DEPLOYING OFFLINE REINFORCEMENT LEARNING WITH HUMAN FEEDBACK

Ziniu Li*

The Chinese University of Hong Kong, Shenzhen ziniuli@link.cuhk.edu.cn

Ke Xu, Liu Liu, Langing Li, Deheng Ye, Peilin Zhao[†]

Tencent AI Lab {kaylakxu, leonliuliu, lanqingli, dericye, masonzhao}@tencent.com

March 14, 2023

ABSTRACT

Reinforcement learning (RL) has shown promise for decision-making tasks in real-world applications. One practical framework involves training parameterized policy models from an offline dataset and subsequently deploying them in an online environment. However, this approach can be risky since the offline training may not be perfect, leading to poor performance of the RL models that may take dangerous actions. To address this issue, we propose an alternative framework that involves a human supervising the RL models and providing additional feedback in the online deployment phase. We formalize this online deployment problem and develop two approaches. The first approach uses model selection and the upper confidence bound algorithm to adaptively select a model to deploy from a candidate set of trained offline RL models. The second approach involves fine-tuning the model in the online deployment phase when a supervision signal arrives. We demonstrate the effectiveness of these approaches for robot locomotion control and traffic light control tasks through empirical validation.

1 Introduction

Reinforcement learning (RL) offers a systematic approach to tackle sequential decision-making tasks [41]. RL methods utilize Markov Decision Processes (MDPs) to model the tasks [35], enabling agents to interact with an environment and enhance decision-making by maximizing long-term returns. Thanks to powerful neural networks, RL methods have achieved outstanding performance, surpassing even master-level expertise in various domains [31, 50, 40, 8, 37, 6, 53, 34].

A popular RL framework for real-world applications involves two key steps. The first step is training parameterized policy models using an offline dataset that has previously been collected by specific behavior policies. The second step is deploying these trained policy models in an online environment. In recent years, significant efforts have been dedicated to training offline RL models [13, 24, 25, 46, 23, 14]. The primary challenge in this approach is the lack of further data collection in the offline setting, which requires the agent to consider the epistemic uncertainty (i.e., subjective uncertainty due to limited samples) when optimizing policies. Consequently, various methods have been proposed and evaluated, which can be found in [29, 12] and their respective references.

Despite its benefits, offline training may not be perfect due to various factors such as dataset quality and hyperparameter choices. As a result, trained models may suffer from overfitting, leading to poorer generalization performance in new scenarios. This is particularly concerning when deploying RL methods in the online phase, as models may take

^{*}This work is done when Ziniu Li works as an intern in Tencent AI Lab.

[†]Corresponding author.

dangerous actions and unexpected results may occur. It is worth noting that the issue of overfitting is widely recognized in the machine learning community, and techniques such as cross-validation and early stopping have been proposed to evaluate model performance before deployment [39]. However, these methods often fail in RL due to the distributional shift problem [36]. In other words, the training and validation distributions may differ significantly in decision-making tasks, making it challenging to assess the offline models' performance without conducting online experiments.

Apart from the evaluation issue mentioned earlier, another concern that arises in industrial applications (such as power system control, autonomous driving, and traffic light control) is the importance of safety and ethics [15]. However, incorporating these factors into the training phase can be challenging as designing proper reward/penalty functions for them requires a significant amount of engineering effort [1]. Fortunately, in many applications, an expert policy (i.e., a human operator) can supervise the deployed RL model and provide feedback. For example, in the case of autonomous driving, people may have different preferences about the control system's decisions. Some may prioritize safety and comfort, while others may prioritize driving efficiency. In such scenarios, it is challenging to consider each person's preferences in the offline training phase, as feedback is only available in the online deployment phase. Therefore, offline RL methods may not perform well in this setting as they are not adaptive during online deployment. Although studies have emerged in the offline-to-online RL setting [49, 38, 28], these works have not considered online human feedback, which is crucial in practical applications.

After taking the above-mentioned considerations into account, it becomes essential to improve the performance of trained RL models during online deployment, particularly when human feedback is available. Please see Figure 1 for illustration. In this context, our objective is not only to maximize the environment return defined by each task but also to ensure that the decisions made by RL models are in line with what human experts expect. Thus, this paper proposes to maximize the concept of *online score*, which integrates both the environment return and human feedback (further elaborated in Section 3.2).

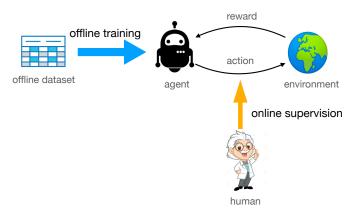


Figure 1: The framework of online deployment with human feedback. In our framework, a human expert supervises the RL models and provides additional feedback to improve their performance.

In this manuscript, we formalize the problem of online deployment with human feedback and propose two approaches to maximize the online score. The first approach is based on model selection, where we assume there are N pre-trained offline models, but their online scores are unknown in advance. To determine which model can achieve the highest online score with minimal trials, we propose using the upper confidence bound (UCB) algorithm [26, 2]. The UCB algorithm estimates the online score of each offline model optimistically and adaptively selects the one to deploy, taking into account the stochastic and uncertain nature of the environment.

For the same online deployment problem, our second approach is based on fine-tuning. In this scenario, we assume that we only have access to one specific offline RL model, but we can improve its performance by fine-tuning it in the online deployment phase using human feedback. Unlike the first approach, the expert provides direct suggestions on action selection, allowing us to improve the model's performance. To leverage the human feedback, we develop imitation-learning-based methods [20] that penalize the discrepancy between the model's and the human's decisions to improve the online score.

We conduct experiments on two tasks: robotics locomotion control [9] and traffic light control [45]. The goal of the first task is to train a robot to perform locomotion behaviors like humans, while in the second task, the objective is to control traffic lights to avoid congestion. We evaluate the performance of our proposed methods on these tasks. Specifically, we show that in the case of model selection, our approach can identify the best model from a candidate set with only about

100 trials. In the case of fine-tuning, we demonstrate that the online performance can be improved significantly with no more than 200 trials.

