https://doi.org/10.26599/AIR.2023.9150020 CAAI Artificial Intelligence Research

OSCAR: OOD State-Conservative Offline Reinforcement Learning for Sequential Decision Making

Yi Ma[1], Chao Wang[2], Chen Chen[3], Jinyi Liu[1], Zhaopeng Meng[1], Yan Zheng[1], and Jianye Hao[1,4] ✉

ABSTRACT

Offline reinforcement learning (RL) is a data-driven learning paradigm for sequential decision making. Mitigating the overestimation of values originating from out-of-distribution (OOD) states induced by the distribution shift between the learning policy and the previously-collected offline dataset lies at the core of offline RL. To tackle this problem, some methods underestimate the values of states given by learned dynamics models or state-action pairs with actions sampled from policies different from the behavior policy. However, since these generated states or state-action pairs are not guaranteed to be OOD, staying conservative on them may adversely affect the in-distribution ones. In this paper, we propose an OOD state-conservative offline RL method (OSCAR), which aims to address the limitation by explicitly generating reliable OOD states that are located near the manifold of the offline dataset, and then design a conservative policy evaluation approach that combines the vanilla Bellman error with a regularization term that only underestimates the values of these generated OOD states. In this way, we can prevent the value errors of OOD states from propagating to in-distribution states through value bootstrapping and policy improvement. We also theoretically prove that the proposed conservative policy evaluation approach guarantees to underestimate the values of OOD states. OSCAR along with several strong baselines is evaluated on the offline decision-making benchmarks D4RL and autonomous driving benchmark SMARTS. Experimental results show that OSCAR outperforms the baselines on a large portion of the benchmarks and attains the highest average return, substantially outperforming existing offline RL methods.

KEYWORDS

offline reinforcement learning; out-of-distribution; decision making

In recent years, deep reinforcement learning (RL) has achieved considerable success in various decision-making domains such as robotics[1], recommendation and advertising systems[2], and video games[3]. However, the trial-and-error learning paradigm of RL can be prohibitive when scaling it to real-world applications such as autonomous driving[4] and healthcare[5], as exploratory actions may cause critical damage to the agent or the environment[6]. Offline RL, also known as batch RL, aims to address this issue by learning policies using only previously collected data by an unknown behavior policy without interacting with the environment during learning, which is a promising direction towards applying RL to safety-critical applications[6].

Since the state space is generally poorly covered by the offline dataset consisting of limited data samples, vanilla RL algorithms may suffer from extrapolation error that Q-values of some state-action pairs may be severely overestimated. This overestimation is aroused by bootstrapping inaccurate Q-values of out-of-distribution (OOD) state-action pairs and policy improvement[7] based on the estimated Q-value function. To this end, most offline RL methods impose various constraints or penalty terms on the loss function of vanilla RL algorithms to encourage the learning policy to be pessimistic about visiting OOD states. For example, some prior works regularize the learning policy to be close to the behavior policy via minimizing the distance between them, which can be measured by Kullback–Leibler (KL) divergence, maximum mean discrepancy (MMD), or other distance metrics[7–10]. CQL[11] instead learns a conservative Q-function by underestimating the Q-values of OOD state-action pairs to reduce the learning policy's willingness of taking actions that may lead to OOD states.

The aforementioned offline RL methods yield policies that are capable of mitigating distribution shifts in the online evaluation stage, i.e., seldom visiting OOD states when interacting with the environment. In this regard, a more straightforward way is to directly underestimate the values of OOD states instead of the state-action values of OOD actions. To this end, COMBO[12] proposes to conservatively estimate the value function by penalizing the value of states generated through rollouting a learned dynamics model. PessORL[13] proposes to detect which states sampled from the offline dataset are OOD by estimating the uncertainty of states using the ensemble dynamics models and further penalize the values of these states on top of CQL practice. However, the above two works achieve state-conservativeness in an implicit way as not all the generated or detected states can be considered as OOD states. For example, depending on the model reliability and the rollout length, the dynamics model in COMBO may generate both OOD and in-distribution states. Similarly, those methods that underestimate the values of state-action pairs cannot guarantee that the actions sampled from a different policy are OOD. In this respect, the learning policy might also be too conservative in visiting in-distribution states or actions.

1 College of Intelligence and Computing, Tianjin University, Tianjin 300350, China
2 Lab for High Technology, Tsinghua University, Beijing 100084, China
3 Department of Automation, Tsinghua University, Beijing 100084, China
4 Noah's Ark Lab, Huawei Technologics, Co., Ltd., Beijing 100084, China
Address correspondence to Jianye Hao, jianye.hao@tju.edu.cn

Additionally, unreliable OOD states can lead to inefficient underestimation of the values of some true OOD states. As a result, value errors at these true OOD states can still be propagated and magnified through value bootstrapping and policy improvement, thus affecting policy learning.

To address the above problems, we propose OOD state-conservative offline reinforcement learning (OSCAR), a model-free algorithm that derives a state-conservative policy by underestimating OOD state values in an explicit manner. We first design an efficient and reliable OOD state generation method by embedding the offline dataset into a low-dimensional manifold, followed by a scheme that perturbs each state in the offline dataset by tiny noise sampled from the orthogonal space of the tangent space of the manifold at the state point to be perturbed. Then we design a policy learning scheme that conducts policy improvement based on a learned Q-function that explicitly underestimates the values of the generated OOD states. We evaluate OSCAR on the D4RL[14] and SMARTS[4] benchmarks, and verify that the proposed method outperforms most existing offline RL algorithms (e.g., CQL, COMBO, UWAC[15], and so on) and achieves competitive performance when compared with EDAC[16], the resource-intensive state-of-the-art method.

Our main contributions are

- The proposed tiny noise generation method ensures that the generated states are not random and exist in the real world while sampling noise from the orthogonal space ensures the generated states are truly OOD.
- Conservativeness is achieved by explicitly generating OOD states instead of the implicit ways as described in Refs. [12, 13]. As a result, OSCAR is more effective in reducing overestimation errors of in-distribution states caused by propagating inaccurate values of OOD states through value bootstrapping and policy improvement.
- We also theoretically prove that the conservative Q-function evaluation method guarantees the underestimation of the values of OOD states.
- OSCAR outperforms several state-of-the-art offline RL methods on different benchmarks.

1 Preliminary

1.1 Markov decision process and RL

A Markov decision process (MDP) is formulated by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, d_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$ is the transition probability distribution, $r: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, d_0 is the initial state distribution, and $\gamma \in (0,1]$ is the discount factor. The goal of RL is to find an optimal policy $\pi(\boldsymbol{a} \mid \boldsymbol{s})$ that maximizes the cumulative discounted rewards $\boldsymbol{E}_{\boldsymbol{s}_t, \boldsymbol{a}_t}\left[\sum_{t=0}^{+\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$, where $\boldsymbol{s}_0 \sim d_0(\cdot)$, $\boldsymbol{a}_t \sim \pi(\cdot \mid \boldsymbol{s}_t)$, and $\boldsymbol{s}_{t+1} \sim T(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$. Q-learning is the major approach for obtaining the optimal $\pi(\boldsymbol{a} \mid \boldsymbol{s})$, which learns a Q-value function $Q(\boldsymbol{s}, \boldsymbol{a})$ that represents the expected cumulative discounted rewards when starting from the state $\boldsymbol{s}$ taking the action $\boldsymbol{a}$ and executing the policy π thereafter. Given the Q-function, the greedy policy can be recovered using maximization scheme. In standard actor-critic approaches, a separate policy is trained to maximize the Q-value. Thus the actor-critic methods alternate between policy evaluation and policy improvement. In policy evaluation, the value function of π is evaluated by iterating the Bellman operator as $\mathcal{B}^\pi Q(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{E}_{\boldsymbol{s}' \sim T(\cdot \mid \boldsymbol{s}, \boldsymbol{a})}\left[r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \boldsymbol{E}_{\boldsymbol{a}' \sim \pi(\cdot \mid \boldsymbol{s}')} Q(\boldsymbol{s}', \boldsymbol{a}')\right]$ with Q^π as the fixed point. In policy improvement, the policy is updated towards maximizing the expected Q-value: $J(\pi) := \boldsymbol{E}_{\boldsymbol{s} \sim \mathcal{S}, \boldsymbol{a} \sim \pi(\cdot \mid \boldsymbol{s})}\left[Q^\pi(\boldsymbol{s}, \boldsymbol{a})\right]$.

1.2 Offline RL

Offline RL follows the setting that we only have access to a fixed dataset $\mathcal{D} = \{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')\}$, which consists of transition tuples collected using an unknown behavior policy π_β. In other words, the dataset $\mathcal{D}$ is sampled from $d^{\pi_\beta}(\boldsymbol{s}, \boldsymbol{a}) := d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a} \mid \boldsymbol{s})$. We define $\widehat{\mathcal{M}}$ as the empirical MDP induced by the dataset $\mathcal{D}$, and use $d(\boldsymbol{s}, \boldsymbol{a})$ and $d(\boldsymbol{s})$ to represent the empirical state-action distribution and state distribution, respectively. We also define the empirical Bellman operator in $\widehat{\mathcal{M}}$ as $\hat{\mathcal{B}}^\pi Q(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{E}_{\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}' \sim \mathcal{D}}\left[r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \boldsymbol{E}_{\boldsymbol{a}' \sim \pi(\cdot \mid \boldsymbol{s}')} Q(\boldsymbol{s}', \boldsymbol{a}')\right]$.

