Drift Q-Learning

Anas Houssaini* **Mohamad H. Danesh*** **Amin Abyaneh**

Scott Fujimoto **Hsiu-Chin Lin** **David Meger**

McGill University
Mila - Quebec AI Institute

Abstract

Offline reinforcement learning requires improving a policy from fixed data while avoiding out-of-distribution actions with unreliable value estimates. Diffusion and flow policies handle this trade-off by modeling the behavior distribution to regularize the RL objective, but they require iterative denoising, solver integrations, and in more efficient variants, distillation or other approximations at inference. We propose **DriftQL**, which combines a drift-based behavioral regularizer with critic-driven policy improvement. The value signal biases the policy toward high-value regions of the data support, while attraction and repulsion together keep generated actions near the data and prevent collapse onto a single mode. DriftQL is implemented as a single network with a unified training objective and generates actions in a *single* forward pass. On D4RL and OGBench, DriftQL consistently outperforms diffusion and flow methods, advancing the state of the art. Under degraded data quality, where the baselines visibly struggle, DriftQL remains close to its clean-data performance, positioning it as a promising alternative to diffusion and flow-based methods while maintaining the simplicity and efficiency of deterministic approaches.

Project page: driftql.github.io

1 Introduction

Offline reinforcement learning (RL) learns policies from fixed datasets without environment interaction [Fujimoto et al., 2019]. Since the value function is trained exclusively on transitions from the dataset, its estimates for out-of-distribution (OOD) actions are unreliable. This necessitates behavior regularization that must balance two competing objectives: constraining the policy toward the data distribution so that value estimates remain trustworthy while still permitting policy improvement away from low-value actions.

Expressive generative models [Sohl-Dickstein et al., 2015, Lipman et al., 2022] are well-suited for this tradeoff as they can represent the full multimodal structure of the behavior distribution, providing broad coverage of the data support while remaining flexible enough to concentrate probability on high-value regions. These properties have driven strong empirical performance across offline RL benchmarks [Janner et al., 2022, Hansen-Estruch et al., 2023, Wang et al., 2023, Park et al., 2025b, Abyaneh et al., 2026]. However, existing approaches carry significant practical limitations. Such models require iterative denoising or heavy solver integrations to produce an action, making them slow at inference time. While distillation-based alternatives can improve inference speed, they require an additional network and use a two-stage training pipeline, increasing complexity.

*Equal contribution.
Corresponding authors: {achraf.elhoussaini, mo.danesh}@mail.mcgill.ca

Preprint.

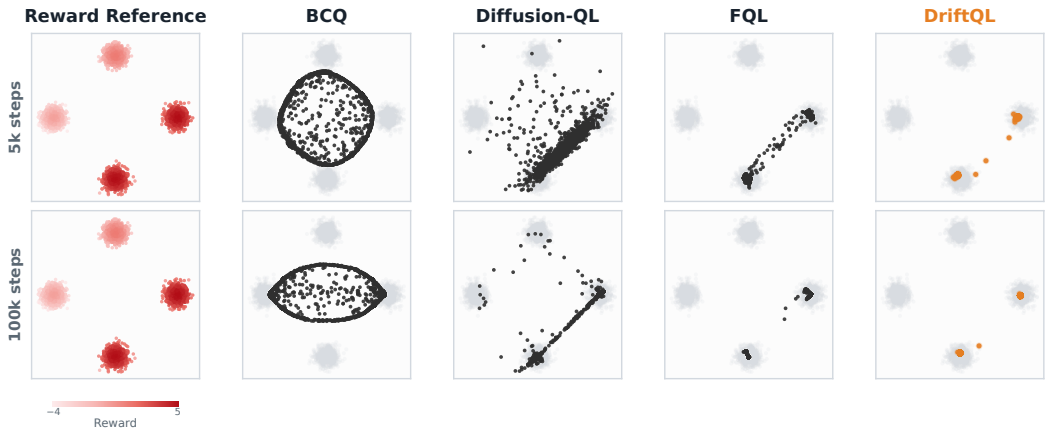

Figure 1: **Value-guided mode selection on a four-Gaussian bandit with tied optima.** We adapt the four-Gaussian bandit of Wang et al. [2023], arranging four isotropic data modes in a cross with the right and bottom modes tied for highest reward. Reward decays with distance from each mode center. The first column shows the reward reference, and gray blobs show dataset support. Rows show policy samples after 5k and 100k training steps. BCQ remains broad and assigns mass to unsupported regions, while Diffusion-QL and FQL move toward high-reward modes but retain scattered samples or bridges between modes. DriftQL concentrates on the two tied high-reward supported modes, with sharper mode selection by 100k steps.

Drifting Models offer a promising alternative [Deng et al., 2026]. They match the expressive capacity of diffusion and flow methods but generate samples in a single forward pass at both training and inference [Lai et al., 2026]. A Drifting Model learns a direct one-step pushforward map from prior noise and conditioning signals to data, supervised by the non-parametric *drifting field* which balances support for data while resisting collapse.

Building on this insight, we propose Drift Q-Learning (**DriftQL**), a one-step generative method with a conditional drift field [Deng et al., 2026] to supervise behavior regularization in offline RL. Drifting models are particularly well-suited to this setting because they reconcile the two requirements identified above: they preserve mass across candidates during training to smooth optimization, while remaining mode-seeking at convergence [Lai et al., 2026] so that probability concentrates on high-value regions rather than the full action distribution. Concretely, our drift field includes a behavior-cloning-like attraction toward observed dataset actions, and additionally repels nearby generated actions from one another, preserving a diverse set of candidates within high-value regions throughout training. Finally, since the drift field defines a single-step transport target, inference requires only one forward pass, avoiding both the discretization and integration errors as well as the computational overhead of iterative denoising in diffusion and flow-based generative modeling. Fig. 1 highlights convergence behavior of DriftQL.

We evaluate DriftQL on D4RL and OGBench [Fu et al., 2020, Park et al., 2025a], where it outperforms diffusion and flow-based baselines while generating actions in a single forward pass with no denoising chains, no solvers, no distillation, and no auxiliary networks. When data quality degrades, DriftQL sustains its performance in various environments and noise levels where other baselines clearly struggle. Our results position DriftQL as a promising alternative to diffusion and flow-based offline RL methods while maintaining the simplicity and efficiency of deterministic approaches.

2 Related Work

Offline RL must improve a policy from fixed data without exploiting unsupported actions whose values are unreliable [Fujimoto et al., 2019]. Early methods addressed this through explicit behavior constraints [Fujimoto et al., 2019, Kumar et al., 2019], while CQL [Kumar et al., 2020] regularizes the critic to suppress overestimated values on OOD actions. IQL [Kostrikov et al., 2021] avoids evaluating OOD actions entirely through expectile regression, and uncertainty-aware methods use critic ensembles to quantify uncertainty and moderate risky policy updates [An et al., 2021, Ghasemipour et al., 2022, Yang et al., 2022, Danesh et al., 2025].

Generative policies for offline RL have largely started with diffusion-based methods that combine expressive behavior modeling with value guidance: DQL [Wang et al., 2023] jointly trains a diffusion policy with behavior cloning and value maximization, IDQL [Hansen-Estruch et al., 2023] couples a diffusion behavior model with implicit Q-learning for policy extraction, EDP [Kang et al., 2023] approximates actions from corrupted samples to avoid running the full denoising chain at training, and BDPO [Gao et al., 2025] derives an analytic KL regularization along the diffusion trajectory within a two-time-scale actor-critic. Flow-based methods further reduce training and inference cost by avoiding iterative sampling: FQL [Park et al., 2025b] trains an expressive flow-matching policy and distills it into an RL-optimized one-step student, FlowQ [Alles et al., 2025] bakes energy-based Q-guidance directly into the flow-matching objective via reweighted regression, and SSCP [Koirala and Fleming, 2025] augments flow matching to predict direct completion vectors for one-shot action generation. Later extensions further accelerate generation through distillation, flow reformulation, or direct one-step training [Chae et al., 2026, Zhang et al., 2026, Nguyen and Yoo, 2026]. DriftQL takes an entirely different route. We train a one-step policy with drifting loss, sidestepping iterative denoising and reliance on a multi-step teacher altogether.

3 Background

Offline RL. Consider a Markov Decision Process defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces (with $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$), $P(s' \mid s, a)$ is the transition function, $r(s, a)$ is the reward, and $\gamma \in [0, 1)$ is the discount factor. An agent behaves according to a policy $\pi(a \mid s)$, aiming to maximize the expected discounted return $J(\pi) = \mathbb{E}_{\tau \sim p_\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$, under trajectories τ induced by π and P [Sutton et al., 1998]. In offline RL, the agent must optimize $J(\pi)$ using only a static dataset $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{M}$ collected by an unknown behavior policy π_β [Fujimoto et al., 2019].

Value-based offline RL algorithms estimate the action-value function $Q^\pi(s, a)$, but suffer from *extrapolation error* on OOD actions. Behavioral regularization addresses this by constraining the learned policy to remain close to the behavior policy [Fujimoto and Gu, 2021, Kumar et al., 2020]. A prominent example of this paradigm is TD3+BC [Fujimoto and Gu, 2021], which augments the standard Q-maximization objective with a BC penalty. The corresponding actor loss is formulated as:

$$\mathcal{L}_{\text{actor}}(\theta) = -\lambda\, \mathbb{E}_{(s,a)\sim\mathcal{D}}[Q(s, \pi_\theta(s))] + \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\|\pi_\theta(s) - a\|^2\right], \tag{1}$$

where $\lambda = \alpha / \mathbb{E}_{(s,a)\sim\mathcal{D}}[|Q(s, a)|]$ normalizes the Q-value gradients to ensure they are properly scaled relative to the BC penalty. A downside to TD3+BC is that π_θ is deterministic, meaning that the regularizer produces a single target per state and cannot maintain mass across multiple candidate actions, a property DriftQL recovers through stochastic generation and a repulsive component.

Generative Drifting Models. Drifting models [Deng et al., 2026] train a one-step conditional generator by constructing a kernel-based vector field that displaces generated outputs toward a target distribution. Let $f_\theta(c, \epsilon)$ map a conditioning input c and Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$ to a generated output $\hat{y}$. For a fixed c, independent noise draws induce the generator distribution q_θ, while p denotes the target distribution for that condition.