This paper is structured as follows. In Section 2, we review prior research in the field. Section 3 provides the necessary background and problem formulation. We then present our proposed methods for model selection and fine-tuning in Sections 4 and 5, respectively. Finally, we present the numerical results in Section 6.

2 Related Work

In this section, we provide an overview of previous research related to the topic of this paper.

Offline Reinforcement Learning. Offline reinforcement learning algorithms aim to train an effective policy using a dataset that has been previously collected. Over the past few decades, offline RL (also known as batch RL) has been extensively studied in terms of algorithm design [11, 13, 24, 25, 46, 23] and theoretical analysis [42, 33, 4, 47]. These works demonstrate that if the dataset has wide coverage and a small concentration coefficient in relation to the optimal policy, it is possible to accurately solve the Bellman equation using finite samples. In practical applications, the relevant theory studies suggest to consider the epistemic uncertainty in policy optimization. For example, BCQ [13] limits the action range to improve the policy, BRAC [46] employs KL regularization during policy optimization, and CQL [25] penalizes Q-values for out-of-distribution actions.

It should be noted that prior studies on offline reinforcement learning algorithms have a significant limitation. These algorithms often employ online evaluation to optimize architectures or find suitable hyper-parameters, which is impractical due to the high cost of online evaluation. In real-world applications, hyperparameter-free heuristics such as selecting the action with the highest Q-value [16] are commonly used, despite the lack of theoretical guarantees. In our experiments, we will compare the performance of our proposed framework with such heuristics.

Online Deployment. Several recent studies have focused on the online deployment problem. For example, [49] proposed a meta-episodic algorithm that addresses the exploration uncertainty issue and ensures uniformly conservative exploration. [38] employed model-based policy and value improvement operators to compute new training targets on existing data points. Additionally, [28] proposed a balanced experience replay scheme to address the online distribution shift issue. However, none of these works considered human feedback during the online deployment phase. In our framework, we consider two types of deployment plans: model selection and fine-tuning, and we review related works in the sequel.

Model Selection. Model selection is well studied in the supervised learning literature [39, 32]. Since we only have access to finite samples in practice, the generalization gap must be carefully considered when training an offline model. In supervised learning, training and testing data are independently and identically drawn from the same distribution. Various techniques such as early stopping and n-fold cross-validation have been developed to address overfitting in the training phase [39]. Nevertheless, reinforcement learning poses a unique challenge to model selection as training and testing data are not from the same distribution due to distributional shift [29].

In contrast, model selection in online learning is well-studied in the literature, particularly in the context of multi-arm bandit (MAB) problems [27]. In MAB, the learner seeks to identify the optimal arm based on partial and bandit feedback. In our scenario, we face a similar problem with partial and bandit feedback, making it suitable for MAB solutions. The upper confidence bound (UCB) algorithm is a popular approach for MAB, which provides nearly minimax optimal performance for regret minimization [26, 2]. Therefore, we propose implementing the UCB algorithm for effective model selection in reinforcement learning.

Fine-Tuning. Fine-tuning is a widely used technique in deep learning, especially in the field of transfer learning [17]. It involves taking a pre-trained model and further refining it for downstream tasks [51, 22, 5]. In the realm of reinforcement learning (RL), researchers have explored the use of imitation learning approaches [20] to initialize a model using human demonstrations, and then improve it further with online RL methods. One example of this is the DQfD algorithm proposed by [19], which combines temporal difference updates with supervised classification of the demonstrator's actions. Another algorithm, LOKI, introduced by [7], starts with a few iterations of imitation learning before switching to a policy gradient RL method. A Bayesian formulation using prior information to fine-tune is considered in [30].

3 Problem Formulation

In this section, we first introduce the background of reinforcement learning (RL) in Section 3.1. Subsequently, we formalize the problem of online deployment with human feedback in Section 3.2.

3.1 Background

Markov Decision Processes. A standard tool to study reinforcement learning is the Markov Decision Process (MDP) [35], which can be described by a tuple $(S, A, p, r, \rho, \gamma)$. Here S and A are the state and action space, respectively. Moreover, p is the system transition function; i.e., p(s'|s,a) determines the probability of the next state s' condition on the current state-action pair (s,a). $r: S \times A \to \mathbb{R}$ specifies the reward signal and $\rho(\cdot)$ is the initial state distribution. Finally, $\gamma \in (0,1)$ is the discounted factor in computing the long-term return.

For a deterministic policy $\pi: \mathcal{S} \to \mathcal{A}$, its expected long-term return is denoted by

$$V(\pi) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r(s_{t}, a_{t}) \mid s_{0} \sim \rho, a_{t} = \pi(s_{t}), s_{t+1} \sim p(\cdot | s_{t}, a_{t}), \forall t \geq 0\right].$$

To further measure the quality of policy π , Q-value function is introduced:

$$Q^{\pi}(s, a) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r(s_{t}, a_{t}) \mid s_{0} = s, a_{0} = a; a_{t} = \pi(s_{t}), s_{t} \sim p(\cdot | s_{t}, a_{t}), \forall t \geq 1\right],$$

i.e., the expected long-term return starting from (s, a). It is well-known that the optimal Q-value function Q^* satisfies the Bellman optimality equation:

$$Q^{\star}(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\max_{a'} Q^{\star}(s',a') \right]. \tag{1}$$

The optimal policy π^* is defined as the greedy policy with respect to Q^* , i.e., $\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a)$. In this manuscript, we mainly consider model-free approaches, so when we mention an RL model, we mean a Q-value function or the greedy policy associated with this Q-value function.

