Supported Policy Optimization for Offline Reinforcement Learning

Jialong Wu[1], **Haixu Wu**[1], **Zihan Qiu**[2], **Jianmin Wang**[1], **Mingsheng Long**[1]✉
[1]School of Software, BNRist, Tsinghua University, China
[2]Institute for Interdisciplinary Information Sciences, Tsinghua University, China
{wujialong0229,qzh11628}@gmail.com, whx20@mails.tsinghua.edu.cn
{jimwang,mingsheng}@tsinghua.edu.cn

Abstract

Policy constraint methods to offline reinforcement learning (RL) typically utilize parameterization or regularization that constrains the policy to perform actions within the support set of the behavior policy. The elaborative designs of parameterization methods usually intrude into the policy networks, which may bring extra inference cost and cannot take full advantage of well-established online methods. Regularization methods reduce the divergence between the learned policy and the behavior policy, which may mismatch the inherent density-based definition of support set thereby failing to avoid the out-of-distribution actions effectively. This paper presents **S**upported **P**olicy **O**p**T**imization (SPOT), which is directly derived from the theoretical formalization of the density-based support constraint. SPOT adopts a VAE-based density estimator to explicitly model the support set of behavior policy and presents a simple but effective density-based regularization term, which can be plugged non-intrusively into off-the-shelf off-policy RL algorithms. SPOT achieves the state-of-the-art performance on standard benchmarks for offline RL. Benefiting from the pluggable design, offline pretrained models from SPOT can also be applied to perform online fine-tuning seamlessly.

1 Introduction

Offline RL [30, 31], where the agent learns from a fixed dataset, collected by arbitrary process, not only provides a bridge between RL and the data-driven paradigm but also eliminates the need to interact with the live environment, which is always expensive or risky in practical scenarios [17, 33, 19]. Unfortunately, the absence of environment interaction also raises a number of challenges. Previous work has shown that the extrapolation error of the Q-function queried by out-of-distribution actions significantly degrades the performance of off-policy algorithms [10].

Avoiding out-of-distribution actions, namely to constrain the learned policy to perform actions within the support set of the behavior policy, is essential to mitigate extrapolation error. To meet this support constraint, policy constraint methods [31] to offline RL utilize either *parameterization* [10, 52, 12] or *regularization* [16, 26, 49] techniques. However, there are still several drawbacks in previous methods of policy constraint, limiting their performance and applications. Firstly, parameterization methods involve elaborate designs of the policy parameterization, typically coupled to generative models of the behavior policy, to *directly* constrain actions taken by the learned policy. But these designs intrude into the architecture of policy networks, which may bring extra inference cost and supplementary difficulties to implement and tune offline RL algorithms. Furthermore, as criticized by Fujimoto and Gu [8], these *intrusive* designs complicate causal attribution of performance gains and transfer of techniques between offline RL algorithms or from well-established online RL algorithms. In contrast, for the second category, regularization methods are designed with a non-intrusive or *pluggable*

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

manner, which is done by simply augmenting a penalty to the actor loss that measures the divergence of the learned policy from the behavior policy [16, 26, 49, 24]. However, these divergence-based regularization methods may mismatch the inherent density-based definition of support set, which applies support constraint *indirectly*. As we show in experiments (Section 5.1), the divergence-based design prevents regularization methods from effectively avoiding out-of-distribution actions and thus limits their performance.

In this work, we aim to design a *pluggable* offline RL method that also *directly* meets the standard formalization of the support constraint based on the density of the behavior policy. We introduce **S**upported **P**olicy **O**p**T**imization (SPOT), a regularization method which can be plugged non-intrusively into off-the-shelf off-policy RL algorithms. SPOT involves a regularization term from the new perspective of explicit estimation of the behavior density. Concretely, SPOT adopts a VAE-based density estimator [21, 39] to explicitly model the support set of behavior policy and presents a simple yet effective regularization term directly applied to the estimated density.

Our method benefits from a closer connection between theory and algorithm, thereby achieving superior performance compared to previous methods. It also profits from the pluggable algorithmic design, which leads to efficient inference and minor algorithmic modifications on top of standard off-policy algorithms. Moreover, a minimal gap between offline learning objectives and standard online learning objectives also enables SPOT to take full advantage of existing online RL algorithms and attain strong online fine-tuning performance after offline RL, exceeding state-of-the-art methods.

The main contributions of this work are three-fold:

- We derive a regularization term for constrained policy optimization in offline RL, based on the theoretical formalization of the density-based support constraint, which directly regularizes the behavior density of actions taken by the learned policy.
- We propose Supported Policy OpTimization (SPOT), a practical algorithm with a neural VAE-based density estimator to implement the proposed regularization term.
- Compared to several strong baselines, SPOT achieves state-of-the-art results on standard offline RL benchmarks [6] and also outperforms previous methods when online fine-tuned after offline RL initialization.

2 Related Work

Our work belongs to the family of policy constraint methods in offline RL, where parameterization or regularization is typically utilized. Comparison between previous policy constraint methods and ours has been summarized in Table 1. In addition to the policy constraint methods, we also review behavior policy modeling in offline RL, which is commonly necessary to both parameterization and regularization methods. Lastly, we briefly discuss a broader range of competitive offline RL approaches.

Table 1: Comparison among policy constraint methods.

Method	Support Constraint	Implementation
BCQ [10] PLAS [52] EMaQ [12]	Explicit Density Constrained	Intrusive Parameterization
BEAR [26] BRAC-p [49] TD3+BC [8]	Implicit Density Constrained	Pluggable Regularization
SPOT (Ours)	Explicit Density Constrained	Pluggable Regularization

Policy constraint via parameterization. Careful parameterization of the learned policy can naturally satisfy the support constraint. For example, BCQ [10] learns a generative model of the behavior policy and trains the actor to perturb randomly generated actions. The policy parameterized by BCQ is to greedily select the one maximizing Q function among a large number of perturbed sampled actions. EMaQ [12] simplifies BCQ by discarding the perturbation models. PLAS [52] learns a policy in the latent space of the generative model and parameterizes the policy using the decoder of the generative model on top of the latent policy.

Policy constraint via regularization. An alternative approach is to use a divergence penalty in order to compel the learned policy to stay close to the behavior policy, such as KL-divergence [16, 49], maximum mean discrepancy (MMD) [26], Fisher divergence [24] and Wasserstein distance [49]. Using divergence penalties alleviates the hard constraint of explicit parameterization but may bear the risk of out-of-distribution actions. Although BEAR [26] attempts to constrain the policy to the

support of the behavior policy, it heavily relies on the empirically-found approximate property of low-sampled MMD. TD3+BC [8] simply adds a behavior cloning (BC) term to the policy update and presents competitive performance on simple locomotion tasks. Note that recent SBAC [50] proposes a new policy learning objective based on performance difference lemma [18], along with a density-based regularization term similar to ours, but our work simply plugs the term into the standard policy training objective to enjoy minimal algorithmic modifications.

Behavior policy modeling in offline RL. Most policy constraint methods need to fit an accurate generative model of the behavior policy, to sample in-distribution actions or estimate behavior density. Conditional variational auto-encoders (CVAE) [21, 42] are typically used by past works [10, 52] to sample actions, while EMaQ [12] opts for using an autoregressive model [11] which enables more accurate sampling. On the other hand, policy class of Gaussian [26, 49] or Gaussian mixture models [24] are commonly used to fit and estimate the density of behavior policy. Instead of explicitly fitting the behavior policy, recent approaches have utilized implicit constraint without ever querying the values of any out-of-sample actions [51, 25]. Although MBS-QL [32] uses ELBO to estimate marginal state distribution, to the best of our knowledge, we are unique in estimating the density of behavior policy based on VAE, for its flexibility to capture almost arbitrary class of distributions [22].

Broader range of offline RL approaches. Besides policy constraint methods based on parameterization or regularization, there exist more types of competitive offline RL approaches. IQL [25] designs a multi-step dynamic programming procedure based on expectile regression, which completely avoids any queries to values of out-of-sample actions. Pessimistic value methods, such as CQL [28], produce a lower bound on the value of the current policy to effectively alleviate overestimation, but their performance may suffer from excessive pessimism. Advantage-weighted regression [37, 35, 47] improves upon behavior policy, while simultaneously enforcing an implicit KL-divergence constraint. Recent sequence modeling methods based on Transformers [45] also show competitive performance in both model-free [5] or model-based [15] paradigm.

3 Background

The reinforcement learning problem [43] is formulated as decision making in the environment represented by a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \rho_0, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\rho_0(s_0)$ is the initial state distribution, $p(s'|s,a)$ is the transition distribution, $r(s,a)$ is the reward function, and γ is the discount factor. The goal in RL is to find a policy $\pi(a|s)$ maximizing the expected return: $\mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$.

