Actively Learning Costly Reward Functions for Reinforcement Learning

André Eberhard[1]
andre.eberhard@kit.edu

Houssam Metni[2]
houssam.metni@etu.unistra.fr

Georg Fahland[1]
georg.fahland@kit.edu

Alexander Stroh[1]
alexander.stroh@kit.edu

Pascal Friederich[1]
pascal.friederich@kit.edu

[1]Karlsruhe Institute of Technology, [2]Université de Strasbourg

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. Therefore, training might become prohibitively expensive without an extensive amount of computational resources and time. 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 ACRL method (**a**ctively learning **c**ostly rewards for **r**einforcement **l**earning), it is possible to train agents in complex real-world environments orders of magnitudes faster. By enabling the application of reinforcement learning 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.

1 Introduction

Reinforcement Learning (RL) techniques have achieved impressive results in a wide range of applications such as robotics [21], games [27, 37, 43] or natural sciences [26, 47]. This success is the result of improvements along multiple independent branches of RL research such as an improved understanding of rewards in difficult environments [34, 1, 8, 45], more sample-efficient training via experience replay [23, 35, 2, 22] or more effective sampling via active learning [11, 9, 6], more powerful algorithms [27, 42] and more efficient and scalable implementations [16, 10, 15] of established techniques. These extensions were primarily developed and benchmarked in simulated environments such as OpenAI Gym [7] or MuJoCo [41], where rewards are well-defined and computationally cheap to obtain. However, in real-world tasks rewards may be either difficult to formulate or to collect. There has been extensive work on how to formulate and quantify rewards in scenarios where agents have to learn from demonstrations [34, 1] or from ranked alternatives [8, 45].

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

These methods are mainly developed within the field of robotics, where feedback frequently is provided by a human supervisor. Thus, since human feedback is relatively expensive, it would be desirable to reduce the number of expert evaluations. Active reward learning techniques aim to reduce the number of expert queries by selecting only the most informative ones, usually employing uncertainty measures within the Bayesian framework [11, 9, 6] as a selection criterion. More recently, *Information Directed Reward Learning (IDRL)* [24] has been proposed to learn an unknown reward function with as few expert queries as possible.

Existing literature focuses on developments in simulated environments and real-world tasks in fields such as robotics. While in the former case rewards are clearly formulated and cheap to obtain, in the latter case rewards are typically difficult to formulate and/or quantify, e.g., in object manipulation tasks [17]. However, in many other fields rewards appear to have different properties than in these scenarios, for which most of existing work has been done. In a natural sciences and engineering context, for example, rewards are frequently the result of computationally demanding optimization procedures or algorithms. Thus, it may not be possible to leverage recent advances in RL to make training more efficient in these scenarios, since frequent reward evaluation may become a major bottleneck during training without an extensive amount of computational resources and horizontal scaling.

In this paper, we present a framework to make training of RL agents feasible in environments where it is prohibitively expensive to evaluate ground-truth rewards at every step. We propose to use neural networks as reward function approximators with active learning to address non-stationarity of state and reward distributions during training. We show that within our framework, which we term **ACRL** [1], neural networks, pre-trained on a relatively small initial dataset and regularly updated during training via an **a**ctive learning approach, can be used as reward proxies and that agents trained within this framework achieve competitive results across different real-world tasks with varying computational **c**ost, thus extending the applicability of **RL** algorithms to a wide range of new applications. This way, our agents are able to generalize in environments with varying constraints which avoids re-optimizations for new instances of a task.

2 Related work

2.1 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, rewards can be immediate or delayed and the reward signal can be binary, discrete or real. In simulated environments like OpenAI Gym and MuJoCo rewards are well-defined and exposed to the agent via a simulator interface. In fields like robotics, rewards can become complex, high-level signals of desired behavior, e.g., to manipulate an object in a particular manner [17]. Since the formulation of a reward function is often difficult in the latter case, early work [34, 1] aimed to infer an unknown reward function solely from demonstration. While alleviating the issue of reward formulation, demonstrations by a human supervisor are costly to obtain. As an alternative, preference-based learning [8, 45] allows feedback to be a relative preference over a set of trajectories rather than a quantitative measure of goodness.