Offline Reinforcement Learning. In the framework of RL, the transition function is assumed to be unknown, so Equation (1) cannot be directly solved. Instead, RL methods typically have access to samples obtained from environments in an online or offline manner. In this manuscript, we consider the offline scenario, where a dataset $\mathcal{D} = \{(s, a, r, s')\}$ is provided to train RL models [11]. In a simple form where the action space is finite, offline RL methods aim to minimize the Bellman error from finite samples:

$$\theta_{\text{critic}} = \underset{\theta}{\operatorname{argmin}} \sum_{(s, a, r, s') \in \mathcal{D}} \left(Q_{\theta}(s, a) - r(s, a) - \gamma \max_{a'} Q_{\theta}(s', a') \right)^{2}, \tag{2}$$

where θ_{critic} is the parameter of Q-value function $Q_{\theta_{\text{critic}}}$ (in the context of actor-critic methods [21], the Q-value function is also called a critic). Then, the policy can be extracted by greedily optimizing $Q_{\theta_{\text{critic}}}$, i.e., $\pi(s) = \underset{argmax_{a \in \mathcal{A}}}{\operatorname{argmax}} Q_{\theta_{\text{critic}}}(s, a)$.

For applications with continuous action control, the above training method cannot be applied as the maximization over \mathcal{A} cannot be directly implemented. In this case, we need an additional actor network $\pi_{\theta_{\text{actor}}}$ to extract the greedy policy from $Q_{\theta_{\text{critic}}}$:

$$\theta_{\text{actor}} = \underset{\theta}{\operatorname{argmax}} \sum_{s \in \mathcal{D}} Q_{\theta_{\text{critic}}}(s, \pi_{\theta}(s)).$$

where $\theta_{\rm actor}$ is the parameter of the policy $\pi_{\theta_{\rm actor}}$ (in the context of actor-critic methods, the policy is also called an actor). Accordingly, the optimization problem (2) becomes

$$\theta_{\text{critic}} = \underset{\theta}{\operatorname{argmin}} \sum_{(s, a, r, s') \in \mathcal{D}} \left(Q_{\theta}(s, a) - r(s, a) - \gamma Q_{\theta}(s', \pi_{\theta_{\text{actor}}}(s')) \right)^{2}.$$

Advanced offline RL methods additionally consider the epistemic uncertainty in the above optimization; please refer to [29] and references therein.

¹Note that there always exists a deterministic policy that can achieve the optimal return [35], so it does not lose generality to consider the deterministic policies.

3.2 Deployment of Offline RL Models

In this section, we explore the challenges involved in deploying trained RL models, which have significant implications for real-world applications. Although a well-trained offline RL model is expected to perform well in online deployment, this is often not the case. One of the primary reasons for this is the difficulty in accurately assessing the offline RL model's performance in the offline scenario. In supervised learning, we can use a validation dataset to choose a model that can generalize well. However, estimating the actual performance of an offline RL model using a validation dataset is challenging due to the distributional shift problem.

Another critical issue in deploying an offline RL model is safety [15]. To address this concern, a human operator often supervises the trained RL model in the online phase to ensure that it does not take dangerous actions. Typically, this human operator has an expert policy $\pi^{\rm E}$. If the RL agent deviates significantly from the expert policy, it incurs a penalty. Therefore, it is desirable for the trained RL model to balance the environment's return and the expert policy's expectation.

In this manuscript, we propose the expected online score as a metric to evaluate the performance of RL models in the online deployment phase, taking into account the challenges discussed earlier. The online score is defined as follows:

$$S = \mathbb{E}\left[\alpha_1 \cdot \sum_{t=1}^{T} r(s_t, \pi(s_t)) - \alpha_2 \cdot \sum_{t=1}^{T} \mathbf{1}\left\{\pi(s_t) \neq \pi^{\mathrm{E}}(s_t)\right\}\right] \quad \text{(discrete action control)}.$$

Here, T is the maximum trajectory length of an episode, $\alpha_1 > 0$ and $\alpha_2 > 0$ are scaling factors, and $\mathbf{1}\left\{\cdot\right\}$ is the indicator function. The expectation is taken over the randomness in environment transitions. For continuous action control tasks, the second term in (3) is too restrictive, so we introduce a relaxation that considers the squared distance between $\pi(s_t)$ and $\pi^{\mathrm{E}}(s_t)$, with a tolerance parameter $\tau > 0$. That is, the second term in (3) is replaced with

$$S = \mathbb{E}\left[\alpha_1 \cdot \sum_{t=1}^{T} r(s_t, \pi(s_t)) - \alpha_2 \cdot \sum_{t=1}^{T} \mathbf{1} \left\{ \left\| \pi(s_t) - \pi^{\mathrm{E}}(s_t) \right\|^2 > \tau \right\} \right] \quad \text{(continuous action control)}. \tag{4}$$

We note that in both equations (3) and (4), the second term acts as a constraint by quantifying the degree of disagreement between the RL models and the human expert.

4 Online Model Selection

In this section, we propose a model-selection-based approach to determine the best offline RL model for online deployment. Suppose we have trained N different offline models using various methods such as different random seeds, training techniques, and hyperparameters. However, it is unclear which model would perform the best in online deployment. Therefore, we aim to select the optimal model by conducting multiple online deployment trials. This task can be viewed as an online model selection problem.

In this manuscript, we focus on the (cumulative) regret criterion as a model selection algorithm. The (cumulative) regret criterion is defined as follows:

$$\operatorname{regret}_{K} = \sum_{k=1}^{K} (S_{k} - S^{\star}), \tag{5}$$

where S_k is the online score obtained when deploying a specific model in iteration k. S^* represents the expected best score that can be obtained by deploying the optimal offline model in hindsight, i.e., $S^* = \max_{i=1,\dots,N} S^i$, where S^i is the online score of the i-th model. Ideally, we want the model selection algorithm to identify the optimal model quickly, leading to sublinear growth of the regret.

