Numbers from *ProcGen* are from [19].

However, in real-world tasks rewards do not follow this principle and may be either difficult to formulate or to collect. Whenever this is the case, a viable approach is to *learn* the reward function. Why and how to learn a reward model differs based on the exact task to perform. Early approaches, rooted in the field of robotics, aim to learn high-level behavior (placing or grasping objects) while having access to only low-level primitives (actuators) of the robot. In these scenarios, the reward function is assumed to be absent as it is difficult to engineer it manually. Instead, an agent must infer it from demonstration [1, 70] or, whenever demonstration is difficult or impossible, from ranked alternatives [17, 89]. Another scenario in which a reward model can be learned is model-based reinforcement learning (MBRL) [47]. In MBRL, the agent not only learns a policy, but also tries to learn the transition dynamics and/or reward function. Both models can then be used for planning, i.e. a hypothetical interaction with the environment with no real-world effects in contrast to executing the policy, which always causes a state transition in an MDP. In these scenarios, a learned reward function can be used to reduce the number of real-world interactions, thus improving sample-efficiency. Since testing algorithms is easier in simulation, the majority of RL research focuses on discrete [6, 91] or continuous [81] control tasks in simulated environments, which are suitable testbeds for general RL research. In these scenarios, researchers primarily study, compare and improve algorithmic components, potentially running environments for billions [6] of timesteps. Due to the availability of cheap rewards, learning the reward function is uncommon in these environments.

Apart from traditional RL research, there is an increasing interest to transfer these methods to other domains, e.g. to the natural sciences or engineering domain. In those scenarios, rewards are frequently the result of computationally demanding optimization procedures or algorithms, which are, even though in simulation, orders of magnitude more costly to evaluate than in benchmark scenarios.

Figure 1 compares different environments and simulators with regard to their number of possible reward evaluations per second. A neural network forward pass is conservatively estimated at 1 ms. Human judgment is assumed to be on the order of 1 min. The time to evaluate one reward for a particular task is based on our experiments in section 4. The drag optimization task is based on a Computational Fluid Dynamics simulation, which runs for approximately 10 min on a single core, but was run on multiple cores in our experiments. Density functional theory refers to expensive quantum chemical calculations. In contrast, distributed architectures such as *Gigastep* [48] are able to process a billion steps per second, while other optimized architectures such as *Brax* [26] report around 100 million steps per second. Even though orders of magnitude slower, popular environments in *OpenAI Gym* [13] such as Atari [6] or *MuJoCo* [81] are still able to process a considerable number of rewards per second, which can further be increased by using parallelization, e.g. as in *EnvPool* [86].

As such, the current trend in RL is to scale systems massively by running multiple independent copies of agent-environment interactions in parallel. While using massive amounts of compute resources may be justified by the outstanding results such as [38, 75], following this trend in scenarios with complex rewards, evaluations have two major drawbacks. First, it may exclude all but the largest institutions to engage in this area of research at all. Second, establishing scale as the default to gather enough data to train agents can be considered *Red AI* [72].

When ground-truth evaluations become orders of magnitude more expensive than in common environments, vanilla RL would require massive amounts of scale, just for evaluating rewards rather than training the policy. For problems in the context of natural sciences and engineering, in which simulations with different computational budgets are needed in order to obtain rewards, it would be more practical to provide proxy rewards in the form of a learned model. As can be seen in figure 1, such tasks, when not evaluated in a parallel or distributed context, can be even slower to evaluate than human feedback. In contrast, a neural network model of the reward function has the potential to speed up reward generation considerably, given that it is possible to learn such a model for a particular task. Human feedback still remains a special case, as it cannot be parallelized. However, even if we would evaluate rewards in the scenarios we consider in a distributed context, it still would require a considerable amount of compute, which may not be available, as we are required to bridge several orders of magnitude in order to reach a reasonable number of training samples.

In this work, we explicitly focus on learning policies with costly rewards (red areas in figure 1) in a model-free setting. We show that it is possible to combine reinforcement and active learning to resolve the issue of reward collection in real-world scenarios, where the relevant domain-specific quantities are difficult to obtain. We show that with our approach, which we term **ACRL** (**A**ctively Learning **C**ostly Reward Functions for **R**einforcement **L**earning)[6], neural networks, pre-trained on a relatively small initial dataset and regularly updated during training via active learning, can be used as reward proxies. We furthermore show that agents trained with our method achieve competitive results across different real-world tasks with varying computational cost, while training several times faster compared to training with ground-truth rewards.

2. Related work

2.1. Active reward learning and sample efficiency

Active reward learning techniques [12, 20, 22] build upon the insight that not all training samples are equally important for learning and aim to select only those samples which are most beneficial for learning. The selection is usually done by some form of uncertainty estimation, often within the Bayesian framework. Reducing the number of state queries is vital in cases where reward evaluation is expensive. While existing work employs active learning to reduce the number of queries for the agent to accelerate convergence of the RL training [52], we employ active learning for the reward model such that predictions become more accurate on states the agent visits during exploration.

In vanilla RL, every observation is used only once to update the agent's policy, making learning slow and sample-inefficient. A popular technique to overcome this is to use *experience replay* [51] in off-policy algorithms, which improves sample-efficiency in terms of sample usage by storing experience in a *replay buffer* and performing parameter updates on batches uniformly sampled from it. Improvements of experience replay use different forms of non-uniform sampling [45, 71], handle sparse and binary reward signals and multi-goal environments [4], and are also extended to a distributed context [36].

In our work, we do not aim to improve sample efficiency in its general sense, even though we dramatically reduce the number of ground-truth environment interactions. Rather, we avoid expensive and repetitive ground-truth evaluations for known regions of the state space by using a reward model. We increase the size of this region over the course of training by providing ground-truth labels for a small fraction of states selected by some sampling method. We use the learned reward model as a drop-in replacement for the actual reward, hence we do not use any form of model-based learning or planning to improve training efficiency. In addition, since our method assumes a modification of the environment's properties, i.e. the reward, it is possible to use techniques such as information directed reward learning (IDRL) [52] with our method to reduce the number of reward model queries. Specifically, we show that we can improve training time without using any of the more advanced techniques in the RL toolbox to keep our solution lean and to avoid common reproducibility issues [34, 37].

2.2. Learning reward functions

In theory, every agent accumulates rewards under a unified mathematical framework. In practice, though, the exact properties of a reward function depend on the task. For example, a reward function could produce immediate (dense) or delayed (sparse) rewards, which themselves could be binary, discrete, or continuous. Sometimes, a reward function is not even available and hence cannot produce any rewards. In this section,

[6] Our code is available at: https://github.com/32af3611/acrl.

we provide an overview about *why* and *how* to learn reward functions in different scenarios. We put most emphasis on *why* it is required to learn a reward function in a variety of cases as opposed to *how* to do so. Investigating in detail the question of why to learn a reward model for expensive computational simulations differs from traditional and more recent work. For example, the reason to learn a reward function in the branch of MBRL is *conceptual*, i.e. methods require it for planning. In the model-free branch of RL, learning a reward function is not a requirement, and we learn a model of it for *practical* reasons as we may have different issues with using the true reward function directly, if it exists.

RL provides a powerful framework for solving problems in which it is not directly possible to specify the correct solution but only an indicator of quality or progress as measured by the reward. Unfortunately, even specifying how well an agent did in certain tasks is difficult in some cases. The field of robotics can provide many such example tasks, e.g. to carry around an object or to place it on a table. To solve such tasks, the paradigm of *learning from demonstration* [70] emerged, which is most commonly also referred to as *apprenticeship learning* [1] or *imitation learning* [70]. In [70], a real-world robotic arm is trained to solve the classic Cart-Pole balancing problem using MBRL. While the task can be learned from scratch, the authors show that by pretraining the policy $a_t = \pi(s_t)$ or both the policy and the dynamics model $s_{t+1} = f(s_t, a_t)$ on only a small number of demonstrations, the task can be learned from fewer trials.