In our work, we focus on problems where the reward function is known, e.g., as the solution of an optimization problem, but difficult to evaluate. This is frequently the case in real-world applications, e.g., in natural sciences and engineering, where the solution to a problem can be formulated as a goal-directed search implemented as an agent's policy. Thus, states of high rewards correspond to more optimal solution spaces of the underlying problem. We therefore propose to replace the known reward function with an approximate model and to jointly train it with the agent to account for non-stationarity of state and reward distributions during exploration. Due to their ability to generalize, our agents are able to solve optimization tasks with varying constraints, which is, in general, not trivially doable using conventional optimization and search methods.

[1] Our code is available at: `https://github.com/32af3611/ai4mat-neurips-workshop-2022`

2.2 Active reward learning

Active reward learning techniques [11, 9, 6] 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, we employ active learning for the reward model such that predictions become more accurate on states the agent visits during exploration. Closest to our method is *Information Directed Reward Learning (IDRL)* [24], which also uses a reward model and active reward learning to accelerate training. Yet, there are important conceptual differences to our method:

- *IDRL* assumes the absence of a reward function, while in our context the reward function is known.
- *IDRL* makes learning more efficient w.r.t. the number of training steps, our method makes training more efficient w.r.t. wall-clock time.
- *IDRL* uses active learning for faster convergence, we use active learning for the exploration of relevant solution spaces.
- *IDRL* relies on the Bayesian framework for uncertainty estimation, while our method requires standard deep learning components only.
- *IDRL* trains multiple policies for query selection which does not scale to tasks where agents are expensive to train. In contrast, our method trains only one agent.

2.3 Sample-efficiency

Active learning techniques improve sample-efficiency in terms of sample collection. 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* [23], 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 [35, 22], handle sparse and binary reward signals and multi-goal environments [2], and are also extended to a distributed context [16].

In our work, we do not aim to increase sample-efficiency of the RL training process. Rather, we avoid expensive 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.

2.4 Efficient implementations

The effects of other extensions within the RL framework have been studied in [15], showing recent advances can be integrated to improve their standalone-performance. From a practical point of view, the authors of [38] 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 improve RL from a technical perspective. 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 [16].

3 Our method: ACRL

Existing literature covers how to learn a reward function in cases of unclear tasks or how to make efficient use of it in cases where it exists and can be evaluated frequently. In contrast to that, in many other tasks the reward function is clearly defined but costly to evaluate. Providing these kinds of rewards to an agent during training thus can become prohibitively expensive even with off-policy learning with experience replay as one may fail to gather enough examples to learn from. In the

following we describe our proposed ACRL framework to alleviate this issue. We use a standard MDP formulation as found in [40].

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 $f(s)$. Let s_0 be a possibly random initial state, the agent then aims to maximize the total cumulative 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 framework requires only a few modifications of the standard training loop, namely the introduction of a reward model and its improvement via active learning. 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 framework 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.

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 $s' = \arg\max_{s \in \mathcal{E}} h(s)$, $D = D \cup \{s'\}$ and subsequently update $\hat{f}$ on the new D, either by training from scratch or fine-tuning. At this point, we assume that 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 like 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. 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 [42] 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.

Algorithm 1 Double Deep-Q-Learning within ACRL

agent A, replay buffer B, initial dataset D, environment E, reward network $\hat{f}$ trained on D
$\hat{f} \leftarrow$ train($\hat{f}, D$) ▷ train reward network
E.reward $\leftarrow \hat{f}$ ▷ E.step() uses $\hat{f}$ instead of f
for episode = 1 to M **do** ▷ training loop
  $s_t \leftarrow$ initial state
  for step = 1 to T **do** ▷ episode loop
    $a_t \leftarrow$ A.action(s_t) ▷ ϵ-greedy
    $s_{t+1}, \hat{r}_{t+1} \leftarrow$ E.step(a_t) ▷ $\hat{r}_{t+1} = \hat{f}(s_t, a_t)$
    $obs \leftarrow (s_t, a_t, \hat{r}_{t+1}, s_{t+1})$
    B.add(obs) ▷ save observation
    $obs \leftarrow$ B.sample() ▷ sample experience from B
    A.optimize(obs) ▷ update parameters
  end for
  if sample state **then** ▷ e.g., periodically
    $s' \leftarrow \arg\max_{s \in \mathcal{E}} h(s)$ ▷ any method
    $y' \leftarrow f(s')$ ▷ calculate ground-truth label
    $D \leftarrow D \cup \{(s', y')\}$
  end if
  if update model **then** ▷ e.g., periodically
    $\hat{f} \leftarrow$ train($\hat{f}, D$) ▷ retrain reward network
    E.reward $\leftarrow \hat{f}$ ▷ update reward network
  end if
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 [28, 32]. States and actions are based on prior work [47], 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 [29, 13, 46]. 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}):