It is worth noting that we only deploy one specific model at a time and receive its corresponding feedback. Thus, we are faced with a bandit feedback setting where the feedback information is partial, leading to an exploration-and-exploitation dilemma [27]. This means that we must try each offline model multiple times (i.e., exploration) before identifying the optimal one for online deployment (i.e., exploitation). To balance the trade-off between exploration and exploitation adaptively, we can employ well-known UCB (upper confidence bound) strategies [26, 2], which construct an optimistic estimate of the feedback to guide exploration. Specifically, in each iteration k, we can use the following rule to make decisions:

$$\underset{i}{\operatorname{argmax}} \operatorname{UCB}_{k}^{i} := \widehat{S_{k}^{i}} + \beta \cdot \operatorname{bonus}_{k}^{i}, \tag{6}$$

where $\widehat{S_k^i}$ is the estimation of the feedback for decision i (explained later), $\beta > 0$ is the scaling factor, and bonus $i = \sqrt{1/n_k^i}$, where n_k^i is the number of times decision i has been attempted up to iteration k. Although this UCB strategy is a greedy approach, it can achieve sublinear regret due to the optimistic estimation [27]. Intuitively, the bonus term encourages exploration, ensuring that each decision is tested sufficiently.

In our scenario, each decision refers to a specific offline model, and each iteration corresponds to a rollout of the policy, i.e., an episode. The feedback obtained in each iteration is a noisy score that combines the long-term environment return with a penalty for violating the human's intents, as follows:

$$s_k = \alpha_1 \cdot \sum_{t=1}^{T} r(s_t, a_t) - \alpha_2 \cdot \sum_{t=1}^{T} \mathbf{1} \left\{ \pi(s_t) \neq \pi^{E}(s_t) \right\}.$$

Here, α_1 and α_2 are scaling factors defined previously, and π^E is the expert policy. A similar formulation can be obtained for continuous action control tasks by replacing the hard constraint with the soft constraint. Note that s_k is a random variable due to the randomness of environment transitions, and we have $S_k = \mathbb{E}[s_k]$.

The online score combines the environment rewards and human preferences for the agent's actions. Trained offline models can have dramatically different online scores. Reward-pursuit models, which have no particular constraints in offline policy optimization, may generate actions that human believe are risky and dangerous. Thus, such models receive a large penalty in the online deployment phase and a low online score. Conversely, conservative models with explicit constraints in offline policy optimization may not achieve excellent environment-defined performance, and radical experts may not like them, resulting in a low online score. In both cases, note that we do not know how the models will perform in the online deployment phase.

We have outlined the procedure for applying the UCB strategy to select offline models in our problem in Algorithm 1. In each iteration, we use the UCB strategy to select the most suitable offline model. The model index is denoted by i_k in iteration k. We compute \widehat{S}_k^i (appeared in (6)) by the empirical mean:

$$\widehat{S}_{k}^{i} = \frac{\sum_{k'=1}^{k} s_{k'} \mathbf{1}(i_{k'} = k)}{n_{k}^{i}},$$
(7)

where n_k^i is the total times of deploying the *i*-th model. In Algorithm 1, we use X_k^j to compute the numerator in (7).

Although Algorithm 1 is straightforward, it has been shown to achieve good practical performance and provides reasonable theoretical guarantees. According to the literature [27], the cumulative regret of Algorithm 2 scales proportionally to $\widetilde{\mathcal{O}}(\sqrt{NK})$, by properly choosing β . In practice, a constant value of β usually performs well. Note that as K goes to ∞ , we have the averaged regret $\widetilde{\mathcal{O}}(\sqrt{NK}/K) \to 0$. The theory also implies that utilizing prior knowledge to select suitable candidate policies with small N can significantly minimize regret.

Algorithm 1 UCB for online model selection

```
Input: N: number of offline models, \beta: exploration coefficient, and offline models M^1, \dots, M^N.
 1: Initialize X_0 \in \mathbb{R}^N \leftarrow 0, n_0 \in \mathbb{R}^N \leftarrow \hat{0}.
 2: for iteration k = 1, 2, \cdots do
           if k \leq N then
 3:
 4:
                i_k = k.
 5:
                i_k = \operatorname{argmax}_i \frac{X_k^i}{n_i^i} + \beta \cdot \sqrt{\frac{1}{n_i^i}}.
 6:
 7:
           Deploy the model M^{i_k} and receive the score s_k.
 8:
           for each j=1,\cdots,N do if j=i_k then
 9:
10:
                      Update: n_k^j \leftarrow n_{k-1}^j + 1 and X_k^j \leftarrow X_{k-1}^j + s_k.
11:
12:
                      Update: n_k^j \leftarrow n_{k-1}^j and X_k^j \leftarrow X_{k-1}^j.
13:
14:
15:
           end for
16: end for
```

As a meta-algorithm, Algorithm 1 can be applied to both discrete and continuous action control tasks. It is important to note that although UCB includes an exploration phase in the model selection process, it differs significantly from

online exploration in a standard RL framework. In Algorithm 1, we consider only well-trained offline models and test them in the online phase, ensuring the quality of the exploration behavior. On the other hand, in a standard online RL framework, agents may attempt dangerous or harmful actions to explore, which can lead to unexpected results in some applications.

An advantage of Algorithm 1 is its minimal computation complexity in the online phase, as it only requires storing a few vectors and updating them with simple calculations. However, the quality of the candidate set determines the online performance of this selection problem. If the quality of the N offline RL models is poor, the final performance of Algorithm 1 may not be acceptable. In such cases, fine-tuning can be used to further improve the performance of the offline models, and we will discuss this method in the following sections.