To compute the drift, positives $y^+ \sim p$ are drawn from the target distribution and negatives $\hat{y}^- \sim q_\theta$ are produced by the current generator using random noise. The drift decomposes as:

$$V_{p,q_\theta}(\hat{y}) = V_p^+(\hat{y}) - V_{q_\theta}^-(\hat{y}), \tag{2}$$

$$V_p^+(\hat{y}) = \frac{1}{Z_p(\hat{y})} \mathbb{E}_{y^+\sim p}\left[k(\hat{y}, y^+)(y^+ - \hat{y})\right], \tag{3}$$

$$V_{q_\theta}^-(\hat{y}) = \frac{1}{Z_q(\hat{y})} \mathbb{E}_{\hat{y}^-\sim q_\theta}\left[k(\hat{y}, \hat{y}^-)(\hat{y}^- - \hat{y})\right]. \tag{4}$$

Here $k(\cdot, \cdot)$ is a similarity kernel with $Z_p(\cdot)$ and $Z_q(\cdot)$ normalizing the kernel weights. Both terms take the form of a mean-shift step, which moves a point toward the kernel-weighted average of a reference set, giving closer samples stronger influence. V_p^+ applies this to the positives, attracting $\hat{y}$ toward target outputs. $V_{q_\theta}^-$ does the same over generated samples, and the minus sign converts the resulting pull into a push away [Lai et al., 2026]. When $q_\theta = p$, the two terms cancel. In practice, the expectations are estimated with M positives $\{y_i^+\}_{i=1}^{M}$ from p and N negatives $\{\hat{y}_j^-\}_{j=1}^{N}$ from

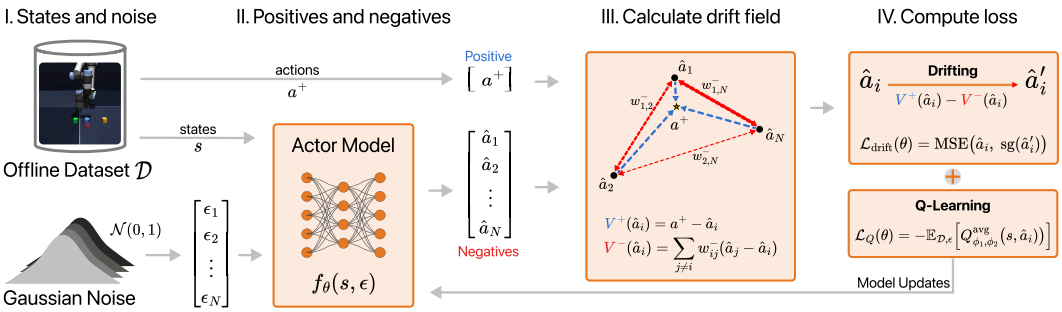

Figure 2: **Overview of DriftQL.** (I) A state s is sampled from the offline dataset $\mathcal{D}$ along with N Gaussian noise vectors ϵ. (II) The actor network processes state and noise to generate candidate actions $\{\hat{a}_i\}_1^N$. (III) The generated actions are subjected to a conditional drift field composed of two forces: an attraction force (V^+) that pulls all actions uniformly toward the true dataset action a^+, and a repulsion force (V^-) that pushes the generated actions away from one another. (IV) At the end, the drifting process dictates the update step, computing a new target action $\hat{a}_i'$ for each sample, calculating the drift loss, and updating the actor network with both value estimation and drifting losses.

the generator, where $\hat{y}_j^- = f_\theta(c, \epsilon_j^-)$ for independent $\epsilon_j^- \sim \mathcal{N}(0, I)$. The generator is trained by regressing each output toward its drifted location:

$$\mathcal{L}_{\text{drift}}(\theta) = \mathbb{E}_{c,\epsilon}\left[\|\hat{y} - \text{sg}\left(\hat{y} + V_{p,q_\theta}(\hat{y})\right)\|_2^2\right], \qquad \hat{y} = f_\theta(c, \epsilon), \tag{5}$$

where $\text{sg}(\cdot)$ denotes stop-gradient. After training, inference is a single evaluation $\hat{y} = f_\theta(c, \epsilon)$.

4 Drift Q-Learning

In this section, we present DriftQL, a behavior-regularized offline RL learning method that builds on the drifting framework. We adopt the one-step pushforward distribution in state-conditioned action space as our policy parameterization. Due to the nature of data in offline RL, constructing the supervising conditional drift field in this situation requires a critical set of changes compared to the original formulation [Deng et al., 2026]. An actor-critic objective unifies distribution modeling with reward-seeking and enables the learning of effective behaviors from challenging offline data. DriftQL's logic is outlined in Fig. 2.

At its core, DriftQL uses a stochastic generator $a = f_\theta(s, \epsilon)$ as the policy. A state $s \in \mathcal{S}$ and a noise vector $\epsilon \sim \mathcal{N}(0, I)$ are mapped to an action $a \in \mathcal{A}$, with ϵ inducing the policy's stochasticity. For each state, the actor produces N candidate actions $\hat{a}_i = f_\theta(s, \epsilon_i)$, $i = 1, \ldots, N$, which are simultaneously transported by a state-conditional drift field:

$$V(\hat{a}_i) = V^+(\hat{a}_i) - V^-(\hat{a}_i). \tag{6}$$

Attraction. In the unconditional drifting field of Deng et al. [2026], V_p^+ is a kernel-weighted mean shift over multiple samples drawn from the target distribution, with each positive weighted by its similarity to the candidate (Eq. 3). However, continuous state and action offline RL is single-positive by construction: the dataset supplies one observed action per state. With a single positive, Eq. 3 contains only one target sample and the empirical estimator of V_p^+ reduces to a single term:

$$V^+(\hat{a}_i) = a^+ - \hat{a}_i. \tag{7}$$

Repulsion. Repulsion retains the mean-shift structure of $V_{q_\theta}^-$ from the original drifting method (Eq. 4): the $N - 1$ other policy samples form a non-degenerate empirical estimator of the model distribution $q_\theta(\cdot \mid s)$, and the resulting force prevents candidates from collapsing onto a^+. Concretely, the repulsion vector is a weighted average of displacements toward neighboring samples, which the minus sign in Eq. 6 converts into a push away from them:

$$V^-(\hat{a}_i) = \sum_{k \neq i} w_{ik}^- (\hat{a}_k - \hat{a}_i), \tag{8}$$

where the weights are obtained by a row-wise softmax over the off-diagonal entries of an $N \times N$ logit matrix,

$$w_{i,:}^- = \text{softmax}\left(\ell_{i,:}^-\right), \tag{9}$$

whose logits come from a Gaussian kernel with temperature τ controlling the sharpness of the fall-off:

$$\ell^-_{ik} = -\frac{\|\hat{a}_i - \hat{a}_k\|_2^2}{2\tau^2\, d_a}. \tag{10}$$

Drift loss. Combining the attraction and repulsion components, each generated action $\hat{a}_i$ is transported toward a drifted candidate $\hat{a}_i^+ = \hat{a}_i + V(\hat{a}_i)$, where $V(\hat{a}_i)$ is evaluated over the jointly generated set $\{\hat{a}_j\}_{j=1}^N$ for state s. The drift loss regresses each $\hat{a}_i$ toward this target, held fixed via a stop-gradient:

$$\mathcal{L}_{\text{drift}}(\theta) = \mathbb{E}_{s\sim\mathcal{D},\,\epsilon_{1:N}}\left[\frac{1}{N}\sum_{i=1}^{N}\left\|\hat{a}_i - \text{sg}\big(\text{clip}(\hat{a}_i^+,\,-1,1)\big)\right\|_2^2\right], \quad \hat{a}_i^+ = \hat{a}_i + V(\hat{a}_i). \tag{11}$$

The stop-gradient (sg) freezes the drifted target $\hat{a}_i^+$, so the actor is updated toward the transported point without backpropagating through the drift-field computation. Since the targets are recomputed from the current policy at every training step, the drift regularizer tracks the evolving actor distribution.

Critic. Following FQL [Park et al., 2025b], we pair the drift actor with a clipped double-Q critic $\{Q_{\phi_1}, Q_{\phi_2}\}$ [Fujimoto et al., 2018], trained via standard Bellman regression.

Actor. The actor jointly minimizes the drift loss and a value-improvement term. The drift loss acts as a behavior regularizer, while the value gradient pushes mass toward high-value modes.

$$\mathcal{L}_{\text{actor}}(\theta) = \alpha\,\mathcal{L}_{\text{drift}}(\theta) + \mathcal{L}_Q(\theta), \qquad \mathcal{L}_Q(\theta) = -\mathbb{E}_{\mathcal{D},\epsilon}\Big[\tfrac{1}{2}\big(Q_{\phi_1} + Q_{\phi_2}\big)\big(s, f_\theta(s,\epsilon)\big)\Big], \tag{12}$$

with $\alpha > 0$ weighting behavioral fidelity against policy improvement. The value gradient $\nabla_\theta\mathcal{L}_Q$ supplies per-candidate reweighting. Candidates are not merely pulled toward a^+, but are biased toward high-value regions of the support, allowing the policy to concentrate on multiple high-value modes when the data contains them.

The drift policy is trained directly against Eq. 12 and, unlike competing generative model approaches [Wang et al., 2023, Park et al., 2025b], inference reduces to a single forward pass $a = f_\theta(s,\epsilon)$. With no need for an auxiliary network, solvers, or denoising chain, we argue that this formulation is the most direct implementation of generative behavior regularization for offline RL policy improvement. Its efficiency is empirically validated in Subsec. 5.4.

Differences from Deng et al. [2026]. The original drifting implementation was developed for high-dimensional, unconditional image generation, where the drift field must single-handedly shape the model distribution toward the data. Two structural challenges arise in that setting, both highlighted by Lai et al. [2026]. First, when kernels are evaluated on learned feature embeddings, relative distances among samples can become nearly uniform, making the softmax kernel weights close to flat and the drift magnitude small even when p and q remain mismatched. Second, the Laplace kernel (when replaced with the Gaussian one) does not give a clean identifiability story: the preconditioned-score decomposition shows that mean-shift differs from the smoothed-score field by a scalar preconditioner and a covariance residual, both depending on the local kernel-reweighted neighborhood, so the equilibrium $V \equiv 0$ no longer forces $q = p$ unless those terms are separately controlled.