A concept related to learning from direct demonstration is *Inverse Reinforcement Learning* (IRL) [60]. The goal in IRL is to extract a reward function given observed, optimal behavior. The extracted reward function should be able to explain the observed behavior, i.e. IRL is indirect in that it first learns a reward function from demonstration and subsequently optimizes a policy using this reward function. In [1], the authors' objective is to learn from demonstration via IRL. They first assume that the unknown, true reward function R^* is defined as a linear combination of state features ϕ, i.e. $R^*(s) = w^* \cdot \phi(s)$. Defining $\mu(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]$ to be a vector of feature expectations, the expected policy performance can be expressed as a linear combination of learned weights w and $\mu(\pi)$. Another assumption is that one can observe sequences of state vectors generated by an expert policy and therefore the expert feature expectations $\mu(\pi_E)$. The authors then propose an iterative algorithm to estimate the reward function using a series of policies $\pi^{(i)}$, expected features $\mu^{(i)} = \mu(\pi^{(i)})$ and weights $w^{(i)}$. Weights $w^{(i)}$ are calculated based on the distance $\mu(\pi_E) - \mu(\pi^{(i)})$. One can then obtain a policy $\pi^{(i+1)}$ by optimizing it with a learned reward function $R = (w^{(i)})^T \phi$. Using the policy $\pi^{(i+1)}$, expected features $\mu^{(i+1)} = \mu(\pi^{(i+1)})$ and weights $w^{(i+1)}$ can be calculated and so on. A major drawback of this algorithm is that it requires learning a policy in every iteration and therefore is limited in the ability to scale. IRL is also considered an ill-posed problem as there are many reward functions which can explain a particular behavior as well as the degenerate reward function induced by $w = 0$, which makes all policies optimal. An advantage compared to direct learning from demonstration is that the policy can be transferred to similar environments by learning a new policy with the learned reward function, while direct learning from imitation depends on the environment's dynamics and cannot be transferred [2]. As such, learning from demonstration, direct or indirect, whenever possible, is an elegant way to avoid hand-crafting a reward function for complex, high-level behavior.

Unfortunately, there are certain cases where it is not easily possible to demonstrate optimal behavior. For example, while a human could demonstrate optimal behavior for a humanoid robot, it is less possible for robots with a different morphology [17]. Under mild assumptions, it is believed that whenever it is not possible to describe optimal behavior, it may be possible to learn behavior from an ordered set of demonstrations, which need not necessarily be optimal [49]. As a result, *Reinforcement Learning from Human Feedback* (RLHF) [17] has emerged as a paradigm. The authors of [49] argue that one can separate the *what*, i.e. the user intention, summarized in form of a reward model, from the *how*, i.e. maximizing this reward via RL. The underlying intuition of this approach is that there are frequent cases where humans are able to compare and order a set of candidate solutions, while being less able to produce optimal solutions directly with sufficient quality or amount. A prominent application of RLHF is *InstructGPT* [63], a chatbot able to solve a large variety of text-based tasks with high quality.

While the above examples of why to learn a reward function are the most prominent ones, there is also literature learning reward functions in a more narrow scope. In a model-free setting, the authors in [68] show that when the reward signal is noisy, corrupted or stochastic, a learned reward model can be used for variance reduction. A different reward concept is that of intrinsic rewards [16], which serve as an exploration bonus in addition to the external reward provided by the environment. Agents are intrinsically rewarded when visiting novel states and therefore, one must be able to quantify how frequently a state has been visited. In large state spaces, a model of state novelty must be learned, for example by distillation [14] or with generative models [43].

Table 1. Categorization of different concepts of reward model learning and their characteristics.

Scenario	Sample efficiency	Difficult specification	Active Learning	Expensive human rewards	Expensive computational rewards
Apprenticeship learning	✘	✓	✓	✓	✘
Inverse reinforcement learning	✘	✓	✓	✓	✘
Reinforcement learning from human feedback	✘	✓	✓	✓	✘
Model-based reinforcement learning	✓	✘	✓	✘	✘
Our scenarios	✓	✘	✓	✘	✓

In the context of learning reward functions, active learning is frequently encountered as a component, as it reduces the number of reward queries to make, which is especially beneficial when rewards are expensive. In RLHF [17], queries are selected based on an ensemble of reward models to estimate uncertainty. Also motivated by the fact that both evaluation and demonstration of tasks in robotics are difficult, the authors of [12] propose to learn a reward with Gaussian Processes, which are also suitable methods for active learning as they can provide uncertainty quantification without ensembles, but do not scale to large datasets. Similarly, the authors of [76] use a binary classifier as a reward model which predicts whether a particular state is a goal state. During training, the user can be queried to provide binary labels, which are relatively easy to obtain as a human is able to quickly distinguish a goal from a non-goal state. The authors in [53] use active learning to reduce the number of samples required in IRL. IDRL [52] actively learns a reward model such that additional queries help with identifying the optimal policy rather than uniformly improving the reward model at all states.

We summarize our categorization of the different concepts of learning a reward function in table 1. Cells are marked with ✓ if a column is the defining characteristic of a particular type and ✘ otherwise. For example, sample efficiency is also a concern when learning from demonstration or feedback but the defining characteristics are that rewards are difficult to formulate and that human feedback is required. Conversely, in MBRL, rewards are not always difficult to specify and human feedback is not always required, as it is a more general concept itself, even though it can also be part of learning from demonstration. When learning reward models, active learning is a frequently encountered component to reduce the number of expensive queries and to speed up training.

In summary, most of the existing literature uses learned reward models to avoid a human to specify a whole reward function or rewards for single transitions. The former is often difficult to do and the latter is very time-consuming when training a policy for many steps. While there exists a large body of literature where RL is applied in simple simulators, the simulations we consider are of scientific nature and are orders of magnitude more difficult to evaluate (see figure 1). In addition, the specification of a reward function for RL as well as an acquisition function for active learning allows, due to the rewards being physically meaningful, to unify how we can learn policies specifically in these environments, which is the primary contribution of our work.

2.3. Efficient implementations

The effects of other extensions within the RL framework have been studied in [35], showing recent advances can be integrated to improve their standalone-performance. From a practical point of view, the authors of [77] provide a unified implementation view of RL algorithms to leverage modern, parallel hardware architectures to further reduce training time.

In our work, we do not aim to leverage massively scaled RL in order to solve the issue of costly rewards. Rather, we propose an extension to restore the effectiveness of these methods in scenarios where their efficiency would be threatened by the reward evaluation bottleneck. We note that our method is scalable and naturally can be integrated into distributed architectures such as those in [24, 36, 58].

2.4. RL for optimization

The idea of optimizing functions with RL was already investigated several decades ago [88]. In [88], the authors used REINFORCE [87]-like algorithms on a set of experiments with known maxima to show that it is possible to learn an adaptive system that generates optima by trial and error. Interestingly, they found that their algorithms converge to suboptimal single-mode solutions in the presence of multiple equally-valued actions and corrected this behavior by maximizing entropy in addition to reward, which today is a standard technique in robust RL and also a core component in one of the current state-of-the-art algorithms, *Soft Actor-Critic* and its variants [18, 32, 33].

Since then, RL has been applied to many instances of combinatorial optimization due to its ability to efficiently explore large spaces without handcrafted heuristics. Canonical NP-hard problems such as the *Travelling Salesman Problem* (TSP) and other graph-related problems have been the focus due to the difficulty of obtaining optimal solutions for these problems [8, 41]. Furthermore, there is an increased interest in using these methods in real-world applications, with applications for road [93] or computer [83] networks. A broader overview of machine learning for combinatorial optimization can be found in [9].

Besides applications in computer science, RL has been applied for optimization and discovery in a variety of other domains. In chemistry and materials science, optimizing properties of molecules or their molecular graphs, respectively, is of major interest. Existing work such as *MolDQN* [94] uses RL to find local modifications of molecules that yield improved properties. Besides single-property optimizations, other work aims for molecules meeting multiple criteria at the same time [29], geometry optimization [3] or design and discovery [27, 62, 64].

