EMPO: A Clustering-Based On-Policy Algorithm for Offline Reinforcement Learning

Jongeui Park
School of Electrical Engineering
Korea Advanced Institute of Science and Technology
Dajeon 34141
jongeui.park@kaist.ac.kr

Myungsik Cho
School of Electrical Engineering
Korea Advanced Institute of Science and Technology
Dajeon 34141
ms.cho@kaist.ac.kr

Youngchul Sung
School of Electrical Engineering
Korea Advanced Institute of Science and Technology
Dajeon 34141
ycsung@kaist.ac.kr

Abstract

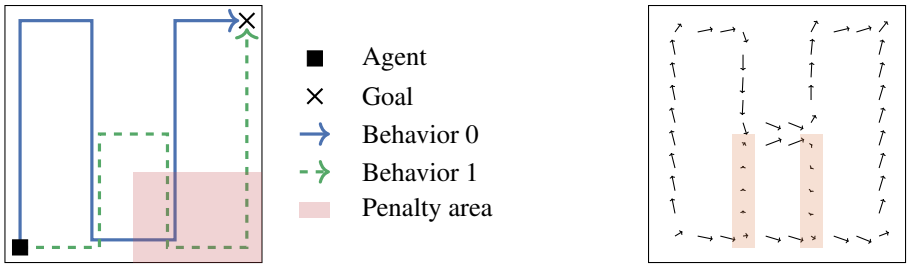
We propose an on-policy algorithm, Expectation–Maximization Policy Optimization (EMPO), for offline reinforcement learning that leverages an EM-based clustering algorithm to recover the behaviour policies used to generate the dataset. By improving each behaviour policy via proximal policy optimization and learning a high-level policy that chooses the optimal cluster at each step, EMPO outperforms existing offline RL algorithms on multiple benchmarks.

1 Introduction

The significant achievements of deep learning in various areas, notably image classification [1], have led to the integration of reinforcement learning (RL) with deep learning, a combination known as deep RL. This approach is seen as a promising method for addressing complex tasks such as mastering simple computer games using raw pixel data [2], achieving super-human performance in complex board games such as Go [3], and enhancing control in robotics [4–8]. In particular, using deep neural networks as function approximators enabled the representation of complex policies and state-action value functions across large state-action spaces.

However, most RL algorithms tend to have notoriously high sample complexity, which led to the birth of a new problem called offline RL. Unlike conventional RL, where an agent learns an optimal policy through trial and error within the environment, in offline RL, environmental interactions are

38th Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2024).



(a) A simple navigation environment.

(b) Behavior cloning policy behaviour

Figure 1: The agent receives a reward of -1 until it reaches the goal and an additional penalty of -1 when it is inside the penalty area. The offline dataset consists of trajectories sampled using two behaviour policies β_0 (Behavior 0) and β_1 (Behavior 1). From (b) we can see that behaviour cloning fails in the shaded region where the two behaviours conflict.

forbidden. Instead, a set $\mathcal{D}$ of trajectories is provided, and the agent has to search for a competent policy only using samples in $\mathcal{D}$.

One of the major obstacles for direct application of online off-policy RL algorithms on offline RL is the extrapolation error of the critic network [9]. As the agent cannot perform an over-estimated out-of-distribution (OOD) action and inspect its outcome, the extrapolation error will never be corrected. Moreover, these errors are propagated through the trajectory causing the errors to accumulate. Policy regularization is a popular approach to mitigate the issue [9–15], where the agent aims to search for a policy with a high estimated expected return in the vicinity of the behaviour policy. Since extrapolation error would be small for actions that are similar to those in the dataset, the agent may anticipate that the estimated expected return is fairly accurate.

Although simple divergence metrics, such as the mean squared error between policy output and actions in the dataset [11], perform surprisingly well in a lot of cases, offline RL algorithms tend to underperform when the dataset comprises multiple behaviours that conflict with each other [14, 16]. Consider a simple navigation environment shown in Figure 1a and an offline dataset that consists of trajectories sampled using either β_0 or β_1, where β_0 and β_1 behaves like "Behavior 0" and "Behavior 1", respectively. Simple behaviour cloning produces a policy depicted in Figure 1b. The policy fails to model the actions in the shaded region, where the two behaviours conflict with each other. As we will later discuss in Section 4.1, this prevents the offline RL algorithms from entering the region, causing them to take the sub-optimal route that passes the penalty area instead.

To overcome this challenge, this paper proposes a novel Expectation–Maximization (EM) based clustering algorithm that can successfully recover the behaviour policies used to create an offline RL dataset. A high-level policy that selects the best behaviour policy to perform at the given state is learned using the clustered dataset. Finally, each cluster is improved via PPO [6], which is known to be robust and stable.

2 Background

Notation For a set X, we denote the family of probability distributions supported on X by $\mathcal{P}(X)$. And for an integer N, we denote the set $\{0, 1, \ldots, N-1\}$ by $[N]$. We also define a clipping function $\mathrm{clip}(x\,; y, z) = \max\{y, \min\{x, z\}\}$. We will also the use the same notation for dimension-wise clipping, that is, for $x \in \mathbb{R}^d$ and $y, z \in \mathbb{R}$, the i-th coordinate of $\mathrm{clip}(x\,; y, z)$ is $\mathrm{clip}(x_i\,; y, z)$.