The optimal state-action value function or Q function $Q^*(s,a)$ measures the expected return starting in state s taking action a and then acting optimally thereafter. A corresponding optimal policy can be obtained through greedy action choices $\pi^*(s) = \arg\max_a Q^*(s,a)$. The Q-learning algorithm [48] learns Q^* via iterating the Bellman optimality operator $\mathcal{T}$, defined as: $\mathcal{T}Q\left(s,a\right) = \mathbb{E}_{s'}\left[r(s,a) + \gamma \max_{a'} Q\left(s',a'\right)\right]$. For large or continuous state space, the value can be represented by function approximators $Q_\theta(s,a)$ with parameters θ. In practice, the parameters θ are updated by minimizing the mean squared Bellman error with an experience replay dataset $\mathcal{D}$ and a target function $Q_{\bar{\theta}}$ [34]: $J_Q(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[Q_\theta(s,a) - r - \gamma \max_{a'} Q_{\bar{\theta}}\left(s',a'\right)\right]^2$.

In a continuous action space, the analytic maximum is intractable. Actor-Critic methods [43, 9, 13] perform action selection with a separate policy function π_ϕ maximizing the value function:

$$J_Q(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[Q_\theta(s,a) - r - \gamma Q_{\bar{\theta}}\left(s', \pi_\phi(s')\right)\right]^2. \tag{1}$$

The policy can be updated following the deterministic policy gradient (DPG) theorem [41]:

$$J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[-Q_\theta\left(s, \pi_\phi(s)\right)\right]. \tag{2}$$

3.1 Offline Reinforcement Learning

In contrast to online RL methods, which interact with environment to collect experience data, offline RL [30, 31] methods learn from a finite and fixed dataset $\mathcal{D} = \{(s,a,r,s')\}$ which has been collected by some unknown behavior policy π_β. Direct application of off-policy methods on offline setting suffers from *extrapolation error* [10, 26], which means that an *out-of-distribution* action a in state s can produce erroneously overestimated values $Q_\theta(s,a)$.

3.2 Support Constraint in Offline RL

To avoid *out-of-distribution* actions from function approximation, support constraint [10, 26, 12] is commonly used, which means in state s to only allow action that has ϵ-support under behavior policy: $\{a : \pi_\beta(a|s) > \epsilon\}$. Kumar *et al.* [26] first introduce the *distribution-constrained operator*, which is instantiated to the *supported operator* in this work:

Definition 3.1. Given behavior policy π_β and threshold ϵ, the *supported backup operator* is defined as

$$\mathcal{T}_\epsilon Q(s,a) = \mathbb{E}_{s'}\left[r(s,a) + \gamma \max_{a':\pi_\beta(a'|s')>\epsilon} Q(s',a')\right] \tag{3}$$

with its fixed point $Q^*_\epsilon(s,a)$ named as the *supported optimal Q function*.

Following the theoretical analysis of Kumar *et al.* [26], we can obtain a bound of how suboptimal Q^*_ϵ may be with respect to the optimal Q function Q^*:

Corollary 3.2. *Let* $\alpha(\epsilon) = \|\mathcal{T}Q^* - \mathcal{T}_\epsilon Q^*\|_\infty = \max_{s,a} |\mathcal{T}Q^*(s,a) - \mathcal{T}_\epsilon Q^*(s,a)|$. *The suboptimality of* $Q^*_\epsilon(s,a)$ *can be upper-bounded as* $\|Q^* - Q^*_\epsilon\|_\infty \leq \frac{1}{1-\gamma}\alpha(\epsilon)$.

Note that the complete theoretical results from Kumar *et al.* [26] also includes a term w.r.t. the bootstrapping error. We refer readers to Kumar *et al.* [26] for more details. Similar results are also presented by Ghasemipour *et al.* [12].

4 Supported Policy Optimization

As aforementioned, performing support constraint is the typical method to mitigate extrapolation error in offline RL. Noticing that the support constraint can be formalized based on the density of behavior policy, we propose the **S**upported **P**olicy **O**p**T**imization (SPOT) as a regularization method from the new perspective of explicit density estimation. Concretely, SPOT involves a new regularization term, which is directly derived from the theoretical formalization of the support constraint. Besides, a conditional VAE is adopted to explicitly estimate the behavior density in the regularization term. Plugged into off-policy RL algorithms, we will finally arrive at the practical algorithm of SPOT.

4.1 Support Constraint via Behavior Density

Similar to how the optimal policy can be extracted from the optimal Q function, the *supported optimal policy* can also be recovered via greedy selection: $\pi^*_\epsilon(s) = \arg\max_{a:\pi_\beta(a|s)>\epsilon} Q^*_\epsilon(s,a)$. For the case of function approximation, it corresponds to a constrained policy optimization problem.

While prior works use specific parameterization of π [10, 12] or divergence penalty [26, 49] to perform support constraint, we propose to directly use behavior density $\pi_\beta(\cdot|s)$ as constraint[1]:

$$\begin{aligned} &\max_\phi \ \mathbb{E}_{s\sim\mathcal{D}}\left[Q_\theta(s,\pi_\phi(s))\right] \\ &\text{s.t.} \ \min_s \ \log \pi_\beta(\pi_\phi(s)|s) > \hat{\epsilon}, \end{aligned} \tag{4}$$

where $\hat{\epsilon} = \log\epsilon$ for notational simplicity. Constraint via behavior density is simple and straightforward in the context of support constraint. We adopt log-likelihood instead of raw likelihood because of its mathematical convenience.

This problem imposes a constraint that the density of behavior policy is lower-bounded at every point in the state space, which is impractical to solve due to the large even infinite number of constraints. Following previous works from both online RL [40] and offline RL [26, 37] w.r.t. constrained policy optimization, we instead use a heuristic approximation that considers the average behavior density:

$$\begin{aligned} &\max_\phi \ \mathbb{E}_{s\sim\mathcal{D}}\left[Q_\theta(s,\pi_\phi(s))\right] \\ &\text{s.t.} \ \mathbb{E}_{s\sim\mathcal{D}}\left[\log \pi_\beta(\pi_\phi(s)|s)\right] > \hat{\epsilon}. \end{aligned} \tag{5}$$

[1] With slight abuse of notation, here $\pi_\beta(\cdot|s)$ stands for an action distribution of the stochastic policy, while $\pi_\phi(s)$ stands for a deterministic action taken by the policy.

Converting the constrained optimization problem into an unconstrained one by treating the constraint term as a penalty, we finally get the policy learning objective as

$$J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[-Q_\theta\left(s, \pi_\phi(s)\right) - \lambda \log \pi_\beta\left(\pi_\phi(s)|s\right)\right], \tag{6}$$

where λ is a Lagrangian multiplier.

4.2 Explicit Estimation of Behavior Density

The straightforward regularization term in Eq. (6) requires access to π_β. While we only have offline data generated by π_β, we can explicitly estimate the probability density at an arbitrary point with the density estimation methods [1].

The variational autoencoder (VAE) [21] is among the best performing neural density-estimation models and we opt to use a conditional variational autoencoder [42] as our density estimator. Typically, $\pi_\beta(a|s)$ can be approximated by a Deep Latent Variable Model $p_\psi(a|s) = \int p_\psi(a|z,s)p(z|s)\mathrm{d}z$ with a fixed prior $p(z|s) = \mathcal{N}(\mathbf{0}, I)$. While the marginal likelihood $p_\psi(a|s)$ is intractable, VAE additionally uses an approximate posterior $q_\varphi(z|a,s) \approx p_\psi(z|a,s)$ and parameters ψ and φ can be optimized jointly with evidence lower bound (ELBO):

$$\begin{aligned}
\log p_\psi(a|s) &\geq \mathbb{E}_{q_\varphi(z|a,s)}\left[\log \frac{p_\psi(a,z|s)}{q_\varphi(z|a,s)}\right] \\
&= \mathbb{E}_{q_\varphi(z|a,s)}\left[\log p_\psi(a|z,s)\right] - \mathrm{KL}\left[q_\varphi(z|a,s)\|p(z|s)\right] \\
&\stackrel{\text{def}}{=} -\mathcal{L}_{\mathrm{ELBO}}(s,a;\varphi,\psi).
\end{aligned} \tag{7}$$

After training a VAE, we can simply use $-\mathcal{L}_{\mathrm{ELBO}}$ to lower-bound $\log p_\psi(a|s)$ and thus approximately lower-bound $\log \pi_\beta$ in Eq. (6). However, there theoretically exists a bias between them, as we know $\log p_\psi(a|s) = -\mathcal{L}_{\mathrm{ELBO}} + \mathrm{KL}(q_\varphi(z|a,s)||p_\psi(z|a,s))$. To obtain an estimation with lower bias, we can use the importance sampling technique [39, 22]:

$$\begin{aligned}
\log p_\psi(a|s) &= \log \mathbb{E}_{q_\varphi(z|a,s)}\left[\frac{p_\psi(a,z|s)}{q_\varphi(z|a,s)}\right] \\
&\approx \mathbb{E}_{z^{(l)}\sim q_\varphi(z|a,s)}\left[\log \frac{1}{L}\sum_{l=1}^{L}\frac{p_\psi(a,z^{(l)}|s)}{q_\varphi(z^{(l)}|a,s)}\right] \\
&\stackrel{\text{def}}{=} \widehat{\log \pi_\beta}(a|s;\varphi,\psi,L).
\end{aligned} \tag{8}$$

Burda *et al.* [3] show that $\widehat{\log \pi_\beta}(a|s;\varphi,\psi,L)$ gives a lower bound of $\log p_\psi(a|s)$ and the bound becomes tighter as L increases. Note that here we adopt sampling of VAE to directly estimate the density of the behavior policy instead of to estimate the divergence [26].

