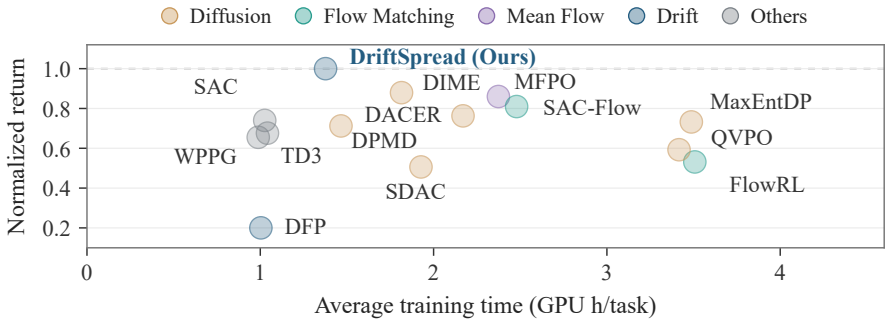
DRIFT, THEN SPREAD: LEARNING ONE-STEP GENERATIVE POLICIES FOR ONLINE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative policies offer expressive action distributions for online reinforcement learning (RL), but many diffusion- and flow-based policies require costly iterative sampling. We propose DRIFTSPREAD, a one-step generative policy based on Drifting for online maximum-entropy (MaxEnt) RL. To incorporate entropy without a tractable policy density, we build on an energy-based view of Drifting and derive critic-guided Drift and entropy-induced Spread from the negative MaxEnt objective. Our method learns these updates sequentially in the same actor, using action regression for value improvement and distribution matching to capture the effect of entropy. The actor updates require neither direct samples from the critic-defined Boltzmann target nor policy-density or score estimation. Under exact fitting with a fixed critic, we prove convergence to the Boltzmann target in the small-step limit followed by the long-time limit. On MuJoCo, DRIFTSPREAD achieves the highest aggregate normalized return among all evaluated methods, while outperforming the diffusion- and flow-based baselines in both training and inference efficiency.

1 INTRODUCTION



(a) Normalized return versus average training time.

Method	SAC	TD3	WPPG	**Ours**	DFP	FlowRL	DACER	MFPO	MaxEnt DP	SDAC	DPMD	SAC-Flow	DIME	QVPO
Steps	1	1	1	**1**	1	1	20	2	20	20	20	4	16	20
ms/action	0.10	0.10	0.17	**0.15**	0.16	0.35	0.43	0.36	0.57	0.85	0.87	1.20	1.24	1.83

(b) Sampling steps and inference time.

Figure 1: **Normalized return versus average training time (a), and average inference time per action (b), across five MuJoCo tasks.**

Generative policies offer expressive action distributions for online reinforcement learning (RL). Diffusion- and flow-based policies can capture multimodal distributions that are difficult to represent with a single Gaussian (Dong et al., 2025; Celik et al., 2025). However, many of these approaches generate actions through iterative sampling, requiring repeated network evaluations during training

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

and inference. Recent MeanFlow-based methods reduce this cost through few-step or one-step action generation (Dong et al., 2026; Chen et al., 2025; Kim et al., 2026).

Drifting offers a different way to train one-step generators (Deng et al., 2026). At each training iteration, it uses target data to move generated samples toward the desired distribution, then trains the generator to reproduce the moved samples from the same inputs. Through these repeated updates, the generator learns to produce samples from the target distribution in a single forward pass at inference. This approach has shown better performance than MeanFlow variants in one-step image generation on ImageNet (Deng et al., 2026). These results motivate exploring Drifting for online RL, where its use remains less studied.

We consider the maximum-entropy (MaxEnt) RL objective, which balances expected return with policy entropy to encourage exploration (Haarnoja et al., 2018). The critic defines a Boltzmann target distribution for policy improvement, but does not directly provide samples from it (Li et al., 2026). We therefore adopt an energy-based view of Drifting (Gretton et al., 2026; Han et al., 2026), using the negative MaxEnt policy-improvement objective as the energy. This energy guides updates to actions generated by the current policy. For the value term, critic gradients tell us how to move actions toward higher values. The entropy term is harder to use: an implicit actor generates actions but does not provide a tractable density for computing their distribution's entropy. We therefore need training targets that capture the effect of entropy without explicitly computing it.

We propose DRIFTSPREAD, a one-step generative policy in the Drifting model family for online MaxEnt RL. Our key idea is to derive two updates from this energy and fit them sequentially to the same actor through repeated Drift–Spread cycles. In each cycle, Drift uses critic gradients to move generated actions toward higher values. The actor then learns to reproduce these updated actions from the same inputs through regression. Spread captures the entropy contribution through diffusion of the action distribution. We approximate this diffusion by adding appropriately scaled independent noise to actions from the updated actor, then train the actor to match the resulting distribution. This allows the actor to learn the effect of entropy without explicitly computing it. For a fixed critic, we prove that the learned policy converges to the Boltzmann target distribution in the small-step, long-time limit.

Our main contributions are: (1) We introduce DRIFTSPREAD, a Drifting model for online MaxEnt RL with expressive policies and one-step action generation. (2) We develop sequential Drift–Spread training through critic-guided regression and entropy regularization to address the challenge of entropy computation for implicit policies. (3) On MuJoCo, DRIFTSPREAD outperforms 13 baselines in aggregate normalized return, with lower training costs than the diffusion, flow, and MeanFlow baselines (Figure 1).

2 RELATED WORK

Generative policies for online RL. Diffusion and flow models provide expressive action distributions for online RL. One line of work optimizes policies by backpropagating critic gradients through diffusion or flow sampling (Wang et al., 2024; Lyu et al., 2025). DIME also differentiates through diffusion sampling while optimizing a variational lower bound on the MaxEnt objective (Celik et al., 2025).

Another line of work constructs training targets for learning critic-defined Boltzmann distributions. DQS uses weighted averages of critic gradients to construct score targets (Jain et al., 2025). SDAC and MaxEntDP instead use critic values to weight denoising supervision and noise estimates, respectively (Ma et al., 2025; Dong et al., 2025). Reverse flow matching unifies these gradient-expectation and noise-expectation formulations through posterior-mean estimation (Li et al., 2026).

These approaches use iterative action sampling, motivating few-step and one-step policies for online RL. MeanFlow was introduced for one-step generative modeling and learns average velocities (Geng et al., 2025). Recent online RL methods use this formulation to learn policies with fewer generation steps. FPMD trains rectified-flow and MeanFlow policies through policy mirror descent, with both variants supporting one-step inference (Chen et al., 2025). MFPO trains few-step MeanFlow policies using velocity estimates for critic-defined Boltzmann targets (Dong et al., 2026), while SOM constructs velocity targets from critic-based score estimates to train one-step MeanFlow policies (Kim et al., 2026).

Drifting models and policy learning. Drifting learns one-step generators by repeatedly moving generated samples and training the generator to reproduce their updated positions. This approach has achieved state-of-the-art performance in one-step image generation on ImageNet (Deng et al., 2026). Subsequent work relates specific Drifting fields to Wasserstein gradient flows (Gretton et al., 2026) and uses optimal-transport couplings to construct sample updates (He et al., 2026). W-Flow derives sample movements from an objective over distributions and trains a one-step generator on the resulting samples (Han et al., 2026).

Drifting has also been applied to policy learning. For offline RL, DriftQL combines critic-guided policy improvement with a Drifting regularizer based on the behavior distribution (Houssaini et al., 2026). For offline-to-online RL, Drifting Field Policy derives a KL-regularized policy update through Wasserstein gradient flow and constructs training targets from actions ranked by the critic (Koo et al., 2026). Our work focuses on online MaxEnt RL from scratch.

3 PRELIMINARIES

3.1 MAXIMUM-ENTROPY RL AND SOFT POLICY ITERATION

We consider a discounted Markov decision process (MDP) with state space $\mathcal{S}$, continuous action space $\mathcal{A} \subseteq \mathbb{R}^d$, transition kernel $P(\cdot \mid s, a)$, reward $r(s, a)$, and discount factor $\gamma \in [0, 1)$. A policy $\pi(a \mid s)$ specifies a density over actions at each state. Maximum-entropy (MaxEnt) RL learns a policy that maximizes discounted rewards together with policy entropy:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \big(r(s_t, a_t) + \lambda \mathcal{H}(\pi(\cdot \mid s_t)) \big) \right], \tag{1}$$

where $\mathcal{H}(\pi(\cdot \mid s)) = -\mathbb{E}_{a \sim \pi(\cdot|s)}[\log \pi(a \mid s)]$ is the conditional differential entropy, and $\lambda > 0$ controls the tradeoff between reward and action diversity. The expectation is over trajectories generated by π from a fixed initial-state distribution.

Soft policy iteration alternates between policy evaluation and policy improvement (Haarnoja et al., 2018). Evaluation holds π fixed and computes its soft action-value function Q^π, which accounts for rewards and future entropy bonuses. This function satisfies $Q^\pi = \mathcal{T}^\pi Q^\pi$, where the soft Bellman operator is

$$(\mathcal{T}^\pi Q)(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \left[\mathbb{E}_{a' \sim \pi(\cdot|s')}[Q(s', a')] + \lambda \mathcal{H}(\pi(\cdot \mid s')) \right], \tag{2}$$

where $s' \sim P(\cdot \mid s, a)$.