Another source of interest is the optimization of airfoils to improve their aerodynamic properties, with all kinds of applications in aeronautics. While traditionally these kinds of problems are tackled with optimization methods such as gradient-based optimization, the authors in [23] argue that these methods, even though computationally efficient in large spaces, are susceptible to poor local minima, and do not work well with non-linear cost functions. While machine learning techniques are less susceptible to these kinds of errors, the authors in [10] point out that using high-fidelity data for training can become prohibitively expensive.

While there is much interest in using RL for different kinds of optimization problems, in many cases, most effort is spent on finding a solution, with the general assumption that it can be verified fast. Methods such as *MolDQN* [94] or *MoleGuLAR* [29] optimize cheap properties, e.g. *logP* and *QED* (for a description of the quantities, see section 4.1), while in the case of airfoil design, lower-fidelity data is used to accelerate data generation [10]. For real-world domains, the validation of solutions may be orders of magnitude slower than in research scenarios, which hinders training agents in environments that require a very large number of steps. In this work, we show that we can alleviate the computational burden of reward evaluation by actively sampling data points and learning a reward model. By using a cheap reward model, we can provide rewards much faster and, in addition, avoid repetitive and thus redundant evaluation of frequently visited states. We show that we can use the original *MolDQN* with an actively learned reward model to optimize properties of molecules which are much more costly to evaluate. We also show that we can train agents for hundreds of thousands of steps without excessive amounts of computational effort in an airfoil optimization task. This does not only contribute to *Green AI* [72] in these scenarios, but also may allow smaller institutions to engage in this area of research.

3. Our method: ACRL

In this section, we introduce our method for the optimization of physical systems whose properties are computationally costly to evaluate. We use a standard MDP formulation as found in [79].

Let $f(s)$ be a quantity or metric associated with state s, f being a known but expensive to evaluate function of s. Without loss of generality, we aim to find a (local) minimum of f, or equivalently, a (locally) optimal state $s^* = \arg\min_s f(s)$. Due to high computational cost as well as non-convexity of f in real-world tasks, we neither can directly solve for s^* nor is it likely that we can find s^* with heuristic search in general. We therefore propose a more principled search of s^* by framing it as a sequential decision-making problem within the RL framework. A natural definition of reward in such environments is $r_t = f(s_{t-1}) - f(s_t)$, i.e. the agent aims to accumulate reward by sequentially visiting states s with decreasing $f(s)$. We note that this formulation lends itself well to attract the agent to minima and is a popular choice in optimization scenarios [41]. Nevertheless, the standard cumulative, discounted return formulation can be used whenever appropriate. However, the potential-based reward formulation should be the primary choice when applicable due to its favorable properties in terms of reward shaping and credit assignment [59].

Let s_0 be a possibly random initial state, the agent then aims to maximize the total cumulative (undiscounted) reward $R_T = \sum_{t=1}^{T} f(s_{t-1}) - f(s_t) = f(s_0) - f(s_T)$. Due to the computational complexity of f, training an agent for a large number of steps may become infeasible or at least very time-consuming.

To reduce the computational burden of state evaluations during training, our method requires only a few modifications of the standard training loop, namely the introduction of a reward model and its improvement via active learning. In general, the interaction of policy and reward model closely mirrors the interaction of policy and value networks in generalized policy iteration [79], except that in our method trajectories generated by the policy are used to improve the reward model, which itself is used to improve the value function. The two steps are then as follows.

The first step is to pre-train an approximate reward model $\hat{f}$, e.g. a neural network, on a small, initial dataset D in a supervised manner. $\hat{f}$ is then used as a drop-in replacement for the true evaluation function f. Doing so is theoretically sound as the reward distribution does not depend on the agent's policy. This allows using our method with both value-based and policy gradient methods without the necessity to change the underlying theory. At this point, we make several mild assumptions about f. In contrast to the general RL setting, we assume that we can evaluate f in any state, thus providing dense and instantaneous rewards on state transitions. Hence, our method is not well-suited for sparse or delayed rewards, as found in conventional RL scenarios, which we do not aim to cover since it is rarely the case that we cannot evaluate any property of a system.

The second step is then to actively improve the reward model during agent training. Since the initial state distribution in D likely differs from states visited by an exploring agent, $\hat{f}$ may have poor extrapolation capabilities which will cause agent training to diverge as estimated state quantities may not have their true value predicted accurately. This particularly applies in scenarios where it is difficult to define *good* initial states, for example in the case of optimization problems where the optimal solution is to be found rather than given. To overcome this issue, we propose to sample a small number of states encountered during agent training and to provide the expensive ground-truth labels for them. In the most general form, we define an acquisition function $h(s)$ which hypothesizes about how beneficial adding the true label $f(s)$ to D is for training the reward model. We then periodically evaluate h for a small fraction of the agent's experience $\mathcal{E}$, e.g. the last N steps, where N is an application-dependent hyperparameter. We set $s' = \underset{s\in\mathcal{E}}{\arg\max}\, h(s)$, $D = D \cup \{s'\}$ and subsequently update $\hat{f}$ on the new D, either by training from scratch or fine-tuning. At this point, we assume that the reward model can be trained reasonably fast such that the training time can be amortized given enough reward evaluations. For example, $h(s)$ may be chosen to be $\hat{f}(s)$, $||\nabla\hat{f}(s)||$ or other sampling techniques such as uniform or uncertainty sampling. We hypothesize that this active learning component allows to explore relevant regions of the state space effectively and efficiently. An important implication of using active learning is that, depending on the task at hand, the initial reward model must not be perfectly accurate, which is a recurring theme in RL when viewed from the perspective of policy iteration. For example, when using our method for optimization tasks where it is unlikely that the optimal solution space is included in the initial dataset D, perfect accuracy in this space is not necessary since the agent moves away from the initially covered space towards a more optimal region. It is thus more important for the reward model to be accurate on the on-policy distribution of states rather than on randomly selected initial data points. The reward model is only required to improve as the agent's policy improves and stabilizes. We found this active learning component to be crucial in our tasks.

A summary of the overall procedure can be found in algorithm 1. We note that even though we use variations of Double DQN [82] agents in all experiments, our method does not assume any particular type of RL implementation and can be integrated into existing implementations with minimal changes, even in asynchronous and distributed settings. Due to the freedom of choice of algorithms, our method can be used for both discrete and continuous optimization problems. The same holds for sampling strategies or active learning approaches in general.

Algorithm 1. Double Deep-Q-Learning within ACRL.

```
1: agent $A$, replay buffer $B$, initial dataset $D$, environment $E$, randomly initialized reward network $\hat{f}$
2: $\hat{f} \leftarrow$ train($\hat{f}, D$)                         ▷ train reward network
3: E.reward $\leftarrow \hat{f}$                                    ▷ E.step() uses $\hat{f}$ instead of $f$
4: for episode = 1 to M do                                          ▷ training loop
5:     $s_t \leftarrow$ initial state
6:     for step = 1 to T do                                         ▷ episode loop
7:         $a_t \leftarrow$ A.action($s_t$)                          ▷$\epsilon$-greedy
8:         $s_{t+1}, \hat{r}_{t+1} \leftarrow$ E.step($a_t$)         ▷$\hat{r}_{t+1} = \hat{f}(s_t, a_t)$
9:         $obs \leftarrow (s_t, a_t, \hat{r}_{t+1}, s_{t+1})$
10:        B.add($obs$)                                             ▷ save observation
11:        $obs \leftarrow$ B.sample()                               ▷ sample experience from B
12:        A.optimize($obs$)                                        ▷ update parameters
13:    end for
14:    if sample state then                                         ▷ e.g. periodically
15:        $s' \leftarrow \arg\max_{s \in \mathcal{E}} h(s)$          ▷ any method
16:        $y' \leftarrow f(s')$                                    ▷ calculate ground-truth label
17:        $D \leftarrow D \cup \{(s', y')\}$
18:    end if
19:    if update model then                                         ▷ e.g. periodically
20:        $\hat{f} \leftarrow$ train($\hat{f}, D$)                 ▷ retrain reward network
21:        E.reward $\leftarrow \hat{f}$                            ▷ update reward network
22:    end if
23: end for
```

4. Applications

4.1. Proof-of-principle: molecular property optimization

The algorithm described above is first used in molecular property optimization tasks as a proof-of-principle. We use two fast-to-evaluate benchmarking properties to evaluate the performance of the algorithm and to choose its optimal hyperparameters. Both the Q-network and the reward network are trained on Morgan fingerprint vectors as molecular representations [57, 67]. States and actions are based on prior work [94], where states are discrete molecular graphs and actions are semantically allowed local graph modifications.