2.1 Problem Setting

An RL problem is usually formulated as a Markov Decision Process (MDP), which is defined as a 6-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0 \rangle$, where $\mathcal{S} \in \mathbb{R}^{d_s}$ is a state space, $\mathcal{A} \in \mathbb{R}^{d_a}$ is an action space, $P\colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is a transition probability function, $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\gamma \in [0, 1]$ is a discount factor, and $\rho_0 \in \mathcal{P}(\mathcal{S})$ is an initial state distribution. The objective is to find a policy $\pi\colon \mathcal{S} \to \mathcal{P}(\mathcal{A})$ that maximizes the expected discounted return $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$, where $s_0 \sim \rho_0$ and for each $t \geq 0$, $a_t \sim \pi(s_t)$ and $s_{t+1} \sim P(s_t, a_t)$.

For offline RL, interactions with the environment is prohibited, and the agent has to learn a policy from a given dataset $\mathcal{D}$ of trajectories. Throughout the paper we will assume that each trajectory $\tau \in \mathcal{D}$ is sampled with a behaviour policy $\beta \in \{\beta_0, \beta_1, \ldots, \beta_{K-1}\}$, where the candidate set $\{\beta_0, \beta_1, \ldots, \beta_{K-1}\}$ is fixed but unknown to the agent.

2.2 Expectation–Maximization (EM) Algorithm

Consider a probabilistic model with observable data $\mathbf{X}$ and a categorical latent variable Z. Denoting the model parameters by $\boldsymbol{\theta}$, the log-likelihood function can be written as

$$\log p(\mathbf{X}\,;\boldsymbol{\theta}) = \log\left(\sum_{Z} p(\mathbf{X}, Z\,;\boldsymbol{\theta})\right).$$

In most cases, directly maximizing the right-hand side is difficult due to the summation inside the logarithm. An Expectation–Maximization algorithm is a popular method that is used in such cases [17]. The algorithm comprises two steps: the E-step and the M-step. In the E-step, we compute the posterior probability

$$\mathcal{Q}_{\hat{\boldsymbol{\theta}}_k}(Z \mid \mathbf{X}) = p(Z \mid \mathbf{X}\,;\hat{\boldsymbol{\theta}}_k) = \frac{p(\mathbf{X}, Z\,;\hat{\boldsymbol{\theta}}_k)}{\sum_{Z'} p(\mathbf{X}, Z'\,;\hat{\boldsymbol{\theta}}_k)} \tag{1}$$

with respect to the current estimate $\hat{\boldsymbol{\theta}}_k$ of $\boldsymbol{\theta}$ after k iterations. In the M-step, we find $\hat{\boldsymbol{\theta}}_{k+1}$ that maximizes the conditional expectation

$$\hat{\boldsymbol{\theta}}_{k+1} = \arg\max_{\boldsymbol{\theta}} \sum_{Z} \mathcal{Q}_{\hat{\boldsymbol{\theta}}_k}(Z \mid \mathbf{X}) \log p(\mathbf{X}, Z\,;\boldsymbol{\theta}) \tag{2}$$

We iterate between the two steps until $\hat{\boldsymbol{\theta}}_k$ converges.

3 Proposed Method

The proposed method Expectation–Maximization Policy Optimization (EMPO) comprises of three parts. In the first part, we apply an EM-based clustering algorithm to recover the behaviour policies $\beta_0, \beta_1, \ldots, \beta_{K-1}$ that were used to generate the dataset $\mathcal{D}$. The algorithm also tells us the β_z that was used to generate each trajectory $\tau \in \mathcal{D}$. Based on this information, we improve each β_z by PPO [6] to obtain π_z. Finally, we learn a high-level policy that selects the best π_z given the current state. We will discuss the three parts in the following three subsections 3.1, 3.2, and 3.3.

3.1 Policy Clustering via Expectation–Maximization

We have assumed in Section 2.1 that each trajectory in $\mathcal{D}$ is sampled with one of the K behaviour policies $\beta_0, \beta_1, \ldots, \beta_{K-1}$. Suppose each behaviour policy β_i can be parametrized using the same neural network but with different parameters, that is, there are parameters $\theta_0, \theta_1, \ldots, \theta_{K-1}$ such that $\beta_i(a \mid s) = \beta(a \mid s\,;\theta_i)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $i \in [K]$. Let $\Theta = \{\theta_0, \theta_1, \ldots, \theta_{K-1}\}$. Slightly abusing the notation, we may write $\beta(a \mid s\,;\theta_z)$ as $\beta(a \mid s, Z = z\,;\Theta)$, where Z is a categorical latent variable supported on $[K]$. This leads us to the probabilistic model in Figure 2. The joint probability distribution can be written as

$$p(\tau, Z\,;\Theta) = \rho_0(s_0)p(Z)\left[\prod_{t=0}^{T-1}\left(\beta(a_t \mid s_t, Z\,;\Theta)P(a_{t+1} \mid s_t, a_t)\right)\right]\beta(a_T \mid s_T, Z\,;\Theta),$$

where we have denoted $(s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$ by τ. Unless additional information is available, we set the prior $p(Z)$ to be a uniform distribution.

Our goal is to find the optimal parameter Θ, which will allow us to compute the posterior probability

$$p(Z \mid \tau\,;\Theta) \propto \prod_{t=0}^{T} \beta(a_t \mid s_t, Z\,;\Theta).$$

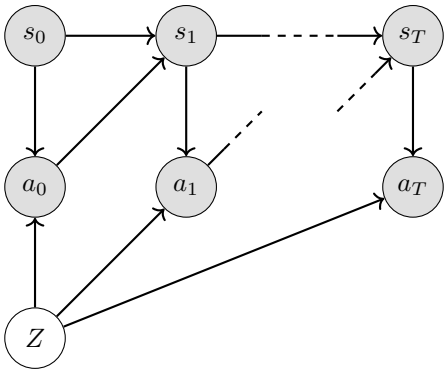


Figure 2: A graphical model of an offline RL dataset.