In summary, the loss function in Eq. (6) can be implemented with the explicit density estimator as follows:

$$J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[-Q_\theta\left(s,\pi_\phi(s)\right) - \lambda\widehat{\log \pi_\beta}(\pi_\phi(s)|s;\varphi,\psi,L)\right]. \tag{9}$$

4.3 Practical Algorithm

The general framework derived above can be built on top of off-policy algorithms with minimal modifications. We choose TD3 [9] as our base algorithm, which recently shows strong resistance to overestimation in offline RL [8] (see Section 5.2 for a detailed discussion).

Q normalization. Following TD3+BC [8], we add a normalization term to policy loss as a default option for better balance between Q value objective and regularization: $J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\frac{-Q_\theta(s,\pi_\phi(s))}{\alpha} - \lambda\widehat{\log \pi_\beta}(\pi_\phi(s)|s;\varphi,\psi,L)\right]$, where $\alpha = \frac{1}{N}\sum_{s_i}\left|Q\left(s_i,\pi_\phi(s_i)\right)\right|$ is the normalization term based on the minibatch $\{s_i\}_{i=1}^N$ with size N.

Simpler density estimator. While $\widehat{\log \pi_\beta}(a|s;\varphi,\psi,L)$ with large L is much tighter, we empirically find there is no further improvement with larger L compared to $L=1$ (see Figure 7 in Appendix for results of ablation study). To make our algorithm simpler, we choose to only use $L=1$ for

Algorithm 1 Supported Policy Optimization (SPOT)

Input: Dataset $\mathcal{D} = \{(s, a, r, s')\}$
// VAE Training
Initialize VAE with parameters ψ and φ
for $t = 1$ **to** T_1 **do**
  Sample minibatch of transitions $(s, a) \sim \mathcal{D}$
  Update ψ, φ minimizing $\mathcal{L}_{\text{ELBO}}(s, a; \varphi, \psi)$ in Eq. (7)
end for
// Policy Training
Initialize the policy network π_ϕ, critic network Q_θ and target network $Q_{\bar{\theta}}$ with $\bar{\theta} \leftarrow \theta$
for $t = 1$ **to** T_2 **do**
  Sample minibatch of transitions $(s, a, r, s') \sim \mathcal{D}$
  Update θ minimizing $J_Q(\theta)$ in Eq. (1)
  Update ϕ minimizing $J_\pi(\phi)$ in Eq. (9)
  Update target network: $\bar{\theta} \leftarrow \tau\theta + (1 - \tau)\bar{\theta}$
end for

a practical estimator, which is just the ELBO estimator of the VAE: $\widehat{\log \pi_\beta}(a|s; \varphi, \psi, L = 1) = -\mathcal{L}_{\text{ELBO}}(s, a; \varphi, \psi)$. Note that with $L = 1$, we can analytically separate out the KL divergence as Eq. (7) to enjoy a simpler and lower-variance update.

Overall algorithm. Putting everything together, the full algorithm is summarized in Algorithm 1. Our algorithm first trains VAE using $\mathcal{L}_{\text{ELBO}}(s, a; \varphi, \psi)$ to obtain a density estimator with sufficient accuracy. Then it turns to policy training analogous to common Actor-Critic methods except that we plug the regularization term computed by the density estimator into the policy loss $J_\pi(\phi)$.

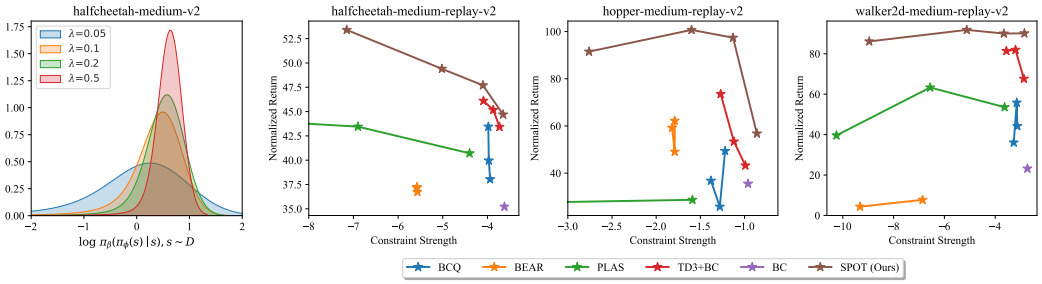
5 Experimental Evaluation

Our experiments aim to evaluate our method comparatively, in contrast to prior offline RL methods, focusing on both offline training and online fine-tuning. We first demonstrate the effect of λ on applying support constraint and show that our method is able to learn a policy with the strongest performance at the same level of constraint strength, compared to previous policy constraint methods. We then evaluate SPOT on D4RL benchmark [6], studying how effective our method is in contrast to a broader range of state-of-the-art offline RL methods. Finally, we study how SPOT compares to prior methods when fine-tuning with online RL from an offline RL initialization, and investigate the computational efficiency of different methods. Code is available at `https://github.com/thuml/SPOT`.

5.1 Analysis of Support Constraint in SPOT

Effect of λ on constraint strength. The coefficient λ in SPOT is essential and corresponds to a specific constraint strength in the constrained policy optimization problem formalized in Eq. (4). To illustrate how λ effects the learned policy, we evaluate behavior density of actions taken by the policy learned with varying values of $\lambda \in [0.05, 0.1, 0.2, 0.5]$ on standard D4RL [6] Gym-MuJoCo domains. Concretely, we plot the distribution of behavior density $\log \pi_\beta(\pi_\phi(s)|s), s \sim \mathcal{D}$ in Figure 1a, where $\log \pi_\beta$ is estimated by our learned density estimator (Eq. (8)) with L set to a sufficient large number 500 for more accurate estimation. As we show, with smaller λ, the learned policy is much more possible to perform actions with low behavior density $\log \pi_\beta(\pi_\phi(s)|s)$. On the other hand, policies learned by higher λ are restricted to take only high-density actions.

Tradeoff between constraint strength and optimality. It has been shown by Kumar *et al.* [26] that the optimality of approximate supported optimal policy is lower-bounded by a tradeoff between keeping the learned policy supported by the behavior policy (controlling extrapolation error) and keeping the supported policy set large enough to capture well-performing policies. If the constraint in Eq. (4) is strong (by a large $\log \epsilon$), the extrapolation error is restrained to be small but the optimal



(a) Effect of λ.

(b) Tradeoff between constraint strength and optimality.

Figure 1: Analysis on constraint strength: (a) With varying values of the coefficient λ, SPOT applies support constraint with different strength, which is demonstrated by the behavior density of actions taken by the learned policy: $\log \pi_\beta(\pi_\phi(s)|s), s \sim \mathcal{D}$. (b) When evaluating the performance at varying levels of constraint strength, SPOT takes shape of the "upper envelope" of all methods, showing that SPOT can always achieve the strongest performance among extensive policy constraint methods. Constraint strength is captured approximately by the 5th-percentile of the distribution $\log \pi_\beta(\pi_\phi(s)|s), s \sim \mathcal{D}$. Extended results can be found in Appendix C.4.

policy under constraint may have poor performance. Otherwise, if the constraint is weak, well-performing policies can be learned though at the risk of the extrapolation error.

We aim to answer the question that *at the same level of constraint being satisfied, is SPOT able to learn a policy with the strongest performance compared to previous policy constraint methods?* We compare SPOT with BC [38], BCQ [10], BEAR [26], PLAS [52] and TD3+BC [8]. Hyperparameters to control constraint strength of various methods are adjusted to several values to form a spectrum of constraint strength (see Appendix C.4 for details). We approximate the satisfied constraint strength $\log \epsilon$ by the 5th-percentile of the distribution $\log \pi_\beta(\pi_\phi(s)|s), s \sim \mathcal{D}$. As shown in Figure 1b, our method SPOT takes shape of the "upper envelope" of all methods, demonstrating that taking advantage of exact standard formulation of support constraint, SPOT is flexible to learn the supported optimal policy and resist extrapolation error at the same time. Note that the divergence-based regularization method BEAR [26] yields a poor performance in our experiments and only satisfies a loose constraint in contrast to other baselines, showing that it cannot prevent out-of-distribution actions effectively and suffers from extrapolation error with indirect divergence regularization.

5.2 Comparisons on Offline RL Benchmarks

Next, we evaluate our approach on the D4RL benchmark [6] in comparison to state-of-the-art methods. We focus on Gym-MuJoCo locomotion domains and much more challenging AntMaze domains, which consists of sparse-reward tasks and requires "stitching" fragments of suboptimal trajectories traveling undirectedly in order to find a path from the start to the goal of the maze.

Baselines. We select the classic BC [38] and state-of-the-art offline RL methods as baselines. For methods based on dynamic programming, we compare to AWAC [35], Onestep RL [2], TD3+BC [8], CQL [28], and IQL [25]. We also include the sequence-modeling method Decision Transformer [5].

Hyperparameter tuning. The weight λ in Eq. (9) is essential for policy constraint. Following prior works [2, 26, 49], we allow access to the online environment to tune a small set of the hyperparameter λ (< 10 choices) , which is a reasonable setup for practical applications. See Appendix C.1 for additional discussion and details.