To compensate, Deng et al. [2026] compute a *joint* kernel coupling positives and negatives, $V_{p,q}(x) \propto \mathbb{E}[k(x,y^+)\,k(x,y^-)\,(y^+ - y^-)]$, apply softmax along both the x and y axes, aggregate drift fields across multiple temperatures, and rescale the resulting force to unit RMS. These mechanisms reshape the effective transport field so that, even under feature-space evaluation with a Laplace kernel, the drift remains a sufficient training signal on its own.

In unconditional image generation, the drift field is the *only* signal shaping the generator, so it must single-handedly drive the model to a unique, identifiable equilibrium $q = p$. This is precisely the job of the symmetrized affinities, cross-weighting, multi-temperature aggregation, and force normalization. Lai et al. [2026] show that without them the bare kernel drift does not guarantee $V \equiv 0 \Rightarrow q = p$, so the machinery exists to suppress that residual ambiguity.

Offline RL removes the need for it on two grounds, decided *before* any tuning. First, we do not target $q = p$ at all. The optimal policy is generally deterministic, and the goal is to *improve* over the behavior policy and concentrate on high-value modes, not to reproduce the data distribution.

Second, and more importantly, the drift is no longer the only signal. The value gradient $\nabla_\theta\mathcal{L}_Q$ supplies an independent, dataset-anchored pull toward high-value support that anchors the actor

Table 1: **Offline RL results.** DriftQL achieves competitive or superior performance in almost all 78 offline learning tasks. We present standard deviations after "±" and denote values at or above 95% of the best performance in bold.

	Gaussian			Diffusion			Flow				Drift
Task	BC	IQL	ReBRAC	IDQL	SRPO	CAC	FAWAC	FBRAC	IFQL	FQL	DriftQL
D4RL											
antmaze (6)	17	57	78	79	74	30 ±3	44 ±3	64 ±7	65 ±7	**84** ±3	**84** ±9
adroit (12)	48	53	**59**	52 ±1	51 ±1	43 ±2	48 ±1	50 ±2	52 ±1	52 ±1	50 ±5
locomotion (9)	50	82	**90**	82	**87**	–	–	–	50 ±3	63 ±2	81 ±23
D4RL Overall	38	64	**76**	71 ±1	71 ±1	–	–	–	56 ±9	66 ±1	**72** ±2
OGBench											
antmaze-large-st (5)	11 ±1	53 ±3	81 ±5	21 ±5	11 ±4	33 ±4	6 ±1	60 ±6	28 ±5	79 ±3	**92** ±4
antmaze-giant-st (5)	0 ±0	4 ±1	26 ±8	0 ±0	0 ±0	0 ±0	0 ±0	4 ±4	3 ±2	9 ±6	**60** ±2
humanoidmaze-med-st (5)	2 ±1	33 ±2	22 ±8	1 ±0	1 ±1	53 ±8	19 ±1	38 ±5	**60** ±1	58 ±5	**62** ±2
humanoidmaze-large-st (5)	1 ±0	2 ±1	2 ±1	1 ±0	0 ±0	0 ±0	0 ±0	2 ±0	**11** ±2	4 ±2	5 ±8
antsoccer-arena-st (5)	1 ±0	8 ±2	0 ±0	12 ±4	1 ±0	2 ±4	12 ±0	16 ±1	33 ±6	60 ±2	**65** ±2
cube-single-st (5)	5 ±1	83 ±3	91 ±2	**95** ±2	80 ±5	85 ±9	81 ±4	79 ±7	79 ±2	**96** ±1	**94** ±3
cube-double-st (5)	2 ±1	7 ±1	12 ±1	15 ±6	2 ±1	6 ±2	5 ±2	15 ±3	14 ±3	**29** ±2	25 ±2
scene-st (5)	5 ±1	28 ±1	41 ±3	46 ±3	20 ±1	40 ±7	30 ±3	45 ±5	30 ±3	56 ±2	**74** ±4
puzzle-3x3-st (5)	2 ±0	9 ±1	21 ±1	10 ±2	18 ±1	19 ±0	6 ±2	14 ±4	19 ±1	30 ±1	**35** ±3
puzzle-4x4-st (5)	0 ±0	7 ±1	14 ±1	**29** ±3	10 ±3	15 ±3	1 ±0	13 ±1	25 ±5	17 ±2	**27** ±3
OGBench Overall	3 ±0	23 ±1	31 ±1	23 ±1	14 ±1	25 ±2	16 ±1	29 ±1	30 ±1	44 ±1	**54** ±1

regardless of the drift's internal balance. The roles therefore separate cleanly, with the drift regularizer keeping generated actions near dataset support while the critic supplies the value-guided improvement signal, so the coefficient α in Eq. 12 directly governs the trade-off between the two. We thus expect the drift to remain stable in DriftQL without the image-oriented machinery, and verify this in App. C. The simplified drift computation matches or improves on the original drifting implementation in the offline RL setting, and a stress test that deliberately breaks the drift's attraction-repulsion balance confirms that the critic, not the machinery, supplies the missing constraint.

5 Experiments

We evaluate DriftQL on a suite of standard offline RL benchmarks designed to test behavioral multi-modality, long-horizon planning, and distributional coverage. Our experiments are structured to comprehensively validate both the downstream performance and the underlying mechanics of our approach. Specifically, we investigate whether DriftQL matches the expressivity of iterative generative policies, discuss computational efficiency during training and inference, and isolate the impact of our drift field adaptations.

5.1 Experimental Setup

Benchmarks. We evaluate our approach on two primary benchmark suites. To test standard continuous control and navigation, we use AntMaze, Adroit, and Locomotion from D4RL [Fu et al., 2020]. To specifically stress-test the policy's ability to handle highly multimodal and suboptimal data distributions with sparse reward signals, we evaluate on the OGBench suite [Park et al., 2025a].

Baselines. We compare DriftQL against a comprehensive set of offline RL algorithms. To represent standard Gaussian policies, we evaluate against Behavioral Cloning (BC), Implicit Q-Learning (IQL) [Kostrikov et al., 2021], and ReBRAC [Tarasov et al., 2023a]. For expressive generative policies, we benchmark against a diverse suite of diffusion and flow-matching methods. Our diffusion baselines include IDQL [Hansen-Estruch et al., 2023], SRPO [Chen et al., 2024], and CAC [Ding and Jin, 2024], and the flow-based ones are FQL [Park et al., 2025b] and its associated flow-based variants: Implicit Flow Q-Learning (IFQL), Flow Advantage-Weighted Actor-Critic (FAWAC), and Flow Behavior-Regularized Actor-Critic (FBRAC) [Park et al., 2025b].

Training and Evaluation. We train DriftQL for 1M gradient steps on state-based OGBench tasks, and 500K steps on D4RL tasks, evaluating the agent every 100K steps. To prevent evaluation bias, we assess DriftQL and all baselines using a fixed number of gradient steps rather than selecting the best performance across epochs. For OGBench, we adhere to the official evaluation protocol [Park et al., 2025a] and report the *average success rate across the last three evaluation epochs*. For

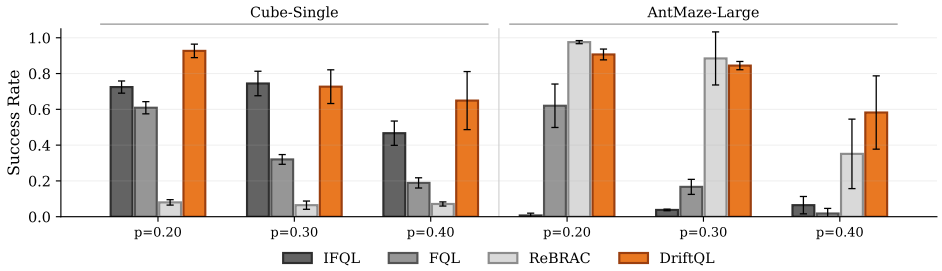

Figure 3: **Robustness under random-action corruption.** Success rate on the default `cube-single` and `antmaze-large` tasks as the random-action fraction increases from $p = 0.20$ to $p = 0.40$. Results are averaged over the 800k, 900k, and 1M evaluation checkpoints and shown with 95% confidence intervals.

D4RL, we follow standard practice and report *performance at the last epoch* [Tarasov et al., 2023b]. Performances are averaged over 8 seeds for all tasks.

5.2 Main Results: Downstream Offline RL Performance

We report success rates on OGBench and normalized scores on D4RL, comparing DriftQL against both unimodal baselines [Kostrikov et al., 2021, Tarasov et al., 2023a] and expressive multi-step generative baselines [Hansen-Estruch et al., 2023, Park et al., 2025b]. Baseline numbers are taken from prior work where available and reproduced with official implementations otherwise with full sourcing detailed in App. A. Our goal is to establish that DriftQL is competitive on complex continuous control without iterative inference.

Table 1 shows that DriftQL is competitive across the full suite, with the largest gains on the hardest tasks. It substantially outperforms all prior methods on long-horizon navigation (`antmaze-large-st`, `antmaze-giant-st`), `scene-st`, and `antsoccer-arena-st`, exceeds baselines on `humanoidmaze-medium-st` and `puzzle-3x3-st`, and matches the strongest baseline on `puzzle-4x4-st` and D4RL `antmaze`. Per-task results are reported in App. A.

5.3 Robustness Under Corrupted Offline Data

We next ask whether the same advantage as Subsec. 5.2 appears when the offline dataset itself becomes less reliable. We use the default `cube-single` and `antmaze-large` tasks from OGBench and modify the official data collection rule by replacing a fraction $p \in \{0.20, 0.30, 0.40\}$ of collected actions with uniformly random actions. Following the protocol used in the main experiments, we report success averaged over the 800k, 900k, and 1M checkpoints. We focus on these default tasks because the compared methods are already competitive in the corresponding clean-data settings, so the corruption study isolates robustness rather than trivial clean-data weakness.

Fig. 3 shows that the same picture reappears under corrupted offline data. On Cube, DriftQL is strongest at every corruption level. On AntMaze, ReBRAC is stronger at milder corruption, but DriftQL degrades more gracefully and becomes the strongest method at the hardest setting, $p = 0.40$.