This can be done by applying an EM algorithm. The E-step equation (1) can be rewritten as

$$\mathcal{Q}_{\hat{\Theta}_k}(Z \mid \tau) = p(Z \mid \tau\,; \hat{\Theta}_k) = \frac{\prod_{t=0}^{T} \beta(a_t \mid s_t, Z; \hat{\Theta}_k)}{\sum_{z=0}^{K-1} \prod_{t=0}^{T} \beta(a_t \mid s_t, z; \hat{\Theta}_k)}. \tag{3}$$

Similarly, the M-step equation (2) can be rewritten as

$$\hat{\Theta}_{k+1} = \arg\max_{\Theta} \sum_{\tau \in \mathcal{D}} \sum_{z=0}^{K-1} \mathcal{Q}_{\hat{\Theta}_k}(z \mid \tau) \log \beta(\tau, z\,; \Theta).$$

As the exact optimization of neural network parameters is intractable, we perform gradient descent with respect to the following loss function:

$$\ell_M^{(k)}(s, a\,; \Theta) = -\sum_{z=0}^{K-1} \mathcal{Q}_{\hat{\Theta}_k}(z \mid \tau(s,a)) \log \beta(a \mid s\,; \theta_z), \tag{4}$$

where $\tau(s,a)$ is the trajectory containing the state-action pair (s,a).

We adopt three heuristics to improve the clustering performance further. The first is to use the exponential moving average of the conditional probability $\mathcal{Q}_{\hat{\Theta}_k}$ to stabilize the M-step. Inspired by on-line EM [18], we introduce a momentum parameter μ and define

$$\mathcal{Q}^{(k+1)}(Z \mid \tau) = \mu \mathcal{Q}^{(k)}(Z \mid \tau) + (1-\mu)\mathcal{Q}_{\hat{\Theta}_k}(Z \mid \tau). \tag{5}$$

and $\mathcal{Q}^{(0)}(Z \mid \tau) = 1/K$ for all $Z \in [K]$ and $\tau \in \mathcal{D}$. The M-step loss (4) becomes

$$\tilde{\ell}_M^{(k)}(s, a\,; \Theta) = -\sum_{z=0}^{K-1} \mathcal{Q}^{(k)}(z \mid \tau(s,a)) \log \beta(a \mid s\,; \theta_z). \tag{6}$$

We train Θ for N_E epochs, where at the start of epoch k, we compute $\mathcal{Q}^{(k)}$ for all $\tau \in \mathcal{D}$, and then apply gradient descent with respect to the loss function $\tilde{\ell}_M^{(k)}$. Secondly, we pretrain each $\beta(a \mid s\,; \theta_z)$ using the entire dataset. This is equivalent to setting

$$\mathcal{Q}^{(0)}(z \mid \tau) = \frac{1}{K} \tag{7}$$

for each τ and repeating the M-step for N_p epochs without performing the E-step. Introducing the two heuristics results in a reasonable clustering performance, but in some rare cases where the initial $\hat{\Theta}_0$ is biased towards a particular θ_i, that is, $\mathcal{Q}_{\hat{\Theta}_0}(i \mid \tau) > \mathcal{Q}_{\hat{\Theta}_0}(z \mid \tau)$ for all $z \neq i$, every trajectory is clustered into a single group. To forestall this skewness, we apply another heuristic, which is to divide the trajectories into K groups randomly and train each policy β_i to be biased towards the i-th group. This is equivalent to randomly sample $z_\tau \in [K]$ for each τ, set

$$\mathcal{Q}^{(-1)}(z \mid \tau) = \begin{cases} 1 & \text{if } z = z_\tau \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

Algorithm 1 EM-based Clustering

Initialize $\hat{\Theta}_0$
Compute $\mathcal{Q}^{(-1)}$ for all τ according to (8) ▷ Heuristic III
for N_b epochs **do**
    Sample a mini-batch $\mathcal{B} = \{(s_0, a_0), (s_1, a_1), \ldots, (s_{B-1}, a_{B-1})\}$
    $\hat{\Theta}_0 \leftarrow \hat{\Theta}_0 - \frac{1}{B}\sum_{(s,a)\in\mathcal{B}} \alpha \nabla_\Theta \hat{\ell}_M^{(-1)}(s, a; \hat{\Theta}_0)$ ▷ (6)
end for
Compute $\mathcal{Q}^{(0)}$ according to (7) ▷ Heuristic II
for N_p epochs **do**
    Sample a mini-batch $\mathcal{B} = \{(s_0, a_0), (s_1, a_1), \ldots, (s_{B-1}, a_{B-1})\}$
    $\hat{\Theta}_0 \leftarrow \hat{\Theta}_0 - \frac{1}{B}\sum_{(s,a)\in\mathcal{B}} \alpha \nabla_\Theta \hat{\ell}_M^{(0)}(s, a; \hat{\Theta}_0)$ ▷ (6)
end for
for $k \leftarrow 1, 2, \ldots, N_E$ **do**
    Compute $\mathcal{Q}^{(k)}$ according to (5) ▷ Heuristic I
    $\hat{\Theta}_k \leftarrow \hat{\Theta}_{k-1}$
    for 1 epoch **do**
        Sample a mini-batch $\mathcal{B} = \{(s_0, a_0), (s_1, a_1), \ldots, (s_{B-1}, a_{B-1})\}$
        $\hat{\Theta}_k \leftarrow \hat{\Theta}_k - \frac{1}{B}\sum_{(s,a)\in\mathcal{B}} \alpha \nabla_\Theta \hat{\ell}_M^{(k)}(s, a; \hat{\Theta}_k)$ ▷ (6)
    end for
end for
$\hat{\Theta}^* \leftarrow \hat{\Theta}_{N_E}$
Compute $\mathcal{Q}_{\hat{\Theta}^*}$ according to (3)
For each $(s, a) \in \mathcal{D}$, compute $\mathbb{A}(s, a)$ according to (9)

and applying the M-step for N_b epochs while skipping the E-step.