Improvement then holds the critic Q fixed and updates the policy at each state to maximize $\mathbb{E}_{a \sim \pi(\cdot|s)}[Q(s, a)] + \lambda \mathcal{H}(\pi(\cdot \mid s))$. When optimizing over all action densities, the maximizing policy has the Boltzmann form

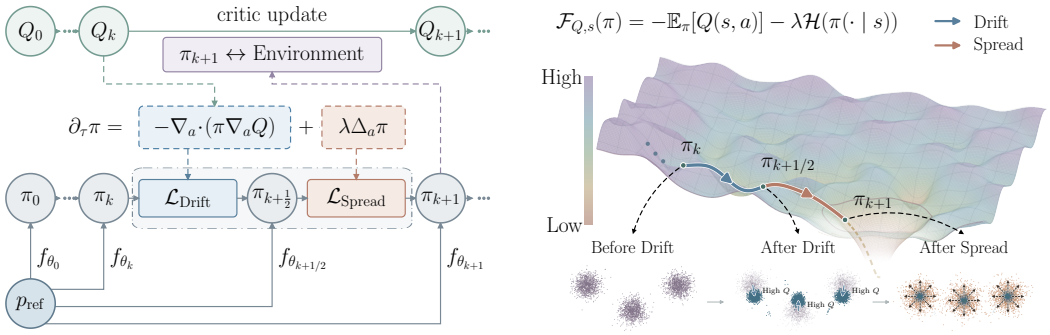
$$\pi^*_Q(a \mid s) = \frac{\exp(Q(s, a)/\lambda)}{Z_Q(s)}, \qquad Z_Q(s) = \int_{\mathcal{A}} \exp(Q(s, a)/\lambda)\, da, \tag{3}$$

assuming $0 < Z_Q(s) < \infty$. Thus, each fixed critic defines a target action distribution for policy improvement. The practical task is to learn a policy that approximates this distribution and generates actions efficiently.

3.2 DRIFTING MODELS AND WASSERSTEIN GRADIENT FLOWS

Drifting trains a one-step generator by repeatedly moving its samples and fitting the generator to their new positions (Deng et al., 2026). The generated distribution evolves during training, while sampling requires only one forward pass.

The original Drifting field combines attraction toward data samples with repulsion from generated samples. Energy-based extensions start from an objective $\mathcal{F}(q)$ over the generated distribution q. For example, $\mathcal{F}(q)$ can measure the difference between q and a target distribution. They use a Wasserstein gradient flow to derive sample movements that reduce this objective (Gretton et al., 2026; Han et al., 2026). This flow describes continuous-time steepest descent of $\mathcal{F}$ under the Wasserstein distance



(a) (b)

Figure 2: **Overview of** DRIFTSPREAD**.** (a) Training alternates between critic learning and sequential Drift–Spread actor updates, using action regression for Drift and distribution matching for Spread. The maps f_θ represent the same actor at different training stages, each generating an action sample from reference noise in one forward pass. (b) The surface schematically represents the negative MaxEnt objective over policy distributions. The samples below illustrate how Drift moves actions toward higher critic values and Spread then broadens the post-Drift distribution to capture the effect of entropy.

(Jordan et al., 1998; Ambrosio et al., 2008). The first variation $\delta\mathcal{F}/\delta q$ describes how $\mathcal{F}$ responds to small changes in q; its negative spatial gradient gives the sample velocity. Under suitable regularity and decay conditions,

$$V_q(x) = -\nabla_x \frac{\delta \mathcal{F}}{\delta q}(q)(x), \qquad \partial_\tau q_\tau + \nabla_x \cdot (q_\tau V_{q_\tau}) = 0. \tag{4}$$

Here, τ denotes continuous flow time. The first equation specifies how individual samples move, and the second describes how their density changes as a result.

To train a generator with this velocity, Han et al. (2026) use a forward Euler step to construct targets for fitting. At iteration k, let q_k denote the distribution of $x = f_{\theta_k}(z)$, where $z \sim p_{\text{ref}}$. When the velocity can be evaluated at q_k, setting $V_k = V_{q_k}$ with a step size $\eta > 0$ gives

$$\widetilde{x} = \text{sg}[x + \eta V_k(x)], \qquad f_{\theta_{k+1}}(z) \approx \widetilde{x}. \tag{5}$$

The generator is fitted using the same input z, while sg keeps the target fixed. The generator thus learns to reproduce the moved samples in a single forward pass.

4 METHOD

DRIFTSPREAD learns a one-step generative policy through two sequential updates to the same actor: critic-guided Drift and entropy-induced Spread. We first derive these updates from the MaxEnt objective (section 4.1), then show how to train the actor using action regression for Drift and distribution matching for Spread (section 4.2). We next analyze fixed-critic convergence under exact fitting (section 4.3), and finally describe the practical training procedure (section 4.4). Figure 2 illustrates the overall method.

4.1 DECOMPOSING MAXENT POLICY IMPROVEMENT

The Drifting framework updates a generator by fitting its outputs to targets displaced by a velocity field. For MaxEnt policy improvement, we derive this field from the negative MaxEnt objective. For a fixed state s, critic Q, and temperature $\lambda > 0$, define the distributional energy

$$\mathcal{F}_{Q,s}(\pi) = -\mathbb{E}_\pi[Q(s,a)] - \lambda \mathcal{H}(\pi(\cdot \mid s)).$$

For smooth positive densities with sufficient decay and finite second moments, the energy–velocity relation in equation (4) gives

$$V_\pi(a \mid s) = -\nabla_a \frac{\delta \mathcal{F}_{Q,s}}{\delta \pi}(a) = \nabla_a Q(s,a) - \lambda \nabla_a \log \pi(a \mid s).$$

The critic gradient is directly available, but the entropy contribution depends on the policy score $\nabla_a \log \pi$, which an implicit actor does not provide directly.

To implement the entropy contribution without estimating this score, we consider the corresponding evolution of the policy distribution. Substituting V_π into the continuity equation gives

$$\partial_\tau \pi = \underbrace{-\nabla_a \cdot (\pi \nabla_a Q)}_{\text{critic-guided drift}} + \underbrace{\lambda \Delta_a \pi}_{\text{entropy-induced spreading}} . \tag{6}$$

The drift term transports actions toward higher critic values. We approximate this transport by an explicit step along the critic gradient.

The spreading term describes diffusion. When this diffusion acts alone, it preserves the mean and increases the variance of each action coordinate by $2\lambda\eta$ over a time interval η. We approximate its effect over a small time step by adding independent noise with the corresponding scale:

$$\epsilon_\eta = \sqrt{2\lambda\eta}\,\xi, \qquad \mathbb{E}[\xi] = 0, \qquad \mathbb{E}[\xi\xi^\top] = I_d.$$

Here, ξ follows a fixed noise distribution, such as Gaussian, Gamma, or uniform noise, standardized to satisfy the moment conditions above.

Let $K_{\lambda,\eta}$ denote the distribution of ϵ_η. Starting from $a \sim \pi_k(\cdot \mid s)$, an ideal cycle applies Drift followed by Spread:

$$a \xrightarrow{\text{Drift}} a + \eta \nabla_a Q(s,a) \xrightarrow{\text{Spread}} a + \eta \nabla_a Q(s,a) + \epsilon_\eta,$$

where $\epsilon_\eta \sim K_{\lambda,\eta}$ is independent of a.

4.2 Learning Drift and Spread sequentially

As illustrated in Figure 2, we update the same actor in two stages, using action regression for Drift and distribution matching for Spread: $\theta_k \xrightarrow{\text{Drift}} \theta_{k+1/2} \xrightarrow{\text{Spread}} \theta_{k+1}$. The critic remains fixed throughout each cycle.

Learning Drift from the critic. Following Drifting (Deng et al., 2026), we turn the critic-gradient step into a regression target. For each input (s, z), we displace the current actor's output along the critic gradient and fit the resulting target using the same input:

$$\widetilde{a}_{k+1/2}(s,z) = \mathrm{sg}\big[f_{\theta_k}(s,z) + \eta \nabla_a Q\big(s, f_{\theta_k}(s,z)\big)\big], \tag{7}$$

$$\mathcal{L}_{\text{Drift}}(\theta;\theta_k) = \tfrac{1}{2}\mathbb{E}_{s,z}\left\| f_\theta(s,z) - \widetilde{a}_{k+1/2}(s,z)\right\|^2. \tag{8}$$

Here, states s are sampled from the replay buffer and $z \sim p_{\text{ref}}$. The stop-gradient operation sg holds the target fixed during fitting. Starting from θ_k, this fit produces the post-Drift actor $f_{\theta_{k+1/2}}$.

Learning Spread through distribution matching. Spread trains the actor to represent the distribution of $a + \epsilon_\eta$, where $a = \mathrm{sg}[f_{\theta_{k+1/2}}(s,z)]$. Let $\widetilde{\pi}_{k+1}(\cdot \mid s)$ denote this target distribution. Because the noise is independent and has zero mean, paired regression using the same input z gives

$$\mathbb{E}_{z,\epsilon_\eta}\|f_\theta(s,z) - a - \epsilon_\eta\|^2 = \mathbb{E}_z\|f_\theta(s,z) - a\|^2 + \mathbb{E}\|\epsilon_\eta\|^2.$$

The noise contributes only a constant, so the post-Drift actor already minimizes this loss. To learn the spreading effect, we instead match the actor and target distributions.

Our actor generates samples but does not provide a tractable action density, making direct KL minimization difficult. We therefore use squared maximum mean discrepancy (MMD), which can be estimated from samples without evaluating their densities (Li et al., 2015):

$$\mathcal{L}_{\text{Spread}}(\theta;\theta_{k+1/2}) = \mathbb{E}_s\,\mathrm{MMD}_h^2\big(\pi_\theta(\cdot \mid s), \widetilde{\pi}_{k+1}(\cdot \mid s)\big), \tag{9}$$

where h is a positive-definite kernel on the action space. The target distribution is held fixed during this update.

We choose a kernel with a frequency representation and compute this objective using its equivalent expression in terms of characteristic functions (CFs) (Ansari et al., 2020). Define the actor's CF

as $\varphi_\theta(\omega \mid s) := \mathbb{E}_z[e^{i\omega^\top f_\theta(s,z)}]$, where $\omega \in \mathbb{R}^d$. Since the spreading noise is independent of the post-Drift action, the target CF is

$$\widetilde{\varphi}_{k+1}(\omega \mid s) = \widehat{K}_{\lambda,\eta}(\omega)\varphi_{\theta_{k+1/2}}(\omega \mid s), \tag{10}$$

where $\widehat{K}_{\lambda,\eta}$ is the known noise CF. We estimate the post-Drift actor's CF from its samples and multiply it by this factor. For Gaussian spreading noise $K_{\lambda,\eta} = \mathcal{N}(0, 2\lambda\eta I_d)$, $\widehat{K}_{\lambda,\eta}(\omega) = e^{-\lambda\eta\|\omega\|^2}$. Appendix A.2.3 gives the CFs for the other noise choices.