However, as the target values for Bellman backups in policy evaluation use actions sampled from the learning policy while the Q-function is trained only on actions sampled from the fixed dataset $\mathcal{D}$, existing vanilla RL algorithms following this recipe suffer from action distribution shift during training. As the learning policy is trained to maximize Q-values, it may be misled towards OOD actions with erroneously high Q-values. By contrast, online RL does not face this overestimation issue since the agent can correct the bias by interacting with the environment. Since it might be costly or infeasible to interact with the environment in the offline setting, typical offline RL methods mitigate this problem either by constraining the learning policy away from OOD actions or by conservatively estimating the Q-values of OOD actions. Nevertheless, despite focusing on staying away from OOD actions, existing model-free offline RL algorithms rarely consider a fact that the policy may suffer from state distribution shift during online evaluation due to the absence of explicitly underestimating the values of OOD states during training.

COMBO tries to mitigate this problem by underestimating the values of out-of-support state-action pairs via rollouts under the learned dynamics model. However, due to the difficulty of fitting accurate dynamics models using limited data, it is challenging to tell whether the states given by the dynamics model are OOD. As a result, the learning policy might also be too conservative on some in-distribution states. Besides, as the learned dynamics model may produce states lying far away from the region covered by the offline dataset, staying conservative on them can hardly help conservative policy evaluation and improvement. Worse still, those states far away may not exist in the real world at all and may shape the landmark of the value function in an uncontrollable way. Therefore, how to generate reliable OOD states and how to alleviate the side effect of the distribution shift on the learning policy using OOD states still remain open problems.

2 Method

To address the above problems, in this section, we present our proposed method named OSCAR which explicitly underestimates the values of OOD states. We achieve this by embedding the states in the offline dataset into a low-dimensional manifold and then sampling OOD states in the orthogonal space close to the boundary of the manifold, as described in Section 2.1. Given the OOD states, we design the offline policy optimization method alternatives between an OOD state-conservative policy evaluation step and a policy improvement step, as described in Section 2.2. We theoretically prove that OSCAR guarantees to underestimate the values of OOD states in Section 2.3. The practical implementation of OSCAR is then demonstrated in Section 2.4 and outlined in Algorithm 1.

Algorithm 1 OSCAR

Input: Offline dataset $\mathscr{D}$, mini-batch size B, target network update rate τ, initial VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, policy π_θ, critic Q_φ, with random parameters ω, θ, φ, and target critic $Q_{\varphi'}$ with $\varphi' \leftarrow \varphi$

1: Initialize the OOD state replay buffer $\mathscr{D}_{\text{ood}} \leftarrow \varnothing$;

2: **for** each training step **do**

3: Sample mini-batch of B states $\boldsymbol{s}$ from $\mathscr{D}$;

4: $\mu, \delta = E_{\omega_1}(\boldsymbol{s})$, $\tilde{\boldsymbol{s}} = D_{\omega_2}(\boldsymbol{z})$, $\boldsymbol{z} \sim \mathcal{N}(\mu, \sigma)$;

5: Update the VAE model by
$\omega \leftarrow \arg\min_{\omega} \sum(\boldsymbol{s} - \tilde{\boldsymbol{s}})^2 + D_{\text{KL}}(\mathcal{N}(\mu, \sigma) \| \mathcal{N}(0,1))$

6: Generate perturbed sampled states using D_{ω_2} as described in Eq. (2) and save them into $\mathscr{D}_{\text{ood}}$;

7: **for** each training step **do**

8: Sample mini-batch of B transitions $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$ from $\mathscr{D}$;

9: Sample mini-batch of B OOD states from $\mathscr{D}_{\text{ood}}$;

10: Evaluate current policy by solving Formula (3) with the conservative loss of Formula (8);

11: Improve policy with SAC-style entropy regularization as described in Formula (9);

12: Update target network: $\varphi' \leftarrow \tau\varphi + (1-\tau)\varphi'$

2.1 OOD state generation

According to the discussion in Section 1.2, given an offline dataset, we aim to generate OOD states that lie beyond the border of the region covered by the offline dataset and meanwhile not too far away from it. To this end, inspired by Ref. [17], we first obtain a low-dimensional manifold representation of the offline dataset and then generate OOD states by adding minor perturbation sampled from the orthogonal subspace of the tangent space where the original states to be perturbed locate. An illustration is given in Fig. 1.

We use a variational autoencoder (VAE) to represent the offline data manifold by learning mappings from low-dimensional latent vectors $\boldsymbol{z} \in Z$ to high-dimensional states $\boldsymbol{s} \in X$, where X is the state space same as that of the offline states, and Z is the latent state space. Let E and D denote the encoder and decoder functions of the VAE, respectively. Given the well-trained VAE, the tangent space of the manifold at a state $\boldsymbol{s}$ is given by the column space of the Jacobian of the VAE decoder, which is defined as

$$J(\boldsymbol{s}) = \left.\frac{\partial D(\boldsymbol{z})}{\partial \boldsymbol{z}}\right|_{\boldsymbol{z}=E(\boldsymbol{s})} \tag{1}$$

Note that although the manifold is generally non-linear, for every point on the manifold, we can obtain its tangent space through local linearization. Given the tangent space $J(\boldsymbol{s})$, its orthogonal subspace can be obtained by calculating the left null space of $J(\boldsymbol{s})$. For computing convenience, we instead calculate the null space of $J^{\text{T}}(\boldsymbol{s})$ (the transpose of $J(\boldsymbol{s})$), which is equivalent to the left null space of $J(\boldsymbol{s})$. Denoted as $N(\boldsymbol{s})$ the null space of $J^{\text{T}}(\boldsymbol{s})$, the base vectors of $N(\boldsymbol{s})$ span the space perpendicular to the tangent space at $\boldsymbol{s}$. Denote $\boldsymbol{v}(\boldsymbol{s}) \sim N(\boldsymbol{s})$ as a unit vector randomly sampled from $N(\boldsymbol{s})$, we can obtain the perturbed state by adding a tiny perturbation along the direction of $\boldsymbol{v}(\boldsymbol{s})$ to the in-distribution $\boldsymbol{s}$:

$$\tilde{\boldsymbol{s}} = \boldsymbol{s} + \alpha \boldsymbol{v}(\boldsymbol{s}) \tag{2}$$

where $\alpha \in \mathbb{R}$ is a hyperparameter that controls the distance between the perturbed state and the unperturbed one. Our goal is to mitigate the overestimation of values of in-distribution states by underestimating the values of OOD states generated by Eq. (2). In this regard, the OOD states close to the boundary of the manifold should be more concerned. This is because the underestimated values of OOD states too far away from the manifold will not be effectively propagated to in-distribution states through value bootstrapping and the learning policy will not benefit from staying conservative on these states. To achieve this, we set α to a relatively small value to generate OOD states near the boundary of the manifold.

Following the above method, we generate one perturbed state for each state sampled from $\mathscr{D}$. The perturbed states together constitute the OOD states datasets $\mathscr{D}_{\text{ood}}$.

2.2 State-conservative offline policy optimization

Conservative policy evaluation. Given the offline dataset $\mathscr{D}$ and the OOD state dataset $\mathscr{D}_{\text{ood}}$, the goal in this part is to obtain a conservative estimation of the Q-function. Following CQL and COMBO, we design our conservative loss by introducing two items: (1) penalizing the Q-values of the state-action pairs with states sampled from the OOD state dataset $\mathscr{D}_{\text{ood}}$ and actions from a policy π; (2) increasing the Q-values of the state-action pairs in the offline dataset $\mathscr{D}$ to further highlight the difference between values of OOD states and those of in-distribution states. Combined with the standard temporal difference (TD) error, the loss for conservative policy evaluation becomes

$$\begin{aligned}\hat{Q}_{k+1}(\boldsymbol{s}, \boldsymbol{a}) \leftarrow \arg\max_{Q} \beta \big(& \boldsymbol{E}_{\boldsymbol{s} \sim \mathscr{D}_{\text{ood}}, \boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s})} Q(\boldsymbol{s}, \boldsymbol{a}) - \\ & \boldsymbol{E}_{\boldsymbol{s}, \boldsymbol{a} \sim \mathscr{D}} Q(\boldsymbol{s}, \boldsymbol{a}) \big) + \frac{1}{2} \boldsymbol{E}_{\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}' \in \mathscr{D}} \left[\left(Q(\boldsymbol{s}, \boldsymbol{a}) - \hat{\mathscr{B}}^{\pi} Q_k(\boldsymbol{s}, \boldsymbol{a}) \right)^2 \right]\end{aligned} \tag{3}$$

where β is a hyperparameter adjusting weights between the conservative loss and the TD error. Note that in contrast to

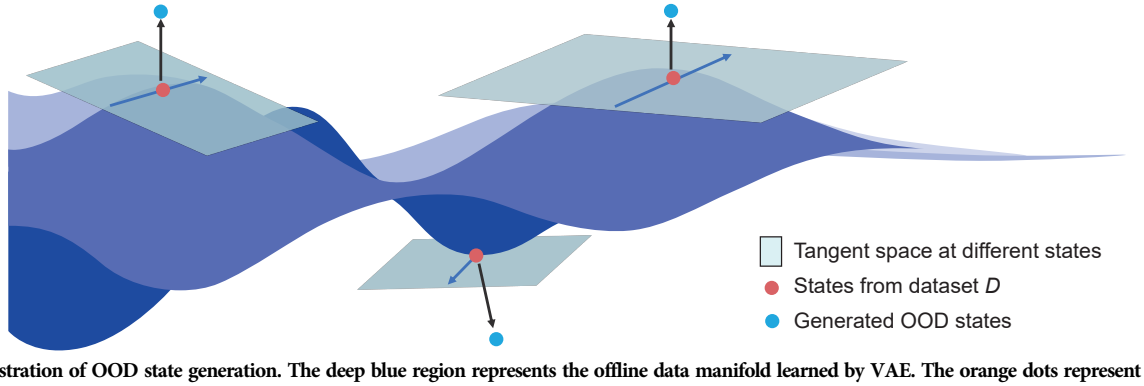

Fig. 1 Illustration of OOD state generation. The deep blue region represents the offline data manifold learned by VAE. The orange dots represent the states from the dataset. The shadow blue region represents the tangent space at different states. The black arrows indicate the orthogonal direction of tangent space at different states. The light blue dots represent the generated OOD states.