After we finish training, we find each state-action pair (s, a)'s assignment through computing the arg max of $\mathcal{Q}_{\hat{\Theta}^*}(z \mid \tau(s, a))$, where $\hat{\Theta}^*$ is the final parameter. We will denote the assignment by $\mathbb{A}(s, a)$, that is,

$$\mathbb{A}(s, a) = \arg\max_{z\in[K]} \mathcal{Q}_{\hat{\Theta}^*}(z \mid \tau(s, a)). \tag{9}$$

The EM-based clustering method is summarized in Algorithm 1.

3.2 Improving Each Policy Cluster

Given that the proposed EM algorithm has clustered the dataset accurately, we are left with a bunch of smaller datasets whose trajectories are sampled using a single policy. This allows us to apply on-policy RL algorithms such as TRPO [4] or PPO [6]. We selected PPO as it is well-known to be robust and stable. PPO utilizes the generalized advantage estimator (GAE) [19] to compute the advantage function in a Monte Carlo manner, but with additional variance reduction techniques. One downside of using a GAE in the offline setting is that we can only compute the advantage estimates for actions in the dataset. When the learned behaviour policy is erroneous, the samples in the dataset may have low likelihoods and thus be clipped away during the training process. To mitigate this issue, sampling from the learned behaviour policy is necessary. Therefore, following Zhuang et al. [15], we train an action-value function network through the SARSA [20] algorithm and a value function network through regression with the sample return.

The detailed algorithm is as follows. We define an action-value network $Q(\cdot, \cdot\,; \psi)\colon \mathbb{R}^{d_s} \times \mathbb{R}^{d_a} \to \mathbb{R}^K$ and a value network $V(\cdot\,; \varphi)\colon \mathbb{R}^{d_s} \to \mathbb{R}^K$. The i-th coordinate of $Q(s, a\,; \psi)$ and $V(s\,; \psi)$ corresponds to the $Q^{\beta_i}(s, a)$ and $V^{\beta_i}(s)$, respectively. For every trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$, in $\mathcal{D}$, we compute the sample return of each state

$$R(s_i) = \sum_{t=0}^{T-i} \gamma^t r_t. \tag{10}$$

Algorithm 2 Policy Improvement

Initialize ψ, φ, and Θ'
$\bar{\psi} \leftarrow \psi$ and $\Theta' \leftarrow \hat{\Theta}^*$
for $k \leftarrow 1, 2, \ldots, N_Q$ **do**
    Sample a mini-batch $\mathcal{B} = \{(s_i, a_i, r_i, s'_i, a'_i)\}_{i=0}^{B-1}$
    $\psi \leftarrow \psi - \frac{1}{B} \sum_{(s,a,r,s',a') \in \mathcal{B}} \alpha \nabla_\psi \ell_Q(s, a, r, s', a'; \psi)$ ▷ (11)
    if $k \equiv 0 \pmod{n_Q}$ **then**
        Update $\bar{\psi}$ according to (13)
    end if
end for
Compute $R(s)$ for all $s \in \mathcal{D}$ according to (10)
for N_V epochs **do**
    Sample a mini-batch $\mathcal{B} = \{(s_0, a_0), (s_1, a_1), \ldots, (s_{B-1}, a_{B-1})\}$
    $\varphi \leftarrow \varphi - \frac{1}{B} \sum_{(s,a) \in \mathcal{B}} \alpha \nabla_\varphi \ell_V(s, a; \varphi)$ ▷ (12)
end for
for N_π epochs **do**
    $\Theta' \leftarrow \Theta' - \frac{1}{|\mathcal{D}|} \sum_{(s,a) \in \mathcal{D}} \alpha \nabla_{\Theta'} \ell_\pi(s, a; \Theta')$ ▷ (14)
end for

Based on this return we can define the following loss functions:

$$\ell_Q(s, a, r, s', a' ; \psi) = \left[Q(s, a ; \psi)^\intercal e_{\mathbb{A}(s,a)} - \left(r + \gamma Q(s', a' ; \bar{\psi})^\intercal e_{\mathbb{A}(s',a')}\right)\right]^2 \tag{11}$$

$$\ell_V(s, a ; \varphi) = \left[V(s ; \varphi)^\intercal e_{\mathbb{A}(s,a)} - R(s)\right]^2, \tag{12}$$

where $\bar{\psi}$ is the target network parameter and $e_i \in \mathbb{R}^K$ is the i-th standard basis vector. The target network parameter $\bar{\psi}$ is updated every n_Q time-steps according to the following equation

$$\bar{\psi} \leftarrow (1 - \tau_Q)\bar{\psi} + \tau_Q \psi, \tag{13}$$

where $0 < \tau_Q \leq 1$ is a hyperparameter. We train the action-value network for N_Q steps and the value network for N_V epochs. For the policy network $\pi_L(\cdot \mid s, z ; \Theta')$, we adopt the same architecture we used for clustering in Section 3.1. The PPO loss can be written as