5 Online Model Fine-Tuning

In this section, we explore fine-tuning approaches to improve the quality of offline models during online deployment. Our inspiration for this method comes from the fine-tuning of deep neural networks in downstream tasks [51, 22, 5], as well as fine-tuning of deep RL models trained from human demonstrations [19, 7, 3]. Specifically, we consider a scenario where a human expert policy can override the model's action when the expert's decision deviates significantly from the model's action. We log these events in the form of (s, a, r, s'), where a is the expert's decision. Later, we extract samples from the log and construct a dataset $\mathcal{D}^o = (s, a, r, s')$, which we use to fine-tune the models. Depending on whether the action space is continuous or discrete, we employ two different fine-tuning methods.

Continuous action control. In this scenario, we have two models: an actor model $\pi_{\theta_{\text{actor}}}$ and a critic model $Q_{\theta_{\text{critic}}}$ (refer to Section 3.1). It is natural to first optimize the critic by minimizing the Bellman error:

$$\theta_{\text{critic}} = \underset{\theta}{\operatorname{argmin}} \sum_{(s, a, r, s') \in \mathcal{D}^{\circ}} \left(Q_{\theta}(s, a) - r(s, a) - \gamma Q_{\theta}(s', \pi_{\theta_{\text{actor}}}(s')) \right)^{2}.$$
 (8)

Then, we can improve the actor by maximizing its Q-value. To effectively follow the expert's guidance, we additionally train the actor by a mean-squared-error between the model's output and the expert's action (c.f. the second term in (9)). This loss function is inspired by imitation learning theory [48]. As a result, the trained actor may perform well in maximizing the environment return and following the expert, achieving a high score defined in (3).

$$\theta_{\text{actor}} = \underset{\theta}{\operatorname{argmax}} \sum_{s \in \mathcal{D}^o} Q_{\theta_{\text{critic}}}(s, \pi_{\theta}(s)) - \sum_{(s, a) \in \mathcal{D}^o} \|\pi_{\theta}(s) - a\|_2^2.$$
(9)

Discrete action control. Different from the previous case, there is no explicit component of policy or actor in the discrete action control applications. Instead, we only have a Q-value function. Hence, we cannot directly apply the above approach. Following [19], we consider a margin-based loss function to increase the gap between the expert's action and the other actions. Concretely, the margin $\Delta > 0$ is a hyper-parameter, and this margin-based loss function incentivizes the expert action value $Q_{\theta_{\text{critic}}}(s,a)$ is at least larger than Q(s,a') for $a' \neq a$ by a margin Δ . In this way, the greedy policy is more likely to select the expert action a.

$$\theta_{\text{critic}} = \underset{\theta}{\operatorname{argmin}} \sum_{(s,a,r,s')\in\mathcal{D}^o} \left\{ \left(Q_{\theta}(s,a) - r(s,a) - \gamma Q_{\theta}(s',\pi_{\text{actor}}(s')) \right)^2 + \max_{a'} \left[Q_{\theta}(s,a') + \ell_{\Delta}(a,a') - Q_{\theta}(s,a) \right] \right\}.$$

$$(10)$$

where $\ell_{\Delta}(a, a') = \Delta$ if $a \neq a'$ and 0 otherwise.

In Algorithm 2, we present our proposed approach based on fine-tuning. However, it's important to note that in real-world scenarios, decisions must be made in real-time. Therefore, it's crucial to record only those state-action pairs where the expert is strongly dissatisfied with the decision made by the RL model. Otherwise, the log would become too large, leading to a significant computation load for the fine-tuning process. We address this issue in Lines 6-9 of Algorithm 2.

6 Experiments

In this section, we conduct experiments to verify the effectiveness of the proposed methods.

Algorithm 2 Online Fine-Tuning

```
Input: Trained offline model.
 1: Initialize \mathcal{D}^o \leftarrow \emptyset.
 2: for iteration k = 1, 2, \cdots do
          for time step t = 1, \dots, T do
 3:
 4:
               Observe the current state s_t.
              Select the action a_t.
 5:
               if ||a_t - \pi^{\mathrm{E}}(s_t)||^2 > \tau (or a_t \neq \pi^{\mathrm{E}}(s_t) for discrete action control) then
 6:
                   Implement the action \pi^{E}(s_t).
 7:
                   Receive the environment reward r_t and observe the next state s_{t+1}.
 8:
                   Update \mathcal{D}^o \leftarrow \mathcal{D}^o \cup \{(s_t, \pi^{\mathrm{E}}(s_t), r_t, s_{t+1})\}.
 9:
10:
               else
                   Implement the action a_t.
11:
12:
                   Receive the environment reward r_t and observe the next state s_{t+1}.
13:
               end if
14:
          end for
          if \mathcal{D}^o is not empty then
15:
              if Action space is continuous then
16:
                   Fine-tune the model by (8) and (9).
17:
18:
               else
19:
                   Fine-tune the model by (10).
20:
               end if
21:
          end if
          \mathcal{D}^o \leftarrow \emptyset.
22:
23: end for
```

6.1 Robotics Locomotion Control

This section focuses on three locomotion control tasks: HalfCheetah-v2, Hopper-v2, and Walker2d-v2, as described in [10]. These tasks aim to train a robot to perform human-like locomotion behaviors, using joint angle and velocity information as states and low-level motor controls as actions. Refer to Figure 2 for a visual representation of the locomotion tasks.

For illustrative purpose, we employ an online RL algorithm, specifically the SAC algorithm [18], to obtain an expert policy. Note that SAC is run for 1 million steps. The replay buffer from SAC serves as our offline dataset.



Figure 2: Illustration of robotics locomotion control, simulated by MuJoCo [43].