Gym-MuJoCo domains. Results for the Gym-MuJoCo domains are shown in Table 2. As we show, SPOT substantially outperforms state-of-the-art methods, especially in suboptimal "medium" and "medium-replay" datasets with a large margin, which further demonstrates the advantages of direct constraint on behavior density proposed by SPOT.

AntMaze domains. Results for the AntMaze domains are shown in Table 3. Note that D4RL recently releases a bug-fixed "-v2" version of AntMaze datasets, and thus we select competitive baselines and rerun their author-provided implementations for comparison. We also include results for BCQ and

Table 2: Performance of SPOT and prior methods on Gym-MuJoCo tasks. m = medium, m-r = medium-replay, m-e = medium-expert. For baselines, we report numbers directly from the IQL paper [25], which provides a unified comparison for "-v2" datasets. For SPOT, we report the mean and standard deviation for 10 seeds.

	BC	AWAC	DT	Onestep	TD3+BC	CQL	IQL	SPOT (Ours)
HalfCheetah-m-e-v2	55.2	42.8	86.8	**93.4**	90.7	91.6	86.7	86.9±4.3
Hopper-m-e-v2	52.5	55.8	**107.6**	103.3	98.0	105.4	91.5	99.3±7.1
Walker-m-e-v2	107.5	74.5	108.1	**113.0**	110.1	108.8	109.6	112.0±0.5
HalfCheetah-m-v2	42.6	43.5	42.6	48.4	48.3	44.0	47.4	**58.4**±1.0
Hopper-m-v2	52.9	57.0	67.6	59.6	59.3	58.5	66.2	**86.0**±8.7
Walker-m-v2	75.3	72.4	74.0	81.8	83.7	72.5	78.3	**86.4**±2.7
HalfCheetah-m-r-v2	36.6	40.5	36.6	38.1	44.6	45.5	44.2	**52.2**±1.2
Hopper-m-r-v2	18.1	37.2	82.7	97.5	60.9	95.0	94.7	**100.2**±1.9
Walker-m-r-v2	26.0	27.0	66.6	49.5	81.8	77.2	73.8	**91.6**±2.8
Gym-MuJoCo total	466.7	450.7	672.6	684.6	677.4	698.5	692.4	**773.0**±30.2

Table 3: Performance of SPOT and prior methods on AntMaze tasks. For baselines, we obtain the results using author-provided implementations on "-v2" datasets. For BCQ and BEAR, we report numbers from D4RL paper [6]. For SPOT, we report the mean and standard deviation for 10 seeds.

	BCQ	BEAR	BC	DT	TD3+BC	PLAS	CQL	IQL	SPOT (Ours)
umaze-v2	78.9	73.0	49.2	54.2±4.1	73.0±34.0	62.0±16.7	82.6±5.7	89.6±4.2	**93.5**±2.4
umaze-diverse-v2	55.0	61.0	41.8	41.2±11.4	47.0±7.3	45.4±7.9	10.2±6.7	**65.6**±8.3	40.7±5.1
medium-play-v2	0.0	0.0	0.4	0.0±0.0	0.0±0.0	31.4±21.5	59.0±1.6	**76.4**±2.7	74.7±4.6
medium-diverse-v2	0.0	8.0	0.2	0.0±0.0	0.2±0.4	20.6±27.7	46.6±24.0	72.8±7.0	**79.1**±5.6
large-play-v2	6.7	0.0	0.0	0.0±0.0	0.0±0.0	2.2±3.8	16.4±17.1	**42.0**±3.8	35.3±8.3
large-diverse-v2	2.2	0.0	0.0	0.0±0.0	0.0±0.0	3.0±6.7	3.2±4.1	**46.0**±4.5	36.3±13.7
AntMaze total	142.8	142.0	91.6	95.4±15.5	120.2±41.7	164.6±84.3	218.0±59.2	**392.4**±30.5	359.6±39.7

BEAR trained on "-v0" datasets, directly from [6]. It is not suitable to compare them with reproduced baselines but we include them in order to highlight that previous policy constraint methods struggle to succeed in training on challenging AntMaze domains.

As shown in Table 3, SPOT performs slightly worse than IQL but outperforms remaining modern offline RL baselines, including the pessimistic value method CQL and the sequence modeling method Decision Transformer. Note that IQL ingeniously designs multi-step dynamic programming and policy extraction steps to apply an implicit constraint for offline RL, but when online fine-tuned after offline RL initialization, IQL is inferior to SPOT. SPOT's pluggable design can take full advantage of existing online RL algorithms (see Section 5.3 and Table 4). To the best of our knowledge, our algorithm is the first to train successfully in challenging AntMaze domains with pluggable modification on top of off-policy RL methods for offline RL.

Ablation. In Figure 2, we perform an ablation study over the components in our method. First, we replace the VAE-based density estimator with a behavioral-cloned Gaussian policy model [26, 49]. As expected, it degrades the performance due to the lack of flexibility to model complex distribution, especially on Gym-MuJoCo medium-expert datasets and AntMaze datasets. Then, we evaluate SPOT without Q normalization and find that the removal provides some benefits as well as damages on different datasets, with only an insignificant impact on total performance. Nevertheless, we include it as a default option following TD3+BC [8].

Lastly, we investigate how the base off-policy algorithm matters. Since SAC [13] has similar off-policy performance with TD3, we attempt to adopt SAC as our base method. As Xu *et al.* [50] argue that the entropy term will do harm to the offline setting, we also implement a variant of SAC without maximum entropy regularization. Surprisingly, both variants are vulnerable to pathological extrapolation error on AntMaze domains and provide poor performance (see Figure 2). We argue that

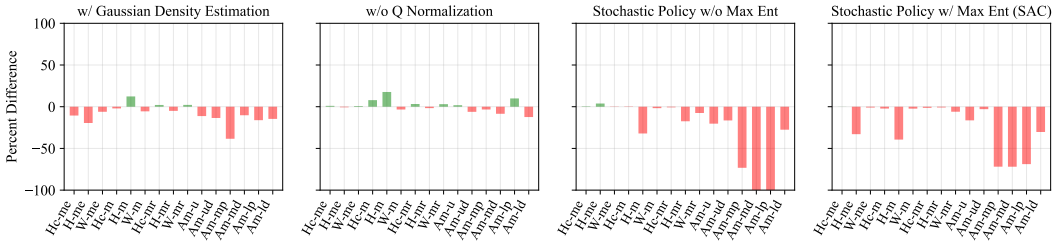


Figure 2: Percent difference of the performance of an ablation of SPOT, compared with the original algorithm. Hc = HalfCheetah, H = Hopper, W = Walker, me = medium-expert, m = medium, mr = medium-replay. Am = AntMaze, u = umaze, m = medium, l = large, d = diverse, p = play. Implementation details and quantitative results can be found in Appendix C.2.

several key native designs of TD3 are beneficial to offline RL. **(1).** For the case of learning a stochastic policy *without* entropy regularization, we find that the learned policy quickly degenerates into a deterministic one (with near-zero standard deviation). A concern with deterministic policies is that they are prone to overfit overestimated actions and propagate the estimation error through Bellman backups, which is even serious in offline RL. Hopefully, TD3 introduces Target Policy Smoothing [9, 44] into Eq. (1), which adds random noise to target actions and can alleviate the effect of error propagation. **(2).** For the case *with* entropy regularization, Bellman backup of a stochastic policy resembles Target Policy Smoothing, then the primary distinction may come from the actor learning objective. While TD3 produces a deterministic action minimizing Eq. (9), stochastic policies are more likely to produce out-of-distribution actions with erroneous estimated values, so the policy gradient may become biased as well as with high variance. On much easier Gym-MuJoCo domains, SAC-style SPOT variants are comparable to the TD3 variant on most of the datasets, but we remark that TD3 indeed extends the limit of policy constraint methods on some tasks, such as hopper-medium. Our analysis suggests that TD3 may be preferable for offline RL as the base off-policy method with native designs (such as "stochastic" critic training and deterministic actor training) addressing function approximation error not only in the online setting but also in the offline setting [8].

5.3 Online Fine-tuning after Offline RL

Pluggable SPOT can be online fine-tuned seamlessly, which means that we only need to gradually decay the constraint strength λ in the online phase in order to avoid excessive conservatism. As AWAC [35] shows that behavior models are hard to update online, we fix the VAE during online fine-tuning. Note that when λ is zero, our algorithm is exactly the standard off-policy RL algorithm that SPOT builds upon. It is beneficial since we enjoy a minimal gap between offline RL and well-established online RL methods and can take full advantage of them for online fine-tuning.

Since IQL [25] is the strongest baseline in our offline experiments, which also has shown superior online performance than prior methods [35, 28] in its paper, and most of the other baselines fail to learn meaningful results, we follow the experiments of IQL and compare to IQL in online fine-tuning. We also compare to our base RL method TD3 [9] trained online from scratch. We use the challenging AntMaze domains [6]. During online fine-tuning of SPOT, the regularization weight λ is linearly decayed to one-fifth of its initial value. See Appendix C.3 for details.

Table 4: Online fine-tuning results on AntMaze tasks, showing initial performance after offline RL and performance after 1M steps of online RL. All numbers are reported by the mean of 5 seeds.