$$\ell_\pi(s, a ; \Theta') = \min\left\{\rho(s, a')A(s, a'), \operatorname{clip}\left(\rho(s, a')1 - \varepsilon_c, 1 + \varepsilon_c\right) A(s, a')\right\}, \tag{14}$$

where $a' \in \mathcal{A}$ is an action sampled from $\beta(\cdot \mid s, \mathbb{A}(s, a) ; \hat{\Theta}^*)$, $0 < \epsilon_c < 1$ is a clipping hyperparameter, and

$$\rho(s, a') = \frac{\pi_L(a' \mid s, \mathbb{A}(s, a) ; \Theta')}{\beta(a' \mid s, \mathbb{A}(s, a) ; \hat{\Theta}^*)}, \qquad A(s, a') = \left(Q(s, a' ; \psi) - V(s ; \varphi)\right)^\intercal e_{\mathbb{A}(s,a)}.$$

We train the policy network for N_π epochs. PPO tends to perform well with extremely large mini-batch sizes [21]. We observed that instead of the conventional mini-batch gradient descent, batch gradient descent greatly improves the training process in terms of performance and stability. Refer to Section 4.3.2 for the corresponding results. The overall policy improvement method is summarized in Algorithm 2.

3.3 Training a High-Level Policy

Policy clustering provides us with K different policies, but we can only use one at test time. Which one should we use? To choose the optimal β_z to execute at each step, we train a high-level Q network $Q_H(s, z ; \psi_H)$ and a V network $V_H(s, z ; \varphi_H)$ through implicit Q-learning [13]. The loss function for each network is

$$\ell_H^{(Q)}(s, a, r, s' ; \psi_H) = \left[Q_H(s, \mathbb{A}(s, a) ; \psi_H) - (r + \gamma V_H(s' ; \varphi_H))\right]^2$$

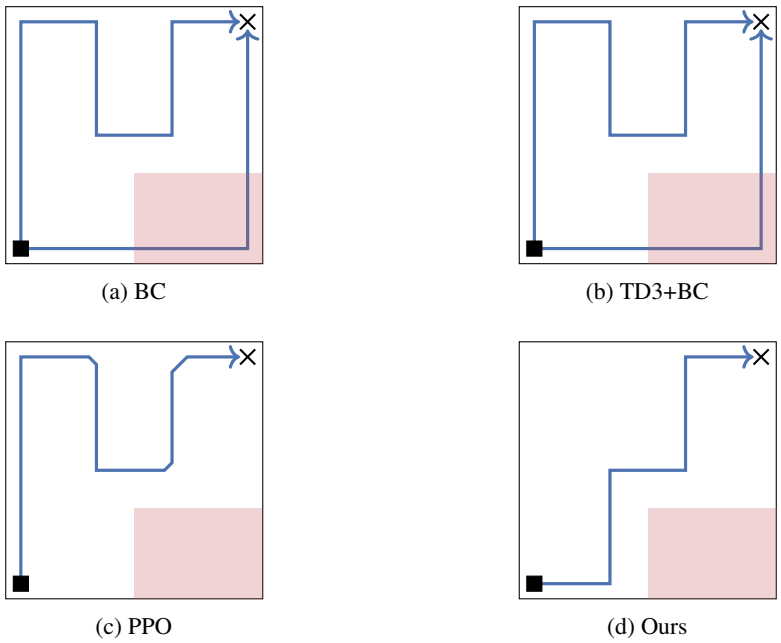
$$\ell_H^{(V)}(s, a ; \varphi_H) = L_2^\tau(Q_H(s, \mathbb{A}(s, a) ; \bar{\psi}_H) - V_H(s ; \varphi_H)),$$

where $\bar{\psi}_H$ is the target network parameter that is updated every n_H steps, τ is the expectile hyperparameter, and

$$L_2^\tau(x) = \begin{cases} \tau x^2 & \text{if } x \geq 0 \\ (1 - \tau)x^2 & \text{otherwise.} \end{cases}$$

Table 1: Comparisons of the test returns for the simple navigation task.

BC	TD3+BC	PPO	Ours
-854.14 ± 214.85	-739.94 ± 214.60	-606.16 ± 329.08	$\mathbf{-345.94 \pm 171.46}$



(a) BC (b) TD3+BC

(c) PPO (d) Ours

Figure 3: Simplified trajectories of the final policies learned by each algorithm.

ψ_H and φ_H are updated simultaneously for N_H steps. We also adopted the n-step return [22] technique to further stabilize the training process.

$\arg\max_z Q_H(s, z\,;\psi_H)$ may result in an erroneous high-level action due to extrapolation errors of out-of-distribution clusters. Hence, while we learn the behaviour policies, we also train a classifier $q(z \mid s\,;\phi)$ to determine the in-distribution clusters given a state s. The classifier is trained using the KL divergence loss between the predicted distribution and $\mathcal{Q}^{(k)}(z \mid \tau(s,a))$, that is,

$$\ell_C^{(k)}(s, a\,;\phi) = \mathrm{KL}(\mathcal{Q}^{(k)}(\cdot \mid \tau(s,a)) \parallel q(\cdot \mid s\,;\phi)).$$

Unlike continuous action spaces, defining in-distribution actions for discrete action spaces is unambiguous; an action is in-distribution if its probability mass function is positive. In practice, an output of the softmax function can never be zero, so we define the set $\mathrm{ID}(s)$ of in-distribution clusters for a given state s as

$$\mathrm{ID}(s) = \left\{ z : q(z \mid s\,;\phi) \geq \frac{1}{b} \max_{z'} q(z' \mid s\,;\phi) \right\},$$

where $b > 0$ is a hyperparameter. The high-level policy π_H would then be

$$\pi_H(s\,;\psi_H) = \operatorname*{arg\,max}_{z \in \mathrm{ID}(s)} Q_H(s, z\,;\psi_H).$$

The resulting policy is $\pi(\cdot \mid s) = \pi_L(\cdot \mid s, \pi_H(s\,;\psi_H)\,;\Theta')$.