5.4 Computational Efficiency and Inference Latency

While diffusion [Ho et al., 2020] and flow-matching models [Lipman et al., 2022] require multiple sequential network evaluations at inference, DriftQL generates actions in a single forward pass. Table 2 summarizes the per-method inference complexity, and Fig. 4 reports wall-clock measurements on the default `antmaze-large-st` task, averaged over 3 seeds on a single NVIDIA RTX 4090. To ensure a fair comparison, all methods use their tuned hyperparameters and share identical actor and critic network sizes.

Inference. DriftQL matches distilled FQL without requiring a separate distillation stage, and runs roughly $2\times$ faster than FQL, $3\times$ faster than Diffusion-QL, and $4\times$ faster than IDQL and IFQL. This advantage stems from the single-pass design: diffusion- and flow-based methods incur sequential ODE or denoising steps, while DriftQL transports the distribution during training and emits actions

Table 2: Inference complexity of expressive offline RL methods. *Sequential NFEs* is the number of network evaluations that cannot be parallelized, i.e. the main bottleneck for inference speed. IDQL and IFQL generate N_c candidates in parallel, each taking K sequential steps. DriftQL has only a single forward pass with no distillation.

Method	Sequential NFEs	Distillation required
Diffusion-QL [Wang et al., 2023]	$K \times N_c$	No
IDQL [Hansen-Estruch et al., 2023]	$K \times N_c$	No
FQL [Park et al., 2025b]	5–20 (ODE) + 1 (distilled)	Yes
IFQL [Park et al., 2025b]	$K \times N_c$	No
DriftQL	**1 (feedforward)**	**No**

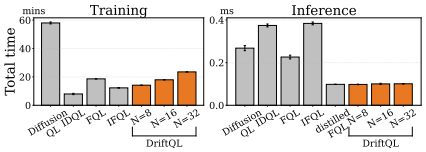

Figure 4: Total training time for 1M steps (left, minutes) and inference latency per step (right, ms), measured on default task from `antmaze-large-st`. All DriftQL variants share the same single-pass inference cost. Training cost scales with N.

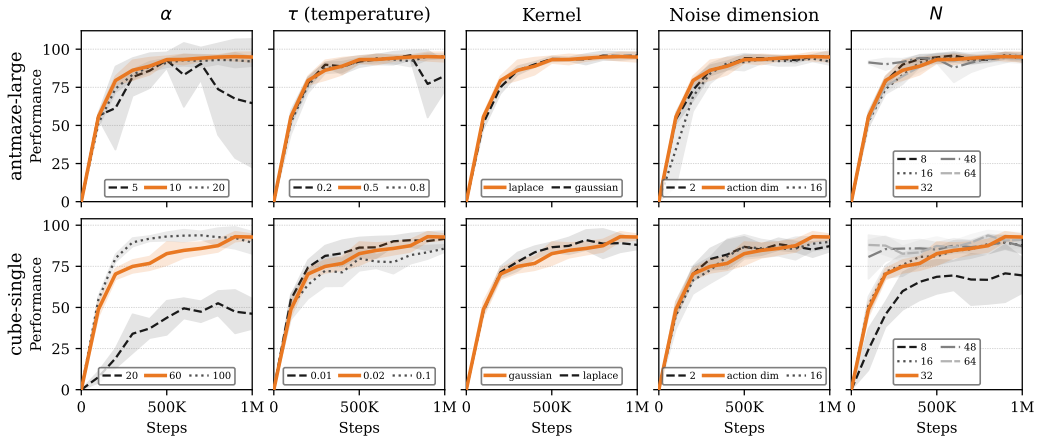

Figure 5: **Ablation Study of DriftQL Hyperparameters.** Success rates across training steps for default tasks in `antmaze-large-navigate` (top) and `cube-single-play` (bottom), averaged over 3 seeds with ± 1 standard deviation shaded. From left to right: the behavioral regularization trade-off (α), the drift kernel temperature (τ), the kernel type (Laplace vs. Gaussian), the noise dimension (z size), and the number of generated action samples per state (N). The default optimal setting for each environment is denoted by a solid orange line, while variations are plotted with grey lines.

directly at test time. Since inference uses one feedforward pass regardless of N, all DriftQL variants share identical latency.

Training. Training cost grows with N due to the $O(N^2)$ pairwise kernel, but the absolute overhead remains modest. At $N = 8$, DriftQL trains faster than FQL, at $N = 16$, it matches FQL, and at $N = 32$, it is only marginally slower. Diffusion-QL is substantially slower than all flow- and drift-based methods due to its longer denoising chain at training time. Larger N thus trades minor additional compute for greater sample diversity in the repulsion field.

5.5 Ablation Study

We isolate the impact of DriftQL's hyperparameters on two representative default environments that exercise different aspects of the drift field: `antmaze-large-navigate` (sparse-reward navigation, 29-dimensional observations and 8-dimensional actions) and `cube-single-play` (robotic cube manipulation, 28-dimensional observations and 5-dimensional actions). All ablations are averaged over 3 seeds, with one variable changed at a time relative to the per-environment defaults specified in App. F. Training curves for each ablation are shown in Fig. 5.

Trade-off parameter α: The scalar α governs the balance between the drift behavioral regularizer and the Q-maximization objective (Eq. 12), and it is the most important hyperparameter of DriftQL, consistent with the broader offline RL literature [Fujimoto and Gu, 2021, Tarasov et al., 2023a, Park et al., 2025b]. As shown in Fig. 5 (1st col), the sensitivity of performance to α differs markedly across environments: on `antmaze-large`, undersized α leads to a late-training collapse, while a slightly oversized α stays close to the default. Yet, on `cube-single`, the degrades are more noticeable at

the extremes. Intuitively, when α is too small the drift constraint provides insufficient support in low-density regions of the dataset, whereas an overly large α suppresses the Q-gradient and prevents improvement beyond the behavioral policy. We therefore recommend tuning α per environment, following the same protocol as Park et al. [2025b]: a coarse sweep over one order of magnitude is generally sufficient to identify a well-performing setting.

Kernel temperature τ: The temperature τ controls the sharpness of the kernel-weighted repulsion (Eq. 10): small τ concentrates repulsive force on the nearest neighbors, while large τ spreads it more uniformly across all generated samples. Fig. 5 (2^{nd} col) shows that DriftQL is broadly robust to τ across both environments, with performance remaining stable over a wide range. The curves separate only at the extremes, where very small τ can destabilize training by concentrating all repulsive mass on a single neighbor, and very large τ effectively removes the distance-sensitivity of the kernel. In practice, the default τ can be used without tuning.

Kernel choice: DriftQL uses a Gaussian kernel for the repulsion weights by default, motivated by its exact connection to the reverse-Fisher divergence on smoothed distributions [Lai et al., 2026]. Fig. 5 (3^{rd} col) compares the Gaussian kernel against the Laplace alternative used in the original drifting-model implementation [Deng et al., 2026]. On `antmaze-large`, the default and the softer setting are essentially indistinguishable, while the sharpest setting trails behind, with the gap appearing only late in training. On `cube-single`, although both kernels perform similarly throughout training, the Gaussian kernel provides an edge late in training.

Noise dimension: The actor $f_\theta(s, \epsilon)$ takes a noise vector $\epsilon \sim \mathcal{N}(0, I)$ whose dimension defaults to the action dimension. Fig. 5 (4^{th} col) shows that this choice is essentially free across a wide range. On `antmaze-large`, the smaller, default, and larger noise dimensions all converge to comparable performance, with only the larger setting showing slightly slower early progress before catching up. On `cube-single`, the picture is similar, with all settings performing similarly, except in the last few steps in training that they deviate slightly. We use the action-dimension default for simplicity.

Number of generated samples N_{gen}: The repulsive component of the drift field is computed over N generated samples per state. Because each step incurs an $O(N^2)$ pairwise kernel cost, N trades compute against the fidelity of the empirical negative distribution. Fig. 5 (5^{th} col) shows how performance varies with N across both environments. On `antmaze-large`, all values for N perform similarly, while on `cube-single`, the smaller N struggles to keep up with the default and larger values, which are comparable. On these two ablation environments, $N \geq 16$ essentially converges to the same performance, but on a broader sweep across the full benchmark we observed cases where $N = 16$ underperformed while $N = 32$ remained stable. Moreover, $N \geq 32$ leads to computational overhead while providing minimal improvements. Given these observations, we adopt $N = 32$ as the default to ensure robustness across environments without retuning per task.

6 Conclusion

We introduced DriftQL, an offline RL learner based on a one-step generative actor and a state-conditioned drifting objective. The method uses a drift regularizer to keep generated actions near dataset support and a critic objective to drive policy improvement. Unlike diffusion and flow policies that require iterative sampling, solvers, or distillation for fast inference, DriftQL trains a single stochastic actor that produces actions with one forward pass. Empirically, DriftQL performs competitively across D4RL and OGBench, with especially strong gains on difficult OGBench navigation and manipulation tasks. It also remains robust under random-action data corruption and retains the inference efficiency of deterministic one-step policies. These results suggest that drifting provides a useful middle ground for offline RL: it gives the actor a stochastic, distributional training signal while avoiding the test-time cost of multi-step generative policies.

Several questions remain open. We provide a detailed discussion of limitations in Appendix E, including the training-time cost of pairwise generated-action interactions, and the assumption of continuous box-bounded action spaces. It would also be useful to study more adaptive drift estimators, including learned or state-dependent kernels and cheaper approximations to the generated-action repulsion term. Finally, this work focuses on purely offline training with low-dimensional state spaces. Extending DriftQL to high-dimensional observations such as images, as well as to offline-to-online fine-tuning and online RL where fast action sampling and continued policy improvement are both important, are natural next steps.

References

Amin Abyaneh, Charlotte Morissette, Mohamad H. Danesh, Anas Houssaini, David Meger, Gregory Dudek, and Hsiu-Chin Lin. Contractive diffusion policies: Robust action diffusion via contractive score-based sampling with differential equations. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=iKJbmx1iuQ.

Marvin Alles, Nutan Chen, Patrick van der Smagt, and Botond Cseke. FlowQ: Energy-guided flow policies for offline reinforcement learning. *arXiv preprint arXiv:2505.14139*, 2025.

Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ZUvaSolQZh3.

Jongseong Chae, Jongeui Park, Yongjae Shin, Gyeongmin Kim, Seungyul Han, and Youngchul Sung. Flow actor-critic for offline reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=wuncwN7iZN.

Huayu Chen, Cheng Lu, Zhengyi Wang, Hang Su, and Jun Zhu. Score regularized policy optimization through diffusion behavior. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=xCRr9DrolJ.

Mohamad H Danesh, Maxime Wabartha, Stanley Wu, Joelle Pineau, and Hsiu-Chin Lin. Safe domain randomization via uncertainty-aware out-of-distribution detection and policy adaptation. *arXiv preprint arXiv:2507.06111*, 2025.

Mingyang Deng, He Li, Tianhong Li, Yilun Du, and Kaiming He. Generative modeling via drifting, 2026. URL https://arxiv.org/abs/2602.04770.

Zihan Ding and Chi Jin. Consistency models as a rich and efficient policy class for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8jdwkUNXb.

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.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 2021.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

Chen-Xiao Gao, Chenyang Wu, Mingjun Cao, Chenjun Xiao, Yang Yu, and Zongzhang Zhang. Behavior-regularized diffusion policy optimization for offline reinforcement learning. In *Forty-second International Conference on Machine Learning*, 2025.

Kamyar Ghasemipour, Shixiang Shane Gu, and Ofir Nachum. Why so pessimistic? Estimating uncertainties for offline RL through ensembles, and why their independence matters. *Advances in Neural Information Processing Systems*, 35:18267–18281, 2022.

Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. IDQL: Implicit Q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 2020.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

Bingyi Kang, Xiao Ma, Chao Du, Tianyu Pang, and Shuicheng YAN. Efficient diffusion policies for offline reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Prajwal Koirala and Cody Fleming. Flow-based single-step completion for efficient and expressive policy learning. *arXiv preprint arXiv:2506.21427*, 2025.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. *CoRR*, abs/2110.06169, 2021. URL https://arxiv.org/abs/2110.06169.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy Q-learning via bootstrapping error reduction. *CoRR*, abs/1906.00949, 2019. URL http://arxiv.org/abs/1906.00949.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 2020.

Chieh-Hsin Lai, Bac Nguyen, Naoki Murata, Yuhta Takida, Toshimitsu Uesaka, Yuki Mitsufuji, Stefano Ermon, and Molei Tao. A unified view of drifting and score-based models. *arXiv preprint arXiv:2603.07514*, 2026.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *ArXiv*, abs/2210.02747, 2022. URL https://api.semanticscholar.org/CorpusID:252734897.

Thanh Xuan Nguyen and Chang D. Yoo. One-step flow Q-learning: Addressing the diffusion policy bottleneck in offline reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=60VgwdzxDM.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations (ICLR)*, 2025a.

Seohong Park, Qiyang Li, and Sergey Levine. Flow Q-learning. In *International Conference on Machine Learning (ICML)*, 2025b.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, 2015.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36:11592–11620, 2023a.

Denis Tarasov, Alexander Nikulin, Dmitry Akimov, Vladislav Kurenkov, and Sergey Kolesnikov. CORL: Research-oriented deep offline reinforcement learning library. *Advances in Neural Information Processing Systems*, 36:30997–31020, 2023b.

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023.

Rui Yang, Chenjia Bai, Xiaoteng Ma, Zhaoran Wang, Chongjie Zhang, and Lei Han. RORL: Robust offline reinforcement learning via conservative smoothing. *Advances in neural information processing systems*, 35:23851–23866, 2022.

Songyuan Zhang, Oswin So, H M Sabbir Ahmad, Eric Yang Yu, Matthew Cleaveland, Mitchell Black, and Chuchu Fan. ReFORM: Reflected flows for on-support offline RL via noise manipulation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=YvFsyRReeN.

Appendix

Table of Contents

A Full Experimental Results

Table 3 reports the complete, per-task performance of DriftQL and all baselines across the full OGBench and D4RL benchmark tasks. The main text (Subsec. 5.2) condenses these into environment-level averages to summarize overall trends. This table provides the unaggregated breakdown to enable full reproducibility and to facilitate fine-grained comparisons for future work.

Table structure. Rows correspond to individual tasks, grouped by environment. The (*) marker denotes the default task used for hyperparameter tuning within each environment in OGBench. All other tasks in the group use the same hyperparameters, following the evaluation protocol of Park et al. [2025b]. Columns are organized by policy class: Gaussian policies (BC, IQL, ReBRAC), diffusion policies (IDQL, SRPO, CAC), flow policies (FAWAC, FBRAC, IFQL, FQL), and DriftQL. Values are averaged over 8 seeds. Standard deviations are reported after "±". Values at or above 95% of the best performance in each row are **bolded**, following Park et al. [2025a].

Sources of baseline numbers. For OGBench tasks and the D4RL Antmaze and Adroit suites, we report baseline numbers directly from Park et al. [2025b]. For D4RL Locomotion, baseline numbers are taken from the original papers: BC, IQL, and ReBRAC from Tarasov et al. [2023b], IDQL from Hansen-Estruch et al. [2023], and SRPO from Chen et al. [2024]. For IFQL and FQL on D4RL Locomotion, no published numbers were available for the exact task variants we evaluate, so we ran both methods ourselves using the official author implementations and tuned hyperparameters following the protocol described by Park et al. [2025b]. For methods denoted with −, due to computational constraints we were unable to obtain reliable results within the allotted budget, and no published numbers were available for those task variants to the best of our knowledge.

Table 3: **Full offline RL results.** We present the full results on the 77 OGBench and D4RL tasks. (*) indicates the default task in each environment. The results are averaged over 8 seeds (4 seeds for pixel-based tasks) unless otherwise mentioned.