Numerous approaches exist for training offline RL models, as seen in [13, 25, 46, 23]. For this study, we opt to use the CQL algorithm [25] due to its simplicity in implementation, although other methods can also be considered. CQL incorporates a penalty term on out-of-distribution actions during training, resulting in five offline RL models with different scales of this penalty term. The performance of these models is presented in Table 5 in the Appendix. In computing the online score, we appropriately scaled the environment reward and penalty terms for the considered tasks; refer to (3):

```
\begin{split} \text{HalfCheetah-v2}: \quad & \alpha_1 = 1/8500, \alpha_2 = 1/1000, \\ \text{Hopper-v2}: \quad & \alpha_1 = 1/3500, \alpha_2 = 1/1000, \\ \text{Walker2d-v2}: \quad & \alpha_1 = 1/4000, \alpha_2 = 1/1000. \end{split}
```

Table 1: Performance of online model selection algorithms for three locomotion control tasks. Note that the metric of human disagreement measures the number of actions on which the human expert disagrees with the model, as defined in (3) and (4). Here digits correspond to the averaged results, the symbol \pm corresponds to the stand deviation of 5 experiments with different random seeds, and the symbol \uparrow indicates that higher values are better, while the symbol \downarrow indicates the opposite (same as other tables).

		Environment Return (†)	Human Disagreement (↓)	Online Score (†)
HalfCheetah-v2 Hopper-v2 Walker2d-v2	Highest Q	8682±34	554±7	$0.46_{\pm 0.01}$
	Random Ensemble	$8246_{\pm 147}$	$162_{\pm 67}$	$0.78_{\pm0.10}$
	Algorithm 1	8308±136	$72_{\pm 11}$	$0.91 \scriptstyle{\pm 0.00}$
	Oracle	8292	59	0.90
Hopper-v2	Highest Q	3387±1	194±1	0.77±0.00
	Random Ensemble	$2756_{\pm 465}$	$225_{\pm 41}$	$0.54 \scriptstyle{\pm 0.17}$
	Algorithm 1	$3390_{\pm 6}$	$198_{\pm 6}$	$0.77 \scriptstyle{\pm 0.00}$
	Oracle	3387	193	0.77
Walker2d-v2	Highest Q	$3627_{\pm 325}$	688±56	$0.23_{\pm 0.02}$
	Random Ensemble	$3685_{\pm 274}$	$452_{\pm 96}$	$0.40_{\pm 0.10}$
	Algorithm 1	$4100_{\pm 46}$	$329_{\pm 9}$	$0.70 \scriptstyle{\pm 0.00}$
	Oracle	4106	328	0.70

The tolerance parameter $\tau = 0.09$ and the maximum trajectory length T = 1000.

6.1.1 Online Model Selection

Using the five trained offline models, we implement model selection during the online deployment phase with Algorithm 1. For Algorithm 1, we set the hyper-parameter β to 1. Our baselines include the following:

- Highest Q: This method selects the model with the highest Q-value function trained by offline datasets, which has been considered in prior work [16].
- Random Ensemble: This method randomly selects a model from the candidate models to deploy. This method is not adaptive during the online phase.

In addition, we also consider the oracle, which directly selects the model that can maximize the online score. Note that this method cannot be used in practice, and its performance serves as the upper limit for all methods.

We present the numerical results in terms of online scores in Figure 3. Notably, we observe that with no more than 100 iterations, the performance of Algorithm 1 is close to the optimal score. On the other hand, other methods like Highest Q and Random Ensemble do not perform well. The performance of the trained policies is reported in Table 1, where the model selected by Algorithm 1 achieves a high online score.

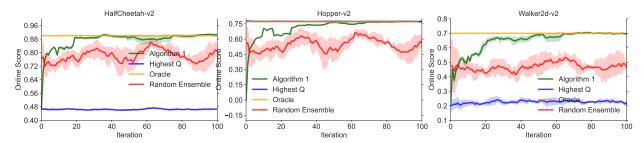


Figure 3: Online score (the higher the better) of Algorithm 1 for the robotics locomotion control tasks. Solid lines correspond to the mean and shaded regions correspond to the 95% confidence interval over 5 random seeds (same as other figures).

		Environment Return (†)	Human Disagreement (↓)	Online Score (†)
HalfCheetah-v2	Without Fine-tuning Algorithm 2	$\begin{array}{ c c }\hline 7362_{\pm 46} \\ 7353_{\pm 90} \end{array}$	93±7 67±10	$\begin{array}{c} 0.76 \scriptstyle{\pm 0.00} \\ 0.80 \scriptstyle{\pm 0.00} \end{array}$
Hopper-v2	Without Fine-tuning Algorithm 2	$\begin{vmatrix} 3368_{\pm 2} \\ 3374_{\pm 2} \end{vmatrix}$	$\begin{array}{c} 264_{\pm7} \\ 81_{\pm5} \end{array}$	$\begin{array}{c} 0.70 \scriptstyle{\pm 0.00} \\ 0.88 \scriptstyle{\pm 0.00} \end{array}$

 $253_{\pm 12}$

 $118_{\pm 14}$

 $0.69 \scriptstyle{\pm 0.00}$

 $0.84_{\pm 0.00}$

 $3787 \scriptstyle{\pm 39}$

 $3843_{\pm 25}$

Table 2: Performance of online fine-tuning algorithms for three locomotion control tasks.

6.1.2 Online Model Fine-Tuning

Walker2d-v2

Without Fine-tuning

Algorithm 2

In this part, we consider the online model fine-tuning approaches for deploying RL models as introduced in Section 5. Without loss of generality, we select the offline model with the worst performance to fine tune (refer to Table 5 in Appendix for the detailed performance of offline models).

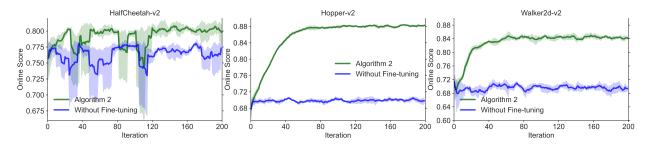


Figure 4: Online score of Algorithm 2 for the robotics locomotion control tasks.

We report the numerical results in Figure 4 and Table 1. From Figure 4, we see that by minimizing the mean squared error to fine-tune the RL model further, its online performance can be improved, compared with that without fine-tuning. This validates the effectiveness of Algorithm 2.