4 Experiments

4.1 Didactic Experiments

To show the importance of policy clustering, we first evaluated our algorithm on a simple navigation environment shown in Figure 1. The agent should output a two-dimensional velocity vector as an

Table 2: Averaged normalized score on D4RL tasks.

Dataset	BC	DT	AWAC	Onestep RL	TD3+BC	CQL	IQL	Ours
halfcheetah-m-e*	55.2	86.8	42.8	93.4	90.7	91.6	86.7	92.6
hopper-m-e	52.5	107.6	55.8	103.3	98.0	105.4	91.5	110.2
walker2d-m-e	107.5	108.1	74.5	113.0	110.1	108.8	109.6	108.4

* m-e stands for medium-expert. All of the experiments were conducted on the v2 dataset.

Table 3: Comparison of the rand index [28].

Dataset	No Heuristics	Heuristic I	Heuristic I, II	**Ours**
halfcheetah-m-e*	59.98	**100.00**	**100.00**	**100.00**
hopper-m-e	69.54	**100.00**	**100.00**	**100.00**
walker2d-m-e	77.07	98.42	99.89	**99.93**

* m-e stands for medium-expert. All of the experiments were conducted on the v2 dataset.

action based on the two-dimensional position vector given as state. The episode terminates when the agent reaches the goal, and until then, it receives a reward of -1. When the agent is inside the penalty area, it receives an additional penalty of -1, that is, the reward is -2 every time-step. We created an offline dataset that consists of trajectories sampled using either β_0 or β_1, where β_0 and β_1 behaves like "Behavior 0" and "Behavior 1" in Figure 1a, respectively.

We trained four algorithms on this dataset: naive behaviour cloning, TD3+BC [11], our algorithm with $K = 1$, which is equivalent to PPO, and our algorithm with $K = 2$. Table 1 compares the performance and Figure 3 depicts the behaviour of each algorithm. Our algorithm is the only algorithm that successfully found the optimal trajectory. Other algorithms failed to enter the shaded region in Figure 1b, where behaviour cloning fails, and had to take a detour.

4.2 Comparisons on Offline RL Benchmarks

Table 2 reports the performance of of our algorithm evaluated on three different datasets in the D4RL benchmark [23]: halfcheetah-medium-expert-v2, hopper-medium-expert-v2, and walker2d-medium-expert-v2. These are the three datasets that satisfy the assumption that each trajectory is sampled using a single behaviour policy. Following Kostrikov et al. [13], we compared it to the performance of following baselines: DT [24], AWAC [25], Onestep RL [26], TD3+BC [11], CQL [27], and IQL [13]. The baseline results were extracted from Kostrikov et al. [13].

4.3 Ablation Studies

4.3.1 Clustering Algorithm

Recall that our clustering algorithm adopted three heuristics to improve the final performance. To investigate the effect of each heuristic, we conducted an ablation study by comparing the rand index [28] of four algorithms: plain EM (No Heuristics), plain EM with the momentum heuristic(Heuristic I), plain EM with the momentum heuristic and pretraining(Heuristic I, II), and ours. Table 3 reports the mean and standard deviation over 10 seeds on the three datasets of the D4RL benchmark: halfcheetah-medium-expert-v2, hopper-medium-expert-v2, and walker2d-medium-expert-v2.[1] We can see that the momentum heuristic has the largest impact on the clustering performance. Refer to Table 5 in Appendix C for the standard deviations and additional ablation study results.

[1] medium-expert datasets of the D4RL benchmark were created by concatenating the corresponding medium and expert datasets, so we have access to the ground truth labels.

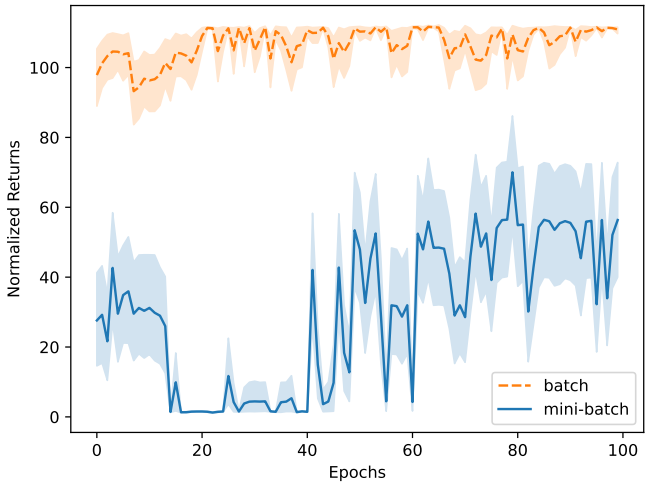


Figure 4: Comparison between min-batch gradient descent and batch gradient descent

4.3.2 Policy Improvement Algorithm

We also analyzed the effect of batch gradient descent on our algorithm using the `hopper-medium-expert-v2` dataset. Figure 4 shows how the performance changes as we increase N_π, the number of policy training epochs. We can see that batch gradient descent not only outperforms the mini-batch gradient descent in terms of normalized return but also is more robust with respect to the choice of N_π. We believe that this phenomena is caused by clipping in the policy loss function (14). When training with mini-batches, the number of unclipped samples inside a min-batch varies from one mini-batch to another. In some mini-batches, the number of unclipped samples would be very low resulting in noisy gradients. By computing the gradient using the entire dataset, we can prevent this from happening.