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

The second benchmarking property used here is the QED score, which is a quantitative estimate of druglikeness based on the concept of desirability [5]. QED is an empirical score quantifying how "drug-like" a molecule is. Both properties are computationally inexpensive and can be calculated using RDKit [31]. 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* [14, 4]. 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:

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

with $\text{gap}(s) = \text{LUMO}(s) - \text{HOMO}(s)$ 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 [20]. This technique has been successfully utilized not only in flat-plate boundary layers [18] but also on more complex curved geometries like airfoils [3]. 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 2a), 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 [42] 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 methods would be a 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 [44] 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 [12].

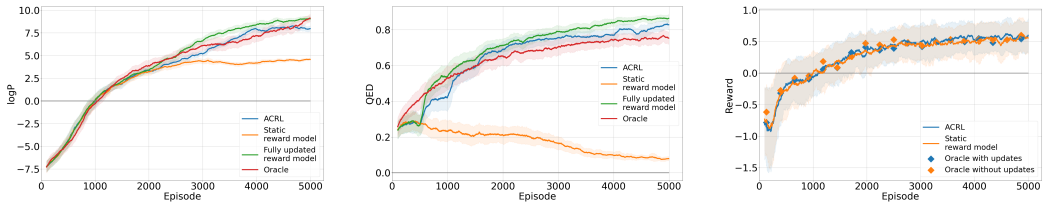
One particular difficulty in training an RL agent in this scenario is the fact that the true state evaluation function f is a Computational Fluid Dynamics (CFD) simulation. On one core of an Intel Xeon Platinum 8368 CPU, the simulation runs for approximately 10 minutes. Due to a fixed mesh size, we found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

5 Results and discussion

5.1 Molecular property optimization

Based on prior work by Zhou *et al.* [47], 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. Figure 1a and 1b 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 ("ACLR") 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.



(a) (b) (c)

Figure 1: 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).

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 [33, 30] 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 Zhou *et al.*[47], 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., [24]) 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 [36]) 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 1). 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 1b) 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 [47] 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 Molecular improvement

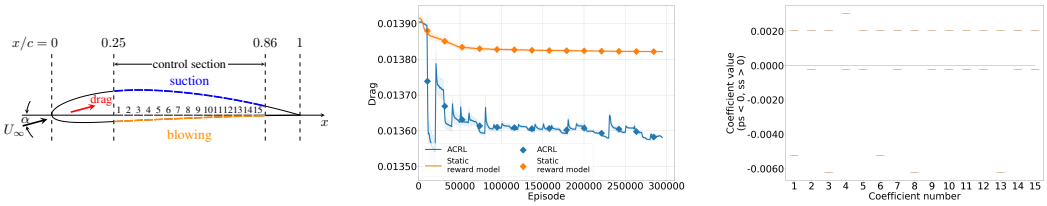
After evaluating the performance of our agent on easy-to-compute properties such as penalized logP and QED, we test our ACRL approach on a molecular improvement task with more costly rewards, where an oracle-based reference study is unfeasible. In particular, we study a RL agent with the goal of independently varying two quantum mechanically calculated energy levels of molecules with only very few, in our case five, modification steps (see Section 4). Figure 1c 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 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 low (see Table 1), 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 1c). 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.

5.3 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 2b shows the evolution of drag during 300000 episodes of training. The discrete jumps of the ACRL model coincide with retraining of the reward model every 10000 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.



(a) (b) (c)

Figure 2: (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.

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.

Task	Oracle queries	Model queries	Speed-up
Mol. opt.	∼4,000	∼200,000	∼50
Mol. imp.	∼4,000	∼25,000	∼6.25
Drag opt.	∼3,000	∼9,000,000	∼3,000

Table 1: Relative speed-up factors for different tasks comparing the number of oracle and model queries.