6.2 Traffic Light Control

This section considers a traffic light control task, as shown in Figure 5. The traffic light at the roadway is the agent that needs to be controlled, while traffic flows serve as the environment. The states of the system include the queue length (i.e., the number of vehicles in incoming lanes) and the current phase (i.e., the movement signal of the traffic light, such as a green light on the west-east). The goal is to minimize queue length and avoid congestion by adaptively selecting the movement signal. We use real traffic data from Hangzhou, China, which was provided by the TSCC competition².

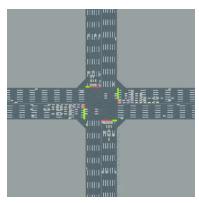


Figure 5: Illustration of the traffic light control, simulated by CityFlow [52].

²https://github.com/tianrang-intelligence/TSCC2019

Table 3: Performance of online model selection algorithms for the traffic light control task.

	Environment Return (†)	Human Disagreement (↓)	Online Score (†)
Highest Q	908±0	$320_{\pm0}$	$0.75_{\pm 0.00}$
Random Ensemble	898±11	$565{\scriptstyle\pm199}$	$0.57 \scriptstyle{\pm 0.03}$
Algorithm 1	916±3	$270_{\pm 17}$	$0.78_{\pm 0.00}$
Oracle	913	270	0.79

Table 4: Performance of online fine-tuning algorithms for the traffic light control task.

	Environment Return (†)	Human Disagreement (↓)	Online Score (†)
Without Fine-tuning	$906_{\pm 0}$	$457{\scriptstyle \pm 0}$	$0.45_{\pm 0.00}$
Algorithm 2	$928_{\pm 0}$	$147\pm o$	$0.77 \scriptstyle{\pm 0.00}$

Since the action space for traffic light control is discrete and finite, we consider using DQN-based approaches [31]. To obtain an expert policy, we first train a double DQN agent [44] for 100 iterations. We then collect an offline dataset of 500K samples by using ϵ -greedy ($\epsilon = 0.2$) to roll out the expert policy. We use CQL [25] to train offline RL models with different penalty scales, resulting in five models. Their performance is summarized in Table 6 in Appendix. We set $\alpha_1 = 1/1000$ and $\alpha_2 = 1/1800$ for this traffic light control task.

6.2.1 Online Model Selection

In this section, we use Algorithm 1 to select offline RL models during the online deployment phase, with a chosen hyper-parameter β of 1. We consider the same baselines as in Section 6.1.1 and present the online score curve in Figure 6. Our observation is consistent with the previous section: after 100 iterations, the performance of Algorithm 1 is comparable to the optimal one. We report the detailed performance of trained policies in Table 3.

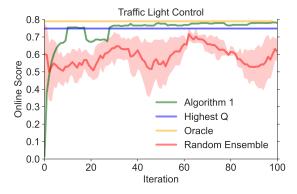


Figure 6: Online score of Algorithm 1 for the traffic light control task.

6.2.2 Online Model Fine-Tuning

In this part, we try to fine-tune the trained RL model in the online deployment phase. Again, we select the model withe the worst performance to fine-tune. The hyperparameter $\Delta=1$ in (10) is used in experiments. We visualize the online score curve in Figure 7. We observe that without fine-tuning, the performance of the deployed RL model is poor. However, by fine-tuning this model via Algorithm 2 with 100 iterations, its performance can be significantly improved. The detailed performance of trained policies is reported in Table 4.

7 Conclusion

This manuscript explores effective deployment strategies for offline reinforcement learning (RL) models in the online phase, leveraging human feedback. Two approaches are proposed: model selection and fine-tuning. Experimental results demonstrate the effectiveness of these methods in achieving high online performance.

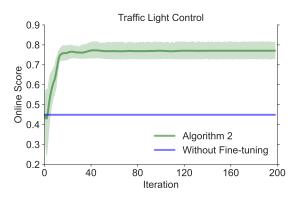


Figure 7: Online score of Algorithm 2 for the traffic light control task.

It should be noted that this work only considers scenarios where the human expert policy and environment are static. In some applications, these factors may change over time, and more sophisticated deployment methods may be required in the future.

References

- [1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, pages 1–8, 2004.
- [2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.
- [3] Y. Aytar, T. Pfaff, D. Budden, T. L. Paine, Z. Wang, and N. de Freitas. Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems 31*, pages 2935–2945, 2018.
- [4] J. Chen and N. Jiang. Information-theoretic considerations in batch reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1042–1051, 2019.
- [5] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1597–1607, 2020.
- [6] X. Chen and J. Wang. Inhomogeneous deep q-network for time sensitive applications. *Artificial Intelligence*, 312: 103757, 2022.
- [7] C. Cheng, X. Yan, N. Wagener, and B. Boots. Fast policy learning through imitation and reinforcement. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*, pages 845–855, 2018.
- [8] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602 (7897):414–419, 2022.
- [9] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 1329–1338, 2016.
- [10] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 1329–1338, 2016.
- [11] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.
- [12] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv*, 2004.07219, 2020.
- [13] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2052–2062, 2019.
- [14] C. Gao, K. Xu, K. Zhou, L. Li, X. Wang, B. Yuan, and P. Zhao. Value penalized q-learning for recommender systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2008–2012, 2022.