COMBO, OSCAR only stays conservative on reliable OOD states.

Policy improvement with state-conservative critic. After learning the conservative Q-function as $\hat{Q}^{\pi}$, the policy can be improved following Formula (4). The conservative Q-function could push the agent towards regions close to the states with higher estimated values.

$$\pi \leftarrow \arg\max_{\pi'} \boldsymbol{E}_{\boldsymbol{s}\sim\mathscr{D},\boldsymbol{a}\sim\pi'(\cdot|\boldsymbol{s})}\hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a}) \tag{4}$$

2.3 Theoretical analysis

In this section, we theoretically analyze our method and show that OSCAR underestimates $V^{\pi}(\boldsymbol{s})$ for $\boldsymbol{s}\in\mathscr{D}_{\text{ood}}$ and any π with a large probability in the tabular setting, where $\mathscr{S}$ and $\mathscr{A}$ are both finite sets. For ease of expression, let P^{π} denote the Hadamard product of the dynamics T and a given policy π in $\mathscr{M}$, and $d_{\text{ood}}(\boldsymbol{s})$ denote the sampling distribution which generates $\mathscr{D}_{\text{ood}}$. Note that symbols in bold in this section represent vectors. For two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $\boldsymbol{x}/\boldsymbol{y}$, $\boldsymbol{x}\cdot\boldsymbol{y}$, and $f(\boldsymbol{x})$ are all element-wise operations. Moreover, we follow Ref. [11] and use $\frac{1}{\sqrt{|\mathscr{D}|}}$ to denote a vector of size $|\mathscr{S}|\times|\mathscr{A}|$ containing square root inverse counts for each state-action pair in Formula (6) or only state counts in Formula (7). The Q-function, found by approximate dynamic programming in iteration k, can be obtained by differentiating Formula (3) with respect to Q_k:

$$\hat{Q}_{k+1}(\boldsymbol{s},\boldsymbol{a}) = \hat{\mathscr{B}}^{\pi}Q_k(\boldsymbol{s},\boldsymbol{a}) - \beta\frac{\rho(\boldsymbol{s},\boldsymbol{a})-d(\boldsymbol{s},\boldsymbol{a})}{d(\boldsymbol{s},\boldsymbol{a})} \tag{5}$$

where $\rho(\boldsymbol{s},\boldsymbol{a}) := d_{\text{ood}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$.

Lemma 1 Assume that $d(\boldsymbol{s},\boldsymbol{a})>0, \forall \boldsymbol{s}\in S, \boldsymbol{a}\in A$, then for $\forall\delta\in(0,1)$, there exists a constant $C_{r,T,\delta}$ such that for any π, the Q-function obtained by recursively applying Formula (3) satisfies the following relationship with probability $1-\delta$:

$$\forall \boldsymbol{s},\boldsymbol{a}, \hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a}) \leqslant Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) - \beta\left[(I-\gamma P^{\pi})^{-1}\left[\frac{\rho-d}{d}\right]\right](\boldsymbol{s},\boldsymbol{a}) + \left[(I-\gamma P^{\pi})^{-1}\left[\frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathscr{D}|}}\right]\right](\boldsymbol{s},\boldsymbol{a}) \tag{6}$$

Defining the expected estimated Q-value under π as $\hat{V}(\boldsymbol{s}) := \boldsymbol{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\hat{Q}(\boldsymbol{s},\boldsymbol{a})$, we reach to Lemma 2.

Lemma 2 Under the same conditions of Lemma 1, with probability $1-\delta$, we have

$$\forall \boldsymbol{s}, \hat{V}^{\pi}(s) \leqslant V^{\pi}(\boldsymbol{s}) - \beta(I-\gamma P^{\pi})^{-1}\left[\frac{1-d/d_{\text{ood}}}{d/d_{\text{ood}}}\cdot E_{\pi}\left[\frac{\pi}{\pi_{\beta}}\right] + D_{\text{CQL}}(\pi,\pi_{\beta})\right](\boldsymbol{s}) + \left[(I-\gamma P^{\pi})^{-1}\left[\frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathscr{D}|}}\right]\right](\boldsymbol{s}) \tag{7}$$

where $D_{\text{CQL}}(\pi,\pi_{\beta})(\boldsymbol{s}) = \boldsymbol{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\frac{\pi(\boldsymbol{a}|\boldsymbol{s})-\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}$ is the CQL distance defined in Ref. [11].

Note that $d(\boldsymbol{s},\boldsymbol{a})>0$ required in Lemmas 1 and 2 is a standard concentration assumption widely used in literatures such as Refs. [11, 12], and $C_{r,T,\delta}$ is the concentration coefficient depending on the desired confidence value δ. Lemmas 1 and 2 can be obtained by adopting some existing theoretical results, and details can be found in Appendix A. It can be further derived that the second term in Formula (7) is positive if $d_{\text{ood}}(\boldsymbol{s})>d(\boldsymbol{s})$, and the third term of Formula (7) is upper bounded by a constant independent of β. Based on these results, we can obtain Theorem 1.

Theorem 1 (Conservative OOD state value estimation) Under the same conditions of Lemma 1 and further assume that $\forall \boldsymbol{s}\in\mathscr{D}_{\text{ood}}$, $d_{\text{ood}}(\boldsymbol{s})>d(\boldsymbol{s})$, then with probability $1-\delta$, $\hat{V}^{\pi}(\boldsymbol{s})\leqslant V^{\pi}(\boldsymbol{s}), \forall\boldsymbol{s}\in\mathscr{D}_{\text{ood}}$, given that β is large enough.

Theorem 1 shows that estimating Q-values using Formula (9) guarantees that OOD state values are conservatively estimated with high probability if $d_{\text{ood}}(\boldsymbol{s})>d(\boldsymbol{s})$.

Remark 1 For OSCAR, the states in $\mathscr{D}_{\text{ood}}$ are assured to lie outside of the manifold of the offline dataset and thus cannot be represented by any vectors in the manifold. In other word, the states in $\mathscr{D}_{\text{ood}}$ appear in $\mathscr{D}$ with low probability while appear in $\mathscr{D}_{\text{ood}}$ with high probability. As a result, the inequality $d_{\text{ood}}(\boldsymbol{s})>d(\boldsymbol{s})$ for $\forall\boldsymbol{s}\in\mathscr{D}_{\text{ood}}$ holds, which is exactly the condition required in Theorem 1.

2.4 Practical implementations

In this section, we introduce the practical implementation of OSCAR. Before training the RL agent, we first train the VAE. We parameterize the VAE using a neural network and update it in an end-to-end manner by minimizing the MSE loss that evaluates the state reconstruction effectiveness and KL divergence that measures the distance between the embedding distribution in the latent space with a zero-mean, unit-variance Gaussian distribution. Then we generate the OOD states using the well-trained VAE and save them into $\mathscr{D}_{\text{ood}}$. Details can be found in lines 1–6 in Algorithm 1.

For the RL agent, we parameterize the Q-function and the policy using the neural network as Q_{φ} and π_{θ}, respectively. Similar to CQL, with the modification to Formula (3) by adding a designed KL divergence regularizer, the first term in Formula (3) can be transformed into a logsumexp term, which can be exactly calculated in discrete action domains while should be approximated via importance sampling in continuous action domains. Therefore, following CQL, we also sample actions both from the current policy and a uniform distribution $\mathscr{U}(\boldsymbol{a})$ over the action space. Then the conservative loss term of Formula (3) is instantiated as

$$\log\left(\frac{1}{2N}\sum_{\substack{\boldsymbol{s}\sim\mathscr{D}_{\text{ood}},\\ \boldsymbol{a}_i\sim\mathscr{U}(\boldsymbol{a}),\pi_{\theta}(\boldsymbol{a}|\boldsymbol{s})}}^{N}\left[\frac{\exp(Q_{\varphi}(\boldsymbol{s},\boldsymbol{a}_i))}{\mathscr{U}(\boldsymbol{a})}\right]\right) - \boldsymbol{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathscr{D}}Q(\boldsymbol{s},\boldsymbol{a}) \tag{8}$$

where N is the number of the sampled actions. We build our algorithm upon SAC, and thus the policy improvement operation argmax in Formula (4) is achieved via minimizing the negative Q-value along with a policy entropy regularization term through gradient descent:

$$J(\pi_{\theta}) := \boldsymbol{E}_{\boldsymbol{s}\sim D,\boldsymbol{a}\sim\pi_{\theta}(\cdot|\boldsymbol{s})}\left[\tau\log(\pi_{\theta}(\boldsymbol{a}|\boldsymbol{s})) - Q_{\varphi}(\boldsymbol{s},\pi_{\theta}(\boldsymbol{a}|\boldsymbol{s}))\right] \tag{9}$$

where τ is a temperature parameter that can be tuned automatically. Details of the RL agent can be found in lines 7–12 in Algorithm 1.

3 Experiment

We provide the empirical evaluations in this section to shed light on the effectiveness of the proposed method OSCAR.