	IQL	SPOT (Ours)
umaze-v2	85.4 → 96.2	**93.2** → **99.2** (+3.0)
umaze-diverse-v2	**70.8** → 62.2	41.6 → **96.0** (+33.8)
medium-play-v2	68.6 → 89.8	**75.2** → **97.4** (+7.6)
medium-diverse-v2	**73.4** → 90.2	73.0 → **96.2** (+6.0)
large-play-v2	40.0 → 78.6	**40.8** → **89.4** (+10.8)
large-diverse-v2	40.4 → 73.4	**44.0** → **90.8** (+17.4)
AntMaze total	**378.6** → 490.4	367.8 → **569.0** (+78.6)

Results are shown in Table 4. While online training from scratch fails in the challenging sparse reward tasks on AntMaze domains, SPOT initialized with offline RL succeeds to learn nearly optimal

policies and performs significantly better than the strongest baseline IQL, especially in the most difficult large maze.

5.4 Computation Cost

Regularization methods, including our SPOT, benefiting from the pluggable design, only need one forward pass of the policy network to do inference, while parameterization methods always need inference through secondary components, such as the generative model or the critic network, which may bring extra time or memory cost. As demonstrated empirically in Figure 3, parameterization methods are usually slower than regularization methods. TD3+BC and our SPOT run more than two times faster compared to the most time-consuming BCQ, while SPOT also has the advantage of parameterization methods, which explicitly constrain the behavior density of learned actions.

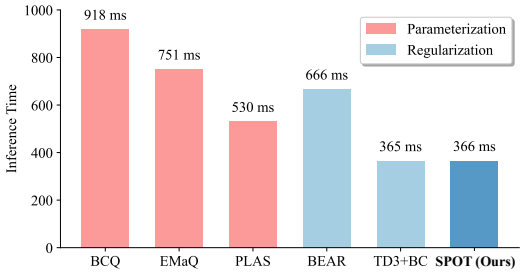


Figure 3: Runtime of various offline RL algorithms interacting with the HalfCheetah environment to produce a 1000-steps trajectory. See Appendix C.5 for the details.

SPOT indeed adds some training overhead due to the VAE-based density estimator, but is still much simpler than most methods. A comparison of training time is provided in Appendix C.5. SPOT lies in the second tier, only slightly worse than PLAS and beaten by the simplest TD3+BC.

6 Conclusion

We present Supported Policy OpTimization (SPOT), a policy constraint method to offline RL built upon off-the-shelf off-policy RL algorithms. SPOT introduces a pluggable regularization term applied directly to the estimated behavior density, which brings a number of important benefits. First, our algorithm is computationally efficient at inference, which only needs one forward process of the policy network for action selection. Second, capturing the standard formulation of the support constraint based on behavior density, it obtains excellent performance across different tasks in the D4RL benchmarks, including standard Gym-MuJoCo tasks and much more challenging AntMaze tasks. Finally, the pluggable design of our algorithm makes it seamless to apply online fine-tuning after offline RL pre-training. Taking full advantage of well-established online methods, SPOT exceeds the state-of-the-art online fine-tuning performance on the challenging AntMaze domains.

Limitations. One limitation of our current method, shared by most policy constraint methods, is that the performance may be limited by the accuracy of estimation of the behavior policy. Advances in generative models, such as diffusion models [14, 46, 4], may improve the real-world performance of offline RL, especially in scenarios with highly multimodal behaviors. An exciting direction for future work would be to develop an effective pluggable constraint mechanism excluding explicit estimation of behavior policy. Another limitation of our work is that we rely on online evaluation to select the best set of hyperparameters. Although this evaluation protocol is commonly adopted by the literature of offline RL, extensive online evaluation is not practical in real-world applications and online evaluation budgets may have a significant impact on final performance [23, 29]. Specifically for SPOT, the selection of regularization weight without requirements for extensive online evaluation is critical and needs to be developed, either by offline policy evaluation [36], by manual tuning based on offline training metrics and conditions [27], or by auto-tuning with a dual optimization [13].

Acknowledgments

We would like to thank many colleagues, in particular, Jincheng Zhong, Haoyu Ma, Yiwen Qiu, and Yuchen Zhang for their valuable discussion and support for this work. This work was supported by the National Key Research and Development Plan (2020AAA0109201), National Natural Science Foundation of China (62022050 and 62021002), Beijing Nova Program (Z201100006820041), and BNRist Innovation Fund (BNR2021RC01002).

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] David Brandfonbrener, William F Whitney, Rajesh Ranganath, and Joan Bruna. Offline rl without off-policy evaluation. In *NeurIPS*, 2021.

[3] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[4] Huayu Chen, Cheng Lu, Chengyang Ying, Hang Su, and Jun Zhu. Offline reinforcement learning via high-fidelity generative behavior modeling. *arXiv preprint arXiv:2209.14548*, 2022.

[5] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *NeurIPS*, 2021.

[6] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[7] Justin Fu, Mohammad Norouzi, Ofir Nachum, George Tucker, Ziyu Wang, Alexander Novikov, Mengjiao Yang, Michael R Zhang, Yutian Chen, Aviral Kumar, et al. Benchmarks for deep off-policy evaluation. In *ICLR*, 2021.

[8] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *NeurIPS*, 2021.

[9] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, 2018.

[10] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.

[11] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked autoencoder for distribution estimation. In *ICML*, 2015.

[12] Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. EMaQ: Expected-max q-learning operator for simple yet effective offline and online rl. In *ICML*, 2021.

[13] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[14] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

[15] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *NeurIPS*, 2021.

[16] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[17] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[18] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

[19] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *CoRL*, 2018.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[21] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[22] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[23] Ksenia Konyushova, Yutian Chen, Thomas Paine, Caglar Gulcehre, Cosmin Paduraru, Daniel J Mankowitz, Misha Denil, and Nando de Freitas. Active offline policy selection. In *NeurIPS*, 2021.

[24] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *ICML*, 2021.

[25] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *ICLR*, 2022.

[26] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *NeurIPS*, 2019.

[27] Aviral Kumar, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine. A workflow for offline model-free robotic reinforcement learning. In *CoRL*, 2021.

[28] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

[29] Vladislav Kurenkov and Sergey Kolesnikov. Showing your offline reinforcement learning work: Online evaluation budget matters. In *ICML*, 2022.

[30] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[31] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[32] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Provably good batch reinforcement learning without great exploration. In *NeurIPS*, 2020.

[33] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[34] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[35] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. AWAC: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[36] Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

[37] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[38] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, Carnegie Melon Univ. Pittsburgh, PA. Artificial Intelligence and Psychology., 1989.

[39] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

[40] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[41] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[42] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *NeurIPS*, 2015.

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

[44] Harm van Seijen, Hado van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of expected sarsa. In *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning: ADPRL*, 2009.

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[46] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[47] Ziyu Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. In *NeurIPS*, 2020.

[48] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[49] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[50] Haoran Xu, Xianyuan Zhan, Jianxiong Li, and Honglei Yin. Offline reinforcement learning with soft behavior regularization. *arXiv preprint arXiv:2110.07395*, 2021.

[51] Yiqin Yang, Xiaoteng Ma, Chenghao Li, Zewu Zheng, Qiyuan Zhang, Gao Huang, Jun Yang, and Qianchuan Zhao. Believe what you see: Implicit constraint approach for offline multi-agent reinforcement learning. In *NeurIPS*, 2021.

[52] Wenxuan Zhou, Sujay Bajracharya, and David Held. PLAS: Latent action space for offline reinforcement learning. In *CoRL*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix D.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code to reproduce the main results of our methods (Table 2, 3, 4) is available at `https://github.com/thuml/SPOT`.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 5 and Appendix C.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.4 and Appendix C.5.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] All the datasets we used are open-sourced and we comply with the license.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] All the data and experiments are simulated without any personally identifiable information.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proofs

A.1 Proof of Corollary 3.2

This proof is adapted from the proof of Theorem 4.1 in Kumar *et al.* [26].

Proof.

$$\begin{aligned}\|Q^* - Q^*_\epsilon\|_\infty &= \|\mathcal{T}Q^* - \mathcal{T}_\epsilon Q^*_\epsilon\|_\infty \\ &\leq \|\mathcal{T}_\epsilon Q^* - \mathcal{T}_\epsilon Q^*_\epsilon\|_\infty + \|\mathcal{T}Q^* - \mathcal{T}_\epsilon Q^*\|_\infty \\ &\leq \gamma \|Q^* - Q^*_\epsilon\|_\infty + \alpha(\epsilon)\end{aligned} \tag{10}$$

Thus we have $\|Q^* - Q^*_\epsilon\|_\infty \leq \frac{1}{1-\gamma}\alpha(\epsilon)$. □

B Missing Background: Parameterization Methods for Policy Constraint

Parameterization methods enforce the learned policy π to be close to the behavior policy π_β with various specific parameterization of π.