The first benchmarking property is the penalized logP score, a widely used metric in the literature for evaluating and benchmarking machine learning models on regression and generative tasks [30, 61, 92]. The logP score is the logarithm of the water-octanol partition coefficient, quantifying the lipophilicity or hydrophobicity of a molecule. Penalized logP additionally takes into account the synthetic accessibility (SA) and the number of long cycles (n_{cycles}):

$$\mathrm{pen.logP} = \mathrm{logP} - \mathrm{SA} - n_{\mathrm{cycles}}. \tag{1}$$

The second benchmarking property used here is the QED score, which is a quantitative estimate of druglikeness based on the concept of desirability [11]. QED is an empirical score quantifying how ‘drug-like’ a molecule is. Both properties are computationally inexpensive and can be calculated using RDKit [66]. We use them as benchmarking properties to study the effect of replacing the ground-truth reward with an approximation and to choose hyperparameters of our algorithm. In both applications, empty initial states are optimized for $T = 40$ steps. We then test our method on a real-life application in molecular improvement with a more costly property value to calculate.

4.2. Application I: molecular design

In our first application, we evaluate ACRL on a molecular design task involving more costly rewards. We aim to optimize electronic properties of molecules such as energies of the highest occupied molecular orbital (HOMO) and the lowest unoccupied molecular orbital (LUMO) by performing sequential modifications. These values can be calculated using semiempirical quantum mechanical methods such as density functional tight binding methods as implemented in *xTB* [7, 31]. xTB-based reward evaluations on one Intel Xeon Gold 6248 CPU range from seconds to minutes, depending on size and structure of the molecule. Compared to other RL applications, this is comparably expensive, especially considering the number of reward evaluations needed during agent training. The algorithm described above is applied using the hyperparameters found in the experiments of section 4.1. Here, the agent learns a more application-oriented optimization goal, i.e. how to decrease the LUMO energy of randomly sampled starting molecules with only $T = 5$ steps per episode, while keeping the HOMO-LUMO gap constant. Therefore, the goal of the agent is to find optimal local improvements of given molecules with a limited number of actions, i.e. changes of the chemical structure. Let s_0 be a randomly sampled molecule at the beginning of an episode, then the improvement of the molecule s_t at timestep t over s_0 is defined as:

$$R(s_t) = -|\mathrm{gap}(s_t) - \mathrm{gap}(s_0)| - (\mathrm{LUMO}(s_t) - \mathrm{LUMO}(s_0)) \tag{2}$$

with $\mathrm{gap}(s) = \mathrm{LUMO}(s) - \mathrm{HOMO}(s)$ being the HOMO-LUMO energy difference of molecule s.

4.3. Application II: optimization of airflow drag around an airfoil

The control technique of wall-normal blowing or/and suction constitutes a promising approach for the reduction of drag in turbulent boundary layers [42]. This technique has been successfully utilized not only in flat-plate boundary layers [39] but also on more complex curved geometries such as airfoils [5]. The majority of studies on the aforementioned control technique, however, considers uniform distribution of the introduced blowing or suction profiles. In our second application, we use ACRL to minimize aerodynamic drag around an airfoil by sequential adjustment of a set of blowing and suction coefficients represented as vectors in $\mathbb{R}^d$ (see figure 3(a)), which form the state space in $\mathbb{R}^{2d}$. As higher coefficients trivially reduce drag, we seek to optimize profiles with a constrained mean value for each side. By choosing a different constraint at the start of each episode, we aim to generalize across multiple instances of optimization. We use a Double DQN [82] agent with discrete actions corresponding to exactly one (or no) modification of an entry of s per step to keep the action space as small as possible. Thus, we seek to find a (near-)optimal state $s^* \in \mathbb{R}^{2d}$ under given constraints. In our experiments, we use an episode length of $T = 30$ steps. While policy gradient methods would be more appropriate for this task, we use Double DQN for the sake of consistency.

Let $d_0 = f(s_0)$ be the drag coefficient of starting state s_0 corresponding to a uniform profile on each side. Our agent then seeks to find a sequence of modifications such that $R_T = \sum_{t=1}^{T} d_{t-1} - d_t = d_0 - d_T$ becomes as large as possible. We note that while the agent seeks to maximize R_T, we are primarily interested in the shape of states s_T close to the (globally) optimal state s^* rather than the exact value of $f(s_T)$.

The incompressible flow around airfoils is analysed using Reynolds-averaged Navier–Stokes equation based simulations in order to assess the effect of localized blowing and suction on the global aerodynamic performance of the airfoil. The simulations are carried out with the open-source CFD-toolbox OpenFOAM [85] using a steady state, incompressible solver. For the current study we consider a flow around the NACA4412 airfoil at the Reynolds number $Re = U_\infty c/\nu = 4 \cdot 10^5$ and the angle of attack $\alpha = 5°$. For a more detailed description of the setup the reader is referred to [25].

One particular difficulty in training an RL agent in this scenario is the fact that the true state evaluation function f is a computational fluid dynamics (CFD) simulation. On one core of an Intel Xeon Platinum 8368 CPU, the simulation runs for approximately 10 min. Due to a fixed mesh size, we found that parallelization beyond 4 cores did not result in a significant speed-up, hence one reward evaluation takes approximately 2 to 3 min and cannot be reduced significantly, which severely limits the applicability of conventional RL algorithms with thousands of sequential reward evaluations.

5. Results and discussion

We summarize our results in table 2. The main quantity of interest in our experiments is the time it takes to evaluate a particular number of rewards to train an agent successfully. Given several queries and the duration for each query, we calculate the total query time for both ground-truth (oracle) and model queries. The standard training protocol for RL agents evaluates the environment's reward function on every step, while our training protocol uses a reward model as a proxy.

Our results are two-fold. First, while not being our primary objective, using a reward proxy directly improves sample efficiency, which is a direct consequence of not using the environment to provide rewards. Second, by using a reward proxy, we can improve computational efficiency. On the one hand, given the high cost of running a full simulation and the low cost of a neural network forward pass, the time to produce enough samples to learn a model exceeds the time it takes to provide rewards for all steps during training. On the other, and more importantly, the combined time to collect ground-truth samples and to run a particular number of environment steps with proxy rewards is significantly lower than if we provided ground-truth rewards in every step. Note that this does not include the total training time of the reward model. The reason we do not include an exact number is that the time to train the reward model depends on hyperparameter settings, the retraining frequency, and the number of environment steps. Even if we estimate model training time on the order of how long it takes to run our most expensive experiment, which trained for

Table 2. Relative and absolute speed-up factors for different tasks comparing the number of oracle and model queries. The time for a neural network forward pass has been conservatively estimated at 1 ms.