Starting from $\theta_{k+1/2}$, we obtain θ_{k+1} by minimizing the following objective with the target CF held fixed:

$$\mathcal{L}_{\text{Spread}}(\theta; \theta_{k+1/2}) = \mathbb{E}_s\mathbb{E}_{\omega\sim\rho}\left|\varphi_\theta(\omega \mid s) - \text{sg}\left[\widetilde{\varphi}_{k+1}(\omega \mid s)\right]\right|^2. \tag{11}$$

We estimate this loss using states sampled from the replay buffer, generated actions, and frequencies drawn from ρ. The actor and target CFs are evaluated at the same frequencies.

Here, ρ determines the kernel through $h(x, y) = \mathbb{E}_{\omega\sim\rho}[\cos(\omega^\top(x - y))]$. When $\rho = \mathcal{N}(0, \ell^{-2}I_d)$, this objective equals squared MMD with the Gaussian kernel $h(x, y) = \exp(-\|x - y\|^2/(2\ell^2))$. We use this choice in our experiments.

Why sequential fitting? Sequential fitting first minimizes the Drift regression loss $\mathcal{L}_{\text{Drift}}$, then constructs and fits the Spread target using the updated actor. A joint alternative constructs both targets from the actor before Drift and minimizes $\mathcal{L}_{\text{Drift}} + \beta\mathcal{L}_{\text{Spread}}$, where $\beta > 0$ weights the Spread MMD loss. Because regression and MMD penalize changes in the policy differently, a single weight need not recover the intended balance between Drift and Spread across action coordinates. Figure 3 illustrates this issue in a two-dimensional Gaussian example. After repeated fitting, Sequential closely matches the Boltzmann target, whereas Joint remains too narrow in one coordinate and too wide in the other, despite tuning its weight. Appendix C analyzes this bias.

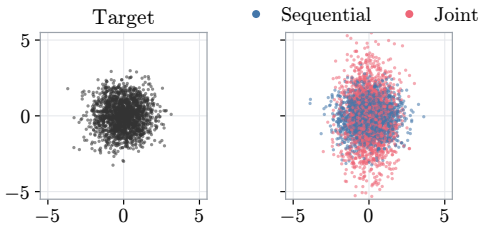


Figure 3: **Sequential versus joint fitting for a Gaussian Boltzmann target.** Left: target samples. Right: samples from the fitted policies. Joint's loss weight is tuned to minimize KL divergence to the known target.

4.3 Theoretical analysis

We next show that sequential fitting can recover the Boltzmann target for a fixed critic. Under exact fitting, Drift reproduces the critic-guided action update, and Spread matches the distribution obtained by adding noise to the post-Drift actions. Each cycle therefore implements the distribution update derived in section 4.1.

Fix a state s, critic Q, and temperature $\lambda > 0$. Let $\pi_k^{(\eta)}$ denote the actor's policy after k cycles with step size η, starting from the same initial policy π_0 for every η. To compare different step sizes at a common evolution time τ, we take $k = \lfloor \tau/\eta \rfloor$.

Theorem 4.1 (Boltzmann convergence)**.** *Under the normalizability assumption $0 < Z_Q(s) < \infty$, suppose that (i) the population Drift regression loss $\mathcal{L}_{\text{Drift}}$ and Spread MMD loss $\mathcal{L}_{\text{Spread}}$ at state s are zero after their respective updates in every cycle, and ρ has full support on $\mathbb{R}^d$; (ii) $Q(s, \cdot)$ is twice continuously differentiable and satisfies $\langle a, \nabla_a Q(s, a)\rangle \le C(1 + \|a\|^2)$ for all a and some finite $C \ge 0$, which prevents the continuous dynamics from escaping to infinity in finite time; and (iii) the Spread noise is $\epsilon_\eta = \sqrt{2\lambda\eta}\,\xi$, where ξ has an η-independent distribution with $\mathbb{E}[\xi] = 0$, $\mathbb{E}[\xi\xi^\top] = I_d$, and $\mathbb{E}\|\xi\|^3 < \infty$, with draws independent across cycles and independent of the current action. Then the fitted policies satisfy*

$$\lim_{\tau\to\infty}\lim_{\eta\downarrow 0}\pi^{(\eta)}_{\lfloor\tau/\eta\rfloor} = \pi^*_Q \qquad \textit{weakly}. \tag{12}$$

The full-support condition on ρ ensures that zero Spread loss implies an exact match to its target distribution. The two limits are taken in order: first, taking $\eta \downarrow 0$ at fixed τ recovers the MaxEnt diffusion in equation (6); then, letting $\tau \to \infty$ brings this diffusion to the Boltzmann target. The proof follows from theorem B.2 in the appendix.

4.4 Implementation Details

We implement DriftSpread as an off-policy actor–critic algorithm that reuses past transitions stored in a replay buffer. Training combines data collection, critic learning, and sequential Drift–Spread actor updates.

Data collection. At each environment step, we sample an action $a \sim \pi_\theta(\cdot \mid s)$ from the current actor and execute it in the environment. We store the resulting transition (s, a, r, s', δ) in the replay buffer for subsequent critic and actor updates.

Critic training. Following prior generative-policy methods (Jain et al., 2025; Li et al., 2026), we use standard critic targets without an entropy term, while retaining entropy regularization in the actor updates. We maintain two critics and use the smaller of their target values to reduce overestimation (Fujimoto et al., 2018). For a replay transition (s, a, r, s', δ), both critics minimize the squared error to the detached target $y = r + \gamma(1-\delta)\min_{i=1,2} \bar{Q}_i(s', a')$, where $a' \sim \pi_{\bar{\theta}}(\cdot \mid s')$. Here, $\bar{Q}_i$ and $\pi_{\bar{\theta}}$ denote the target critics and target policy, respectively, and δ indicates termination.

Actor training. After each critic update, we hold the critics fixed and perform one Drift–Spread cycle using states sampled from the replay buffer. For these states, the current actor generates the action samples used to construct the Drift targets. We take one optimizer step on the Drift regression loss $\mathcal{L}_{\text{Drift}}$. We then construct the Spread target from the updated actor and take one optimizer step on the Spread MMD loss $\mathcal{L}_{\text{Spread}}$.

The full training procedure is provided in Appendix A.3.

5 Experiments

We first use a toy experiment to examine the role of Spread in learning a Boltzmann target distribution. We then compare control performance and computational efficiency on MuJoCo. We also investigate how the choice of spreading noise and sequential versus joint fitting affect policy learning.

5.1 Toy Experiment

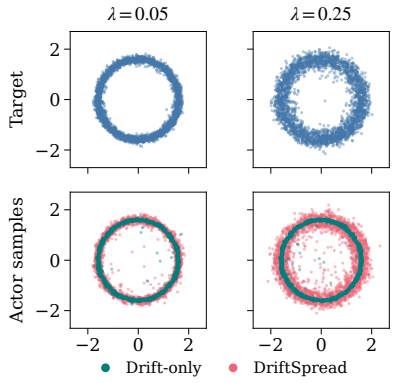


Figure 4: **Learning a ring-shaped Boltzmann distribution.**

We first evaluate DriftSpread on a two-dimensional Boltzmann target $\pi^*_{Q,\lambda}(a) \propto \exp(Q(a)/\lambda)$, with $Q(a) = -(\|a\|_2^2 - r_0^2)^2/(4\sigma_Q^2)$. The target concentrates around a ring of radius r_0, with a width that increases with temperature λ. To assess the role of Spread, we compare DriftSpread with Drift-only using the same MLP actor architecture. Neither method uses samples from the target distribution during training.

Figure 4 shows target samples in blue in the top row, and samples from Drift-only (teal) and DriftSpread (red) in the bottom row. At the higher temperature, the blue target samples spread more widely around the ring. The teal samples from Drift-only remain tightly concentrated near the ring at both temperatures. By contrast, the red samples from DriftSpread capture the wider spread of the higher-temperature target, showing the effect of Spread on the learned distribution.

5.2 MuJoCo Benchmarks

Experimental setup. We evaluate the control performance and computational efficiency of DriftSpread on five MuJoCo-v3 tasks: HalfCheetah, Hopper, Walker2d, Ant, and Humanoid. For each task, every method is trained for one million environment interactions using five random seeds.

We use SAC (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) as standard actor–critic baselines. To compare with more expressive generative policies, we include diffusion-based methods: DIME

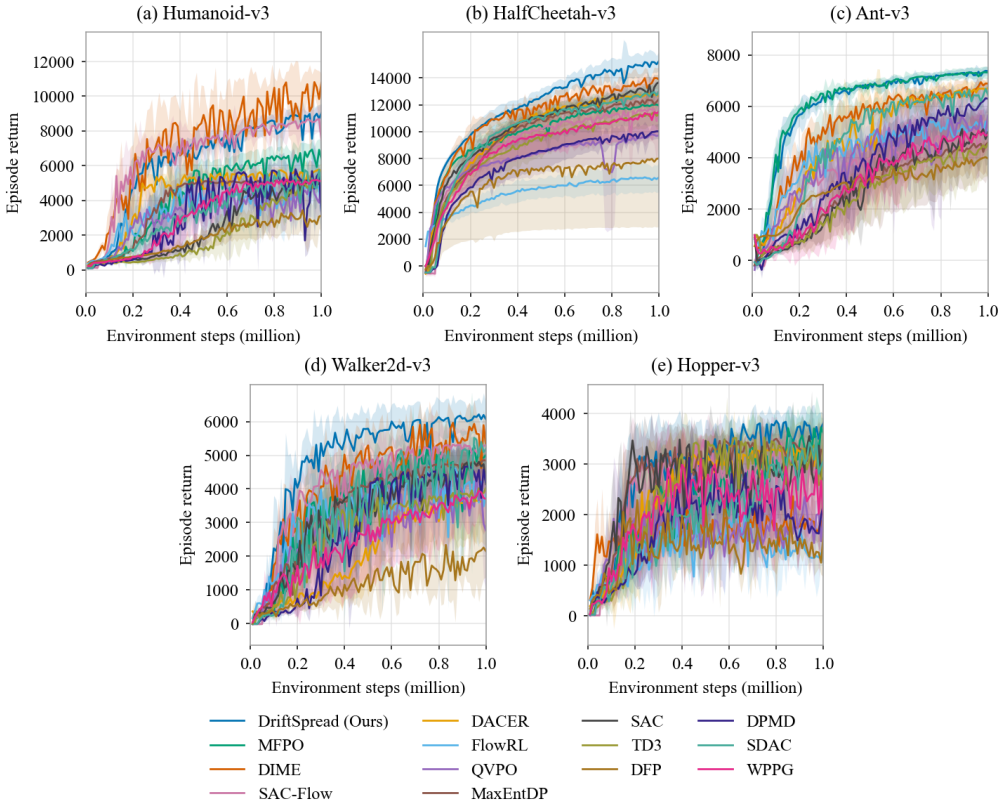


Figure 5: **Comparison with 13 baselines on five MuJoCo-v3 locomotion tasks.** Curves show five-seed mean evaluation returns with sample standard deviations shaded and no smoothing.