BCQ [10] learns a generative model of behavior policy π_β and trains the actor as a perturbation model ξ_ϕ to perturb the randomly generated actions. The policy parameterized by BCQ is to greedily select the one maximizing Q function among a large number N of perturbed sampled actions $a_i + \xi_\phi(s, a_i)$:

$$\pi(a|s) := \underset{a_i + \xi_\phi(s,a_i)}{\arg\max}\ Q_\theta\left(s, a_i + \xi_\phi\left(s, a_i\right)\right) \text{ for } a_i \sim \pi_\beta(a|s), i = 1, \ldots, N. \tag{11}$$

EMaQ [12] simplifies BCQ by discarding the perturbation models:

$$\pi(a|s) := \underset{a_i}{\arg\max}\ Q_\theta\left(s, a_i\right) \text{ for } a_i \sim \pi_\beta(a|s), i = 1, \ldots, N. \tag{12}$$

PLAS [52] learns a policy $z = \pi_\phi(s)$ in the latent space of the generative model and parameterizes the policy using the decoder $D_\beta : z \mapsto a$ of the generative model on top of the latent policy:

$$\pi(a|s) := D_\beta(\pi_\phi(s)). \tag{13}$$

C Implementation Details and Extended Results

C.1 Benchmark Experiments (Table 2 and 3)

Data. We use the datasets from the D4RL benchmark [6], of the latest versions, which are "v2" for both Gym-MuJoCo and AntMaze domains.

Baselines. We report Gym-MuJoCo results of baselines directly from IQL paper [25] and rerun competitive baselines for AntMaze tasks on "v2" datasets, taking their official implementations:

- BC [38]: modified from `https://github.com/sfujim/TD3_BC`
- Decision Transformer [5]: `https://github.com/kzl/decision-transformer`
- TD3+BC [8]: `https://github.com/sfujim/TD3_BC`
- CQL [28]: `https://github.com/aviralkumar2907/CQL`
- IQL [25]: `https://github.com/ikostrikov/implicit_q_learning/`

Implementation details. Our algorithm SPOT consists of two stages, namely VAE training and policy training. We will introduce the details of both stages respectively.

For VAE training, our code is based on the implementation of BCQ: `https://github.com/sfujim/BCQ/tree/master/continuous_BCQ`. Following TD3+BC [8], we normalize the states in the dataset for Gym-MuJoCo domains but do not normalize the states for AntMaze domains. Hyperparameters used by VAE are in Table 5.

For policy training, our code is based on the implementation of TD3+BC [8]. Following IQL [25], we subtract 1 from rewards for the AntMaze datasets. Hyperparameters used by policy training are in Table 6.

Table 5: Hyperparameters of VAE training in SPOT.

	Hyperparameter	Value
VAE training	Optimizer	Adam [20]
	Learning rate	1×10^{-3}
	Batch size	256
	Number of iterations	10^5
	KL term weight	0.5
	Normalized states	*True* for Gym-MuJoCo *False* for AntMaze
VAE architecture	Encoder hidden dim	750
	Encoder layers	3
	Latent dim	$2 \times$ action dim
	Decoder hidden dim	750
	Decoder layers	3

For evaluation, we average mean returns overs 10 evaluation trajectories on the Gym-MuJoCo tasks, and average over 100 evaluation trajectories on the AntMaze tasks.

Table 6: Hyperparameters of policy training in SPOT.

	Hyperparameter	Value
TD3	Optimizer	Adam [20]
	Critic learning rate	3×10^{-4}
	Actor learning rate	3×10^{-4} for Gym-MuJoCo 1×10^{-4} for AntMaze
	Batch size	256
	Discount factor	0.99
	Number of iterations	10^6
	Target update rate τ	0.005
	Policy noise	0.2
	Policy noise clipping	0.5
	Policy update frequency	2
Architecture	Actor hidden dim	256
	Actor layers	3
	Actor dropout	0.1 for Gym-MuJoCo 0.0 for AntMaze
	Critic hidden dim	256
	Critic layers	3
SPOT	λ	$\{0.05, 0.1, 0.2, 0.5, 1.0, 2.0\}$ Gym-MuJoCo $\{0.025, 0.05, 0.1, 0.25, 0.5, 1.0\}$ AntMaze

Hyperparameter tuning. The weight of regularization term λ is essential for SPOT to control different strengths of the constraint. Following prior works [2, 26, 49], we allow access to the environment to tune a small (< 10) set of the hyperparameter λ. The hyperparameter sets for Gym-MuJoCo and AntMaze domains can be seen in Table 6. We tune hyperparameters using 3 seeds but then evaluate the best hyperparameter by training with totally new 10 seeds and then report final results on these additional 10 seeds.

As discussed in Brandfonbrener *et al.* [2], it's a reasonable setup for applications like robotics, where we can test a limited number of trained policies on a real system. Beyond the scope of this work, we believe that to make offline RL more widely applicable, better approaches for offline policy evaluation and selection are indispensable. Fortunately, it has attracted wide attention by the community and we refer the reader to [36, 7]. Note that Paine *et al.* [36] demonstrate that policies learned by policy constraint methods have the advantage of being easier to evaluate and rank offline, since the methods encourage learned policies to stay close to the behavior policy.

Learning curves. Learning curves of best tuned λ for each dataset is presented in Figure 4 and Figure 5. Learning curves of different λ for Gym-MuJoCo domains is presented in Figure 6.

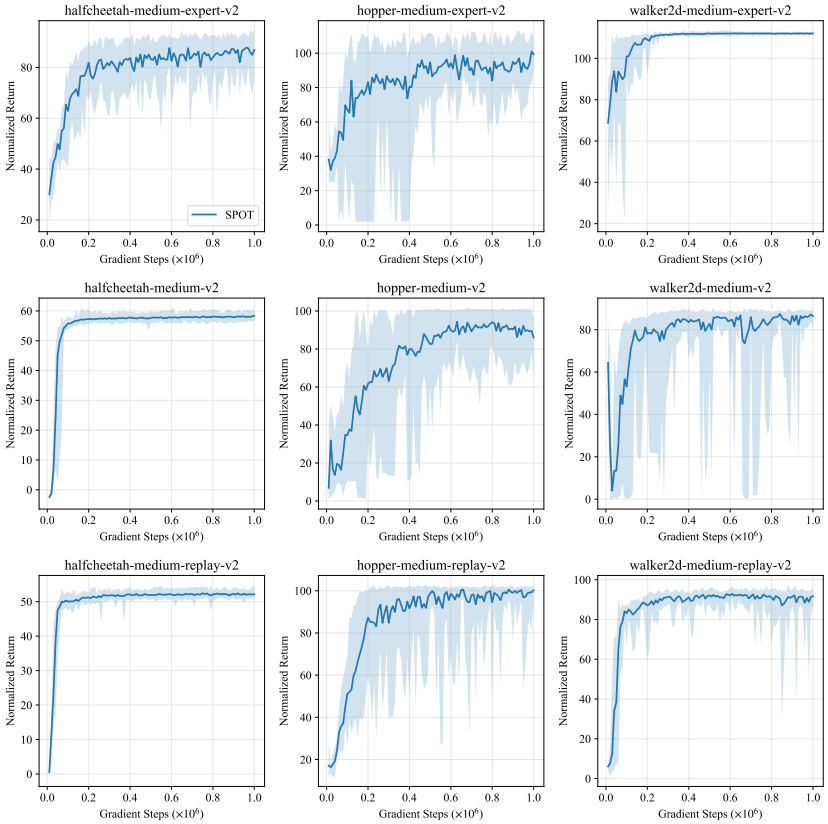


Figure 4: Learning curves of best-tuned λ for Gym-MuJoCo domains. Error bars indicate min and max over 10 seeds.

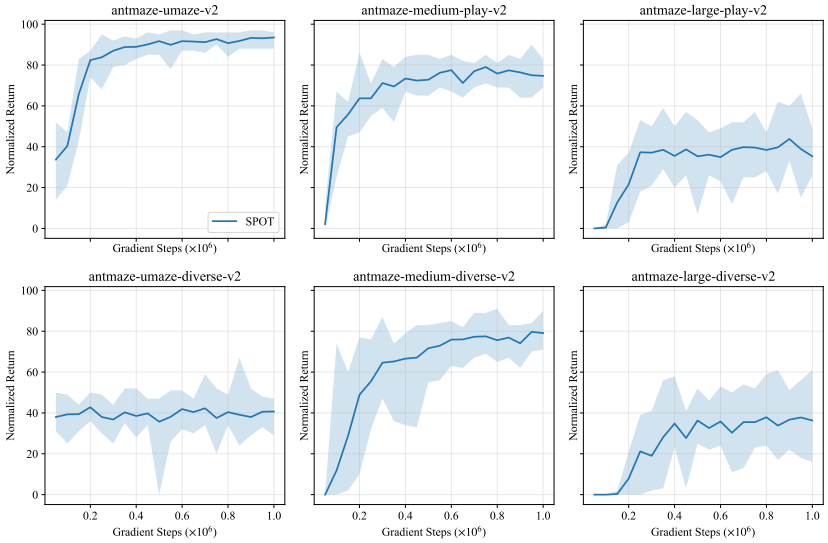


Figure 5: Learning curves of best-tuned λ for AntMaze domains. Error bars indicate min and max over 10 seeds.