Task	Oracle queries	Model queries	Model training	Oracle duration	Model duration	Oracle time	Model time	Oracle training	Sample efficiency
Mol. opt.	∼4000	∼200 000	12 h	∼1 s	∼0.001 s	∼1 h	∼0.05 h	∼56 h	+50×
Mol. imp.	∼4000	∼25 000	12 h	∼1 m	∼0.001 s	∼66 h	∼0.007 h	∼416 h	+6.25×
Drag opt.	∼3000	∼9 000 000	12 h	∼3 m	∼0.001 s	∼150 h	∼2.5 h	∼450 000 h	+3000×

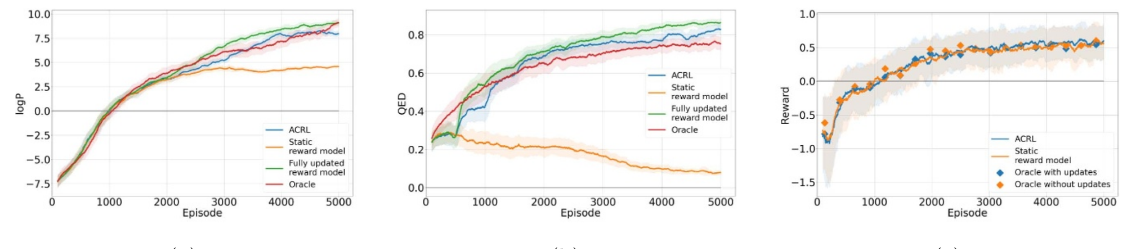

Figure 2. Evolution of the reward reached by the agent during the optimization of logP and QED: The red curve was obtained by training the agent on real (oracle-based) rewards, while the blue, orange and green curves are the ACRL model, the static reward model and a fully updated reward model, respectively. Due to high computational costs, only ACRL and static models can be tested in (c). In (c), solid lines represent modelled rewards while diamonds represent ground-truth rewards.

approximately ten hours, we can train agents with our approach several times faster. This is especially pronounced in the airfoil design experiments as the simulation is expensive to run and training requires a large number of steps, resulting in even one potential training run being prohibitively expensive to evaluate.

In the following sections, we show that we achieve our computational savings without sacrificing the performance of the resulting policy. Our results indicate that using an actively learned reward model is especially beneficial in the non-stationary or low-data regimes. When large databases exist for a particular problem (section 5.1), it should be possible to train a reward model of sufficient quality. In the high-data regime, we therefore cannot expect large improvements against a high-quality reward model in general. However, in our benchmark scenarios (see figures 2(a) and (b)), we can observe two important properties of ACRL agents. First, they match, even though using a relatively small number of queries, the performance of agents trained with more powerful initial but static reward models, with results being similar to the original *MolDQN* [94]. Second, a reward model trained on some fixed prior distribution cannot aid the learning process in general when specifically working on tasks such as molecular discovery, which are inherently out-of-distribution. In such cases, active learning is a crucial component (see figure 2(b)) as the reward model might fail to generalize well enough outside its prior distribution. Therefore, our approach is efficient and flexible and is especially useful when working with expensive quantities which have not been studied extensively before (see sections 5.2 and 5.3). In the following sections, we provide a more detailed description of the exact training protocol for each of our tasks before we draw our conclusion.

5.1. Speedup of molecular property optimization

Based on prior work [94], we used cheap chemistry benchmarking properties logP and QED as a proof of concept to evaluate how the use of actively learned rewards performs in comparison to the real reward. Figures 2(a) and (b) show the performance of three different agents with NN-approximated rewards compared to a reference agent (‘oracle-based reward’) trained on the real reward. One of the reward approximation agents is only trained once in the beginning (‘static’). One of the agents (‘ACRL’) uses a reward model which is updated at regular intervals using additional oracle queries selected based on uncertainty sampling. The last agent (‘full update’) is updated after every episode using oracle queries of all states encountered in that episode (i.e. closest to the reference agent which directly uses oracle queries for training). After approximately 2000 episodes in case of logP optimization and already at the beginning of QED optimization, the performances of the agents start to differ. While the performance of the static agent stagnates, all three other agents show similar performance.

The failure of the **static agent** to learn is due to the low generalization ability of the initial reward model itself, which is trained on the QM9 dataset [65, 69] containing approximately 134.000 molecules with up to 9 non-hydrogen atoms. To some extent, the weak generalization can be attributed to not using state-of-the-art graph neural networks. However, we decided to use the same molecular representation and model as in the original Q-networks in [94], i.e. fingerprint representations and MLPs. Furthermore, during the learning process, the molecules generated (especially after a high number of episodes) contain many more atoms, which explains why the static reward model fails to correctly estimate the real property values. The strength of this effect depends on the property studied. In the logP optimization task, the property values reached with a static reward model follow the general trend of learning with real reward, even though the final performance after 5000 episodes is lower. In case of QED optimization, the static reward model fails to predict QED-values for molecules outside the training distribution. As a consequence, the RL agent learns to exploit errors of the static reward model and finds adversarial examples, rather than samples with desirable properties.

The active learning component within **ACRL agent** allows the reward model to learn from molecules outside its initial training distribution, thus improving reward evaluations during agent training. By only selecting a small subset of labels obtained using oracle queries to be added to the training set, the objective of the ACLR agent is to mimic the reference agent's real behavior as closely as possible. This includes finding (nearly) optimal points (see e.g. [52]) to be selected for retraining of the reward model to minimize its errors while at the same time minimizing the number of costly oracle queries. We experimented with different sampling strategies (see SI), from which a query-of-committee model (see [73]) performed best. Therefore, in the ACLR model used in the molecular design and improvement tasks, three reward models were trained independently to form a query-of-committee model. The three reward models are retrained after 500 episodes with the initial training set along with all 400 new molecules generated during the agent learning process and their computed real property values. The selection of new oracle queries to extend the dataset is based on the disagreement between the three reward models measured by the standard deviation of the predictions. However, our work is independent of the particular sampling strategy (even random sampling of visited states can work well in some applications), as long as the reward model's training distribution follows the exploration of the RL agent. Overall, the speed-up achieved by the ACRL model in this experiment compared to the fully updated and oracle-based model is 50 (see table 2). The relationship between speed-up and rewards reached is analyzed in the SI.

Fully updating the reward model on oracle queries of all samples ('full update') aids the learning process. In case of logP optimization and even stronger in case of QED optimization, learning by fully updating the reward model has even surpassed learning with actual reward values at certain episodes. One potential reason for that can be that the exploration of the fully updated agent is stronger than that of the reference agent (see SI), which needs to be confirmed in future work. However, in practice, fully updating the reward model by adding every single generated point (along with its real property value) to the initial dataset and retraining the neural network is as expensive as training the reference agent, so it cannot be applied to tasks with costly rewards.

In order to understand why the fully updated reward model in some cases (e.g. figure 2(b)) outperforms the oracle-based training, we analyzed the effect of additional noise and thus exploration which might be induced by replacing oracle-based rewards with (noisy) approximated rewards. We therefore varied the ϵ-greedy strategy of the learning process. In particular, we varied final ϵ values (i.e. probabilities of random actions) and the form of the ϵ-decay function used on the learning process. However, none of the changes in ϵ-decay could improve the learning behaviour, i.e. the ϵ-decay rate and function used by [94] was optimal. Therefore, for the rest of the simulations we used a fully exponential decay reaching approximately 1% randomness in episode 5000. The results of this study are available in the supplementary information section. Further study of the improvement effect due to a fully updated reward model is part of ongoing work as it has the potential to improve the performance of RL agents with little computational overhead.

5.2. Speedup of molecular improvement

After evaluating the performance of our agent on easy-to-compute properties such as penalized logP and QED, we test our ACRL approach on a molecular improvement task with more costly rewards, where an oracle-based reference study is unfeasible. In particular, we study a RL agent with the goal of independently varying two quantum mechanically calculated energy levels of molecules with only very few, in our case five, modification steps (see section 4). Figure 2(c) shows the evolution of the ACRL and the static reward agents' rewards as a function of the training episode. We observe that the reward becomes positive after approximately 1000 episodes and stagnates after approximately 2000 episodes. Therefore, the agent has learned to improve given (arbitrary) molecules, since the reward value of the starting reference molecule is zero, each episode starts with a randomly sampled molecule, and any molecule with negative reward would have less desirable properties than the initial one. This suggests that even though the agent deals with different starting reference molecules at each episode, it has managed to learn a strategy to increase the reward in a limited number of steps.