(Celik et al., 2025), DACER (Wang et al., 2024), QVPO (Ding et al., 2024), MaxEntDP (Dong et al., 2025), and DPMD and SDAC (Ma et al., 2025), as well as the flow-based methods FlowRL (Lyu et al., 2025) and SAC-Flow (Zhang et al., 2026). To compare with policies designed for efficient action generation, we include MeanFlow-based MFPO (Dong et al., 2026), which generates actions in a few steps. We further include DFP (Koo et al., 2026) and WPPG (Zhu et al., 2026), two recent one-step policies based on Wasserstein gradient flows.

To keep these comparisons consistent, we align batch size, network capacity, and update-to-data ratio across methods. We implement DRIFTSPREAD in JAX and use the same framework for all baselines.[1] Training and inference time are measured on the same NVIDIA RTX 4090 GPU. Full configurations are provided in Appendix E.

Performance and efficiency. Figure 1 shows that DRIFTSPREAD achieves the highest aggregate normalized return among the compared methods, with lower training costs than the diffusion, flow, and MeanFlow baselines. It also reduces average inference time to 0.15 ms per action, compared with 1.24 ms for DIME and 0.36 ms for the two-step MeanFlow policy MFPO. These comparisons show improvements in both control performance and computational efficiency.

Among one-step generative policies, DRIFTSPREAD achieves higher aggregate returns than DFP and WPPG with similar inference times, although these baselines require slightly less training time. DFP was originally designed for offline-to-online fine-tuning (Koo et al., 2026), whereas our experiments evaluate online learning from scratch. WPPG fits noisy action targets through paired regression using the same generator inputs (Zhu et al., 2026). This regression averages out independent zero-mean

[1]For baselines originally implemented in PyTorch, we port their official implementations to JAX for consistent runtime comparisons.

target noise in expectation. Our Spread update instead learns this distribution through distribution matching. This difference may contribute to the observed performance gap.

The per-task learning curves (Figure 5) complement these aggregate results. DRIFTSPREAD achieves the highest late-training mean returns on HalfCheetah-v3 and Walker2d-v3, while remaining competitive with the strongest baselines on the other tasks.

Impact of Spreading Noise. We examine how the choice of noise in Spread affects policy learning. We compare Gaussian noise with the Gamma-mixture, Resolvent, and Uniform alternatives defined in Appendix A.2.3. All four choices have zero mean and the same covariance. We keep the actor architecture and training procedure unchanged within each task.

Figure 6 shows that Gaussian noise achieves the highest or near-highest late-training mean returns across the three tasks. On Humanoid-v3 and HalfCheetah-v3, Gaussian reaches higher returns than Gamma and Uniform, while Resolvent remains closer to Gaussian. The differences are smaller on Ant-v3. These results support Gaussian noise as our default choice.

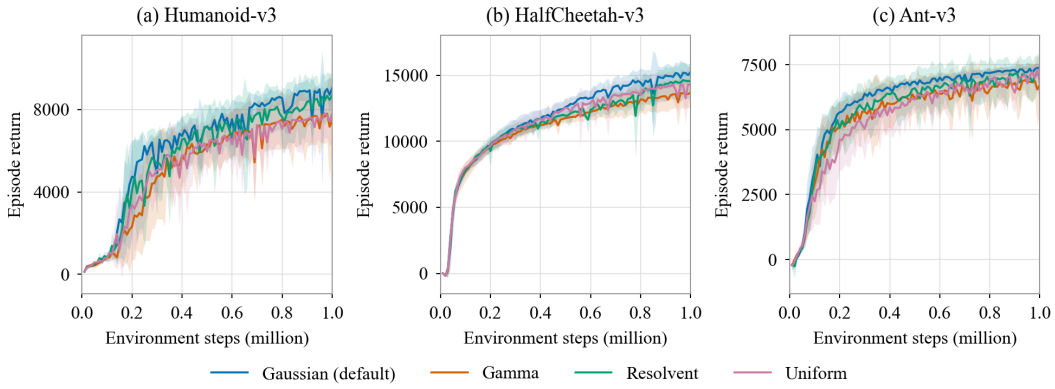


Figure 6: **Effect of spreading noise on MuJoCo.** The four noise choices use the same actor learning rates within each task.

Sequential versus joint fitting. In addition to the Gaussian example in Figure 3, we also compare sequential and joint fitting on MuJoCo. On Humanoid-v3, sequential fitting achieves higher mean returns than joint fitting across all tested loss weights. The learning curves are provided in Appendix D.2.

6 CONCLUSION AND LIMITATIONS

We introduced DRIFTSPREAD, a Drifting-based approach to learning expressive one-step policies for online RL. Building on an energy-based view of Drifting, we derive critic-guided Drift and entropy-induced Spread from the MaxEnt objective. The two updates are learned sequentially by the same actor: Drift moves actions toward higher values, while Spread matches a broader distribution constructed from the updated actor. This design incorporates the effect of entropy without estimating policy densities or scores, while allowing each action sample to be generated in one forward pass. Under exact fitting, our analysis establishes convergence to the fixed-critic Boltzmann target in the small-step, long-time limit. MuJoCo experiments show that DRIFTSPREAD combines strong control performance with efficient training and inference, supporting Drifting as a practical approach to generative policy learning.

Our work leaves two directions for future research. First, our evaluation is limited to simulation environments. Testing on real-world control tasks would help assess robustness to observation noise and changes in system dynamics. Second, we use a fixed spreading-noise distribution and scale during training. Adapting the spreading strength as learning progresses may better balance exploration and value improvement.

AI USE STATEMENT

We used generative AI tools (GPT-5.6 Sol and GPT-6 Sol) to draft and polish manuscript text, assist with coding and identify related work. These tools were not used to generate synthetic data. All AI-assisted material was reviewed by at least one coauthor, and AI-assisted code was checked manually. The main technical contributions are the authors' original work. The authors take responsibility for the final content, including the text, claims, code, and other research artifacts produced with AI assistance.

ETHICS STATEMENT

Our experiments use synthetic examples and MuJoCo simulations. The study involves no human participants or personal or sensitive data, and the method has not been deployed in safety-critical settings. We are not aware of ethical concerns specific to this work beyond those common to reinforcement learning research.

REPRODUCIBILITY STATEMENT

We provide complete training algorithm loop of DRIFTSPREADin Experimental settings and implementation details are provided in Appendix E. We use publicly available MuJoCo simulation environments. Source code and configurations for the main reinforcement-learning experiments are available in an anonymous repository at `https://anonymous.4open.science/r/DriftSpread-7A36`. All experiments were run on an NVIDIA RTX 4090 GPU.

REFERENCES

Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient Flows in Metric Spaces and in the Space of Probability Measures*. Birkhäuser, 2nd edition, 2008. doi: 10.1007/978-3-7643-8722-8.

Abdul Fatir Ansari, Jonathan Scarlett, and Harold Soh. A characteristic function approach to deep implicit generative modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7478–7487, 2020. URL `https://openaccess.thecvf.com/content_CVPR_2020/html/Ansari_A_Characteristic_Function_Approach_to_Deep_Implicit_Generative_Modeling_CVPR_2020_paper.html`.

Patrick Billingsley. *Convergence of Probability Measures*. John Wiley & Sons, 2nd edition, 1999. doi: 10.1002/9780470316962.

Onur Celik, Zechu Li, Denis Blessing, Ge Li, Daniel Palenicek, Jan Peters, Georgia Chalvatzaki, and Gerhard Neumann. DIME: Diffusion-based maximum entropy reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 6958–6977, 2025. URL `https://proceedings.mlr.press/v267/celik25a.html`.

Tianyi Chen, Haitong Ma, Na Li, Kai Wang, and Bo Dai. One-step flow policy mirror descent. *arXiv preprint arXiv:2507.23675*, 2025. URL `https://arxiv.org/abs/2507.23675`.

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

Shutong Ding, Ke Hu, Zhenhao Zhang, Kan Ren, Weinan Zhang, Jingyi Yu, Jingya Wang, and Ye Shi. Diffusion-based reinforcement learning via Q-weighted variational policy optimization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 53945–53968, 2024. doi: 10.52202/079017-1708.

Xiaoyi Dong, Jian Cheng, and Xi Sheryl Zhang. Maximum entropy reinforcement learning with diffusion policy. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 13963–13983, 2025. URL `https://proceedings.mlr.press/v267/dong25b.html`.

Xiaoyi Dong, Xi Sheryl Zhang, and Jian Cheng. Mean flow policy optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026. URL https://arxiv.org/abs/2604.14698.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1587–1596, 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 75460–75482, 2025. doi: 10.52202/085713-2534.

Arthur Gretton, Li Kevin Wenliang, Alexandre Galashov, James Thornton, Valentin De Bortoli, and Arnaud Doucet. On the Wasserstein gradient flow interpretation of drifting models. *arXiv preprint arXiv:2605.05118*, 2026. URL https://arxiv.org/abs/2605.05118.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870, 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Jiaqi Han, Puheng Li, Qiushan Guo, Renyuan Xu, Stefano Ermon, and Emmanuel J. Candès. One-step generative modeling via Wasserstein gradient flows. *arXiv preprint arXiv:2605.11755*, 2026. URL https://arxiv.org/abs/2605.11755.

Ping He, Om Khangaonkar, Hamed Pirsiavash, Yikun Bai, and Soheil Kolouri. Sinkhorn-drifting generative models. *arXiv preprint arXiv:2603.12366*, 2026. URL https://arxiv.org/abs/2603.12366.