	Gaussian Policies			Diffusion Policies			Flow Policies				Drift
Task	BC	IQL	ReBRAC	IDQL	SRPO	CAC	FAWAC	FBRAC	IFQL	FQL	DriftQL
antmaze-large-navigate-singletask-task1-v0 (*)	0 ±0	48 ±9	**91** ±10	0 ±0	0 ±0	42 ±7	1 ±1	70 ±20	24 ±17	80 ±8	**95** ±2
antmaze-large-navigate-singletask-task2-v0	6 ±3	42 ±6	**88** ±4	14 ±8	4 ±4	1 ±1	0 ±1	35 ±12	8 ±3	57 ±10	**85** ±9
antmaze-large-navigate-singletask-task3-v0	29 ±5	72 ±7	51 ±18	26 ±8	3 ±2	49 ±10	12 ±4	83 ±15	52 ±17	**93** ±3	**97** ±1
antmaze-large-navigate-singletask-task4-v0	8 ±3	51 ±9	84 ±7	62 ±25	45 ±19	17 ±6	10 ±3	37 ±18	18 ±8	80 ±4	**91** ±1
antmaze-large-navigate-singletask-task5-v0	10 ±3	54 ±22	**90** ±2	2 ±2	1 ±1	55 ±6	9 ±5	76 ±8	38 ±18	83 ±4	**92** ±1
antmaze-giant-navigate-singletask-task1-v0 (*)	0 ±0	0 ±0	27 ±22	0 ±0	0 ±0	0 ±0	0 ±0	0 ±1	0 ±0	4 ±5	**32** ±2
antmaze-giant-navigate-singletask-task2-v0	0 ±0	1 ±1	16 ±17	0 ±0	0 ±0	0 ±0	0 ±0	4 ±7	0 ±0	9 ±7	**79** ±1
antmaze-giant-navigate-singletask-task3-v0	0 ±0	0 ±0	34 ±22	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	0 ±1	**43** ±2
antmaze-giant-navigate-singletask-task4-v0	0 ±0	0 ±0	5 ±12	0 ±0	0 ±0	0 ±0	0 ±0	9 ±4	0 ±0	14 ±23	**64** ±3
antmaze-giant-navigate-singletask-task5-v0	1 ±1	19 ±7	49 ±22	0 ±1	0 ±0	0 ±0	0 ±0	6 ±10	13 ±9	16 ±28	**85** ±5
humanoidmaze-medium-navigate-singletask-task1-v0 (*)	1 ±0	32 ±7	16 ±9	1 ±1	0 ±0	38 ±19	6 ±2	25 ±8	**69** ±19	19 ±12	28 ±2
humanoidmaze-medium-navigate-singletask-task2-v0	1 ±0	41 ±9	18 ±16	1 ±1	1 ±1	47 ±35	40 ±2	76 ±10	85 ±11	**94** ±3	87 ±2
humanoidmaze-medium-navigate-singletask-task3-v0	6 ±2	25 ±5	36 ±13	0 ±1	2 ±1	**83** ±18	19 ±2	27 ±11	49 ±49	74 ±18	56 ±3
humanoidmaze-medium-navigate-singletask-task4-v0	0 ±0	0 ±1	15 ±16	1 ±1	1 ±1	5 ±4	1 ±1	1 ±2	1 ±1	3 ±4	**39** ±3
humanoidmaze-medium-navigate-singletask-task5-v0	2 ±1	66 ±4	24 ±20	1 ±1	3 ±3	91 ±5	31 ±7	63 ±9	**98** ±2	**97** ±2	**99** ±2
humanoidmaze-large-navigate-singletask-task1-v0 (*)	0 ±0	3 ±1	2 ±1	0 ±0	0 ±0	1 ±1	0 ±0	0 ±1	6 ±2	**7** ±6	2 ±1
humanoidmaze-large-navigate-singletask-task2-v0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0
humanoidmaze-large-navigate-singletask-task3-v0	1 ±1	7 ±3	8 ±4	3 ±1	1 ±1	2 ±3	1 ±1	10 ±2	**48** ±10	11 ±7	23 ±1
humanoidmaze-large-navigate-singletask-task4-v0	1 ±0	1 ±0	1 ±1	0 ±0	0 ±0	0 ±1	0 ±0	0 ±0	1 ±1	**2** ±3	1 ±1
humanoidmaze-large-navigate-singletask-task5-v0	0 ±1	1 ±1	**2** ±2	0 ±0	0 ±0	0 ±0	0 ±0	1 ±1	0 ±0	1 ±3	1 ±0
antsoccer-arena-navigate-singletask-task1-v0	2 ±1	14 ±5	0 ±0	44 ±12	2 ±1	1 ±3	22 ±2	17 ±3	61 ±25	**77** ±4	**79** ±4
antsoccer-arena-navigate-singletask-task2-v0	2 ±2	17 ±7	0 ±1	15 ±12	3 ±1	0 ±0	8 ±1	8 ±2	75 ±3	**88** ±3	**91** ±2
antsoccer-arena-navigate-singletask-task3-v0	0 ±0	6 ±4	0 ±0	0 ±0	0 ±0	8 ±19	11 ±5	16 ±3	14 ±22	**61** ±6	**60** ±3
antsoccer-arena-navigate-singletask-task4-v0 (*)	1 ±0	3 ±2	0 ±0	0 ±1	0 ±0	0 ±0	12 ±3	24 ±4	16 ±9	39 ±6	**48** ±5
antsoccer-arena-navigate-singletask-task5-v0	0 ±0	2 ±2	0 ±0	0 ±0	0 ±0	0 ±0	9 ±2	15 ±4	0 ±1	36 ±9	**48** ±9
cube-single-play-singletask-task1-v0	10 ±5	88 ±3	89 ±5	**95** ±2	89 ±7	77 ±28	81 ±9	73 ±33	79 ±4	**97** ±2	**94** ±3
cube-single-play-singletask-task2-v0 (*)	3 ±1	85 ±8	92 ±4	**96** ±2	82 ±16	80 ±30	81 ±9	83 ±13	73 ±3	**97** ±2	**93** ±2
cube-single-play-singletask-task3-v0	9 ±3	91 ±5	93 ±3	**99** ±1	**96** ±2	**98** ±1	87 ±4	82 ±12	88 ±4	**98** ±2	**95** ±2
cube-single-play-singletask-task4-v0	2 ±1	73 ±6	**92** ±3	**93** ±4	70 ±18	**91** ±2	79 ±6	79 ±20	79 ±6	**94** ±3	**92** ±3
cube-single-play-singletask-task5-v0	3 ±3	78 ±9	87 ±8	**90** ±6	61 ±12	80 ±20	78 ±10	76 ±33	77 ±7	**93** ±3	**90** ±3
cube-double-play-singletask-task1-v0	8 ±3	27 ±5	45 ±6	39 ±19	7 ±6	21 ±8	21 ±7	47 ±11	35 ±9	**61** ±9	49 ±1
cube-double-play-singletask-task2-v0 (*)	0 ±0	1 ±1	7 ±3	16 ±10	0 ±0	2 ±2	2 ±1	22 ±12	9 ±5	**36** ±6	23 ±5
cube-double-play-singletask-task3-v0	0 ±0	0 ±0	4 ±1	17 ±8	0 ±1	3 ±1	1 ±1	4 ±2	8 ±5	**22** ±5	9 ±3
cube-double-play-singletask-task4-v0	0 ±0	0 ±0	1 ±1	0 ±1	0 ±0	0 ±1	0 ±0	0 ±1	1 ±1	**5** ±2	3 ±1
cube-double-play-singletask-task5-v0	0 ±0	4 ±3	4 ±2	1 ±1	0 ±0	3 ±2	2 ±1	2 ±2	17 ±6	19 ±10	**43** ±2
scene-play-singletask-task1-v0	19 ±6	94 ±3	**95** ±2	**100** ±0	94 ±4	**100** ±1	87 ±8	**96** ±8	**98** ±3	**100** ±0	**100** ±0
scene-play-singletask-task2-v0 (*)	1 ±1	12 ±3	50 ±13	33 ±14	2 ±2	50 ±40	18 ±8	46 ±10	0 ±0	76 ±9	**89** ±3
scene-play-singletask-task3-v0	1 ±1	32 ±7	55 ±16	**94** ±4	4 ±4	49 ±16	38 ±9	78 ±14	54 ±19	**98** ±1	**93** ±3
scene-play-singletask-task4-v0	2 ±2	0 ±1	3 ±3	4 ±3	0 ±0	0 ±0	6 ±1	4 ±4	0 ±0	5 ±1	**83** ±10
scene-play-singletask-task5-v0	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	0 ±0	**2** ±2
puzzle-3x3-play-singletask-task1-v0	5 ±2	33 ±6	**97** ±4	52 ±12	89 ±5	**97** ±2	25 ±9	63 ±19	**94** ±3	90 ±4	87 ±9
puzzle-3x3-play-singletask-task2-v0	1 ±1	4 ±3	1 ±1	0 ±1	0 ±1	0 ±0	4 ±2	2 ±2	1 ±2	16 ±5	**39** ±2
puzzle-3x3-play-singletask-task3-v0	1 ±1	3 ±2	3 ±1	0 ±0	0 ±0	0 ±0	1 ±0	1 ±1	0 ±0	10 ±3	**20** ±3
puzzle-3x3-play-singletask-task4-v0 (*)	1 ±1	2 ±1	2 ±1	0 ±0	0 ±0	0 ±0	1 ±1	2 ±2	0 ±0	**16** ±5	10 ±6
puzzle-3x3-play-singletask-task5-v0	1 ±0	3 ±2	5 ±3	0 ±0	0 ±0	0 ±0	1 ±1	2 ±2	0 ±0	16 ±3	**19** ±1
puzzle-4x4-play-singletask-task1-v0	1 ±1	12 ±2	26 ±4	48 ±5	24 ±9	44 ±10	1 ±2	32 ±9	49 ±9	34 ±8	**72** ±8
puzzle-4x4-play-singletask-task2-v0	0 ±0	7 ±4	12 ±4	14 ±5	0 ±1	0 ±0	0 ±1	5 ±3	4 ±4	**16** ±5	4 ±2
puzzle-4x4-play-singletask-task3-v0	0 ±0	9 ±3	15 ±3	34 ±5	21 ±10	29 ±12	1 ±1	20 ±10	**50** ±14	18 ±5	**47** ±10
puzzle-4x4-play-singletask-task4-v0 (*)	0 ±0	5 ±2	10 ±3	**26** ±6	7 ±4	1 ±1	0 ±0	5 ±1	21 ±11	11 ±3	10 ±3
puzzle-4x4-play-singletask-task5-v0	0 ±0	4 ±1	7 ±3	**24** ±11	1 ±1	0 ±0	0 ±1	4 ±3	2 ±2	7 ±3	2 ±1
antmaze-umaze-v2	55	77	**98**	94	**97**	66 ±5	90 ±6	94 ±3	92 ±6	**96** ±2	**96** ±2
antmaze-umaze-diverse-v2	47	54	84	80	82	66 ±11	55 ±7	82 ±9	62 ±12	**89** ±5	**86** ±9
antmaze-medium-play-v2	0	66	**90**	84	81	49 ±24	52 ±12	77 ±7	56 ±15	78 ±7	81 ±5
antmaze-medium-diverse-v2	1	74	**84**	**85**	75	0 ±1	44 ±15	77 ±6	60 ±25	71 ±13	75 ±8
antmaze-large-play-v2	0	42	52	64	54	0 ±0	10 ±6	32 ±21	55 ±9	**84** ±7	**83** ±6
antmaze-large-diverse-v2	0	30	64	68	54	0 ±0	16 ±10	20 ±17	64 ±8	**83** ±4	**84** ±6
pen-human-v1	71	78	**103**	76 ±10	69 ±7	64 ±8	67 ±5	77 ±7	71 ±12	53 ±6	51 ±10
pen-cloned-v1	52	83	**103**	64 ±7	61 ±7	56 ±10	62 ±10	67 ±9	80 ±11	74 ±11	63 ±9
pen-expert-v1	110	128	**152**	140 ±6	134 ±4	103 ±9	118 ±6	119 ±7	139 ±5	142 ±6	**145** ±6
door-human-v1	2	3	−0	6 ±2	3 ±3	5 ±2	2 ±1	4 ±2	**7** ±2	0 ±0	0 ±0
door-cloned-v1	−0	**3**	0	0 ±0	0 ±0	1 ±0	0 ±1	0 ±0	2 ±2	2 ±1	0 ±0
door-expert-v1	**105**	**107**	**106**	**105** ±1	**105** ±0	98 ±3	**103** ±1	**104** ±1	**104** ±2	**104** ±1	**105** ±1
hammer-human-v1	**3**	2	0	2 ±1	1 ±1	2 ±0	2 ±1	2 ±1	**3** ±1	1 ±1	1 ±1
hammer-cloned-v1	1	2	5	2 ±1	2 ±1	1 ±1	1 ±0	2 ±1	2 ±1	**11** ±9	1 ±1
hammer-expert-v1	127	**129**	**134**	125 ±4	127 ±0	92 ±11	118 ±3	119 ±9	117 ±9	125 ±3	**127** ±9
relocate-human-v1	**0**	**0**	**0**	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0	**0** ±0
relocate-cloned-v1	−0	0	**2**	−0 ±0	−0 ±0	−0 ±0	−0 ±0	1 ±1	−0 ±0	−0 ±0	0 ±0
relocate-expert-v1	**108**	**106**	**108**	**107** ±1	**106** ±2	93 ±6	**105** ±3	**105** ±2	**104** ±3	**107** ±1	**102** ±5
halfcheetah-medium-v2	42	47	**64**	51	**60**	–	–	–	55 ±1	59 ±1	59 ±1
halfcheetah-medium-replay-v2	36	45	**51**	46	**51**	–	–	–	44 ±1	50 ±1	**53** ±1
halfcheetah-medium-expert-v2	56	96	**104**	96	92	–	–	–	79 ±2	89 ±1	96 ±1
hopper-medium-v2	54	59	**102**	65	96	–	–	–	37 ±6	62 ±6	75 ±6
hopper-medium-replay-v2	30	95	95	92	**102**	–	–	–	25 ±1	36 ±6	83 ±6
hopper-medium-expert-v2	52	99	**110**	**109**	100	–	–	–	69 ±2	71 ±11	**110** ±3
walker2d-medium-v2	63	**81**	**86**	**83**	**84**	–	–	–	35 ±2	69 ±5	**81** ±1
walker2d-medium-replay-v2	22	73	**84**	**85**	**85**	–	–	–	13 ±1	39 ±5	58 ±7
walker2d-medium-expert-v2	99	**110**	**112**	**113**	**114**	–	–	–	94 ±8	96 ±4	**110** ±1

B Algorithmic Details

Alg. 1 presents the complete pseudocode for the DriftQL's drift loss. At each training step, a batch of transitions (s, a^+, r, s') is sampled from the offline dataset $\mathcal{D}$. The behavioral regularization term is computed by generating N action samples $\{\hat{a}_i\}$ from the actor network $f_\theta(s, \epsilon_i)$ with i.i.d. noise vectors $\epsilon_i \sim \mathcal{N}(0, I)$. These samples serve a dual role: they are the particles being transported toward the data distribution, and they simultaneously supply the empirical negatives used to compute the repulsive component of the drift field.