In contrast to the property optimization task discussed before, the performance of the ACRL and the static reward agents are equal within the confidence intervals. A likely explanation for this observation is that

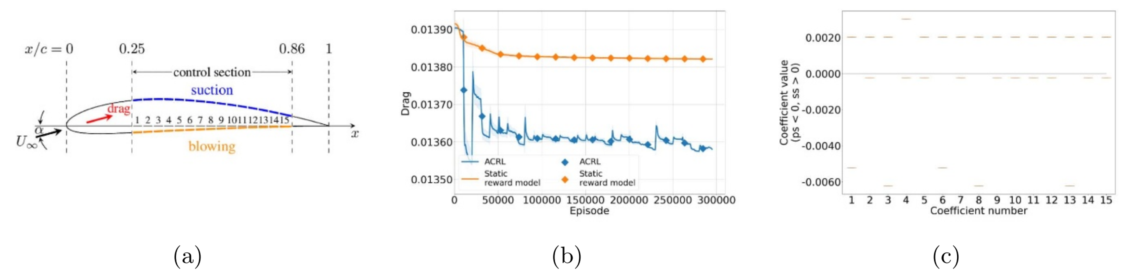

Figure 3. (a) blowing/suction distribution discretized with 30 coefficients corresponding to 15 sections on each side of the considered airfoil. (b) drag evolution of two independent runs. (c) coefficient distribution for low-drag profiles. In (b), solid lines represent modelled rewards while diamonds represent ground-truth rewards.

the number of steps per epoch in this task is limited to five, whereas 40 steps were possible in the prior task. Therefore, the agent here cannot generate molecules that are far outside the initial distribution of starting molecules, i.e. the QM9 dataset. Furthermore, the ratio of reward model queries to oracle queries in this experiment is comparably high (see table 2), meaning that the ACRL reward model is updated on a high fraction of actually encountered molecules. A reference calculation with oracle based rewards or a fully updated reward model to check if the ACRL model found near-optimal results (within the DQN framework) are computationally too costly here and thus unfeasible. However, we compared the predictions of the reward models for randomly selected molecules throughout the training process to oracle predictions (see points in figure 2(c)). We found excellent agreement, indicating that the ACRL as well as the static reward models are reliable. Thus, the solutions found are not exploiting weaknesses of the reward models, nor is the training limited by wrong predictions of the reward models. Therefore, it is likely that the found solutions are of comparable quality as ones that a hypothetical oracle-based RL model would find. The speed-up achieved in this experiment compared to a hypothetical oracle based model is 6.25, which still has room for improvement, given the high reliability of the reward models. Even though the ACRL agent does not exceed the performance of a static reward model trained on QM9, these results have another important implication. While QM9 contains all molecules with up to nine heavy atoms, of which around 130 000 exist and on which the static model has been trained on, the ACRL agent matches its performance using only 4000 ground-truth queries. While large databases exist for molecules, this may not be the case in other and especially new domains. This showcases the effectiveness of our approach in the low-data regime.

5.3. Speedup of optimization of airflow drag around an airfoil

Our ACRL method is applicable to a large number of different tasks in natural sciences and engineering, not only limited to chemistry. Therefore, in this section we present the results of a task in engineering, namely the reduction of airflow drag around an airfoil, e.g. an airplane wing (see section 4). The objective in this task was to find a set of coefficients minimizing drag and to analyze the resulting profiles. Figure 3(b) shows the evolution of drag during 300 000 episodes of training. The discrete jumps of the ACRL model coincide with retraining of the reward model every 100 00 episodes. As higher mean constraints are highly correlated with lower drag, we choose samples for ground-truth evaluation based on reward rather than drag. Dots represent oracle-based ground-truth evaluations of random profiles sampled during training.

The results demonstrate that the ACRL agent is able to find profiles with significantly lower drag coefficients than the static reward model. They also show that in this task (in contrast to the molecular improvement task) it is crucial to actively update the reward model during training. This is related to the fact that in order to improve upon the initially uniform profile, the RL agent has to perform a constrained optimization in high-dimensional real space (30-dimensional in our case). Accurate reward model predictions require sufficient coverage of the relevant space within the initial dataset which is difficult to assert because the relevant region is, in general, not known, which is also true for many other real-world problems. As a consequence, an agent trained without active updates of the reward model only slightly improves upon a uniform profile. At the same time, model updates result in sharp drops of both predicted and ground-truth drag especially in the beginning of training as the relative effect of new ground-truth samples is high and the RL agent probably exploits wrong predictions of the early-stage reward models. This effect decreases as more and more samples are obtained along the trajectories towards the low-drag region in parameter space.

Figure 3(c) shows the distribution of a small number of low-drag profiles sampled with ground-truth labels during training. The resulting profiles are non-trivial and have a regular, alternating pattern of coefficients with physically explainable meaning [40, 54, 78]. We note that due to various limitations of the simulated environment such as discretization of action space, a limited number of coefficients due to limitations in OpenFOAM simulations (used as an oracle) and limited episode length, these results are only locally optimal w.r.t. our setup. Yet, we find consistent, physically interpretable and highly non-trivial results.

6. Conclusion

We introduced ACRL, an approach to learn reward functions in RL in the context of computationally expensive rewards, which models the reward of given applications using machine learning models. Because optimal regions in the search spaces are not known *a priori* and thus typically are not included in initial training sets, we use active learning while exploring the state space to update the reward model over the course of training. We first showed that it is possible to train agents with an incrementally improving reward model on existing benchmark tasks using cheap benchmark quantities. We then showed in two more realistic scenarios that by learning a reward model jointly with our policy, we can reduce the time for reward evaluations by several orders while still being able to produce meaningful results. In turn, it becomes feasible to train agents without massive distribution within reasonable timeframes, which saves computational resources and energy and at the same time accelerates research since resources can be spent on training models rather than evaluating rewards.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/32af3611/acrl.

Acknowledgment

We would like to thank the Federal Ministry of Economics and Energy under Grant No. KK5139001AP0. We acknowledge support by the Federal Ministry of Education and Research (BMBF) Grant No. 01DM21001B (German-Canadian Materials Acceleration Center). We acknowledge funding by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) within Priority Programme SPP 2331.

ORCID iDs

André Eberhard https://orcid.org/0009-0006-6880-8809
Houssam Metni https://orcid.org/0000-0003-0675-7322
Alexander Stroh https://orcid.org/0000-0003-0850-9883
Pascal Friederich https://orcid.org/0000-0003-4465-1465

References

[1] Abbeel P and Andrew Y N 2004 Apprenticeship learning via inverse reinforcement learning *ICML '04: Proc. 21st Int. Conf. on Machine Learning* (ACM) p 1
[2] Adams S, Cody T and Beling P A 2022 A survey of inverse reinforcement learning *Artif. Intell. Rev.* **55** 4307–46
[3] Ahuja K, Green W H and Yi-Pei Li 2021 Learning to optimize molecular geometries using reinforcement learning *J. Chem. Theory Comput.* **17** 818–25
[4] Andrychowicz M, Wolski F, Ray A, Schneider J, Fong R, Welinder P, McGrew B, Tobin J, Abbeel O P and Zaremba W 2017 Hindsight experience replay *Advances in Neural Information Processing Systems* vol 30, ed I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan and R Garnett (Curran Associates, Inc)
[5] Atzori M, Vinuesa R, Fahland G, Stroh A, Gatti D, Frohnapfel B and Schlatter P 2020 Aerodynamic effects of uniform blowing and suction on a NACA4412 airfoil *Flow Turbul. Combust.* **105** 735–59
[6] Badia A P, Piot B, Kapturowski S, Sprechmann P, Vitvitskyi A, Daniel Guo Z, and Blundell C 2020 Agent57: outperforming the atari human benchmark *CoRR* (arXiv:2003.13350)
[7] Bannwarth C, Caldeweyher E, Ehlert S, Hansen A, Pracht P, Seibert J, Spicher S and Grimme S 2020 Extended tight-binding quantum chemistry methods *WIREs Comput. Mol. Sci.* **11** e01493
[8] Bello I, Pham H, Quoc V L, Norouzi M and Bengio S 2016 Neural combinatorial optimization with reinforcement learning (arXiv:1611.09940)
[9] Bengio Y, Lodi A and Prouvost A 2021 Machine learning for combinatorial optimization: a methodological tour d'horizon *Eur. J. Oper. Res.* **290** 405–21
[10] Bhola S, Pawar S, Balaprakash P and Maulik R 2023 Multi-fidelity reinforcement learning framework for shape optimization *J. Comput. Phys.* **482** 112018
[11] Bickerton R, Paolini G, Besnard J'emy, Muresan S and Hopkins A 2012 Quantifying the chemical beauty of drugs *Nat. Chem.* **4** 90–98