Anas Houssaini, Mohamad H. Danesh, Amin Abyaneh, Scott Fujimoto, Hsiu-Chin Lin, and David Meger. Drift Q-learning. *arXiv preprint arXiv:2606.00350*, 2026. URL https://arxiv.org/abs/2606.00350.

Vineet Jain, Tara Akhound-Sadegh, and Siamak Ravanbakhsh. Sampling from energy-based policies using diffusion. *Reinforcement Learning Journal*, 6:2291–2307, 2025. URL https://rlj.cs.umass.edu/2025/papers/Paper273.html.

Richard Jordan, David Kinderlehrer, and Felix Otto. The variational formulation of the Fokker–Planck equation. *SIAM Journal on Mathematical Analysis*, 29(1):1–17, 1998. doi: 10.1137/S0036141096303359.

Kyungyoon Kim, Donghyeon Ki, Hee-Jun Ahn, and Byung-Jun Lee. Score-based one-step MeanFlow policy optimization. *arXiv preprint arXiv:2605.23365*, 2026. URL https://arxiv.org/abs/2605.23365.

Juil Koo, Mingue Park, Jiwon Choi, Yunhong Min, and Minhyuk Sung. Drifting field policy: A one-step generative policy via Wasserstein gradient flow. *arXiv preprint arXiv:2605.07727*, 2026. URL https://arxiv.org/abs/2605.07727.

Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1718–1727. PMLR, 2015. URL https://proceedings.mlr.press/v37/li15.html.

Zeyang Li, Sunbochen Tang, and Navid Azizan. Reverse flow matching: A unified framework for online reinforcement learning with diffusion and flow policies. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026. URL https://arxiv.org/abs/2601.08136.

Lei Lyu, Yunfei Li, Yu Luo, Fuchun Sun, Tao Kong, Jiafeng Xu, and Xiao Ma. Flow-based policy for online reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 93967–93990, 2025. doi: 10.52202/085713-3143.

Haitong Ma, Tianyi Chen, Kai Wang, Na Li, and Bo Dai. Efficient online reinforcement learning for diffusion policy. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 41837–41853, 2025. URL https://proceedings.mlr.press/v267/ma25d.html.

G. O. Roberts and R. L. Tweedie. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996. doi: 10.2307/3318418.

Yinuo Wang, Likun Wang, Yuxuan Jiang, Wenjun Zou, Tong Liu, Xujie Song, Wenxuan Wang, Liming Xiao, Jiang Wu, Jingliang Duan, and Shengbo Eben Li. Diffusion actor-critic with entropy regulator. In *Advances in Neural Information Processing Systems*, volume 37, pp. 54183–54204, 2024. doi: 10.52202/079017-1717.

Yixian Zhang, Shu-ang Yu, Tonghe Zhang, Mo Guang, Haojia Hui, Kaiwen Long, Yu Wang, Chao Yu, and Wenbo Ding. SAC Flow: Sample-efficient reinforcement learning of flow-based policies via velocity-reparameterized sequential modeling. In *International Conference on Learning Representations*, pp. 121983–122016, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/c66a3f3d3317da8d4b9c1579e2bc97d5-Abstract-Conference.html.

Zhaoyu Zhu, Shuhan Zhang, Rui Gao, and Shuang Li. Wasserstein proximal policy gradient. *arXiv preprint arXiv:2603.02576*, 2026. URL https://arxiv.org/abs/2603.02576.

APPENDIX CONTENTS

A Additional Method Details . . . 13
A.1 Deriving Drift and Spread . . . 13
A.2 Computing the Actor Losses . . . 14
A.3 Complete Training Algorithm . . . 16
B Bolzmann Convergence . . . 17
B.1 Setting and Assumptions . . . 17
B.2 Exact Fitting Recovers the Action Update . . . 18
B.3 Convergence to the Boltzmann Target . . . 18
C Sequential versus Joint Fitting . . . 19
C.1 A Two-dimensional Gaussian Example . . . 19
C.2 Numerical Comparison and Weight Selection . . . 21
D Additional Experimental Results . . . 22
D.1 Effect of Temperature on the Learned Distribution . . . 22
D.2 Sequential Fitting versus Joint-loss Weighting . . . 23
D.3 Sensitivity to the Frequency Scale . . . 24
E Experimental Details . . . 24
E.1 Ring-target Experiment . . . 24
E.2 MuJoCo Implementations and Training Settings . . . 25
E.3 Evaluation and runtime measurements . . . 27
E.4 Frequency Settings for the Noise Comparison . . . 27

A ADDITIONAL METHOD DETAILS

We first derive Drift and Spread from the MaxEnt actor objective. We then explain how the actor learns these updates through action regression and distribution matching, and give the complete training algorithm.

A.1 DERIVING DRIFT AND SPREAD

Fix a state s, critic Q, and temperature $\lambda > 0$. We consider actions $a \in \mathbb{R}^d$. The negative MaxEnt policy-improvement objective is

$$\mathcal{F}_{Q,s}(\pi) = -\mathbb{E}_{a\sim\pi(\cdot|s)}[Q(s,a)] - \lambda\mathcal{H}(\pi(\cdot \mid s)). \tag{13}$$

When $0 < Z_Q(s) < \infty$, substituting the Boltzmann density from equation (3) gives $\mathcal{F}_{Q,s}(\pi) = \lambda D_{\mathrm{KL}}(\pi\|\pi^*_Q) - \lambda \log Z_Q(s)$. Thus, minimizing this energy over policy distributions gives the Boltzmann target.

From the objective to distribution updates. Assume the policy density is smooth and positive, the energy is finite, and the density decays sufficiently fast for the following calculations. Differentiating the energy with respect to the density gives its first variation:

$$\frac{\delta\mathcal{F}_{Q,s}}{\delta\pi}(\pi)(a) = -Q(s,a) + \lambda(1 + \log\pi(a \mid s)). \tag{14}$$

The sample velocity given by the Wasserstein gradient flow in equation (4) is therefore

$$V_\pi(a \mid s) = \nabla_a Q(s,a) - \lambda\nabla_a \log\pi(a \mid s). \tag{15}$$

The critic gradient is directly available, but the entropy term depends on $\nabla_a \log\pi$, which the implicit actor does not provide. To avoid computing this term, we consider how the policy density changes. Substituting the velocity into the continuity equation and using $\pi\nabla_a \log\pi = \nabla_a\pi$ gives

$$\partial_\tau\pi = -\nabla_a \cdot (\pi\nabla_a Q) + \lambda\Delta_a\pi. \tag{16}$$

The first term moves actions toward higher critic values. The second spreads their distribution through diffusion.

Approximating Drift and Spread. For a step size $\eta > 0$, Drift approximates the first term by moving each action to $a^{\mathrm{D}} = a + \eta \nabla_a Q(s, a)$. Spread approximates the diffusion term by adding independent noise $\epsilon_\eta = \sqrt{2\lambda\eta}\,\zeta$ to the post-Drift actions. The distribution of ζ does not depend on η and satisfies

$$\mathbb{E}[\zeta] = 0, \qquad \mathbb{E}[\zeta\zeta^\top] = I_d, \qquad \mathbb{E}\|\zeta\|^3 < \infty. \tag{17}$$

The added noise therefore has mean zero and covariance $2\lambda\eta I_d$; it need not be Gaussian.

To see why this approximates diffusion, consider a smooth function g with bounded derivatives through order three. A second-order Taylor expansion gives

$$\mathbb{E}_\zeta[g(a + \epsilon_\eta)] = g(a) + \lambda\eta\Delta_a g(a) + O(\eta^{3/2}). \tag{18}$$

The zero mean removes the linear term, and the covariance determines the quadratic term. To first order in η, this is the change produced by diffusion with coefficient λ.

Applying Drift followed by Spread gives

$$a^+ = a + \eta\nabla_a Q(s, a) + \epsilon_\eta, \qquad \epsilon_\eta = \sqrt{2\lambda\eta}\,\zeta. \tag{19}$$

Noise draws are independent across cycles and independent of the current action. Appendix A.2.3 gives the noise distributions used in our experiments.

Gaussian noise is a useful special case. Adding $\epsilon_\eta \sim \mathcal{N}(0, 2\lambda\eta I_d)$ gives the exact entropy-only diffusion over duration η. The other listed choices approximate this diffusion and share its small-step limit, although their targets can differ at finite step sizes. Even with Gaussian noise, the complete Drift–Spread cycle approximates the combined evolution over a finite step.

Appendix B proves convergence under the stated regularity and exact-fitting assumptions, with the critic held fixed. We next describe how the actor learns the Drift targets through regression and the Spread target through distribution matching.

A.2 Computing the Actor Losses

Drift provides an updated action for each generator input, so it can be learned through regression. Spread specifies a broader action distribution. We explain why this requires a different loss, then give the sample-based losses used in training. Throughout this subsection, the actor generates $a = f_\theta(s, z)$ with $z \sim p_{\mathrm{ref}}$.

A.2.1 Why Spread Uses Distribution Matching

Spread trains the actor to represent the distribution produced by adding noise to its post-Drift actions. This allows the next Drift update to start from the broader distribution.

Fix a state s and write $\bar{a}(z) = \mathrm{sg}[f_{\theta_{k+1/2}}(s, z)]$. If we apply regression to the noisy target $\bar{a}(z) + \epsilon_\eta$ using the same input z, then

$$\begin{aligned}\mathbb{E}_{z,\epsilon_\eta}\|f_\theta(s, z) - \bar{a}(z) - \epsilon_\eta\|^2 &= \mathbb{E}_z\|f_\theta(s, z) - \bar{a}(z)\|^2 + \mathbb{E}\|\epsilon_\eta\|^2 \\ &= \mathbb{E}_z\|f_\theta(s, z) - \bar{a}(z)\|^2 + 2\lambda\eta d.\end{aligned} \tag{20}$$

The cross term vanishes because the noise is independent of z and has zero mean. Its remaining contribution is a constant, so the post-Drift actor already minimizes this regression loss. The loss therefore does not teach the actor to reproduce the added spread.

We instead use the Spread MMD loss in equation (9) to match the actor and target distributions.