The drift vector V_i for each particle decomposes into an attraction term V_i^+, which displaces $\hat{a}_i$ toward the single observed dataset action a^+, and a repulsion term V_i^-, which is a kernel-weighted average of displacements pointing *toward* neighboring generated samples. Subtracting it from V_i^+ produces the net repulsive effect (Eq. 6). The repulsion weights are normalized via a row-wise softmax over the negatives.

The critic is updated via standard Bellman regression with a double ensemble and conservative minimum-Q targets to mitigate overestimation. The actor then jointly minimizes the drift loss $\mathcal{L}_{\text{drift}}$ and maximizes the mean/min ensemble Q-value, with the tradeoff governed by a single scalar α. Fig. 6 shows detailed training checkpoints of the four-Gaussian bandit from Fig. 1, tracking how each method's generated actions evolve over training.

Algorithm 1 Drift Field Computation (actor loss, per training step)

Require: State s, dataset action a^+, policy f_θ, temperature τ, kernel type $\in \{\text{Laplace}, \text{Gaussian}\}$ ▷ *Setup*

draw $\epsilon_1, \ldots, \epsilon_{N_{\text{gen}}} \sim \mathcal{N}(0, I)$ ▷ *Generate policy samples*

$\hat{a}_i \leftarrow \text{clip}\big(f_\theta(s, \epsilon_i), -1, 1\big)$ for each i

$d_{ik}^- \leftarrow \|\hat{a}_i - \hat{a}_k\|_2 / \sqrt{d_a}$ for each $i \neq k$ ▷ *Generated-action distances*

$d_{ii}^- \leftarrow +\infty$ ▷ *Mask self-repulsion*

$\ell_{ik}^- \leftarrow \begin{cases} -d_{ik}^-/\tau, & \text{Laplace}, \\ -(d_{ik}^-)^2/(2\tau^2), & \text{Gaussian} \end{cases}$ ▷ *Kernel logits*

$w_{ik}^- \leftarrow \frac{\exp(\ell_{ik}^-)}{\sum_{k' \neq i} \exp(\ell_{ik'}^-)}$ for $k \neq i$, and $w_{ii}^- \leftarrow 0$ ▷ *Softmax over other generated actions*

$V_i^+ \leftarrow a^+ - \hat{a}_i$ ▷ *Attraction toward the dataset action*

$V_i^- \leftarrow \sum_{k \neq i} w_{ik}^- (\hat{a}_k - \hat{a}_i)$ ▷ *Model-side mean-shift term*

$V_i \leftarrow V_i^+ - V_i^-$ ▷ *Full drift*

$\text{target}_i \leftarrow \text{sg}(\text{clip}(\hat{a}_i + V_i, -1, 1))$ ▷ *Stop-gradient drifted target*

$\mathcal{L}_{\text{drift}} \leftarrow \frac{1}{N_{\text{gen}}} \sum_i \|\hat{a}_i - \text{target}_i\|_2^2$ ▷ *Drift loss*

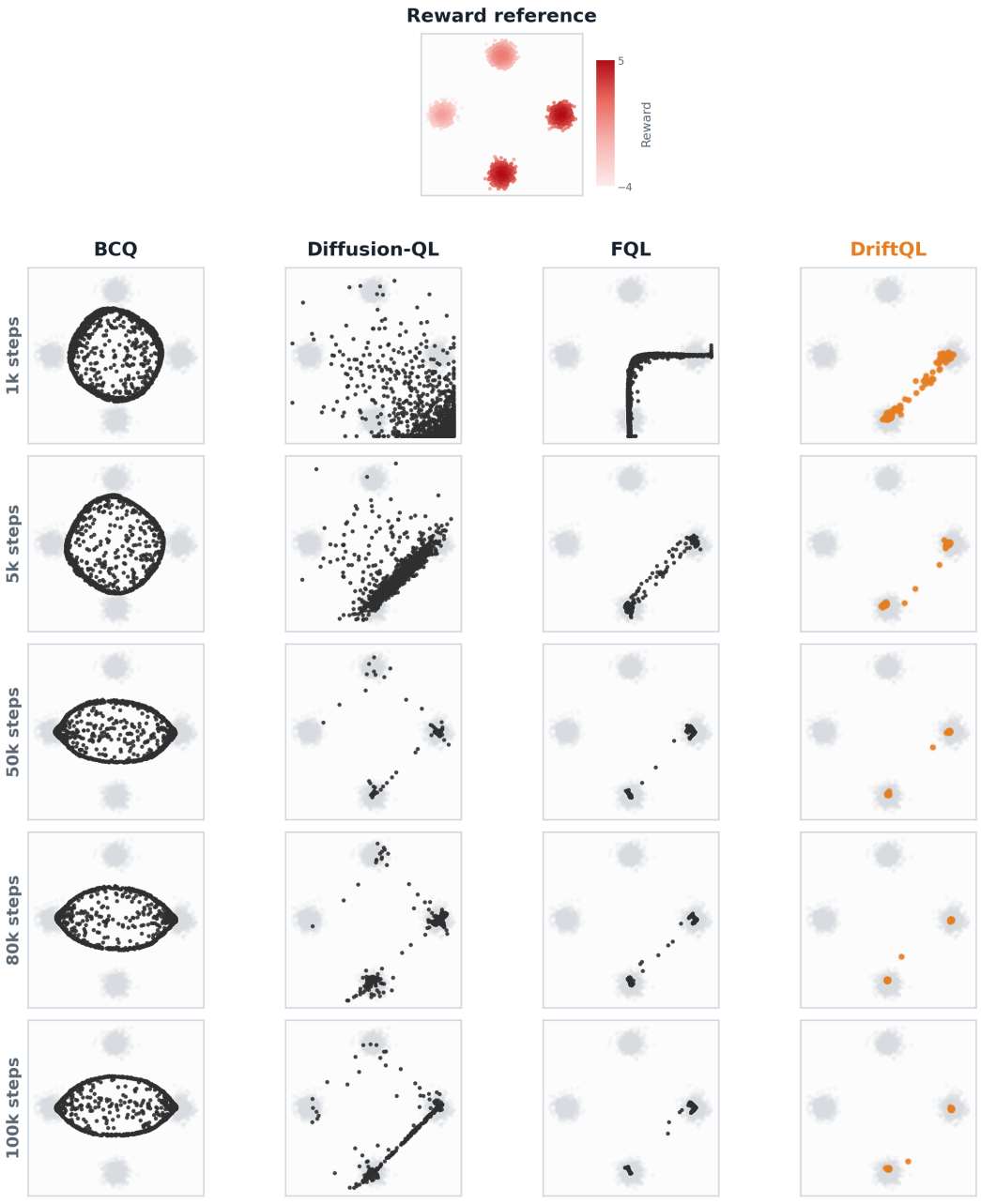

Figure 6: Detailed training checkpoints of the bandit example.

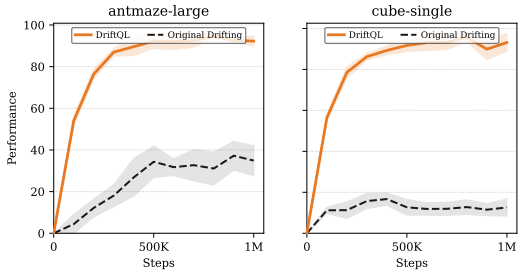

Figure 7: **DriftQL vs. original drifting in the offline RL setting.** Performance on OGBench AntMaze Large and Cube Single (default) over 1M training steps, averaged across three seeds. Shaded regions denote standard deviation.

C DriftQL vs. Original Drifting

This ablation isolates the effect of the drift computation. Both variants share the same state-conditioned actor, critic, Q-maximization term, number of generated samples, and single dataset action a^+ per state. Only the drift regularizer differs.

The original drifting implementation [Deng et al., 2026] pools generated samples, positives, and optional negatives into a joint set, computes symmetrized affinities, cross-weights positive and negative terms, aggregates over several kernel radii, and normalizes the resulting force. DriftQL instead uses the decomposition from Sec. 4: the dataset action serves as a direct anchor $V^+(\hat{a}_i) = a^+ - \hat{a}_i$, while the remaining generated actions estimate the current policy through $V^-(\hat{a}_i) = \sum_{j \neq i} w^-_{ij}(\hat{a}_j - \hat{a}_i)$.

This distinction matters because the drift term is not a standalone training signal: it is paired with a critic that pushes the actor toward high predicted value. As Lai et al. [2026] noted, finite-sample choices such as softmax normalization, positive/negative balance, batch-dependent scaling, and temperature aggregation reshape the effective transport field. The original computation introduces batch- and kernel-dependent rescalings that interact with the Q gradient in hard-to-predict ways. Our formulation removes these, so the behavioral correction is controlled directly by α in Eq. 12.

Fig. 7 shows that our formulation is more suited when combined with Q maximization. We swept over multiple values of α and the kernel temperature for the original drift variant and found significant instability across runs, with performance highly sensitive to both hyperparameters. DriftQL keeps the same attraction-repulsion structure but uses a drift field better suited to low-dimensional, state-conditioned action spaces.

C.1 Anti-symmetry: the critic absorbs the symmetrization machinery

Sec. 4 argues that in offline RL the value gradient, not the symmetrization machinery of Deng et al. [2026], is what anchors the actor, so the machinery can be dropped. We test this prediction directly with a destructive anti-symmetry probe. The attraction V^+ and repulsion V^- are anti-symmetric by construction: at the population fixed point $p = q$ the two cancel. We deliberately break this balance by rescaling the attraction weight by a multiplier $\rho \in [0.01, 50]$ relative to the repulsion (default $\rho = 1$). For $\rho \neq 1$ the field no longer cancels at $p = q$, the finite-sample analogue of the residual ambiguity that Lai et al. [2026] (Thm. 2) flag at the population level. We compare PURE-DRIFT (drift only, no Q) against FULL-DRIFTQL (drift $+Q$), logging divergence when actions exit a fixed bounding box.