Figure 2c 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 [19, 25, 39]. 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 extension to standard reinforcement learning methods 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 not included in initial training sets, we use active learning during the exploration of the state space to update the reward model. This way, the number of reward evaluations can be significantly reduced, leading to a speed-up of up to 3000 in the presented applications. This reduction enables the

application of reinforcement learning algorithms, in our case Double DQN, to real-world problems where reward evaluation can become prohibitively expensive, and where conventional optimization methods cannot be applied due to dynamic constraints which require generalization across multiple problem instances. We show in three applications - one for benchmarking purposes and two real applications with expensive rewards - that training an agent with modelled rewards yields reliable results and non-trivial solutions in complex environments. The transferability and wide applicability of our approach paves way for the exploration of a wide range of real-world application areas using reinforcement learning methods.

Acknowledgements
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.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 1, New York, NY, USA, 2004. ACM.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[3] M. Atzori, R. Vinuesa, G. Fahland, A. Stroh, D. Gatti, B. Frohnapfel, and P. Schlatter. Aerodynamic Effects of Uniform Blowing and Suction on a NACA4412 Airfoil. *Flow Turbul. Combust.*, 105:735–759, April 2020.

[4] Christoph Bannwarth, Eike Caldeweyher, Sebastian Ehlert, Andreas Hansen, Philipp Pracht, Jakob Seibert, Spicher Spicher, and Stefan Grimme. Extended tight-binding quantum chemistry methods. *WIREs Comput. Mol. Sci.*, 11:e01493, 2020.

[5] Richard Bickerton, Gaia Paolini, Jérémy Besnard, Sorel Muresan, and Andrew Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4:90–8, 02 2012.

[6] Erdem Biyik, Nicolas Huynh, Mykel J. Kochenderfer, and Dorsa Sadigh. Active preference-based gaussian process regression for reward learning. In Marc Toussaint, Antonio Bicchi, and Tucker Hermans, editors, *Robotics: Science and Systems XVI, Virtual Event / Corvalis, Oregon, USA, July 12-16, 2020*, 2020.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[8] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[9] Yuchen Cui and Scott Niekum. Active reward learning from critiques. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6907–6914. IEEE, 2018.

[10] Steven Dalton and iuri frosio. Accelerating reinforcement learning through gpu atari emulation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19773–19782. Curran Associates, Inc., 2020.

[11] Christian Daniel, Oliver Kroemer, Malte Viering, Jan Metz, and Jan Peters. Active reward learning with a novel acquisition function. *Autonomous Robots*, 39(3):389–405, 2015.

[12] G. Fahland, A. Stroh, B. Frohnapfel, M. Atzori, R. Vinuesa, P. Schlatter, and D. Gatti. Investigation of Blowing and Suction for Turbulent Flow Control on Airfoils. *AIAA J.*, 59(11):4422–4436, July 2021.

[13] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, jan 2018.

[14] Stefan Grimme, Christoph Bannwarth, and Philip Shushkov. 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(5):1989–2009, 2017.

[15] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[16] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[17] Rishabh Jangir, Guillem Alenya, and Carme Torras. Dynamic cloth manipulation with deep reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4630–4636. IEEE, 2020.

[18] Y. Kametani and K. Fukagata. Direct numerical simulation of spatially developing turbulent boundary layers with uniform blowing or suction. *J. Fluid Mech.*, 681:154–172, August 2011.

[19] Yukinori Kametani, Koji Fukagata, Ramis Örlü, and Philipp Schlatter. Drag reduction in spatially developing turbulent boundary layers by spatially intermittent blowing at constant mass-flux. *Journal of Turbulence*, 17(10):913–929, 2016.

[20] R. B. Kinney. Skin-friction drag of a constant-property turbulent boundary layer with uniform injection. *AIAA Journal*, 5(4):624–630, 1967.

[21] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[22] Seung-Hyun Kong, I Made Aswin Nahrendra, and Dong-Hee Paek. Enhanced off-policy reinforcement learning with focused experience replay. *IEEE Access*, 9:93152–93164, 2021.

[23] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992.