3.1 Setup

We first compare our proposed OSCAR to prior offline RL methods on the continuous control MuJoCo datasets from the D4RL benchmark[14], including three environments (halfcheetah, hopper, and walker2d) and six dataset types (medium, medium-replay, full-replay, expert, medium-expert, and random). We choose offline RL algorithms that use different forms of behavior

regularization as baselines, including: policy constraint based methods TD3PlusBC[18], value regularization based algorithm CQL[11], uncertainty-based algorithms UWAC[15], and EDAC[16]. These baselines can fully reflect the state of the arts. We run each algorithm for one million training steps and report the normalized average return of each policy. The normalized average return is computed using the D4RL built-in env.get_normalized_score (returns) function where the return is the accumulated un-discounted rewards of an episode. Each algorithm is evaluated with three different seeds and the performance of each policy is evaluated for 10 episodes. The results of EDAC and UWAC are obtained using the open-sourced codes provided by the authors at EDAC implementation and UWAC implementation, respectively. The results of COMBO in Table 1 without standard deviation values are directly taken from the original paper while the others are obtained using the d3rlpy implementation (detailed explanations are presented in Appendix B). For the medium-expert and expert datasets, the hyperparameter β of OSCAR to adjust weights between the conservative loss and TD error is set to 5. For the other datasets, we set $\beta = 0.5$. We also set the number of Q-networks in OSCAR to three. More implementation details of different algorithms and comparisons with more algorithms are provided in Appendices B and D, respectively.

To further show the effectiveness of OSCAR, we also provide results of different algorithms to control a ego-agent trained on Waymo datasets linked in SMARTS to complete different autonomous driving scenarios[4] including lane turning, merging, cruising, cutting in, and overtaking. In each driving scenario, the ego-agent should drive towards their respective goal locations. Detailed description of the scenarios is provided in the SMARTS platform. We compare OSCAR with EDAC, CQL, and COMBO trained on these datasets and evaluate the performance in the different scenarios using the simulator provided in SMARTS. We set $\alpha = 0.01$ and $\beta = 1.0$ in OSCAR, and train all the algorithms for 300 000 gradient steps.

3.2 Results on D4RL and SMARTS

The results of OSCAR and the baselines on D4RL MuJoCo are presented in Table 1, and we highlight the top 2 best results on each dataset. It can be concluded that our method reaches the state-of-the-art performance compared with baselines after one million gradient steps. We also evaluate the successful rate of completing the goals in different scenarios in Fig. 2. As we can see from it, our method achieves the highest successful rate (i.e., 89%) over all scenarios with faster convergence speed. The faster learning speed benefits from the OOD states generation in the driving scenario. As the Waymo datasets only contain the expert data, thus with the help of OOD states (which may be non-expert), OSCAR can estimate the Q-value of states in datasets more accurately. These results demonstrate the effectiveness of the proposed OSCAR.

In particular, OSCAR outperforms both CQL and COMBO. The major difference between our proposed OSCAR and these two baselines is that OSCAR explicitly underestimates the values of states that are guaranteed to be OOD and located near the original states. We can reach the conclusion that OSCAR significantly benefits from conservative value estimation of the controllable OOD state generation method. Note that OSCAR achieves performance comparable with the current best baseline EDAC that trains for 3 million steps (refer to Appendix D). It can also be observed from Table 1 that OSCAR obtains relatively lower returns on random datasets, especially on the walker2d dataset. We think this phenomenon can be attributed to the difficulty of learning high-quality manifolds for random datasets.

3.3 Measurement of OOD states

To have a further investigation of the difference between states

Table 1 Comparison of OSCAR to prior work by normalized return.

Dataset	UWAC	COMBO	EDAC	TD3PlusBC	CQL	OSCAR
halfcheetah-random	2.3 ± 0.0	**38.8**	28.4 ± 0.3	10.7 ± 1.3	24.4 ± 2.2	**28.5 ± 1.0**
halfcheetah-medium	42.0 ± 0.5	54.2	**64.2 ± 2.1**	48.8 ± 0.3	49.0 ± 0.2	**58.3 ± 1.6**
halfcheetah-medium-replay	36.1 ± 4.4	**55.1**	**63.3 ± 1.7**	44.6 ± 0.7	49.1 ± 0.2	54.3 ± 0.4
halfcheetah-full-replay	62.3 ± 2.3	6.5 ± 3.2	**82.5 ± 2.3**	74.3 ± 2.6	**81.3 ± 0.2**	80.6 ± 1.7
halfcheetah-medium-expert	43.0 ± 0.3	**90.0**	72.2 ± 32.6	83.4 ± 2.4	81.1 ± 6.0	**91.4 ± 1.7**
halfcheetah-expert	92.5 ± 0.7	−1.8 ± 1.7	4.8 ± 1.1	**97.1 ± 0.4**	94.5 ± 6.1	**94.5 ± 2.1**
hopper-random	2.6 ± 0.1	**17.9**	7.7 ± 0.3	8.9± 0.3	8.9 ± 1.0	**15.1 ± 12.2**
hopper-medium	49.7 ± 7.4	**94.9**	**101.3 ± 0.8**	60.5 ± 3.4	67.6 ± 1.9	82.1 ± 4.1
hopper-medium-replay	30.8 ± 13.1	73.1	**101.5 ± 0.6**	53.4 ± 17.8	97.5 ± 1.3	**103.8 ± 0.7**
hopper-full-replay	21.9 ± 7.9	89.7 ± 24.7	106.1 ± 0.1	89.0 ± 13.8	**106.3 ± 0.2**	**107.7 ± 0.6**
hopper-medium-expert	50.9 ± 7.8	**111.1**	88.1 ± 32.3	102.0 ± 6.5	106.5 ± 7.1	**111.2 ± 1.4**
hopper-expert	**111.4 ± 0.8**	1.6 ± 0.8	29.8 ± 16.7	**108.4 ± 3.6**	107.9 ± 3.8	103.4 ± 7.1
walker2d-random	**2.8 ± 0.2**	**7.0**	0.0 ± 0.0	1.8 ± 0.9	−0.3 ± 0.3	2.7 ± 1.3
walker2d-medium	78.3 ± 2.8	75.5	**89.8 ± 0.4**	85.0 ± 0.4	83.2 ± 0.3	**88.0 ± 0.9**
walker2d-medium-replay	25.5 ± 7.1	56.0	81.7 ± 1.1	**84.2 ± 5.0**	83.4 ± 5.9	**89.2 ± 3.2**
walker2d-full-replay	25.6 ± 31.2	74.0 ± 32.1	98.6 ± 1.2	94.6 ± 1.4	**98.9 ± 0.8**	**98.6 ± 0.8**
walker2d-medium-expert	107.2 ± 2.8	96.1	**113.9 ± 0.4**	**111.0 ± 0.4**	108.7 ± 0.8	110.2 ± 0.2
walker2d-expert	108.0 ± 0.6	0.4 ± 0.8	37.7 ± 51.1	**110.3 ± 0.2**	**110.2 ± 0.5**	108.6 ± 0.4
Average	49.6	52.2	65.1	70.4	75.5	79.3

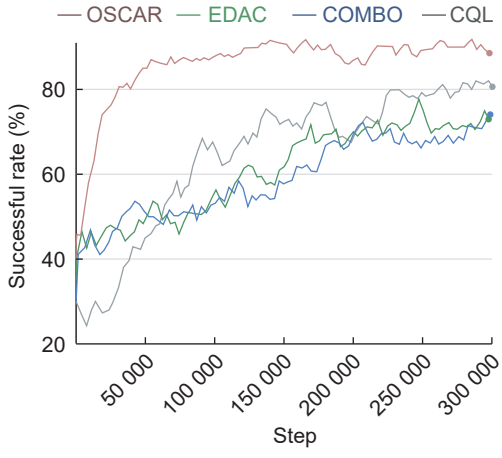

Fig. 2 Successful rate in all step scenarios.

given by OSCAR and COMBO, we compare the ℓ_2 distances between the generated states of OSCAR and COMBO and the original states in Table 2. For COMBO, states are generated by rollouting the learned dynamics model using its learned policy for one to five steps, as done in Ref. [12]. As we can see in Table 2, OSCAR has full control of the distances between generated states and the original states via adjusting α while COMBO's learned dynamics model yields states that are far away from the original ones even when the rollout length is set to one. The quality of OOD states further affects policy learning, as we can see, OSCAR $(\alpha = 0.01)$ obtains much higher returns than COMBO. We can also observe that when we increase α to 1.0, the performance of OSCAR drops to −0.4, much lower than that of COMBO, although the average distance of its generated states to the original states is much lower than that of COMBO. One of the potential reasons is that OSCAR generates OOD states that are assured to be in the orthogonal space but COMBO does not, and thus the average vertical distance of the states generated by COMBO to the manifold could be smaller than that of OSCAR $(\alpha = 1.0)$. Based on these observations, we can conclude that staying conservative on reliable OOD states that lie near the manifold of the offline dataset can significantly improve the performance of the learning policy.

3.4 Performance under different β

We evaluate OSCAR on five walker2d datasets to measure its sensitivity to the conservative loss weight β by taking values from $\{0, 0.5, 1.0, 3.0, 5.0, 10.0\}$. Note that when $\beta = 0$, the algorithm degenerates to SAC. As we can see from Table 3, for each dataset, OSCAR can achieve superior performance over a wide range of β values. Besides, we can also find out that for the datasets including trajectories with higher returns (i.e., medium-expert and expert datasets), policy learning should be more conservative. That is because datasets with higher returns tend to have relatively smaller data coverage of the entire state space. In such cases, if the degree of conservativeness is not enough, the agent may visit regions of low returns that are too far from the distribution of datasets with high probability during the online interactions. For datasets with lower returns, the coverage is relatively wider. As deviating from its distribution may help the agent visit regions with higher returns, the degree of conservatism can be appropriately reduced. Similar considerations are also demonstrated in Refs. [12, 18]. Results in Table 3 show that larger β on these datasets can guide the policy towards a more conservative direction and thus perform better. Correspondingly, being less conservative on suboptimal data (i.e., medium, medium-replay, and full-replay datasets) helps the policy achieve better performance.