5 Related Work

There are multiple prior work concerned with offline RL datasets with heterogeneous behaviours. Wang et al. [14] utilizes a diffusion model [29, 30] to capture the multi-modality of the true behaviour policy. They update the policy using a loss function similar to TD3+BC [11], where they replaced the L_2 regularizer with the diffusion loss. Li et al. [31] trains a mixture of Gaussian policy on the dataset via likelihood maximization and then obtains a closed-form estimate of the best possible action near the behaviour policy. Unlike EMPO, these two work do not explicitly cluster the trajectories.

Mao et al. [32] incorporates an EM based algorithm to learn diverse policies from a given offline RL dataset. Wang et al. [16] proposes a sophisticated trajectory clustering algorithm that can also automatically determine the cluster size. In contrast to these two works, we also discuss how to find an optimal policy through carefully stitching the learned behaviour policies.

6 Conclusion

In this work we introduced the Expectation–Maximization Policy Optimization (EMPO) algorithm that can recover the behaviour policies used to create the offline RL dataset and can interleave them into a one competent policy. Future work could explore cases where trajectories are sampled from continuously changing behaviour policies. Additionally, we could incorporate model-based planing on the cluster space that could enable long-term planning.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 2012, pp. 1106–1114.

[2] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[3] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nat.* 529.7587 (2016), pp. 484–489.

[4] John Schulman et al. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 1889–1897.

[5] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016.

[6] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017).

[7] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1582–1591.

[8] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1856–1865.

[9] Scott Fujimoto, David Meger, and Doina Precup. "Off-Policy Deep Reinforcement Learning without Exploration". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2052–2062.

[10] Aviral Kumar et al. "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019, pp. 11761–11771.

[11] Scott Fujimoto and Shixiang Shane Gu. "A Minimalist Approach to Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. 2021, pp. 20132–20145.

[12] Yifan Wu, George Tucker, and Ofir Nachum. "Behavior Regularized Offline Reinforcement Learning". In: *CoRR* abs/1911.11361 (2019).

[13] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. "Offline Reinforcement Learning with Implicit Q-Learning". In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[14] Zhendong Wang, Jonathan J. Hunt, and Mingyuan Zhou. "Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[15] Zifeng Zhuang et al. "Behavior Proximal Policy Optimization". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[16] Qiang Wang et al. "Dataset Clustering for Improved Offline Policy Learning". In: *CoRR* abs/2402.09550 (2024).

[17] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, Jan. 2006.

[18] Olivier Cappé and Eric Moulines. "On-line expectation–maximization algorithm for latent data models". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71.3 (2009), pp. 593–613.

[19] John Schulman et al. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016.

[20] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[21] Viktor Makoviychuk et al. “Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*. 2021.

[22] Richard S. Sutton. “Learning to Predict by the Methods of Temporal Differences”. In: *Mach. Learn.* 3 (1988), pp. 9–44.

[23] Justin Fu et al. “D4RL: Datasets for Deep Data-Driven Reinforcement Learning”. In: *CoRR* abs/2004.07219 (2020).

[24] Lili Chen et al. “Decision Transformer: Reinforcement Learning via Sequence Modeling”. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. 2021, pp. 15084–15097.

[25] Ashvin Nair et al. “Accelerating Online Reinforcement Learning with Offline Datasets”. In: *CoRR* abs/2006.09359 (2020).

[26] David Brandfonbrener et al. “Offline RL Without Off-Policy Evaluation”. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. 2021, pp. 4933–4946.

[27] Aviral Kumar et al. “Conservative Q-Learning for Offline Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 2020.

[28] William M Rand. “Objective criteria for the evaluation of clustering methods”. In: *Journal of the American Statistical association* 66.336 (1971), pp. 846–850.

[29] Jascha Sohl-Dickstein et al. “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 2256–2265.

[30] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising Diffusion Probabilistic Models”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 2020.

[31] Jiachen Li et al. “Offline Reinforcement Learning with Closed-Form Policy Improvement Operators”. In: *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 20485–20528.

[32] Yihuan Mao et al. “Stylized Offline Reinforcement Learning: Extracting Diverse High-Quality Behaviors from Heterogeneous Datasets”. In: *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. 2024.

[33] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.4.29. 2018.

[34] Jonathan Heek et al. *Flax: A neural network library and ecosystem for JAX*. Version 0.8.4. 2023.

[35] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *J. Mach. Learn. Res.* 12 (2011), pp. 2825–2830.

A Navigation Environment Details

The observation space is $\mathcal{S} = [0, 30] \times [0, 30]$, the action space is $\mathcal{A} = [-0.2, 0.2] \times [-0.2, 0.2]$, and the penalty area is $\mathcal{S}_p = [15, 30] \times [0, 10]$. The starting location of the agent is sampled uniformly at random from $[0, 0.1] \times [0, 0.1]$ and the goal position is fixed to $g = (30, 30)$. If the current state is $s = (s_0, s_1) \in \mathcal{S}$ and the action is $a = (a_0, a_1) \in \mathcal{A}$, the next state $s' = (s'_0, s'_1) \in \mathcal{S}$ is

$$s' = \text{clip}(s + a\,; 0, 30).$$

The reward function is

$$r(s, a, s') = \begin{cases} 0 & \text{if } \|s' - g\|_2 < 0.1, \\ -2 & \text{if } s' \in \mathcal{S}_p, \\ -1 & \text{otherwise.} \end{cases}$$

The episode terminates if either the agent has approached the goal ($\|s' - g\|_2 < 0.1$) or the number of time-steps exceeded 1000.