[24] David Lindner, Matteo Turchetta, Sebastian Tschiatschek, Kamil Ciosek, and Andreas Krause. Information directed reward learning for reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 3850–3862. Curran Associates, Inc., 2021.

[25] OA Mahfoze, A Moody, A Wynn, RD Whalley, and S Laizet. Reducing the skin-friction drag of a turbulent boundary-layer flow with low-amplitude wall-normal blowing within a bayesian optimization framework. *Physical Review Fluids*, 4(9):094601, 2019.

[26] Mufti Mahmud, Mohammed Shamim Kaiser, Amir Hussain, and Stefano Vassanelli. Applications of deep learning and reinforcement learning to biological data. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2063–2079, 2018.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[28] H. L. Morgan. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113, 1965.

[29] AkshatKumar Nigam, Pascal Friederich, Mario Krenn, and Alán Aspuru-Guzik. Augmenting genetic algorithms with deep neural networks for exploring the chemical space. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[30] Raghunathan Ramakrishnan, Pavlo Dral, Matthias Rupp, and Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 08 2014.

[31] RdKit. Rdkit: Open-source cheminformatics, 2006.

[32] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010. PMID: 20426451.

[33] Lars Ruddigkeit, Ruud van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864—2875, November 2012.

[34] Stefan Schaal. Learning from demonstration. In M. C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1997.

[35] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[36] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 287–294, New York, NY, USA, 1992. Association for Computing Machinery.

[37] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[38] Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning, 2018.

[39] A Stroh, Y Hasegawa, Philipp Schlatter, and B Frohnapfel. Global effect of local skin friction drag reduction in spatially developing turbulent boundary layer. *Journal of Fluid Mechanics*, 805:303–321, 2016.

[40] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[41] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[42] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[43] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, October 2019.

[44] H. Weller and H. Jasak. OpenFOAM, 2011.

[45] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136):1–46, 2017.

[46] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6412–6422, 2018.

[47] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N. Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific Reports*, 9(1), Jul 2019.

7 Supplementary Information

Comparison of querying strategies for retraining

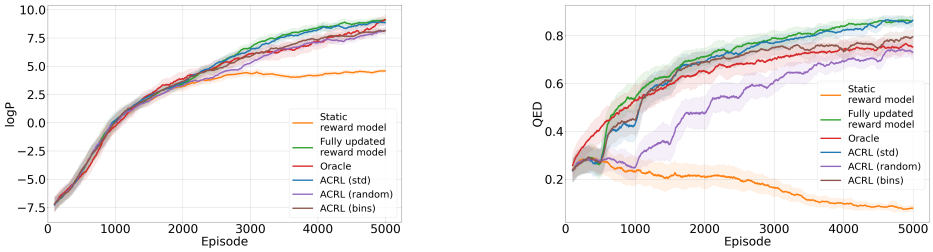
The three strategies for selection of points presented were compared on a constant number of selected points (800) (Figure 3). In each case, three reward models are initially trained with three different train-test splits of the original QM9 dataset and used for prediction later on. The mean of the three reward models' predictions is used as a final reward for the agent to maximize, and the standard deviation of these are calculated to give an idea of the uncertainty on prediction. Based on these models, three selection modes are studied. In the first setting, the models are retrained by randomly sampling a number of points from the initial QM9 test set as well as newly generated points during the learning process. In a second setting, the points with the highest standard deviations of model predictions are sampled and the models are updated using these points as train set. This is based on the assumption that the points with the highest standard deviations of model predictions (points in which the three models "disagree") are more likely to come from outside the original train distribution, thus potentially representing the points that the model needs to learn from in order to improve its predictions during the agent training process. In this sense, having three reward models instead of only one could provide a good basis for the selection of points. Finally, a third approach in sampling points is based on classifying the previously obtained test set and newly generated points in different bins before selecting points with the highest standard deviation (of model predictions). This stems from the fact that in certain situations, points with the highest standard deviations could represent outliers, and therefore could prevent the reward models from learning the main trend of the data. The strategy of bin-based selection offers a solution to this by selecting points with the highest standard deviations while respecting the initial data distribution. It is important to note that all these sampling processes are done separately for each model to be retrained, since their respective starting train and test sets are not the same to start with. Therefore, the three models will never be updated on the same training data, and will thus provide independent predictions, depending on the points they were trained on. Overall, the standard deviation based sampling method performed best and was thus used in the main part of this paper.