3.5 Comparison of ensemble Q-networks

In practice, we set the number of Q-networks in OSCAR to three, which improves the Q-value estimation accuracy of OOD data with minor additional computational overhead. Figure 3 shows the minimum required number of Q-networks to achieve the best-reported results of OSCAR and EDAC. We omit the results on the full-replay datasets as both methods can achieve superior performance using two Q-networks on these datasets. Compared with EDAC, which is the strongest baseline, OSCAR needs much

Table 2 Comparison between the generated states of OSCAR and COMBO and the original states, and corresponding average return. OSCAR (*n*) means α=*n* in OSCAR. COMBO (*n*) means the generated states are obtained by *n* step rollout, of which the algorithm is implemented using d3rlpy.

Algorithm	ℓ_2 distance												Average return
	halfcheetah-medium-v2				halfcheetah-medium-replay-v2				halfcheetah-random-v2				
	min	mean	median	max	min	mean	median	max	min	mean	median	max	
OSCAR (0.01)	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	47.0
OSCAR (1.0)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	−0.4
COMBO (1)	2.0	23.5	23.3	57.8	1.4	22.3	21.5	66.3	2.2	23.5	23.1	60.5	
COMBO (2)	2.9	29.6	29.6	71.6	1.7	27.5	26.8	80.4	2.7	24.7	24.3	70.1	
COMBO (3)	2.4	33.0	32.9	69.8	2.0	28.0	28.6	76.7	2.5	22.4	22.0	63.7	27.2
COMBO (4)	2.2	34.5	35.6	66.1	1.6	26.4	25.8	67.4	2.4	22.2	21.9	59.0	
COMBO (5)	2.6	31.5	30.5	69.3	1.3	22.7	21.9	66.1	2.6	23.8	23.4	61.5	

Table 3 Performance over various β on walker2d datasets by normalized return.

Dataset	Normalized return					
	β=0	β=0.5	β=1.0	β=3.0	β=5.0	β=10.0
medium	0.0 ± 0.2	88.0 ± 0.9	84.1 ± 1.2	76.1 ± 2.6	78.9 ± 1.1	64.0 ± 7.2
medium-replay	17.4 ± 18.7	89.2 ± 3.2	89.6 ± 2.5	84.0 ± 2.0	79.5 ± 2.7	48.8 ± 23.9
full-replay	91.7 ± 10.4	98.2 ± 0.8	94.5 ± 0.7	94.6 ± 0.3	91.8 ± 2.1	91.2 ± 1.2
medium-expert	−0.1 ± 0.0	75.2 ± 52.4	112.9 ± 1.1	110.6 ± 0.3	110.2 ± 0.2	107.0 ± 2.9
expert	1.0 ± 0.9	83.3 ± 5.6	112.0 ± 0.5	110.5 ± 0.5	108.6 ± 0.4	108.5 ± 0.5

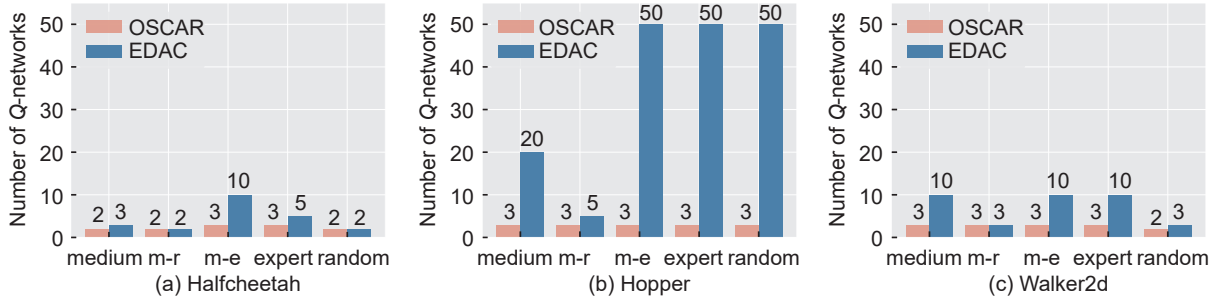

Fig. 3 Minimum number of *Q*-networks to achieve the best-reported results of OSCAR and EDAC. The *x*-axis indicates the different datasets where "m-r" represents the medium-replay dataset and "m-e" represents the medium-expert dataset. The *y*-axis indicates the number of *Q*-networks.

fewer *Q*-networks to reach superior performance, significantly saving computing resources. More computation cost comparison of different algorithms can be found in Appendix C.

4 Discussion

The goal of offline RL is to derive policies from a static dataset consisting of transitions collected via interaction with the environment[6]. It has been verified that offline RL can be applicable in multiple fields including negotiated dialogue[19], robotic manipulation[1], healthcare[5], and so on. The RL community has developed some offline RL algorithms based on the idea of adopting explicit or implicit regularization to implement conservative learning policies. Similar to the vanilla RL, these methods can also be divided into model-free ones and model-based ones.

Model-free offline RL. The core idea of the existing model-free offline RL algorithms is to explicitly or implicitly regularize the learning policy to be close to the behavioral policy. The most direct way is to explicitly measure the distance between the learning policy and the dataset at each state using direct action constraints[7], Wasserstein distance[9], KL divergence[9, 10, 20–22] or MMD[8]. The other way is to implicitly regularize the learning policy via designing regularized variants of importance sampling based approaches[23, 24] or underestimating the *Q*-values of state-action pairs, which is achieved by quantifying their uncertainties[8, 9, 25] or fitting conservative *Q*-value of sampled actions[11]. Different from these works, OSCAR uses both the offline dataset as well as the generated data sampled from the outside of the closed boundary of in-distribution. This helps OSCAR estimate the values of OOD data more conservatively compared to that of in-distribution data during training and thus avoid accessing OOD states during online evaluation.

Model-based offline RL. Most model-based offline RL methods[26–29] quantify the uncertainty of the learned dynamics models and introduce the uncertainty into the policy learning process. For example, Refs. [26, 27] conservatively estimate the state-action values via adding uncertainty-based penalization into reward function. In contrast to these methods, COMBO[12] learns a conservative policy by underestimating the values of states generated through model rollouts, which is the most similar method to OSCAR that seeks to underestimate the *Q*-values of OOD states. However, the generated states of OSCAR is controllable and much more reliable than that of COMBO.

5 Conclusion

In this paper, we propose OSCAR, an OOD state-conservative offline RL method, to address the potential adverse effect on value estimation of the in-distribution states induced by staying conservative on unreliable OOD states. We explicitly generate reliable OOD states that locate near the manifold of the offline dataset, and then design a conservative policy evaluation approach to only underestimates the values of these generated OOD states. The two techniques together prevent the propagating of value errors of OOD states to the values of in-distribution states through value bootstrapping and policy improvement. We also theoretically prove that the proposed conservative policy evaluation approach guarantees the underestimation of the values of OOD states. Experiments on the D4RL and SMARTS benchmarks demonstrate that OSCAR outperforms the previous popular offline RL algorithms and achieves competitive performance compared with the state-of-the-art method EDAC while consuming much less computing overhead. It is worth noting that OSCAR obtains relatively lower returns on random datasets. We think this can be attributed to the difficulty of learning high-quality manifolds for random datasets. We will continue to develop better manifold construction methods as well as other OOD state generation methods in the future to address this limitation.

Appendix

A Proof

Before we proceed, some lemmas are provided first.

Lemma A1 (Hoeffding's inequality) Suppose $X_1, X_2, \ldots, X_n$ is a sequence of independent, identically distributed random variables with mean μ. Let $Y_n = n^{-1}\sum_{i=1}^{n} X_i$. Suppose that $X_i \in [b_-, b_+]$ with probability 1, then

$$Y_n - \boldsymbol{E}(X) \leqslant (b_+ - b_-)\sqrt{\ln(1/\delta)/2n} \tag{A1}$$

We also provide a lemma below to bound the difference between the empirical Bellman backup operator and the true Bellman backup operator, by making some modifications to the discussion in Appendix B of Ref. [30].

Lemma A2 With probability $1-\delta$, where $\delta \in (0,1)$, we have

$$\forall Q, \boldsymbol{s}, \boldsymbol{a} \in \mathscr{D}, \left|\hat{\mathscr{B}}^{\pi}Q(\boldsymbol{s},\boldsymbol{a}) - \mathscr{B}^{\pi}Q(\boldsymbol{s},\boldsymbol{a})\right| \leqslant \left(\frac{1}{1-\gamma}\sqrt{\frac{1}{2}\ln\frac{2|\mathscr{S}\times\mathscr{A}|}{\delta}}R_{\max}\frac{1}{\sqrt{|\mathscr{D}(\boldsymbol{s},\boldsymbol{a})|}}\right) \tag{A2}$$

Proof Since r is bounded by $R_{\max}$, then $Q(\boldsymbol{s},\boldsymbol{a})$ is bounded in $[0, \frac{R_{\max}}{1-\gamma}]$, so $\hat{\mathscr{B}}^{\pi}Q(\boldsymbol{s},\boldsymbol{a})$, $\mathscr{B}^{\pi}Q(\boldsymbol{s},\boldsymbol{a})$ and $|\hat{\mathscr{B}}^{\pi}Q(\boldsymbol{s},\boldsymbol{a}) - \mathscr{B}^{\pi}Q(\boldsymbol{s},\boldsymbol{a})|$ are bounded in $[0, \frac{R_{\max}}{1-\gamma}]$ as well.