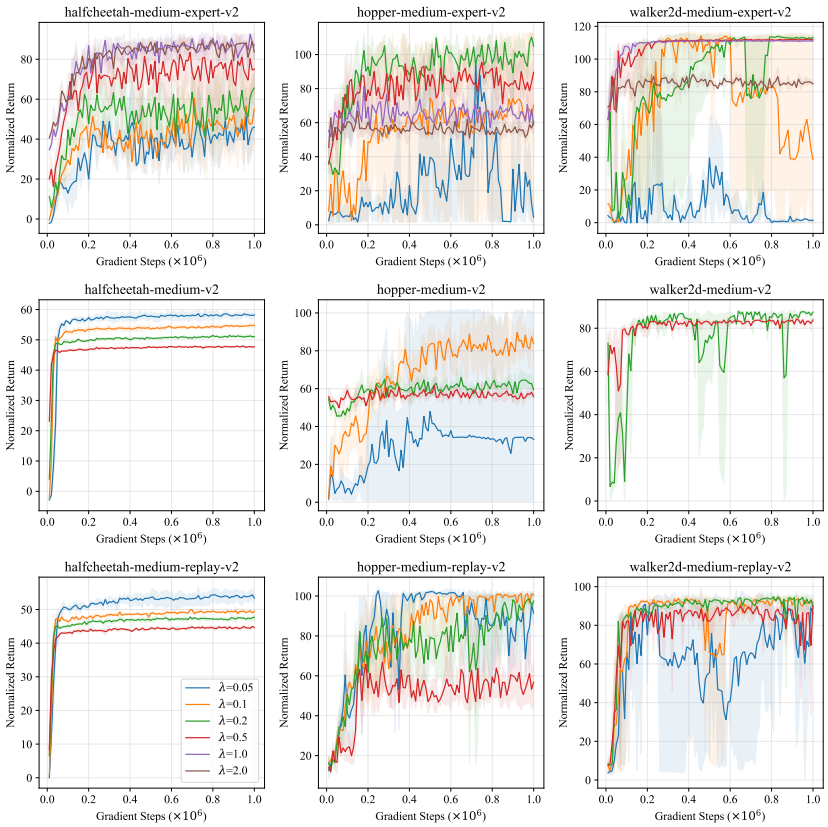


Figure 6: Learning curves of different λ for Gym-MuJoCo domains. Error bars indicate min and max over 3 seeds. We only search for 4 values on "medium-replay" and "medium" datasets and SPOT fails to train on walker-medium-v2 dataset with $\lambda = 0.05$ and $\lambda = 0.1$ since the constraint is too loose.

C.2 Ablation Study

Ablation study with variants of SPOT (Figure 2). We consider several variants of SPOT to perform an ablation study over the components in our method:

- 'Gaussian': We train a parameteric model $\widehat{\pi_\beta}$, which fits a tanh-Gaussian distribution to the actions a given the states s: $\widehat{\pi_\beta}(\cdot|s) = \tanh \mathcal{N}(\mu(\cdot|s), \sigma(\cdot|s))$, following [26, 49]. Then we use $\widehat{\pi_\beta}$ as our density estimator in the actor learning objective. Hyperparameters for training Gaussian behavior models are almost the same as Table 5.
- 'w/o Q Norm': We remove the Q normalization trick from SPOT.
- 'Stoch w/ Ent': We adopt SAC [13] as the base off-policy method of SPOT. Training objectives can be written as:

$$J_Q(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[Q_\theta(s,a) - r - \gamma \mathbb{E}_{a'\sim\pi_\phi(s')} \left[Q_{\bar{\theta}}\left(s',a'\right)\right]\right]^2$$

$$J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_\phi(s)} \left[-Q_\theta\left(s,a\right) + \alpha \log \pi_\phi(a|s) - \lambda \widehat{\log \pi_\beta}(a|s;\varphi,\psi,L)\right].$$

  The learning stochastic policy is parameterized as a tanh-Gaussian distribution and the temperature α is adjusted automatically, as [13]. We do not include the entropy term when performing the backup, as CQL [28]. Hyperparameters for training are almost the same as Table 6.
- 'Stoch w/o Ent': This variant is almost the same as the above one, except that we remove the entropy term:

$$J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_\phi(s)} \left[-Q_\theta\left(s,a\right) - \lambda \widehat{\log \pi_\beta}(a|s;\varphi,\psi,L)\right].$$

We present quantitative results of variants of SPOT in Table 7, corresponding to Figure 2.

Table 7: Quantitative results of ablation study on Gym-MuJoCo and AntMaze domains. For variants of SPOT, we report the mean and standard deviation for 5 seeds.

	SPOT variants				SPOT
	Gaussian	w/o Q Norm	Stoch w/o Ent	Stoch w/ Ent	
HalfCheetah-m-e-v2	77.7±8.4	**87.8**±5.5	87.3±3.7	87.0±3.8	86.9±4.3
Hopper-m-e-v2	79.9±21.7	98.6±16.4	**103.1**±8.6	66.6±24.9	99.3±7.1
Walker-m-e-v2	105.2±6.3	**112.9**±0.9	111.7±0.8	110.9±0.4	112.0±0.5
HalfCheetah-m-v2	57.2±0.6	**63.0**±1.9	58.2±0.9	57.1±0.8	58.4±1.0
Hopper-m-v2	96.6±3.2	**101.2**±0.1	58.4±8.3	52.1±2.4	86.0±8.7
Walker-m-v2	81.6±1.0	83.5±1.0	84.9±3.7	84.4±0.3	**86.4**±2.7
HalfCheetah-m-r-v2	53.3±1.4	**53.9**±0.9	51.8±0.4	51.5±1.1	52.2±1.2
Hopper-m-r-v2	95.2±10.4	98.5±6.1	82.7±12.8	99.3±3.7	**100.2**±1.9
Walker-m-r-v2	93.6±2.2	**94.4**±2.6	84.7±6.4	86.1±10.3	91.6±2.8
AntMaze-u-v2	83.0±4.9	**95.2**±1.9	74.5±20.0	78.2±17.5	93.5±2.4
AntMaze-u-d-v2	35.2±27.0	38.2±3.9	34.0±11.8	39.5±3.9	**40.7**±5.1
AntMaze-m-p-v2	46.0±13.8	72.2±5.3	20.0±28.6	21.0±36.4	**74.7**±4.6
AntMaze-m-d-v2	71.0±3.7	72.4±8.9	0.0±0.0	22.2±38.5	**79.1**±5.6
AntMaze-l-p-v2	29.6±4.8	**38.8**±15.9	0.0±0.0	11.0±19.1	35.3±8.3
AntMaze-l-d-v2	31.0±10.1	31.8±6.2	26.3±10.0	25.3±9.91	**36.3**±13.7

Effect of L in density estimation. As a way of investigating the effect of the tightness of density estimation (see Section 4.2) on the final performance, we evaluate different values of the number of samples L used in density estimation. We present result of $L \in \{1, 5, 10\}$ for Gym-MuJoCo domains in Figure 7. We note that all variants yield similar performance, which demonstrates that for this circumstance the ELBO estimator is a good enough density estimator. Thus unless otherwise specified, we adopt $L = 1$ as a default option in all of our experiments.

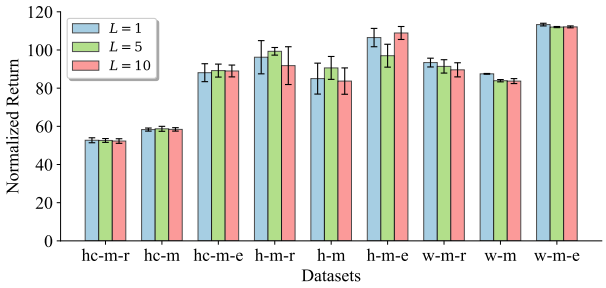


Figure 7: Comparing performance with different number of samples L used in density estimation. hc = HalfCheetah, h = Hopper, w = Walker, m = medium, m-r = medium-replay, m-e = medium-expert. Bars indicate the mean and standard deviation for 3 seeds.

C.3 Online Fine-tuning (Table 4)

Baselines. We takes the official implementation of IQL [25]: `https://github.com/ikostrikov/implicit_q_learning/` as our baseline.

Implementation details. We online fine-tune our models for 1M gradient steps after offline RL. In the online RL phase, we collect data actively in the environment with exploration noise 0.1 and add the data to the replay buffer. We linearly decay the regularization term weight λ in the online phase. We find that in the challenging AntMaze domains with high-dimensional state and action space as well as sparse reward, bootstrapping error is serious even in the online phase, thus we stop decay when λ reaches the 20% of its initial value at the 0.8M-th step. We also have experimented with gradually relaxing the implicit constraint of IQL by increasing its inverse temperature [35, 25]

but we find that IQL does not benefit from this. Additionally, we find that a larger discount factor γ is of great importance in the antmaze-large datasets, thus we set $\gamma = 0.995$ when fine-tuning on antmaze-large datasets, for both SPOT and IQL to ensure a fair comparison. All other training details are kept the same between the offline RL phase and the online RL phase.

Learning curves. Learning curves of online fine-tuning of SPOT and the baseline IQL are presented in Figure 8.