The offline RL datasets was generated using subgoal-reaching policies. A subgoal-reaching policy $\pi(s\,; g_s)$ for a subgoal g_s is defined as

$$\pi(s\,; g_s) = \text{clip}(\text{clip}(g_s - s\,; -0.2, 0.2) + 0.1\varepsilon\,; -0.2, 0.2), \tag{15}$$

were ε is a two-dimensional standard Gaussian noise. We generated the samples according to Algorithm 3 using two different list of subgoals: $[(0, 30), (10, 30), (10, 0), (20, 0), (20, 30)]$ and $[(10, 0), (10, 15), (20, 15), (20, 0), (30, 0)]$.

Algorithm 3 Dataset generation from a list of subgoals

Input: a list of subgoals $[g_s^{(1)}, g_s^{(2)}, \ldots, g_s^{(N)}]$
Initialize an empty dataset $\mathcal{D}$
while $\mathcal{D}$ has less than 1 000 000 elements **do**
    $s \leftarrow$ `env.reset()`
    for $g_s \leftarrow [g_s^{(1)}, g_s^{(2)}, \ldots, g_s^{(N)}]$ **do**
        while $\|s - g_s\|_2 \geq 0.1$ **do**
            $a \leftarrow \pi(s\,; g_s)$ ▷ (15)
            $s', r, d \leftarrow$ `env.step`(a)
            Add (s, a, r, d) to $\mathcal{D}$
            $s \leftarrow s'$
        end while
    end for
    while $\|s - g\|_2 \geq 0.1$ **do**
        $a \leftarrow \pi(s\,; g)$ ▷ (15)
        $s', r, d \leftarrow$ `env.step`(a)
        Add (s, a, r, d) to $\mathcal{D}$
        $s \leftarrow s'$
    end while
end while

B Experiment Details

For the didactic experiments we used a 4-layer multilayer perceptron (MLP) with hidden layers of size 64. For the D4RL experiments we used a 4-layer MLP with hidden layers of size 256. Table 4 shows the rest of the hyperparameters used in the experiments. The algorithm was implemented upon the JAX [33] framework using the Flax [34] library.

C Additional Experimental Results

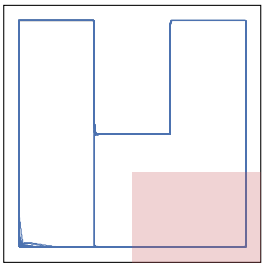
Figure 5 shows the actual trajectories of the final polices learned by each algorithms. We trained each algorithm for 10 different seeds and sampled 10 trajectories from each policy. Although our

algorithm sometimes took the suboptimal route as in Figure 5d, it occurred very rarely; only twice out of the 100 roll-outs.

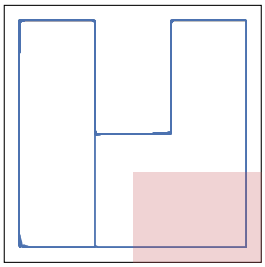
Table 5 shows additional ablation study results. The rand index [28] was calculated using the `scikit-learn` [35] library and the mean and standard deviation was calculated over 10 seeds.

Table 4: Hyperparameters used in experiments

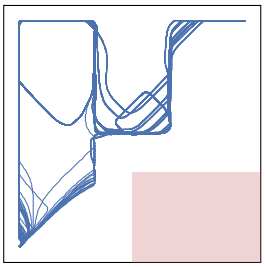
	Didactic	D4RL
γ	0.999	0.99
N_b	3	3
N_p	20	20
N_E	60	100
μ	0.9	0.9
batch size	256	256
learning rate	3×10^{-4}	3×10^{-4}
K	2	2
N_Q	200 000	1 500 000
n_Q	2	2
τ_Q	0.005	0.005
N_V	10	40
N_π	10	100
n_H	500	500
N_H	1 000 000	1 000 000
τ	0.9	0.7
b	5	5
n-step	3	3



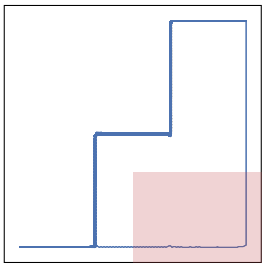
(a) BC



(b) TD3+BC



(c) PPO



(d) Ours

Figure 5: Actual trajectories of the final policies learned by each algorithm.

Table 5: Detailed ablation study results

	$N_p = 0$		$N_p = 20$	
	$N_b = 0$	$N_b = 3$	$N_b = 0$	$N_b = 3$
`halfcheetah-medium-expert-v2`				
$\mu = 0$	59.98 ± 21.09	64.95 ± 24.12	99.98 ± 0.04	99.98 ± 0.04
$\mu = 0.9$	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
`hopper-medium-expert-v2`				
$\mu = 0$	69.54 ± 21.02	95.30 ± 14.85	100.00 ± 0.00	100.00 ± 0.00
$\mu = 0.9$	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
`walker2d-medium-expert-v2`				
$\mu = 0$	77.07 ± 23.49	91.85 ± 15.25	96.58 ± 4.50	95.16 ± 4.87
$\mu = 0.9$	98.42 ± 2.35	98.71 ± 2.52	99.89 ± 0.08	99.93 ± 0.08