Considering that for any $\boldsymbol{s}, \boldsymbol{a}$, $\hat{\mathscr{B}}^{\pi}Q(\boldsymbol{s},\boldsymbol{a})$ can be equivalently expressed as an expectation of random variables,

$$\hat{\mathcal{B}}^{\pi}Q(\boldsymbol{s},\boldsymbol{a})=\frac{1}{|\mathscr{D}(\boldsymbol{s},\boldsymbol{a})|}\sum_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}',r\in\mathscr{D}}\left(r(\boldsymbol{s},\boldsymbol{a})+\gamma\boldsymbol{E}_{\boldsymbol{a}'\sim\pi}Q(\boldsymbol{s}',\boldsymbol{a}')\right) \tag{A3}$$

Note that each term in the RHS of Eq. (A3) has the expected value:

$$\begin{aligned}&\boldsymbol{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}',r\in\mathscr{D}}[r(\boldsymbol{s},\boldsymbol{a})+\gamma\boldsymbol{E}_{\boldsymbol{a}'\sim\pi}Q(\boldsymbol{s}',\boldsymbol{a}')]=\\&\boldsymbol{E}_{\boldsymbol{s}'\in T(\cdot|\boldsymbol{s},\boldsymbol{a})}[r(\boldsymbol{s},\boldsymbol{a})+\gamma\boldsymbol{E}_{\boldsymbol{a}'\sim\pi}Q(\boldsymbol{s}',\boldsymbol{a}')]=\mathcal{B}^{\pi}Q(\boldsymbol{s},\boldsymbol{a})\end{aligned} \tag{A4}$$

Thus, by invoking Hoeffding's inequality at each of the $|\mathscr{S}|\times|\mathscr{A}|$ state-action pairs, and taking a union bound, we see that with probability at least $1-\delta$, Formula (A2) holds.

Based on the above results, we can derive the proofs for Lemma 1, Lemma 2, and Theorem 1.

Proof of Lemma 1 The overestimation due to the empirical Backup operator can be bounded with high probability $1-\delta$, as follows:

$$\begin{aligned}&\hat{Q}_{k+1}(\boldsymbol{s},\boldsymbol{a})\leqslant\mathcal{B}^{\pi}Q_k(\boldsymbol{s},\boldsymbol{a})-\\&\beta\frac{\rho(\boldsymbol{s},\boldsymbol{a})-d(\boldsymbol{s},\boldsymbol{a})}{d(\boldsymbol{s},\boldsymbol{a})}+\frac{C_{r,T,\sigma}R_{\max}}{(1-\gamma)\sqrt{|\mathscr{D}(\boldsymbol{s},\boldsymbol{a})|}}\end{aligned} \tag{A5}$$

where we pick $C_{r,T,\sigma}:=\sqrt{\frac{1}{2}\ln\frac{2|\mathscr{S}\times\mathscr{A}|}{\delta}}$ by Lemma A2. Then we reason about the fixed point of the update of Formula (A5) and yield Lemma 1 straightforwardly.

Proof of Lemma 2 We take expectation on the two sides of Formula (A5), the second term in Formula (A5) becomes

$$\begin{aligned}&\boldsymbol{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\frac{\rho(\boldsymbol{s},\boldsymbol{a})-d(\boldsymbol{s},\boldsymbol{a})}{d(\boldsymbol{s},\boldsymbol{a})}=\\&\boldsymbol{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\frac{\pi(\boldsymbol{a}|\boldsymbol{s})-\frac{d(\boldsymbol{s})}{d_{\text{ood}}(\boldsymbol{s})}\cdot\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}{\frac{d(\boldsymbol{s})}{d_{\text{ood}}(\boldsymbol{s})}\cdot\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}=\\&\boldsymbol{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\frac{\frac{d(\boldsymbol{s})}{d_{\text{ood}}(\boldsymbol{s})}\cdot\pi(\boldsymbol{a}|\boldsymbol{s})-\frac{d(\boldsymbol{s})}{d_{\text{ood}}(\boldsymbol{s})}\cdot\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})+(1-\frac{d(\boldsymbol{s})}{d_{\text{ood}}(\boldsymbol{s})})\cdot\pi(\boldsymbol{a}|\boldsymbol{s})}{d(\boldsymbol{s})d_{\text{ood}}(\boldsymbol{s})\cdot\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}=\\&\boldsymbol{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\frac{\pi(\boldsymbol{a}|\boldsymbol{s})-\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}+\frac{1-d(\boldsymbol{s})/d_{\text{ood}}(\boldsymbol{s})}{d(\boldsymbol{s})/d_{\text{ood}}(\boldsymbol{s})}\cdot\boldsymbol{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}\end{aligned} \tag{A6}$$

Considering that the first term of Eq. (A6) is exactly the CQL distance between $\pi(\cdot|\boldsymbol{s})$ and $\pi_{\beta}(\cdot|\boldsymbol{s})$, then we obtain Lemma 2.

Proof of Theorem 1 Since we assume that $d_{\text{ood}}(\boldsymbol{s})>d(\boldsymbol{s})$, then $\hat{V}^{\pi}(\boldsymbol{s})\leqslant V_{\pi}(\boldsymbol{s})$, given that β satisfies the following condition:

$$\beta\geqslant\max_{\boldsymbol{s},\boldsymbol{a}\in D}\frac{C_{r,T,\sigma}R_{\max}}{(1-\gamma)\sqrt{|\mathscr{D}(\boldsymbol{s},\boldsymbol{a})|}}\cdot$$

$$\max_{\boldsymbol{s}\in\mathscr{D}}\left[D_{\text{CQL}}(\boldsymbol{s})+\frac{1-d(\boldsymbol{s})/d_{\text{ood}}(\boldsymbol{s})}{d(\boldsymbol{s})/d_{\text{ood}}(\boldsymbol{s})}\cdot\boldsymbol{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}\right]^{-1} \tag{A7}$$

which means that if β is large enough, $\hat{V}^{\pi}(s)\leqslant V_{\pi}(s)$. Thus Theorem 1 is proved.

B Implementation Detail

The experiments are conducted on an Intel(R) Xeon(R) Gold 6134 processor based Ubuntu 18.04.6 LTS Server, which consists of one processor of 16 cores, running at 3.20 GHz with 32 KB of L1, 1024 KB of L2, 25 344 KB of L3 cache, and 128 GB of memory and 1 Quadro RTX 5000 GPU. The MuJoCo Gym datasets we use in our experiments are v2 versions, which fixes some bugs as reported in the repository of D4RL. Our codes are implemented with Python 3.7.11 and PyTorch 1.6.0. Unless otherwise specified, for the baseline algorithms, we use the implementation of d3rlpy[31] following the MIT license. We also implement OSCAR based on d3rlpy. We give some details in the following.

COMBO. As the authors do not release the official implementation codes, we run the COMBO algorithm using d3rlpy. However, although we check carefully to follow the settings of rollout length, Q-function and policy learning rates, conservative coefficient, and so on, we could not reproduce similar results to that reported in the original paper. Therefore, in Section 3.2, the reported COMBO results on random, medium, medium-replay, and medium-expert datasets are directly taken from the original paper. For the full-replay and expert datasets, we use the implementation in d3rlpy.

CQL. We follow the official implementation guidelines from the official codes. We use the fixed conservative weight, for all the medium datasets, we set the conservative weight to 10; for the rest datasets, we set the conservative weight to 5. Besides, the actor learning rate is set to 1×10^{-4} while the critic is set to 3×10^{-4}.

EDAC. We use the official implementation and set different hyperparameters for different datasets as demonstrated in the original paper. For the Q-networks size, the ensemble number is set to 10 for halfcheetah and walker2d datasets and 50 for hopper datasets. For the ensemble gradient diversity term, the number is set to 0 for halfcheetah-random and hopper-random, 5.0 for halfcheetah-medium-expert, walker2d-expert, and walker2d-medium-expert, and 1.0 for the other datasets.

OSCAR. OSCAR consists of two parts, OOD state generating and RL learning. For the OOD state generating part, the dimension z_{dim} of the embedded state in VAE is set to int(state_dim/2)−1 where state_dim is the state dimension of each environment. For the medium-expert and expert datasets, the hyperparameter α that controls how far the perturbed sampled state is from the in-distribution state is set to 0.005. For the other datasets, we set $\alpha=0.01$. For the RL learning part, we follow the SAC implementation in d3rlpy. For the medium-expert and expert datasets, the hyperparameter β to adjust weights between the conservative loss and TD error is set to 5. For the other datasets, we set $\beta=0.5$. The other settings of OSCAR are shown in Table A1. Note that for all the baseline algorithms of d3rlpy, we use the same batch size and critic-actor architectures.

C Computation Cost Comparison

We compare the computation cost of OSCAR and CQL on

Table A1 Hyperparameters for OSCAR.

	Hyperparameter	Value
VAE	Learning rate	1×10^{-3}
	Number of epochs	10 000
	Batch size	128
	Hidden units	[128, 128]
RL	Critic number	3
	Critic learning rate	3×10^{-4}
	Actor learning rate	3×10^{-4}
	Temperature learning rate	3×10^{-4}
	Batch size	256
	Action samples number	10
	Critic hidden units	[256, 256, 256]
	Actor hidden units	[256, 256, 256]

halfcheetah-expert-v2. To ensure the fairness of the comparison, the two algorithms are implemented using the d3rlpy repository. We report the GPU memory consumption in Table A2. The runtime of one training epoch, which contains 1 000 gradient steps, is also reported. It can be concluded that our method consumes less GPU memory than CQL. The slightly longer running time than CQL mainly comes from sampling from the OOD state replay buffer and the additional Q-network.

As the official implementation of EDAC is based on RLkit, which is a different framework from d3rlpy, it is unfair to directly compare its computation cost with OSCAR and CQL implemented used d3rlpy. But we can conclude from the EDAC paper that EDAC runs slightly faster than CQL implemented based on RLkit while with about 30% more memory consumption. Thus we can infer that EDAC also consumes much more GPU memory than OSCAR due to the large amount of ensemble Q-networks.