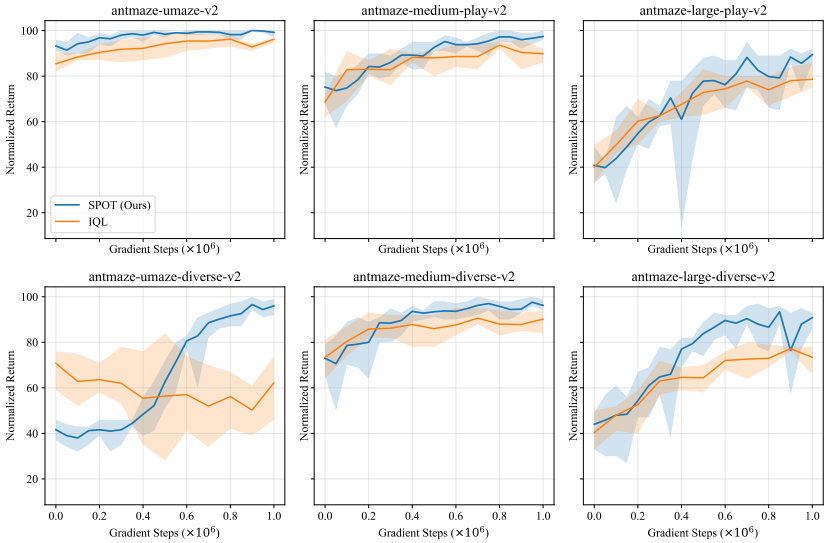


Figure 8: Learning curves of online fine-tuning for AntMaze domains of SPOT and the baseline IQL. Error bars indicate min and max over 5 seeds.

C.4 Analytic Experiments (Figure 1b)

Intuition. We assume that the same constraint strength implies the same risk of extrapolation error on Q estimation (related theoretical bound can be found in [26]). Benefiting from the exact constraint formulation (Eq. (4)), SPOT can fully exploit feasible actions that is ϵ-supported: $\{a : \pi_\beta(a|s) > \epsilon\}$. However, other kinds of constraints may deviate from the density-based formulation of ϵ-support set, thus feasible actions under these constraints may only constitute a subset of the minimal support set that covers them. Under the risk of Q estimation error but only exploiting a subset of the ϵ-supported actions, baseline methods may limit their optimality and provide a fragile tradeoff between satisfied constraint strength and optimality. To quantitatively illustrate this, we conduct experiments comparing the performance of different methods under the same constraint strength.

Data. We use the Gym-MuJoCo "medium-replay" and "medium" datasets from the D4RL benchmark [6], of the latest versions "v2".

Baselines. We choose several policy constraint methods as our baselines, taking their official implementation or the implementations evaluated by D4RL benchmarks:

- BCQ [10]: `https://github.com/rail-berkeley/d4rl_evaluations/tree/master/bcq`
- BEAR [26]: `https://github.com/rail-berkeley/d4rl_evaluations/tree/master/bear`
- PLAS [52]: `https://github.com/Wenxuan-Zhou/PLAS`
- TD3+BC [8]: `https://github.com/sfujim/TD3_BC`

Constraint strength control. Constraint strength of different policy constraint methods can be controlled by their own unique hyperparameters. By varying the values of these hyperparameters, we can get a spectrum of constraint strength. Hyperparameters that we adjust for each method are summaries as follows:

- BCQ [10]: max perturbation $\Phi \in \{0.02, 0.05, 0.1\}$. The smaller the value is, the stronger the constraint will be.
- BEAR [26]: MMD threshold $\varepsilon \in \{0.02, 0.05, 0.1\}$. The smaller the value is, the stronger the constraint will be.
- PLAS [52]: max latent action $\sigma \in \{1.0, 2.0, 3.0\}$. The smaller the value is, the stronger the constraint will be.
- TD3+BC [8]: Q term weight $\alpha \in \{1.0, 2.5, 4.0\}$. The smaller the value is, the stronger the constraint will be.
- SPOT (Ours): regularization term weight $\lambda \in \{0.05, 0.1, 0.2, 0.5\}$.The larger the value is, the stronger the constraint will be.

Quantitative results. We run baselines with varying hyperparameters for 3 seeds and present results in Table 8. We find that BEAR is unstable and rerunning for different seeds does not fix it, thus we exclude failed results from the Figure 1b and Figure 9.

Table 8: Quantitative results with 3 seeds for baselines in analytic experiments. hc = HalfCheetah, h = Hopper, w = Walker, m = medium, m-r = medium-replay.

	BCQ			BEAR			PLAS		
	Φ=0.02	Φ=0.05	Φ=0.1	ε=0.02	ε=0.05	ε=0.1	σ=1.0	σ=2.0	σ=3.0
hc-m-v2	44.2±1.6	46.7±0.4	49.3±0.8	42.8±0.1	42.8±0.1	17.3±28.8	43.3±0.3	44.8±0.1	45.0±1.0
h-m-v2	55.2±2.1	59.5±1.4	54.3±2.7	51.0±1.4	51.9±2.8	1.9±1.9	56.9±7.4	55.0±3.2	52.9±5.5
w-m-v2	80.9±1.4	70.3±11.1	69.2±1.1	-0.2±0.1	19.8±34.7	-0.3±0.0	74.1±2.3	78.4±4.9	73.4±8.3
hc-m-r-v2	38.0±2.5	40.0±1.6	43.5±1.0	36.7±0.6	37.2±0.8	37.2±0.5	40.7±1.5	43.5±0.4	44.3±0.7
h-m-r-v2	49.4±14.7	25.8±5.2	36.8±9.7	59.2±14.8	49.1±13.7	62.2±6.9	28.7±3.8	27.4±4.6	26.2±5.6
w-m-r-v2	44.2±20.8	55.8±11.8	36.1±5.5	1.5±1.5	7.7±6.0	4.3±1.7	53.6±31.8	63.3±18.9	39.6±27.5

Missing graphs. Due to space limitation, we only present results for "medium-replay" datasets for Figure 1b. The complete graphs are presented in Figure 9.

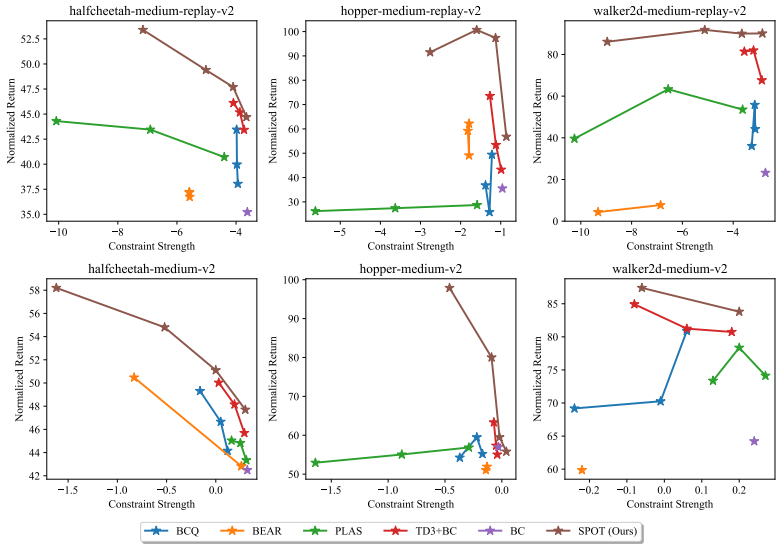


Figure 9: Full graphs of analysis on tradeoff between constraint strength and optimality, extending Figure 1b.

C.5 Computation Cost

Inferece time (Figure 3). We evaluate the runtime of different offline RL methods, that interact with the HalfCheetah environment to produce a full 1000-steps trajectory. For parameterization methods, we evaluate BCQ (num. of sampled actions $N = 100$) [10], PLAS [52], EMaQ (num. of sampled actions $N = 100$) [12] and for regularization methods, we evaluate BEAR (num. of sampled actions $p = 10$) [26], TD3+BC [8] and our SPOT. All numbers of runtime of Figure 3 are the mean of 100 trajectories. We compare different methods with consistent model size to ensure fairness.

Train time. Table 9 presents train time of 1M steps of various offline RL algorithms. All train time experiments were run with author-provided implementations on a single TITAN V GPU and Intel Xeon Gold 6130 CPU at 2.10GHz.

Table 9: Train time of 1M steps of various offline RL algorithms.

	BCQ	BEAR	PLAS	CQL	TD3+BC	SPOT
Train time	5h 25m	12h 30m	3h 5m	14h 20m	1h 58m	3h 25m

D Broader Impact

Social impacts. Offline reinforcement learning has the potential to enable or scale-up practical applications for reinforcement learning, such as robotics, recommendation, healthcare, or educational applications, where data collecting is always expensive or risky, and offline logged data can lead to a better real-world performance by either pure offline or offline2online learning. A limitation to offline RL is that the learned policy, regularized by the offline data, may contain biases originally from the data-collecting policy.

Academic research. Developing a simple and effective offline RL algorithm is the primary aim behind our work. We situate our work in the literature on policy constraint methods for offline RL, covering discussion w.r.t empirical performance, implementation simplicity, and computation efficiency. We identify that a standard off-policy RL algorithm plugged with a VAE-based explicit support constraint is sufficient for exceeding most of substantially more complicated methods on both standard and challenging benchmarks, which may encourage researchers to revisit the progress of offline RL and derive new and better offline RL methods.