Table 4 and Fig. 8 confirm the prediction. PURE-DRIFT is fragile: a small under-weighting of attraction ($\rho \leq 0.2$) makes the actor diverge, because V^- pushes samples apart with nothing to anchor them, exactly the failure the original machinery is built to prevent. FULL-DRIFTQL, by contrast, is essentially flat across more than three orders of magnitude. The Q gradient supplies an independent, value-driven pull toward high-value support, so even when the drift's attraction is artificially crippled the critic keeps the actor from drifting away. This is the empirical content of the claim in Sec. 4: in offline RL the critic already supplies the constraint that cross-weighting,

Table 4: Anti-symmetry stress test. ρ rescales the attraction weight relative to repulsion. PURE-DRIFT diverges or fails to concentrate for $\rho \leq 1$, recovering only when attraction dominates ($\rho \gtrsim 5$). FULL-DRIFTQL is essentially flat across more than three orders of magnitude.

V^+ multiplier ρ	PURE-DRIFT (Y-var)	FULL-DRIFTQL (Y-var)
0.01	*DIVERGED*	0.0011
0.05	*DIVERGED*	0.0009
0.10	*DIVERGED*	0.0006
0.20	*DIVERGED*	0.0006
0.50	0.1523	0.0006
1.00	0.0673	0.0006
2.00	0.0152	0.0010
5.00	0.0009	0.0007
10.00	0.0006	0.0006
20.00	0.0006	0.0006
50.00	0.0005	0.0006

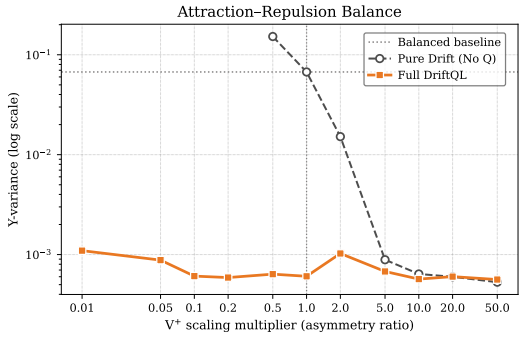

Figure 8: **Anti-symmetry stress test.** Final Y-variance vs. attraction-rescaling multiplier ρ, log–log. The pure-drift baseline is unstable across most of the range. It diverges for $\rho \leq 0.2$ (off-plot) and only stabilizes once attraction strongly dominates repulsion. FULL-DRIFTQL is essentially constant across the entire sweep. The critic gradient supplies a separate value-driven pull that keeps the actor stable even when the drift field is severely mis-balanced.

multi-temperature pooling, and RMS rescaling supply in the unconditional setting, so DriftQL can drop them.

D Network Architectures

To effectively parameterize the drift policy and value functions, we utilize MLPs optimized for continuous control tasks. The architecture consists of a stochastic generator (Actor) and a double-ensemble Q-network (Critic).

As detailed below, the Critic network applies Layer Normalization after each linear transformation to stabilize value estimation and prevent early divergence during offline training. Conversely, the Actor network maps the concatenated state and noise inputs through a deep network without Layer Normalization, culminating in a bounded action output. Both networks rely on a wide and deep structure (4 hidden layers of 512 units) to ensure sufficient expressivity when modeling multimodal behavioral distributions and complex value landscapes.

DriftQL Network Architectures

Inputs:

- Observation $s \in \mathbb{R}^S$
- Action $a \in \mathbb{R}^A$ (Dataset action or generated sample)
- Noise $\epsilon \sim \mathcal{N}(0, I) \in \mathbb{R}^A$

1. Critic (Value) Network $Q_\phi(s, a)$:

```
# Double ensemble architecture (num_ensembles=2)
# Applies LayerNorm to stabilize value bootstrapping
concat = Concatenate([s, a]) # dim: S + A
x_q = Linear(S + A, 512) -> LayerNorm -> GELU
x_q = Linear(512, 512)   -> LayerNorm -> GELU
x_q = Linear(512, 512)   -> LayerNorm -> GELU
x_q = Linear(512, 512)   -> LayerNorm -> GELU
Q   = Linear(512, 1)
```

2. Actor (Generator) Network $f_\theta(s, z)$:

```
# Generates pushforward distribution via noise z
# No LayerNorm applied (actor_layer_norm=False)
concat = Concatenate([s, z]) # dim: S + A
x_a = Linear(S + A, 512) -> GELU
x_a = Linear(512, 512)   -> GELU
x_a = Linear(512, 512)   -> GELU
x_a = Linear(512, 512)   -> GELU
a_raw = Linear(512, A)
action = Clip(a_raw, -1.0, 1.0)
```

E Limitations

We discuss the limitations of DriftQL along three axes: the scope of theoretical support, computational considerations, and the scope of our empirical evaluation.

Theoretical guarantees only partially transfer. The reverse-Fisher interpretation of Lai et al. [2026] is established for the original unconditional drifting objective with coupled cross-normalization between attraction and repulsion. Our construction departs from this in three ways: we condition the drift field on state, we normalize attraction and repulsion independently, and we operate in the single-positive regime. The motivation for these changes is empirical and structural (Sec. 4), not theoretical, and we do not claim that the $O(\tau^4)$ smoothed reverse-Fisher rate or the $O(D^{-(1+2a)})$ minimizer discrepancy bound carry over unchanged. Extending the analysis of Lai et al. [2026] to conditional, single-positive drift fields optimized jointly with a learned critic is an open problem.

Training-time cost offsets the inference-time gain. Although DriftQL requires only a single forward pass at evaluation, each training step draws N generator samples per state and computes an $N \times N$ pairwise kernel (Alg. 1). However, wall-clock training time is close to baselines. We found $N = 32$ to be a reasonable operating point, but this trades sample diversity for throughput and has not been tuned for wall-clock parity against baselines.

Scope of action spaces. The drift field as formulated assumes a continuous, Euclidean, box-bounded action space. We rely on the $\sqrt{d_a}$-scaled Euclidean kernel and a terminal $\mathrm{clip}(\cdot, -1, 1)$. Other types of action spaces are not supported without a redesign of both the kernel and the clipping step.

F Hyperparameters

In this section, we detail the environment-specific hyperparameters used for evaluating DriftQL. To prioritize simplicity and minimize tuning overhead, the majority of our algorithmic configurations, such as network architectures, learning rates, optimization parameters, and the number of generated drift samples (N), remain entirely fixed across all benchmark tasks, as defined in our core implementation. Table 5 lists these fixed default hyperparameters shared across all environments.

Table 5: Default hyperparameters for DriftQL fixed across all tasks.

Hyperparameter	**Value**	***Drift Field Configurations***	**Value**
Optimizer	Adam	Number of generated samples (N)	32
Learning rate	3×10^{-4}	Drift batch size	256
Minibatch size	256	Drift step size (η)	1.0
Actor MLP dimensions	$[512, 512, 512, 512]$	Distance dimensionality scaling	True
Critic MLP dimensions	$[512, 512, 512, 512]$	Drift normalize	False
Nonlinearity	GELU	Drift ϵ	10^{-12}
Actor layer normalization	False		
Critic layer normalization	True		
Target network update rate (τ)	0.005		
Normalize Q-loss	False		

However, to account for the highly varied reward scales, dataset suboptimalities, and dimensionalities across the OGBench and D4RL suites, we tune three hyperparameters for each environment category. These parameters are:

- **`drift_temp`** (τ): The temperature scalar controlling the sharpness of the similarity kernel in the drift field computation.
- α: The trade-off coefficient balancing the drifting behavioral cloning loss against the Q-maximization objective.
- **`kernel`**: The distance kernel used to compute the drift field logits (either `laplace` or `gaussian`).

Table 6 summarizes the exact hyperparameter configurations used for each task domain. For OGBench environments, we follow the standard evaluation protocol by tuning these parameters on the default task (indicated by (*)) and applying the best configuration to the remaining four tasks within that environment suite.

Table 6: Task-specific hyperparameters for DriftQL.

Task Domain	`drift_temp` (τ)	α	`kernel`
OGBench Locomotion			
`antmaze-large-navigate-*`	0.5	10	
`antmaze-giant-navigate-*`	0.2	10	
`humanoidmaze-medium-navigate-*`	0.5	65	laplace
`humanoidmaze-large-navigate-*`	0.2	32	
`antsoccer-arena-navigate-*`	0.5	10	
OGBench Manipulation			
`cube-single-play-*`	0.02	60	
`cube-double-play-*`	0.2	100	
`scene-play-*`	0.2	250	gaussian
`puzzle-3x3-play-*`	0.5	50	
`puzzle-4x4-play-*`	0.8	300	
D4RL Antmaze			
`antmaze-umaze-v2`		15	laplace
`antmaze-umaze-diverse-v2`		12	laplace
`antmaze-medium-play-v2`		5	gaussian
`antmaze-medium-diverse-v2`	0.5	8	gaussian
`antmaze-large-play-v2`		3	gaussian
`antmaze-large-diverse-v2`		5	gaussian
D4RL Adroit			
`pen-cloned-v1`	0.05	1500	gaussian
`pen-expert-v1`	0.9	2000	laplace
`pen-human-v1`	0.05	2000	laplace
`door-*-v1`	0.2	4500	laplace
`hammer-*-v1`	0.05	2500	laplace
`relocate-*-v1`	0.2	5000	laplace
D4RL Locomotion			
`halfcheetah-medium-expert-v2`	0.5	300	laplace
`halfcheetah-medium-replay-v2`	0.5	10	laplace
`halfcheetah-medium-v2`	0.5	3	laplace
`hopper-medium-expert-v2`	0.1	600	gaussian
`hopper-medium-replay-v2`	0.1	100	gaussian
`hopper-medium-v2`	0.1	100	gaussian
`walker2d-medium-expert-v2`	0.1	1000	laplace
`walker2d-medium-replay-v2`	0.1	300	laplace
`walker2d-medium-v2`	0.1	1000	laplace