A.2.2 Losses Computed from Actor Samples

Drift regression. Sample B states s_b from the replay buffer and N_{D} independent inputs $z^{\mathrm{D}}_{b,n} \sim p_{\mathrm{ref}}$ per state. For each current action $a^k_{b,n} = f_{\theta_k}(s_b, z^{\mathrm{D}}_{b,n})$, we construct the fixed target

$$\widetilde{a}^{k+1/2}_{b,n} = \mathrm{sg}\big[a^k_{b,n} + \eta\nabla_a Q(s_b, a^k_{b,n})\big]\,. \tag{21}$$

The empirical Drift loss is

$$\widehat{\mathcal{L}}_{\mathrm{Drift}}(\theta;\theta_k) = \frac{1}{2BN_{\mathrm{D}}} \sum_{b=1}^{B} \sum_{n=1}^{N_{\mathrm{D}}} \left\| f_\theta(s_b, z^{\mathrm{D}}_{b,n}) - \widetilde{a}^{k+1/2}_{b,n} \right\|^2 . \tag{22}$$

The actor prediction and its target use the same input. The stop-gradient operation holds the target fixed, so only the prediction contributes gradients to the actor parameters. One optimizer step from θ_k gives $\theta_{k+1/2}$.

Computing Spread with characteristic functions. We construct the Spread target from this updated actor. Write $K_\eta = K_{\lambda,\eta}$ for the spreading-noise distribution and $m_\eta = \widehat{K}_{\lambda,\eta}$ for its characteristic function (CF). By independence, the target CF is $m_\eta(\omega)\varphi_{\theta_{k+1/2}}(\omega \mid s)$. We use this expression to compute the Spread MMD loss as in equation (11).

For each replay state, draw N_{CF} fresh inputs $z^{\mathrm{CF}}_{b,n} \sim p_{\mathrm{ref}}$. Draw M_ω frequencies $\omega_\ell \sim \rho$, shared across the batch. For any actor parameters θ, estimate its CF by

$$\widehat{\varphi}_\theta(\omega_\ell \mid s_b) = \frac{1}{N_{\mathrm{CF}}} \sum_{n=1}^{N_{\mathrm{CF}}} e^{i\omega_\ell^\top f_\theta(s_b, z^{\mathrm{CF}}_{b,n})} = \widehat{c}^{\,\theta}_{b,\ell} + i\widehat{s}^{\,\theta}_{b,\ell}. \tag{23}$$

Here, $\widehat{c}^{\,\theta}$ and $\widehat{s}^{\,\theta}$ are the averages of the corresponding cosines and sines. Using the post-Drift actor, form the fixed target

$$\widehat{\varphi}^{\mathrm{tar}}_{b,\ell} = \mathrm{sg}\big[m_\eta(\omega_\ell)\widehat{\varphi}_{\theta_{k+1/2}}(\omega_\ell \mid s_b)\big] = \widehat{c}^{\,\mathrm{tar}}_{b,\ell} + i\widehat{s}^{\,\mathrm{tar}}_{b,\ell}. \tag{24}$$

The actor and target use the same inputs and frequencies. Their squared CF difference can be computed with real-valued operations:

$$\widehat{\mathcal{L}}_{\mathrm{Spread}}(\theta;\theta_{k+1/2}) = \frac{1}{BM_\omega} \sum_{b=1}^{B} \sum_{\ell=1}^{M_\omega} \left[(\widehat{c}^{\,\theta}_{b,\ell} - \widehat{c}^{\,\mathrm{tar}}_{b,\ell})^2 + (\widehat{s}^{\,\theta}_{b,\ell} - \widehat{s}^{\,\mathrm{tar}}_{b,\ell})^2 \right]. \tag{25}$$

One optimizer step from $\theta_{k+1/2}$ gives θ_{k+1}. We compare the CFs separately at each state before averaging over the batch. This is a finite-sample approximation to the population MMD objective; the exact-fitting results concern the population objective.

Using the noise characteristic function. The known noise CF lets us average over the spreading noise analytically. For fixed post-Drift actions $a_1, \ldots, a_{N_{\mathrm{CF}}}$ and independent $\epsilon_n \sim K_\eta$,

$$\mathbb{E}_{\epsilon_1,\ldots,\epsilon_{N_{\mathrm{CF}}}} \left[\frac{1}{N_{\mathrm{CF}}} \sum_{n=1}^{N_{\mathrm{CF}}} e^{i\omega^\top(a_n+\epsilon_n)} \right] = m_\eta(\omega) \frac{1}{N_{\mathrm{CF}}} \sum_{n=1}^{N_{\mathrm{CF}}} e^{i\omega^\top a_n}. \tag{26}$$

We therefore construct the target CF by multiplying the estimated actor CF by m_η. Actor inputs and frequencies are still sampled. Computing the inner products and sample averages costs $O(BN_{\mathrm{CF}}M_\omega d)$ operations, in addition to the actor's forward and backward passes.

A.2.3 Spreading Noise and Characteristic Functions

We use Gaussian spreading noise by default and compare three alternatives in Appendix **??**. All four choices use the same actor and loss. For a post-Drift distribution $\pi_{k+1/2}(\cdot \mid s)$, the Spread target is

$$\widetilde{\pi}_{k+1} = K_\eta * \pi_{k+1/2}, \qquad \varphi_{\widetilde{\pi}_{k+1}}(\omega \mid s) = m_\eta(\omega)\varphi_{\pi_{k+1/2}}(\omega \mid s). \tag{27}$$

The noise distribution K_η determines the target. The frequency distribution ρ determines the MMD kernel used to compare that target with the actor.

Noise distributions and their characteristic functions. Write $c = \lambda\eta$. For the Gaussian and mixture constructions, let $\xi \sim \mathcal{N}(0, I_d)$ and $v_\kappa \sim \mathrm{Gamma}(\kappa, \text{rate } \kappa)$ be independent, with fixed $\kappa > 0$. For uniform noise, let the coordinates of U be independent and uniform on $[-1, 1]$. The resulting noise distributions are listed in Table 1. Each has mean zero and covariance $2cI_d$.

For the Gamma mixture, conditioning on v_κ gives

$$\mathbb{E}e^{i\omega^\top\sqrt{2cv_\kappa}\,\xi} = \mathbb{E}e^{-cv_\kappa\|\omega\|^2} = \left(1 + \frac{c\|\omega\|^2}{\kappa}\right)^{-\kappa}. \tag{28}$$

Resolvent is the case $\kappa = 1$, for which $v_1 \sim \mathrm{Exp}(1)$. The Uniform CF follows by multiplying the CFs of its independent coordinates, where $\mathrm{sinc}(x) = \sin(x)/x$ and $\mathrm{sinc}(0) = 1$.

Table 1: Spreading noise and characteristic functions, with $c = \lambda\eta$ and $a_\eta = \sqrt{6c}$. The Gamma construction scales a Gaussian vector by an independent Gamma-distributed variance factor.

Noise	Characteristic function $m_\eta(\omega)$	Noise sample ϵ_η
Gaussian	$e^{-c\|\omega\|^2}$	$\sqrt{2c}\,\xi$
Resolvent	$(1 + c\|\omega\|^2)^{-1}$	$\sqrt{2cv_1}\,\xi$
Gamma mixture	$(1 + c\|\omega\|^2/\kappa)^{-\kappa}$	$\sqrt{2cv_\kappa}\,\xi$
Uniform	$\prod_{j=1}^{d} \mathrm{sinc}(a_\eta\omega_j)$	$a_\eta U$

Connection to the diffusion update. Gaussian noise gives the exact entropy-only diffusion over duration η. The other choices agree with it to first order: for each fixed ω,

$$m_\eta(\omega) = 1 - \lambda\eta\|\omega\|^2 + o(\eta). \tag{29}$$

All four noise distributions can be written as $\epsilon_\eta = \sqrt{2\lambda\eta}\,\zeta$, where ζ satisfies equation (17). They therefore satisfy the noise assumptions in theorem B.2 and share the same diffusion limit. Their target distributions can differ at finite step sizes.

A.2.4 Choosing the Frequency Scale

We sample Gaussian frequencies $\omega = \sigma_\omega Z$, where $Z \sim \mathcal{N}(0, I_d)$, for every spreading-noise choice. The corresponding MMD kernel is $h(a, a') = \exp(-\sigma_\omega^2\|a - a'\|^2/2)$. Thus, the frequency scale determines the Gaussian kernel's bandwidth.

For Gaussian spreading noise, $m_\eta(\omega) = e^{-\lambda\eta\|\omega\|^2}$. If $\lambda\eta\|\omega\|^2$ is very small, the target CF is almost unchanged by Spread. We choose a default scale that accounts for both the spreading-noise size and the action dimension. For independent $\epsilon_{\mathrm{G}} \sim \mathcal{N}(0, 2\lambda\eta I_d)$ and ω, we set

$$\mathbb{E}[(\omega^\top\epsilon_{\mathrm{G}})^2] = 2\lambda\eta d\,\sigma_{\mathrm{G}}^2 = 1, \qquad \sigma_{\mathrm{G}} = (2\lambda\eta d)^{-1/2}. \tag{30}$$

A.3 Complete Training Algorithm

Algorithm 1 combines the actor updates with off-policy critic learning. Environment transitions are stored in a replay buffer $\mathcal{D}$ and reused for training. Each gradient iteration updates the critics first, then performs one Drift step and one Spread step using the replay states.

Critic update. Following section 4.4, the critics estimate discounted rewards without an entropy term in the Bellman target. Entropy regularization remains in the actor objective.

Let Q_{ϕ_1}, Q_{ϕ_2} be the current critics, $\bar{Q}_1, \bar{Q}_2$ their target networks, and $f_{\bar{\theta}}$ the target actor. For each replay transition $(s_b, a_b, r_b, s'_b, \delta_b)$, draw $z'_b \sim p_{\mathrm{ref}}$ and set $a'_b = f_{\bar{\theta}}(s'_b, z'_b)$. Both critics regress toward the fixed target

$$y_b = \mathrm{sg}\left[r_b + \gamma(1 - \delta_b)\min_{i\in\{1,2\}} \bar{Q}_i(s'_b, a'_b)\right]. \tag{31}$$

Using the smaller target value reduces overestimation (Fujimoto et al., 2018). Each critic minimizes $B^{-1}\sum_b (Q_{\phi_i}(s_b, a_b) - y_b)^2$. Here, $\delta_b = 1$ for an environment termination; time-limit truncations retain bootstrapping when required by the task. This update requires no policy-density estimate.