Comparison of the number of points selected for retraining

Given the best strategy (standard deviation based), the effect of the number of points selected for reward model retraining was studied (Figure 4). The minimal number of points that was consistently comparable to the real reward was 400 points.

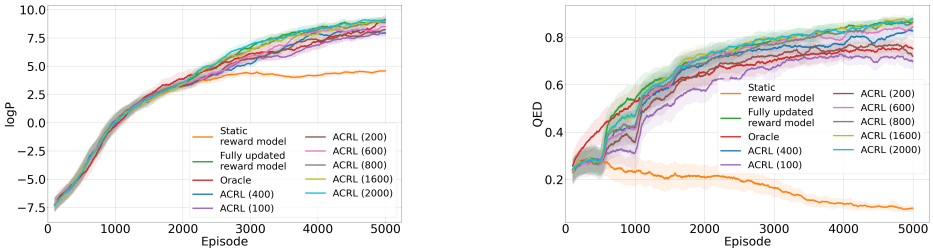
Study of the effect of varying degrees of randomness on learning

The ϵ values in the MolDQN paper start with values of 100% and decrease exponentially at each episode until they reach a percentage of 1% at episode 5000. In this study, our aim was to compare the effect of the ϵ end value (last value of randomness) on the real reward agent at episode 4800, considering that the agent does not choose any more random actions from episode 4800 to 5000 (to better guide it towards the end goal). Results are shown in Figure 5. We concluded that increasing the



(a) logp (b) QED

Figure 3: Comparison of different selection modes, random (red), standard deviation based (yellow) and bin-based (purple) on 800 sampled points.



(a) logp (b) QED

Figure 4: Comparison of reward models retrained on a varying number of points selected based on standard deviation

ϵ end values (increasing randomness, with the hope of favoring exploration) did not help the agent reach better rewards.

Study of the effect of the decay function form

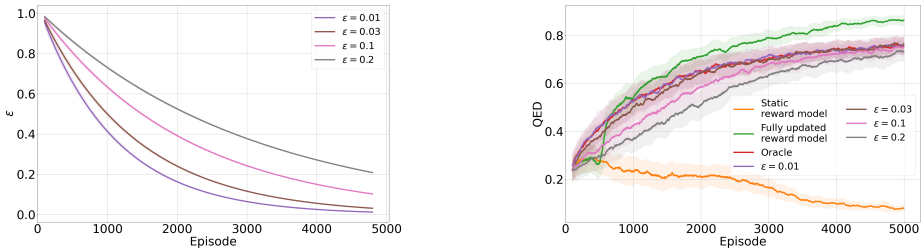
The ϵ-decay function used in MolDQN is an exponential decay function. In this study, we choose to study the effect of varying decay function forms by adding a linear component to the exponential function with varying fractions. The equation is the following:

$$\epsilon(t) = \epsilon_0(\lambda(1 - \beta t) + (1 - \lambda)\alpha^t) \tag{3}$$

with $1 - \beta t$ and α^t the linear and exponential components respectively, ϵ_0 the starting randomness (at 100%), β and α constants that depend on the starting and end values (we choose an end value of 1% at episode 4800), and λ the fraction (relative importance) of the linear component. At a λ of 0, the decay-function is fully exponential, and at a λ of 1, it is fully linear. Results are shown in Figure 6. We conclude here that a fully exponential function is more convenient for the learning of the agent.