- [15] J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015.
- [16] D. Garg, P. Gupta, P. Malhotra, L. Vig, and G. Shroff. Batch-constrained distributional reinforcement learning for session-based recommendation. arXiv, 2012.08984, 2020.
- [17] I. Goodfellow, Y. Bengio, and A. Courville. Deep Learning. MIT Press, 2016.
- [18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1856–1865, 2018.
- [19] T. Hester, M. Vecerík, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. P. Agapiou, J. Z. Leibo, and A. Gruslys. Deep q-learning from demonstrations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 3223–3230, 2018.
- [20] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys*, 50(2):1–35, 2017.
- [21] V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances In Neural Information Processing Systems 12*, pages 1008–1014, 1999.
- [22] S. Kornblith, J. Shlens, and Q. V. Le. Do better imagenet models transfer better? In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019.
- [23] I. Kostrikov, R. Fergus, J. Tompson, and O. Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *Proceedings of the 38th International Conference on Machine Learning*, pages 5774–5783, 2021.
- [24] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems* 32, pages 11761–11771, 2019.
- [25] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems 33*, pages 1179–1191, 2020.
- [26] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6 (1):4–22, 1985.
- [27] T. Lattimore and C. Szepesvári. Bandit Algorithms. Cambridge University Press, 2020.
- [28] S. Lee, Y. Seo, K. Lee, P. Abbeel, and J. Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, pages 1702–1712, 2022.
- [29] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv*, 2005.01643, 2020.
- [30] Z. Li, Y. Li, Y. Zhang, T. Zhang, and Z.-Q. Luo. Hyperdqn: A randomized exploration method for deep reinforcement learning. In *Proceedings of th 10th International Conference on Learning Representations*, 2022.
- [31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.
- [32] M. Mohri, A. Rostamizadeh, and A. Talwalkar. Foundations of machine learning. MIT Press, 2018.
- [33] R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(5), 2008.
- [34] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *arXiv*, 2203.02155, 2022.
- [35] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.
- [36] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the 13rd International Conference on Artificial Intelligence and Statistics*, pages 661–668, 2010.
- [37] L. Roveda, A. Testa, A. A. Shahid, F. Braghin, and D. Piga. Q-learning-based model predictive variable impedance control for physical human-robot collaboration. *Artificial Intelligence*, 312:103771, 2022.
- [38] J. Schrittwieser, T. Hubert, A. Mandhane, M. Barekatain, I. Antonoglou, and D. Silver. Online and offline reinforcement learning by planning with a learned model. *Advances in Neural Information Processing Systems 34*, pages 27580–27591, 2021.

- [39] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.
- [40] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.
- [41] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. MIT press, 2018.
- [42] C. Szepesvári and R. Munos. Finite time bounds for sampling based fitted value iteration. In *Proceedings of the 22nd international conference on Machine learning*, pages 880–887, 2005.
- [43] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 5026–5033, 2012.
- [44] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016.
- [45] H. Wei, G. Zheng, H. Yao, and Z. Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2496–2505, 2018.
- [46] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.
- [47] T. Xie and N. Jiang. Batch value-function approximation with only realizability. In *Proceedings of the 38th International Conference on Machine Learning*, pages 11404–11413, 2021.
- [48] T. Xu, Z. Li, and Y. Yu. Error bounds of imitating policies and environments. In *Advances in Neural Information Processing Systems 33*, pages 15737–15749, 2020.
- [49] W. Xu, K. Xu, H. Bastani, and O. Bastani. Safely bridging offline and online reinforcement learning. *arXiv*, 2110.13060, 2021.
- [50] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo, Q. Chen, Y. Yin, H. Zhang, T. Shi, L. Wang, Q. Fu, W. Yang, and L. Huang. Mastering complex control in MOBA games with deep reinforcement learning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 6672–6679, 2020.
- [51] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems* 27, pages 3320–3328, 2014.
- [52] H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, and Z. Li. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The World Wide Web Conference*, pages 3620–3624, 2019.
- [53] Y. Zhang, P. Zhao, Q. Wu, B. Li, J. Huang, and M. Tan. Cost-sensitive portfolio selection via deep reinforcement learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):236–248, 2022.

A Experiment Details

Our algorithm implementation of SAC and CQL is based on the tianshou framework³. We use λ to indicate the scale of the penalty term in CQL. The performance of offline models is reported in Table 5 for robotics locomotion control tasks and in Table 6 for the traffic light control task. We note that results in Table 1, Table 2, Table 3, and Table 4 are based on the average of last 10 iterations.

³https://github.com/thu-ml/tianshou

Table 5: Performance of trained offline models by CQL for robotics locomotion control tasks. Statistics are estimated from 20 evaluation episodes (same with Table 6). Note that these statistics cannot be obtained in the offline training phase.

		Environment return (†)	Human Disagreement (↓)	Online Score (†)
HalfCheetah-v2	Model 1 ($\lambda = 0$)	8731	559	0.47
	Model 2 ($\lambda = 1$)	8732	131	0.90
	Model 3 ($\lambda = 5$)	8292	59	0.92
	Model 4 ($\lambda = 10$)	8076	49	0.90
	Model 5 ($\lambda = 100$)	7328	104	0.76
	Model 1 ($\lambda = 0$)	39	22	0.00
Hommon v.O	Model 2 ($\lambda = 1$)	3450	266	0.72
Hopper-v2	Model 3 ($\lambda = 5$)	3387	193	0.77
	Model 4 ($\lambda = 10$)	3309	299	0.65
	Model 5 ($\lambda = 100$)	3312	303	0.64
Walker2d-v2	Model 1 ($\lambda = 0$)	3640	699	0.21
	Model 2 ($\lambda = 1$)	4106	328	0.70
	Model 3 ($\lambda = 5$)	3726	695	0.24
	Model 4 ($\lambda = 10$)	3767	302	0.64
	Model 5 ($\lambda = 100$)	3667	360	0.56

Table 6: Performance of trained offline models by CQL for the traffic light control task.

	Environment return (†)	Human Disagreement (↓)	Online Score (†)
Model 1 ($\lambda = 0$)	835	1641	0.01
Model 2 ($\lambda = 1$)	908	320	0.75
Model 3 ($\lambda = 5$)	906	457	0.68
Model 4 ($\lambda = 10$)	913	270	0.78
Model 5 ($\lambda = 100$)	918	262	0.79