Actor update. After updating the critics, we hold their parameters fixed and use $Q(s, a) = \min_{i\in\{1,2\}} Q_{\phi_i}(s, a)$ to construct the Drift targets. One optimizer step on $\widehat{\mathcal{L}}_{\mathrm{Drift}}$ gives $\theta_{k+1/2}$. We then construct the Spread target from this updated actor and take one step on $\widehat{\mathcal{L}}_{\mathrm{Spread}}$ to obtain θ_{k+1}. The evolution step η controls the target displacement and spreading-noise scale; it is separate from the optimizer learning rates.

Target-network update. On the target-update schedule, we average each target parameter toward its current value:

$$\bar{\theta} \leftarrow (1 - \tau_{\mathrm{targ}})\bar{\theta} + \tau_{\mathrm{targ}}\theta_{k+1}, \qquad \bar{\phi}_i \leftarrow (1 - \tau_{\mathrm{targ}})\bar{\phi}_i + \tau_{\mathrm{targ}}\phi_i. \tag{32}$$

Here, $\bar{\phi}_i$ denotes the parameters of $\bar{Q}_i$.

Algorithm 1 DRIFTSPREAD: online training

Require: Actor f_θ, critics Q_{ϕ_1}, Q_{ϕ_2}, and replay buffer $\mathcal{D}$

Initialize target networks from the actor and critics; set $k \leftarrow 0$ and observe state s.
while training continues **do**
  Sample $z \sim p_{\mathrm{ref}}$ and take action $a \leftarrow f_{\theta_k}(s, z)$.
  Observe (r, s', δ) and store (s, a, r, s', δ) in $\mathcal{D}$.
  for each scheduled gradient iteration **do**
    Sample a batch of B transitions from $\mathcal{D}$.
    Compute the targets in equation (31) and update both critics by squared-error regression.
    Set $Q(s, a) \leftarrow \min_{i\in\{1,2\}} Q_{\phi_i}(s, a)$ and hold the critics fixed.
    Draw N_{D} actor inputs per replay state and construct the targets in equation (21).
    Take one step on $\widehat{\mathcal{L}}_{\mathrm{Drift}}$ to obtain $\theta_{k+1/2}$.
    Draw N_{CF} fresh actor inputs per state and M_ω frequencies.
    Construct the fixed Spread target from $f_{\theta_{k+1/2}}$ using equation (24).
    Take one step on $\widehat{\mathcal{L}}_{\mathrm{Spread}}$ to obtain θ_{k+1}.
    Update the target networks on schedule; set $k \leftarrow k + 1$.
  end for
  if the episode has ended **then**
    Reset the environment and observe s.
  else
    $s \leftarrow s'$.
  end if
end while

B BOLZMANN CONVERGENCE

We prove the convergence result in theorem 4.1. The argument connects the actor losses to the Boltzmann target in three steps. Exact fitting makes each Drift–Spread cycle equivalent to moving an action along the critic gradient and then adding noise. As the step size decreases, these updates approach a continuous diffusion. Over time, this diffusion converges to the Boltzmann target.

B.1 SETTING AND ASSUMPTIONS

Fix a state s, critic Q, and temperature $\lambda > 0$. We work on $\mathbb{R}^d$ and omit state conditioning from the policy notation. Write $b(a) = \nabla_a Q(s, a)$. The target is the Boltzmann distribution $\pi^*_Q(a) = Z_Q(s)^{-1} \exp(Q(s, a)/\lambda)$, where $Z_Q(s) = \int_{\mathbb{R}^d} \exp(Q(s, a)/\lambda)\, da$.

Assumption B.1 (Conditions on the critic)**.** The target is normalizable, $0 < Z_Q(s) < \infty$. The critic $Q(s, \cdot)$ is twice continuously differentiable, and there is a finite constant $C \geq 0$ such that

$$\langle a, b(a)\rangle \leq C(1 + \|a\|^2) \qquad \text{for all } a \in \mathbb{R}^d. \tag{33}$$

The smoothness condition makes b locally Lipschitz. The bound in equation (33) controls how strongly the drift pushes actions away from the origin and prevents the continuous process from escaping to infinity in finite time.

For step size $\eta > 0$, the Spread noise is $\epsilon_{\eta,k} = \sqrt{2\lambda\eta}\,\zeta_k$. The variables ζ_k are independent copies of a random vector ζ whose distribution does not depend on η, with

$$\mathbb{E}\zeta = 0, \qquad \mathbb{E}[\zeta\zeta^\top] = I_d, \qquad \mathbb{E}\|\zeta\|^3 < \infty. \tag{34}$$

Each noise draw is independent of the current action. These conditions include Gaussian noise and the other noise choices in Appendix A.2.3. They allow different finite-step updates while giving the same small-step limit.

B.2 Exact Fitting Recovers the Action Update

We first show what minimizing the two actor losses achieves. Let $\pi_k^{(\eta)}$, $\pi_{k+1/2}^{(\eta)}$, and $\pi_{k+1}^{(\eta)}$ denote the actor distributions before Drift, after Drift, and after Spread, respectively. Each target is held fixed during its update, and the Spread target is constructed from the post-Drift actor.

Proposition B.1 (Action update under exact fitting)**.** *Suppose that, at the fixed state s, the population Drift regression loss and Spread MMD loss are zero after their respective updates in every cycle. If the frequency distribution ρ has full support on $\mathbb{R}^d$, then $\pi_k^{(\eta)}$ is the distribution of the random action sequence*

$$A_{k+1}^{(\eta)} = A_k^{(\eta)} + \eta b(A_k^{(\eta)}) + \sqrt{2\lambda\eta}\,\zeta_k, \qquad A_0^{(\eta)} \sim \pi_0. \tag{35}$$

Proof. Zero Drift regression loss means that, for almost every generator input z, the updated actor reproduces its target:

$$f_{\theta_{k+1/2}}(s,z) = f_{\theta_k}(s,z) + \eta b\big(f_{\theta_k}(s,z)\big).$$

Thus, if $A_k^{(\eta)}$ follows the current policy, the post-Drift policy is the distribution of $A_k^{(\eta)} + \eta b(A_k^{(\eta)})$.

Spread then targets the distribution obtained by adding independent noise to these post-Drift actions. Denote this distribution by $\widetilde{\pi}_{k+1}$ and write φ_μ for the characteristic function of a distribution μ. By the characteristic-function expression of the Spread loss in equation (11), zero loss gives

$$\mathbb{E}_{\omega\sim\rho}\left|\varphi_{\pi_{k+1}^{(\eta)}}(\omega) - \varphi_{\widetilde{\pi}_{k+1}}(\omega)\right|^2 = 0.$$

Both characteristic functions are continuous. If they differed at any frequency, they would also differ in a neighborhood of that frequency. Full support of ρ assigns positive probability to every such neighborhood, contradicting zero loss. The characteristic functions therefore agree everywhere. Since a characteristic function uniquely determines a distribution, $\pi_{k+1}^{(\eta)} = \widetilde{\pi}_{k+1}$. Combining the two updates proves equation (35), and repeating this argument establishes the result for every cycle. □

B.3 Convergence to the Boltzmann Target

Each cycle in equation (35) advances the evolution time by η. To compare different step sizes at the same time τ, we take $k = \lfloor \tau/\eta \rfloor$ and use the same initial policy π_0 for every η. The limiting continuous process is

$$dA_\tau = b(A_\tau)\,d\tau + \sqrt{2\lambda}\,dW_\tau, \qquad A_0 \sim \pi_0, \tag{36}$$

where W_τ is standard d-dimensional Brownian motion, independent of A_0. This is the Langevin diffusion whose density follows the MaxEnt evolution in equation (6). Let p_τ denote its distribution at time τ.

Theorem B.2 (Boltzmann convergence under exact fitting)**.** *Under assumption B.1, the noise conditions above, and the exact-fitting conditions in proposition B.1, for every initial policy π_0 and every fixed finite time τ,*

$$\pi_{\lfloor \tau/\eta \rfloor}^{(\eta)} \Rightarrow p_\tau \qquad \text{as } \eta \downarrow 0. \tag{37}$$

Moreover, $p_\tau \Rightarrow \pi_Q^*$ as $\tau \to \infty$. Consequently,

$$\lim_{\tau\to\infty}\lim_{\eta\downarrow 0} \pi_{\lfloor \tau/\eta \rfloor}^{(\eta)} = \pi_Q^* \qquad \text{weakly}. \tag{38}$$

Here, $\Rightarrow$ denotes weak convergence: expectations of every bounded continuous function converge.

Proof. Small-step limit. First fix an initial action $A_0 = x$ and a finite time interval $[0, H]$. The drift is locally Lipschitz, so the diffusion has a unique solution until a possible escape to infinity. Applying Itô's formula to $1 + \|A_\tau\|^2$, with the process stopped when it leaves a ball, and using equation (33) rules out such an escape in finite time. Thus the solution exists throughout $[0, H]$.

The accumulated noise in the discrete updates is

$$W_{k\eta}^{(\eta)} = \sqrt{\eta}\sum_{j=0}^{k-1}\zeta_j.$$

Interpolate these values linearly between grid times. The functional central limit theorem implies that $W^{(\eta)}$ converges in distribution to Brownian motion on $[0, H]$. Along any sequence of step sizes tending to zero, the Skorokhod representation theorem allows these paths to be constructed on a common probability space so that they converge uniformly almost surely (Billingsley, 1999).

We next show that this convergence of the noise paths also gives convergence of the action paths. For each positive integer R, smoothly modify b outside the ball of radius R so that the modified drift b_R is bounded and globally Lipschitz. Use the same noise paths to form both the discrete updates and the diffusion with this modified drift. Within each time step, the change in the interpolated action is bounded by the drift increment and the noise increment, both of which tend uniformly to zero. Subtracting the two path equations and applying Grönwall's inequality therefore shows that the interpolated discrete action paths converge uniformly to the diffusion paths with drift b_R.