Comparison of different mean constraints

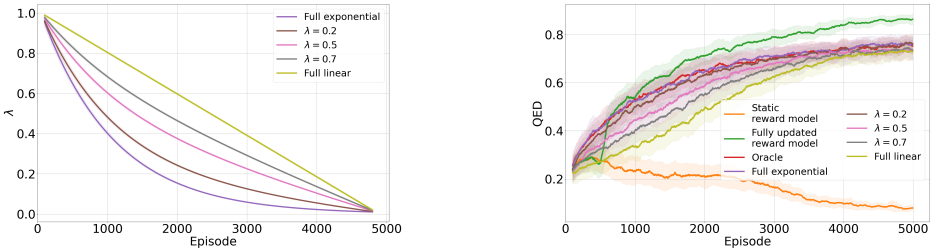
In this section, we present a more detailed description of the results for the drag optimization task. Our initial training dataset contains 5000 random profiles with a mean value centered around ± 0.002 on each side. For a constraint configuration matching the distribution of the initial dataset, Figures 7a and 7b show the evolution of drag and reward, respectively. In order to test extrapolation and generalization capabilities of our ACRL method, we repeated the same experiment with a larger constraint interval which lies outside the initial training distribution. The results in Figures 8a and 8b show that ACRL is able to explore the underrepresented space well, in contrast to a static reward model which fails to guide the agent to explore the relevant solution spaces. The higher variance stems from the fact that higher mean values (or equivalently, higher total volume) trivially reduce drag. Thus, in this experiment the agent encounters a higher diversity of states in terms of their constraint. Even though most of the observed states lie outside the initial training distribution, an ACRL agent is still able to explore the relevant low-drag space. The importance of actively updating



(a) ϵ-values' exponential decay with varying end points

(b) Results on the QED task

Figure 5: The effect of increasing randomness by reaching different ϵ end values with the same exponential decay function.

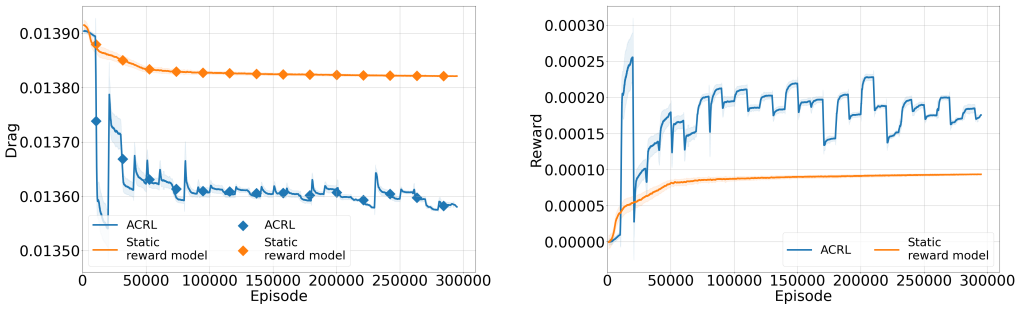


(a) Different forms of ϵ-decay functions

(b) Results on the QED task

Figure 6: The effect of different ϵ-decay functions with the same randomness end value.

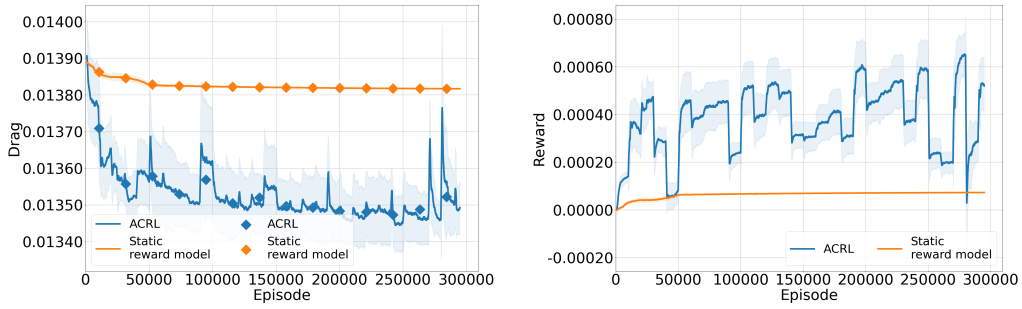
the reward model during training is reflected by the results of agents using a static reward model in both experiments. Both agents trained with a static reward model achieve very similar results in terms of drag, even though drag distributions vary considerably between the experiments. Only the ACRL agents are able to capture the variance of drag well, which is especially high in Figure 8 due a larger constraint interval. In contrast, static models fail to move outside their initially modelled distribution, even though they predict drag values of encountered states very accurately.



(a)

(b)

Figure 7: Results for different mean constraints in $[0.0019, 0.0021]$.



(a) (b)

Figure 8: Results for different mean constraints in $[0.0015, 0.0025]$.