D Extended Comparison Results with More Baselines

In this section, we give the results of more baselines on D4RL MuJoCo datasets. In addition to the baselines mentioned in Section 3.2, we also compare some other popular offline RL algorithms, i.e., BCQ[7], BEAR[8], CRR[32], IQL[33], MOPO[27], MOReL[26], DT[34], and TT[35]. We also take the results of EDAC after training for 3 million steps from the original paper. All the results are presented in Tables A3 and A4.

Table A2 Computation cost comparison.

Method	GPU memory (MB)	Runtime (s/epoch)
OSCAR	887	49
CQL	909	38

E Visualization of Different Generated States

We visualize the distribution of (1) original states from the offline dataset, (2) OOD data generated using OSCAR by setting $\alpha = 5$, and (3) states generated through model rollouts in COMBO. We excerpt 500 states and use t-SNE as the dimensionality reduction method to reduce the states to 2-dimension. Then we plot all the dimension-reduced states in Fig. A1. It can be observed that the states generated by OSCAR is very close to the original states, which proves that the generated OOD states are in the near boundary of the manifold of the original data. However, with the increasement of the rollout length, the generated states of COMBO gradually deviate away from the distribution of the original states.

Acknowledgment

This work was supported by the National Key R&D Program of China (No. 2022ZD0116402) and the National Natural Science Foundation of China (No. 62106172).

Article History

Received: 11 July 2023; Accepted: 31 August 2023

References

[1] A. Mandlekar, F. Ramos, B. Boots, S. Savarese, F. F. Li, A. Garg, and D. Fox, IRIS: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data, in *Proc. 2020 IEEE Int. Conf. Robotics and Automation* (*ICRA*), Paris, France, 2020, pp. 4414–4420.

[2] X. Hao, Z. Peng, Y. Ma, G. Wang, J. Jin, J. Hao, S. Chen, R. Bai, M. Xie, M. Xu, et al., Dynamic knapsack optimization towards efficient

Table A3 Extended normalized average returns of algorithms on D4RL Gym tasks. CQL and EDAC denote the results reported in the original papers, respectively. Other results denoted as "our" are obtained using the official codes or d3rlpy and training for 1 million steps as described in the experimental sections.

Dataset	BCQ (our)	BEAR (our)	CRR (our)	TD3PlusBC (our)	CQL[11]	CQL (our)	EDAC[16]	EDAC (our)	OSCAR
halfcheetah-random	2.3 ± 0.0	2.3 ± 0.0	14.0 ± 8.2	10.69 ± 1.28	35.4	24.42 ± 2.15	28.4 ± 1.0	28.35 ± 0.33	28.5 ± 1.0
halfcheetah-medium	46.8 ± 0.2	42.5 ± 0.2	51.2 ± 0.5	48.8 ± 0.27	44.4	49.00 ± 0.18	65.9 ± 0.6	64.17 ± 2.14	58.3 ± 1.6
halfcheetah-medium-replay	41.5 ± 1.1	38.9 ± 0.7	46.2 ± 0.1	44.64 ± 0.73	46.2	49.12 ± 0.20	61.3 ± 1.9	63.33 ± 1.74	54.3 ± 0.4
halfcheetah-full-replay	71.5 ± 1.6	61.6 ± 0.2	72.3 ± 2.6	74.25 ± 2.64	-	81.34 ± 0.23	84.6 ± 0.9	82.48 ± 2.26	80.6 ± 1.7
halfcheetah-medium-expert	92.3 ± 1.4	44.6 ± 2.1	44.4 ± 9.1	83.36 ± 2.39	62.4	81.10 ± 6.04	106.3 ± 1.9	72.18 ± 32.62	91.4 ± 1.7
halfcheetah-expert	93.4 ± 3.4	92.6 ± 0.4	36.6 ± 2.0	97.13 ± 0.42	104.8	94.47 ± 6.09	106.8 ± 3.4	4.79 ± 1.10	94.5 ± 2.1
hopper-random	7.9 ± 0.5	6.4 ± 0.3	3.6 ± 2.5	8.92 ± 0.27	10.8	8.85 ± 0.98	25.3 ± 10.4	7.65 ± 0.25	15.1 ± 12.2
hopper-medium	60.7 ± 4.6	52.3 ± 2.0	53.5 ± 7.1	60.52 ± 3.38	86.6	67.64 ± 1.88	101.3 ± 0.6	101.32 ± 0.77	82.1 ± 4.1
hopper-medium-replay	55.9 ± 20.0	39.5 ± 1.7	45.9 ± 6.2	53.37 ± 17.86	48.6	97.52 ± 1.34	101.0 ± 0.5	101.53 ± 0.62	103.8 ± 0.7
hopper-full-replay	37.8 ± 6.8	68.5 ± 19.2	60.5 ± 19.8	88.98 ± 13.80	-	106.27 ± 0.17	105.4 ± 0.7	106.07 ± 0.10	107.7 ± 0.6
hopper-medium-expert	79.9 ± 22.8	49.4 ± 6.7	29.1 ± 5.9	101.98 ± 6.47	111.0	106.54 ± 7.14	110.7 ± 0.1	88.11 ± 32.28	111.2 ± 1.4
hopper-expert	84.1 ± 6.8	78.0 ± 3.9	54.0 ± 7.2	108.36 ± 3.63	109.9	107.95 ± 3.79	110.1 ± 0.3	29.81 ± 16.70	103.4 ± 7.1
walker2d-random	5.0 ± 0.1	13.4 ± 10.0	1.8 ± 2.6	1.81 ± 0.85	7.0	−0.25 ± 0.27	16.6 ± 7.0	−0.03 ± 0.03	2.7 ± 1.3
walker2d-medium	71.6 ± 7.1	78.6 ± 5.7	32.9 ± 32.8	85.02 ± 0.37	74.5	83.19 ± 0.32	92.5 ± 0.8	89.83 ± 0.44	88.0 ± 0.9
walker2d-medium-replay	45.1 ± 11.6	5.5 ± 3.1	67.8 ± 2.6	84.18 ± 4.97	32.6	83.35 ± 5.92	87.1 ± 2.3	81.72 ± 1.08	89.2 ± 3.2
walker2d-full-replay	79.4 ± 14.7	83.0 ± 5.8	79.6 ± 9.1	94.61 ± 1.41	-	98.92 ± 0.81	99.8 ± 0.7	98.61 ± 1.21	98.6 ± 0.8
walker2d-medium-expert	111.3 ± 0.9	110.5 ± 0.3	78.8 ± 9.1	110.98 ± 0.42	98.7	108.65 ± 0.82	114.7 ± 0.9	113.90 ± 0.40	110.2 ± 0.2
walker2d-expert	108.5 ± 2.5	110.7 ± 0.5	15.1 ± 20.1	110.30 ± 0.15	121.6	110.18 ± 0.50	115.1 ± 1.9	37.70 ± 51.09	108.6 ± 0.4

Table A4 Extended normalized average returns of algorithms on D4RL Gym tasks. COMBO, IQL, DT, and TT denote the result reported in the original papers, respectively. MOPO is taken from Ref. [36] and MOReL is taken from Ref. [16]. Other results denoted as "our" are obtained using the official codes or d3rlpy and training for 1 million steps as described in the experimental sections.

Dataset	COMBO[12]	COMBO (our)	MOReL[26]	MOPO[27]	DT[34]	TT[35]	IQL[33]	OSCAR
halfcheetah-random	38.8	25.7 ± 3.0	38.9 ± 1.8	35.9 ± 2.9	-	-	-	28.5 ± 1.0
halfcheetah-medium	54.2	0.5 ± 6.9	60.7 ± 4.4	73.1 ± 2.4	42.6	46.9	47.4	58.3 ± 1.6
halfcheetah-medium-replay	55.1	55.4 ± 4.2	44.5 ± 5.6	69.2 ± 1.1	36.6	41.9	44.2	54.3 ± 0.4
halfcheetah-full-replay	-	6.5 ± 3.2	70.1 ± 5.1	-	-	-	-	80.6 ± 1.7
halfcheetah-medium-expert	90.0	0.3 ± 1.2	80.4 ± 11.7	70.3 ± 21.9	86.8	95.0	86.7	91.4 ± 1.7
halfcheetah-expert	-	−1.8 ± 1.7	8.4 ± 11.8	81.3 ± 21.8	-	-	-	94.5 ± 2.1
hopper-random	17.9	2.3 ± 1.7	38.1 ± 10.1	16.7 ± 12.2	-	-	-	15.1 ± 12.2
hopper-medium	94.9	1.2 ± 0.7	84.0 ± 17.0	38.3 ± 34.9	67.6	61.1	66.3	82.1 ± 4.1
hopper-medium-replay	73.1	62.0 ± 42.7	81.8 ± 17.0	32.7 ± 9.4	82.7	91.5	94.7	103.8 ± 0.7
hopper-full-replay	-	89.7 ± 24.7	94.4 ± 20.5	-	-	-	-	107.7 ± 0.6
hopper-medium-expert	111.1	2.0 ± 1.0	105.6 ± 8.2	60.6 ± 32.5	107.6	110.0	91.5	111.2 ± 1.4
hopper-expert	-	1.6 ± 0.8	80.4 ± 34.9	62.5 ± 29.0	-	-	-	103.4 ± 7.1
walker2d-random	7.0	5.2 ± 2.4	16.7 ± 7.0	4.2 ± 5.7	-	-	-	2.7 ± 1.3
walker2d-medium	75.5	−0.2 ± 0.0	72.8 ± 11.9	41.2 ± 30.8	74.0	79.0	78.3	88.0 ± 0.9
walker2d-medium-replay	56.0	63.3 ± 23.2	27.1 ± 9.6	73.7 ± 9.4	66.6	82.6	73.9	89.2 ± 3.2
walker2d-full-replay	-	74.0 ± 32.1	60.7 ± 15.6	-	-	-	-	98.6 ± 0.8
walker2d-medium-expert	96.1	0.4 ± 0.8	85.7 ± 14.0	77.4 ± 27.9	108.1	101.9	109.6	110.2 ± 0.2
walker2d-expert	-	0.4 ± 0.8	62.6 ± 29.9	62.4 ± 3.2	-	-	-	108.6 ± 0.4