[12] Biyik E, Huynh N, Kochenderfer M J and Sadigh D 2020 Active preference-based gaussian process regression for reward learning *Robotics: Science and Systems XVI (Virtual Event / Corvalis) (Oregon, USA, 12–16 July 2020)* ed M Toussaint, A Bicchi and T Hermans (arXiv:2005.02575)
[13] Brockman G, Cheung V, Pettersson L, Schneider J and Schulman J Tang J and Zaremba W 2016 OpenAI gym (arXiv:1606.01540)
[14] Burda Y, Edwards H, Storkey A and Klimov O 2018 Exploration by random network distillation (arXiv:1810.12894)
[15] Chen L, Lee K, Srinivas A and Abbeel P 2021 Improving computational efficiency in visual reinforcement learning via stored embeddings *Advances in Neural Information Processing Systems* vol 34 pp 26779–91
[16] Chentanez N, Barto A and Singh S 2004 Intrinsically motivated reinforcement learning *Advances in Neural Information Processing Systems* p 17
[17] Christiano P F, Leike J, Brown T, Martic M, Legg S and Amodei D 2017 Deep reinforcement learning from human preferences *Advances in Neural Information Processing Systems* p 30
[18] Christodoulou P 2019 Soft actor-critic for discrete action settings (arXiv:1910.07207)
[19] Cobbe K, Hesse C, Hilton J and Schulman J 2020 Leveraging procedural generation to benchmark reinforcement learning *Int. Conf. on Machine Learning* (PMLR) pp 2048–56
[20] Cui Y and Niekum S 2018 Active reward learning from critiques *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)* (IEEE) pp 6907–14
[21] Dalton S and frosio iuri 2020 Accelerating reinforcement learning through gpu atari emulation *Advances in Neural Information Processing Systems* vol 33, ed H Larochelle, M Ranzato, R Hadsell, M F Balcan and H Lin (Curran Associates, Inc) pp 19773–82
[22] Daniel C, Kroemer O, Viering M, Metz J and Peters J 2015 Active reward learning with a novel acquisition function *Auton. Robots* **39** 389–405
[23] Dussauge T P, Je Sung W, Pinon Fischer O J and Mavris D N 2023 A reinforcement learning approach to airfoil shape optimization *Sci. Rep.* **13** 9753
[24] Espeholt L *et al* 2018 Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures *Int. Conf. on Machine Learning* (PMLR) pp 1407–16
[25] Fahland G, Stroh A, Frohnapfel B, Atzori M, Vinuesa R, Schlatter P and Gatti D 2021 Investigation of blowing and suction for turbulent flow control on airfoils *AIAA J.* **59** 4422–36
[26] Freeman C D, Frey E, Raichuk A, Girgin S, Mordatch I and Bachem O 2021 Brax–a differentiable physics engine for large scale rigid body simulation (arXiv:2106.13281)
[27] Fromer J C and Coley C W 2023 Computer-aided multi-objective optimization in small molecule discovery *Patterns* **4** 100678
[28] Fujimoto S, Hoof H and Meger D 2018 Addressing function approximation error in actor-critic methods *Int. Conf. on Machine Learning* (PMLR) pp 1587–96
[29] Goel M, Raghunathan S, Laghuvarapu S and Deva Priyakumar U 2021 Molegular: molecule generation using reinforcement learning with alternating rewards *J. Chem. Inf. Modeling* **61** 5815–26
[30] Gómez-Bombarelli R *et al* 2018 Automatic chemical design using a data-driven continuous representation of molecules *ACS Cent. Sci.* **4** 268–76
[31] Grimme S, Bannwarth C and Shushkov P 2017 A robust and accurate tight-binding quantum chemical method for structures, vibrational frequencies and noncovalent interactions of large molecular systems parametrized for all spd-block elements (z = 1–86) *J. Chem. Theory Comput.* **13** 1989–2009
[32] Haarnoja T, Zhou A, Abbeel P and Levine S 2018 Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor *Int. Conf. on Machine Learning* (PMLR) pp 1861–70
[33] Haarnoja T *et al* 2018 Soft actor-critic algorithms and applications (arXiv:1812.05905)
[34] Henderson P, Islam R, Bachman P, Pineau J, Precup D and Meger D 2018 Deep reinforcement learning that matters *Proc. AAAI Conference on Artificial Intelligence* vol 32 (available at: https://dl.acm.org/doi/abs/10.5555/3504035.3504427)
[35] Hessel M, Modayil J, Van Hasselt H, Schaul T, Ostrovski G, Dabney W, Horgan D, Piot B, Azar M and Silver D 2018 Rainbow: combining improvements in deep reinforcement learning *32nd AAAI Conf. on Artificial Intelligence*
[36] Horgan D, Quan J, Budden D, Barth-Maron G, Hessel M, van Hasselt H and Silver D 2018 Distributed prioritized experience replay *6th Int. Conf. on Learning Representations, ICLR 2018 (Vancouver, BC, Canada, 30 April–3 May 2018)* (Conference Track Proceedings)
[37] Huang S, Fernand Julien Dossa R, Raffin A, Kanervisto A and Wang W 2022 The 37 implementation details of proximal policy optimization *The ICLR Blog Track 2023* (available at: https://openreview.net/forum?id=Hl6jCqIp2j)
[38] Jumper J *et al* 2021 Highly accurate protein structure prediction with alphafold *Nature* **596** 583–9
[39] Kametani Y and Fukagata K 2011 Direct numerical simulation of spatially developing turbulent boundary layers with uniform blowing or suction *J. Fluid Mech.* **681** 154–72
[40] Kametani Y, Fukagata K, Örlü R and Schlatter P 2016 Drag reduction in spatially developing turbulent boundary layers by spatially intermittent blowing at constant mass-flux *J. Turbul.* **17** 913–29
[41] Khalil E, Dai H, Zhang Y, Dilkina B and Song L 2017 Learning combinatorial optimization algorithms over graphs *Advances in Neural Information Processing Systems* p 30 (available at: https://dl.acm.org/doi/10.5555/3295222.3295382)
[42] Kinney R B 1967 Skin-friction drag of a constant-property turbulent boundary layer with uniform injection *AIAA J.* **5** 624–30
[43] Klissarov M, Islam R, Khetarpal K and Precup D 2019 Variational state encoding as intrinsic motivation in reinforcement learning *Task-Agnostic Reinforcement Learning Workshop at Proc. Int. Conf. on Learning Representations* vol 15 pp 16–32
[44] Kober J, Bagnell J A and Peters J 2013 Reinforcement learning in robotics: a survey *Int. J. Robot. Res.* **32** 1238–74
[45] Kong S H, Nahrendra I M A and Paek D-H 2021 Enhanced off-policy reinforcement learning with focused experience replay *IEEE Access* **9** 93152–64
[46] Kostrikov I, Yarats D and Fergus R 2020 Image augmentation is all you need: Regularizing deep reinforcement learning from pixels (arXiv:2004.13649)
[47] Kuvayev Rich Sutton L 1996 Model-based reinforcement learning with an approximate, learned model *Proc. 9th Yale Workshop on Adaptive and Learning Systems* vol 1996 pp 101–5
[48] Lechner M, Yin L, Seyde T, Wang T-H, Xiao W, Hasani R, Rountree J and Rus D 2023 Gigastep-one billion steps per second multi-agent reinforcement learning *37th Conf. on Neural Information Processing Systems Datasets and Benchmarks Track*
[49] Leike J, Krueger D, Everitt T, Martic M, Maini V and Legg S 2018 Scalable agent alignment via reward modeling: a research direction (arXiv:1811.07871)