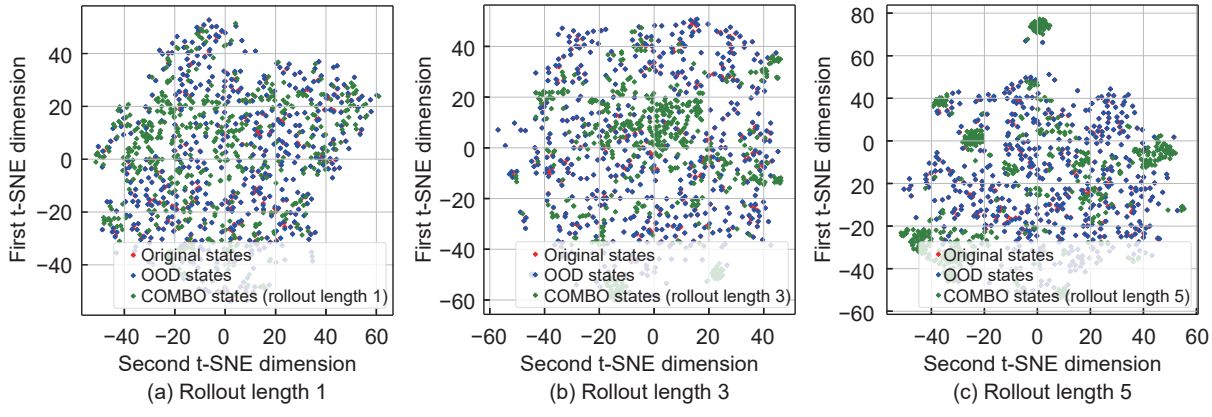

Fig. A1 Visualization of different generated states on halfcheetah-medium-replay-v2.

multi-channel sequential advertising, in *Proc. 37th Int. Conf. Machine Learning* (*ICML*), virtual, 2020, pp. 4060–4070.

[3] 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*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] M. Zhou, J. Luo, J. Villella, Y. Yang, D. Rusu, J. Miao, W. Zhang, M. Alban, I. Fadakar, Z. Chen, et al., Smarts: An open-source scalable multi-agent RL training school for autonomous driving, in *Proc. 4th Conf. Robot Learning* (*CoRL*), Cambridge, MA, USA, 2020, pp. 264–285.

[5] L. Wang, W. Zhang, X. He, and H. Zha, Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation, in *Proc. 24th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, London, UK, 2018, pp. 2447–2456.

[6] S. Levine, A. Kumar, G. Tucker, and J. Fu, Offline reinforcement learning: Tutorial, review, and perspectives on open problems, arXiv preprint arXiv: 2005.01643, 2020.

[7] S. Fujimoto, D. Meger, and D. Precup, Off-policy deep reinforcement learning without exploration, in *Proc. 36th Int. Conf. Machine Learning* (*ICML*), Long Beach, CA, USA, 2019, pp. 2052–2062.

[8] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, Stabilizing off-policy Q-learning via bootstrapping error reduction, in *Proc. 33rd Conf. Nerual Information Processing Systems* (*NeurIPS 2019*), Vancouver, Canada, 2019, pp. 11761–11771.

[9] Y. Wu, G. Tucker, and O. Nachum, Behavior regularized offline reinforcement learning, arXiv preprint arXiv: 1911.11361, 2019.

[10] N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, and M. Riedmiller, Keep doing what worked: Behavioral modelling priors for offline reinforcement learning, arXiv preprint arXiv: 2002.08396, 2020.

[11] A. Kumar, A. Zhou, G. Tucker, and S. Levine, Conservative Q-learning for offline reinforcement learning, in *Proc. 34th Conf. Neural Information Processing Systems* (*NeurIPS 2020*), virtual, 2020.

[12] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, COMBO: Conservative offline model-based policy

optimization, arXiv preprint arXiv: 2102.08363, 2021.
[13] J. Li, C. Tang, M. Tomizuka, and W. Zhan, Dealing with the unknown: Pessimistic offline reinforcement learning, in *Proc. 5th Conf. Robot Learning* (*CoRL*), London, UK, 2021, pp. 1455–1464.
[14] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, D4RL: Datasets for deep data-driven reinforcement learning, arXiv preprint arXiv: 2004.07219, 2020.
[15] Y. Wu, S. Zhai, N. Srivastava, J. M. Susskind, J. Zhang, R. Salakhutdinov, and H. Goh, Uncertainty weighted actor-critic for offline reinforcement learning, in *Proc. 38th Int. Conf. Machine Learning* (*ICML*), virtual, 2021, pp. 11319–11328.
[16] G. An, S. Moon, J. H. Kim, and H. O. Song, Uncertainty-based offline reinforcement learning with diversified Q-ensemble, in *Proc. 35th Conf. Neural Information Processing Systems* (*NeurIPS 2021*), virtual, 2021.
[17] S. Vernekar, A. Gaurav, V. Abdelzad, T. Denouden, R. Salay, and K. Czarnecki, Out-of-distribution detection in classifiers via generation, arXiv preprint arXiv: 1910.04241, 2019.
[18] S. Fujimoto and S. S. Gu, A minimalist approach to offline reinforcement learning, in *Proc. 35th Conf. Neural Information Processing Systems* (*NeurIPS 2021*), virtual, 2021, pp. 20132–20145.
[19] S. Verma, J. Fu, S. Yang, and S. Levine, CHAI: A CHatbot AI for task-oriented dialogue with offline reinforcement learning, in *Proc. 2022 Conf. North American Chapter Association for Computational Linguistics: Human Language Technologies*, Seattle, WA, USA, 2022, pp. 4471–4491.
[20] N. Jaques, A. Ghandeharioun, J. Shen, C. Ferguson, À. Lapedriza, N. J. Jones, S. Gu, and R. W. Picard, Way off-policy batch deep reinforcement learning of implicit human preferences in dialog, arXiv preprint arXiv: 1907.00456, 2019.
[21] W. Zhou, S. Bajracharya, and D. Held, PLAS: Latent action space for offline reinforcement learning, arXiv preprint arXiv: 2011.07213, 2020.
[22] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, arXiv preprint arXiv: 1910.00177, 2019.
[23] R. S. Sutton, A. R. Mahmood, and M. White, An emphatic approach to the problem of off-policy temporal-difference learning, arXiv preprint arXiv: 1503.04269, 2015.
[24] O. Nachum, B. Dai, I. Kostrikov, Y. Chow, L. Li, and D. Schuurmans, AlgaeDICE: Policy gradient from arbitrary experience, arXiv preprint arXiv: 1912.02074, 2019.
[25] R. Agarwal, D. Schuurmans, and M. Norouzi, An optimistic perspective on offline reinforcement learning, in *Proc. 37th Int. Conf. Machine Learning* (*ICML*), virtual, 2020, pp. 104–114.
[26] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims, MOReL: Model-based offline reinforcement learning, in *Proc. 34th Conf. Neural Information Processing Systems* (*NeurIPS 2020*), virtual, 2020.
[27] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma, MOPO: Model-based offline policy optimization, in *Proc. 34th Conf. Neural Information Processing Systems* (*NeurIPS 2020*), virtual, 2020.
[28] T. Matsushima, H. Furuta, Y. Matsuo, O. Nachum, and S. Gu, Deployment-efficient reinforcement learning via model-based offline optimization, in *Proc. 9th Int. Conf. Learning Representations* (*ICLR*), virtual, 2021.
[29] A. Argenson and G. Dulac-Arnold, Model based offline planning, in *Proc. 9th Int. Conf. Learning Representations* (*ICLR*), virtual, 2021.
[30] J. Buckman, C. Gelada, and M. G. Bellemare, The importance of pessimism in fixed-dataset policyoptimization, in *Proc. 9th Int. Conf. Learning Representations* (*ICLR*), virtual, 2021.
[31] T. Seno and M. Imai, d3rlpy: An offline deep reinforcement library, in *Proc. Offline Reinforcement Learning Workshop at Neural Information Processing Systems*, virtual, 2021.
[32] Z. Wang, A. Novikov, K. Zolna, J. Merel, J. T. Springenberg, S. E. Reed, B. Shahriari, N. Y. Siegel, Ç. Gülçehre, N. Heess, et al., *Critic regularized regression*, in *Proc. 34th Conf. Neural Information Processing Systems* (*NeurIPS 2020*), virtual, 2020.
[33] I. Kostrikov, A. Nair, and S. Levine, Offline reinforcement learning with implicit Q-learning, in *Proc. Offline Reinforcement Learning Workshop at Neural Information Processing Systems*, virtual, 2021.
[34] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, Decision transformer: Reinforcement learning via sequence modeling, in *Proc. 35th Conf. Neural Information Processing Systems* (*NeurIPS 2021*), virtual, 2021, pp. 15084–15097.
[35] M. Janner, Q. Li, and S. Levine, Offline reinforcement learning as one big sequence modeling problem, in *Proc. 35th Conf. Neural Information Processing Systems* (*NeurIPS 2021*), virtual, 2021, pp. 1273–1286.
[36] C. Bai, L. Wang, Z. Yang, Z. Deng, A. Garg, P. Liu, and Z. Wang, Pessimistic bootstrapping for uncertainty-driven offline reinforcement learning, presented at the 10th Int. Conf. Learning Representations (ICLR), virtual, 2022.