[50] Lillicrap T P, Hunt J J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D and Wierstra D 2015 Continuous control with deep reinforcement learning (arXiv:1509.02971)
[51] Lin L-J 1992 Self-improving reactive agents based on reinforcement learning, planning and teaching *Mach. Learn.* **8** 293–321
[52] Lindner D, Turchetta M, Tschiatschek S, Ciosek K and Krause A 2021 Information directed reward learning for reinforcement learning *Advances in Neural Information Processing Systems* vol 34, ed M Ranzato, A Beygelzimer, Y Dauphin, P S Liang and J W Vaughan (Curran Associates, Inc) pp 3850–62
[53] Lopes M, Melo F and Montesano L 2009 Active learning for reward estimation in inverse reinforcement learning *Joint European Conf. on Machine Learning and Knowledge Discovery in Databases* (Springer) pp 31–46
[54] Mahfoze O A, Moody A, Wynn A, Whalley R D and Laizet S 2019 Reducing the skin-friction drag of a turbulent boundary-layer flow with low-amplitude wall-normal blowing within a bayesian optimization framework *Phys. Rev. Fluids* **4** 094601
[55] Mahmud M, Shamim Kaiser M, Hussain A and Vassanelli S 2018 Applications of deep learning and reinforcement learning to biological data *IEEE Trans. Neural Netw. Learn. Syst.* **29** 2063–79
[56] Mnih V *et al* 2015 Human-level control through deep reinforcement learning *Nature* **518** 529–33
[57] Morgan H L 1965 The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service *J. Chem. Doc.* **5** 107–13
[58] Nair A *et al* 2015 Massively parallel methods for deep reinforcement learning (arXiv:1507.04296)
[59] Ng A Y, Harada D and Russell S 1999 Policy invariance under reward transformations: theory and application to reward shaping *Int. Conf. on Machine Learning* vol 99 pp 278–87 (available at: https://dl.acm.org/doi/10.5555/645528.657613)
[60] Ng A Y, Russell S *et al* 2000 Algorithms for inverse reinforcement learning *Int. Conf. on Machine Learning* vol 1 p 2 (available at: https://dl.acm.org/doi/10.5555/645529.657801)
[61] Nigam A, Friederich P, Krenn M and Aspuru-Guzik A 2020 Augmenting genetic algorithms with deep neural networks for exploring the chemical space *8th Int. Conf. on Learning Representations, ICLR 2020, (Addis Ababa, Ethiopia, 26–30 April 2020)* (OpenReview.net)
[62] Olivecrona M, Blaschke T, Engkvist O and Chen H 2017 Molecular de-novo design through deep reinforcement learning *J. Cheminf.* **9** 1–14
[63] Ouyang L *et al* 2022 Training language models to follow instructions with human feedback *Advances in Neural Information Processing Systems* vol 35 pp 27730–44
[64] Pereira T, Abbasi M, Ribeiro B and Arrais J P 2021 Diversity oriented deep reinforcement learning for targeted molecule generation *J. Cheminform.* **13** 1–17
[65] Ramakrishnan R, Dral P, Rupp M and von Lilienfeld A 2014 Quantum chemistry structures and properties of 134 kilo molecules *Sci. Data* **1** 08
[66] RdKit 2006 Rdkit: Open-source cheminformatics
[67] Rogers D and Hahn M 2010 Extended-connectivity fingerprints *J. Chem. Inf. Model.* **50** 742–54
[68] Romoff J, Henderson P, Piché A, Francois-Lavet V and Pineau J 2018 Reward estimation for variance reduction in deep reinforcement learning (arXiv:1805.03359)
[69] Ruddigkeit L, van Deursen R, Blum L C and Reymond J-L 2012 Enumeration of 166 billion organic small molecules in the chemical Universe database GDB-17 *J. Chem. Inf. Model.* **52** 2864—2875
[70] Schaal S 1997 Learning from demonstration *Advances in Neural Information Processing Systems* vol 9, ed M C Mozer, M Jordan and T Petsche (MIT Press)
[71] Schaul T, Quan J, Antonoglou I and Silver D 2015 Prioritized experience replay (arXiv:1511.05952)
[72] Schwartz R, Dodge J, Smith N A and Etzioni O 2020 Green ai *Commun. ACM* **63** 54–63
[73] Seung H S, Opper M and Sompolinsky H 1992 Query by committee *Proc. 5th Annual Workshop on Computational Learning Theory, COLT '92* (Association for Computing Machinery) pp 287–94
[74] Silver D *et al* 2016 Thore Graepel and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search *Nature* **529** 484–9
[75] Silver D *et al* 2017 Mastering chess and shogi by self-play with a general reinforcement learning algorithm (arXiv:1712.01815)
[76] Singh A, Yang L, Hartikainen K, Finn C and Levine S 2019 End-to-end robotic reinforcement learning without reward engineering (arXiv:1904.07854)
[77] Stooke A and Abbeel P 2018 Accelerated methods for deep reinforcement learning (arXiv:1803.02811)
[78] Stroh A, Hasegawa Y, Schlatter P and Frohnapfel B 2016 Global effect of local skin friction drag reduction in spatially developing turbulent boundary layer *J. Fluid Mech.* **805** 303–21
[79] Sutton R S and Barto A G 2018 *Reinforcement Learning: An Introduction* (MIT Press)
[80] Tassa Y *et al* 2018 Deepmind control suite (arXiv:1801.00690)
[81] Todorov E, Erez T and Tassa Y 2012 Mujoco: a physics engine for model-based control *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* (IEEE) pp 5026–33
[82] Van Hasselt H, Guez A and Silver D 2016 Deep reinforcement learning with double q-learning *Proc. AAAI Conf. on Artificial Intelligence* vol 30 (available at: https://dl.acm.org/doi/10.5555/3016100.3016191)
[83] Vesselinova N, Steinert R, Perez-Ramirez D F and Boman M 2020 Learning combinatorial optimization on graphs: a survey with applications to networking *IEEE Access* **8** 120388–416
[84] Vinyals O *et al* 2019 Grandmaster level in StarCraft II using multi-agent reinforcement learning *Nature* **575** 350–4
[85] Weller H and Jasak H 2011 OpenFOAM (available at: www.openfoam.com/)
[86] Weng J *et al* 2022 Envpool: a highly parallel reinforcement learning environment execution engine *Advances in Neural Information Processing Systems* vol 35 pp 22409–21
[87] Williams R J 1992 Simple statistical gradient-following algorithms for connectionist reinforcement learning *Reinforcement Learn.* 5–32
[88] Williams R J and Peng J 1991 Function optimization using connectionist reinforcement learning algorithms *Connect. Sci.* **3** 241–68
[89] Wirth C, Akrour R, Neumann G and Fürnkranz J 2017 A survey of preference-based reinforcement learning methods *J. Mach. Learn. Res.* **18** 1–46
[90] Yarats D, Fergus R, Lazaric A and Pinto L 2021 Mastering visual continuous control: improved data-augmented reinforcement learning (arXiv:2107.09645)

[91] Ye W, Liu S, Kurutach T, Abbeel P and Gao Y 2021 Mastering atari games with limited data *Advances in Neural Information Processing Systems* vol 34 pp 25476–88

[92] You J, Liu B, Ying Z, Pande V S, Leskovec J Larochelle H 2018 Graph convolutional policy network for goal-directed molecular graph generation *Advances in Neural Information Processing Systems 31: Annual Conf. on Neural Information Processing Systems 2018, NeurIPS 2018, (Montréal, Canada 3–8 December 2018)* ed S Bengio, H M Wallach, K Grauman, N'o Cesa-Bianchi and R Garnett pp 6412–22

[93] James J Q, Yu W and Jiatao G 2019 Online vehicle routing with neural combinatorial optimization and deep reinforcement learning *IEEE Trans. Intell. Transp. Syst.* **20** 3806–17

[94] Zhou Z, Kearnes S, Li Li, Zare R N and Riley P 2019 Optimization of molecules via deep reinforcement learning *Sci. Rep.* **9** 10752