The original diffusion stays within a finite ball on $[0, H]$ almost surely. Choose an integer R larger than its maximum norm by at least one. For this R, modifying the drift has no effect on the diffusion path. Uniform convergence implies that the corresponding discrete paths also remain inside the ball of radius R for all sufficiently small step sizes. They therefore follow the original update as well. This proves convergence of the original action paths. By continuity of A_τ, it also gives $A^{(\eta)}_{\lfloor \tau/\eta \rfloor} \Rightarrow A_\tau$ for every fixed $\tau \leq H$.

The argument holds for every initial action x. Integrating bounded continuous test functions over $x \sim \pi_0$ and applying bounded convergence extends the result to any initial policy. Proposition B.1 then gives equation (37) for the fitted actors.

Long-time limit. The Boltzmann density satisfies

$$\lambda \nabla_a \pi^*_Q(a) = \pi^*_Q(a) b(a).$$

Thus the drift and diffusion terms balance at π^*_Q, making it a stationary distribution of equation (36). To establish convergence to this distribution, we apply Roberts & Tweedie (1996, Theorem 2.1). Its conditions hold because π^*_Q is positive, $\nabla_a \log \pi^*_Q = b/\lambda$ is continuously differentiable, and

$$\langle a, \nabla_a \log \pi^*_Q(a) \rangle \leq \frac{C}{\lambda}(1 + \|a\|^2).$$

With time rescaled by 2λ, their diffusion is exactly equation (36). The theorem gives convergence to π^*_Q from every initial action. Averaging over π_0 gives $p_\tau \Rightarrow \pi^*_Q$. Combining this result with equation (37) proves equation (38) and hence theorem 4.1. □

The order of the limits is essential. We first decrease the step size while keeping the evolution time fixed, and then let the limiting diffusion run for longer. The result concerns exact fitting with a fixed critic; it does not assert exact convergence for a finite step size or for the practical algorithm with changing critics.

C Sequential versus Joint Fitting

Sequential fitting first learns the Drift target and then constructs the Spread target from the updated actor. A joint alternative constructs both targets from the actor before Drift and minimizes $\mathcal{L}_{\text{Drift}} + \beta \mathcal{L}^{\text{old}}_{\text{Spread}}$. Here, $\beta > 0$ weights the Spread MMD loss, and "old" indicates that its target uses the pre-Drift actor. We examine whether choosing this weight can make joint fitting recover the intended balance between value and entropy. We analyze a Gaussian example, then test weight selection in this example.

C.1 A Two-dimensional Gaussian Example

We use a target that the actor can represent exactly, so that limited model capacity cannot explain a mismatch. Fix one state, omit state conditioning, and set $\lambda = 1$ and $\kappa = (16, 1/4)^\top$. The critic and its Boltzmann target are

$$Q(a) = -\frac{1}{2}\sum_{i=1}^{2} \kappa_i a_i^2, \qquad \pi^*_Q = \mathcal{N}(0, \operatorname{diag}(1/16, 4)). \tag{39}$$

The target is narrow in the first action coordinate and wide in the second. Our actor is $f_\sigma(z) = \mathrm{diag}(\sigma_1, \sigma_2)z$, where $z \sim \mathcal{N}(0, I_2)$ and $\sigma_i > 0$. Its distribution is denoted by π_σ, and the target is represented by $\sigma^* = (1/4, 2)^\top$.

The two fitting procedures. Take $0 < \eta < 1/16$ and Gaussian Spread noise $\mathcal{N}(0, 2\eta I_2)$. For an actor with standard deviations σ, Drift multiplies coordinate i by $1 - \eta\kappa_i$, while adding the Spread noise increases its variance by 2η. If both targets are constructed from this actor, their standard deviations are

$$\sigma_{\mathrm{D},i} = (1 - \eta\kappa_i)\sigma_i, \qquad \sigma_{\mathrm{S},i}^{\mathrm{old}} = \sqrt{\sigma_i^2 + 2\eta}. \tag{40}$$

Sequential first fits σ_{D} and then fits the distribution obtained by adding noise to that updated actor. Joint instead fits both targets in equation (40) at once. For candidate standard deviations σ', its loss is

$$\mathcal{J}_{\eta,\beta}(\sigma'; \sigma) = \frac{1}{2}\|\sigma' - \sigma_{\mathrm{D}}\|^2 + \beta\mathcal{M}\big(\sigma', \sigma_{\mathrm{S}}^{\mathrm{old}}\big), \tag{41}$$

where $\mathcal{M}(v, w) = \mathrm{MMD}_h^2(\pi_v, \pi_w)$ uses the Gaussian kernel $h(a, a') = \exp(-\|a - a'\|^2/2)$. The first term is the Drift regression loss for this actor. The reference σ, and hence both targets, are held fixed when optimizing σ'.

If repeated exact joint fitting leaves an actor unchanged, that actor must satisfy

$$\nabla_{\sigma'}\mathcal{J}_{\eta,\beta}(\sigma'; \sigma)\big|_{\sigma'=\sigma} = 0. \tag{42}$$

We call a positive solution of this equation a stationary point of Joint. The following result shows that such points remain different from the target, even when the weight is retuned as the step size decreases.

Proposition C.1 (A persistent mismatch under joint fitting)**.** *For any positive weights β_η and corresponding positive stationary points σ_η of Joint,*

$$\liminf_{\eta\downarrow 0} D_{\mathrm{KL}}(\pi_{\sigma_\eta}\|\pi_Q^*) \geq c_{\mathrm{stat}} \approx 0.50735. \tag{43}$$

In contrast, repeated exact Sequential fitting converges to a Gaussian distribution with variances

$$\sigma_{i,\mathrm{seq}}^2 = \frac{1}{\kappa_i(1 - \kappa_i\eta/2)} \longrightarrow \frac{1}{\kappa_i} \qquad \text{as } \eta \downarrow 0. \tag{44}$$

Its stationary KL divergence therefore tends to zero.

Proof. Sequential fitting. Let $x_{i,k}$ denote the variance of coordinate i before cycle k. Exact Drift fitting contracts this variance, and the subsequent Spread fitting adds 2η:

$$x_{i,k+1} = (1 - \eta\kappa_i)^2 x_{i,k} + 2\eta.$$

Because $0 < 1 - \eta\kappa_i < 1$, this recursion converges to the fixed variance in equation (44). These variances approach the Boltzmann target variances $1/\kappa_i$ as $\eta \downarrow 0$.

Joint fitting. For the Gaussian actor and kernel used here, the MMD loss has the closed form

$$\begin{aligned}\mathcal{M}(v, w) = &\prod_{i=1}^{2}(1 + 2v_i^2)^{-1/2} + \prod_{i=1}^{2}(1 + 2w_i^2)^{-1/2} \\ &- 2\prod_{i=1}^{2}(1 + v_i^2 + w_i^2)^{-1/2}.\end{aligned} \tag{45}$$

To write its gradient compactly, set $x_i = \sigma_i^2$, $r_i = 1 + 2x_i$, $G_0 = (r_1 r_2)^{-1/2}$, and $G_\eta = ((r_1 + 2\eta)(r_2 + 2\eta))^{-1/2}$. Differentiate equation (41) with respect to the candidate σ', keeping the reference σ fixed, and then set $\sigma' = \sigma$. The stationary-point condition becomes

$$\eta\kappa_i = 2\beta_\eta\left(\frac{G_0}{r_i} - \frac{G_\eta}{r_i + 2\eta}\right), \qquad i \in \{1, 2\}. \tag{46}$$

The same weight β_η must satisfy this equation for both coordinates. For variances bounded above and bounded away from zero, the expression in parentheses expands uniformly as

$$\frac{G_0}{r_i} - \frac{G_\eta}{r_i + 2\eta} = \eta G_0 \left(\frac{3}{r_i^2} + \frac{1}{r_1 r_2} \right) + O(\eta^2).$$

Dividing the two equations in equation (46) eliminates the weight before taking the small-step limit. Any finite positive limiting variances must therefore satisfy

$$64 = \frac{r_2(3r_2 + r_1)}{r_1(3r_1 + r_2)}, \qquad \frac{r_2}{r_1} = t := \frac{21 + \sqrt{697}}{2} \approx 23.7004. \tag{47}$$

At the Boltzmann target, however, $r_1 = 1 + 2/16$ and $r_2 = 1 + 2 \cdot 4$, giving $r_2/r_1 = 8$. Thus the balance required by the joint loss is incompatible with the target variances.

The KL lower bound. For this Gaussian family,

$$D_{\mathrm{KL}}(\pi_\sigma \| \pi_Q^*) = \frac{1}{2} \sum_{i=1}^{2} \big(\kappa_i x_i - 1 - \log(\kappa_i x_i) \big).$$

This expression diverges if either variance tends to zero or infinity. If the KL lower limit is finite, we can therefore choose a subsequence attaining that limit whose variances converge to finite positive values. Those values must satisfy equation (47). Writing $x = x_1$ and $y = x_2 = tx + (t-1)/2$, we obtain

$$c_{\mathrm{stat}} = \min_{x>0} \frac{1}{2} \left[16x - 1 - \log(16x) + \frac{y}{4} - 1 - \log\left(\frac{y}{4}\right) \right] \approx 0.50735. \tag{48}$$

The function is strictly convex and attains its minimum at $x = (\sqrt{457} - 17)/(64 + t)$. If the KL lower limit is infinite, the bound holds immediately. This proves equation (43). □

The example shows why weighting the two losses need not reproduce the intended Drift–Spread update. Joint balances their gradients using one weight, but the required balance differs across action coordinates. Sequential instead fits the two distribution changes in order, with Spread acting on the result of Drift. The conclusion concerns the joint target construction and Gaussian MMD loss defined above.

C.2 Numerical Comparison and Weight Selection

Setup. We test the Gaussian example using the same initial actor, $\sigma_0 = (0.7, 0.5)^\top$, for both methods. We use $\eta = 0.005$ and run 8,000 cycles, corresponding to evolution time $\tau = 40$. Sequential fits each target exactly, while Joint numerically minimizes equation (41) in each cycle using damped Newton steps with analytic derivatives. All action and kernel expectations are evaluated analytically, so the comparison has no sampling error in the losses.

To give Joint a direct way to select its weight, we use the known Boltzmann target for evaluation. We run each of 121 logarithmically spaced weights in $[10^{-1}, 10^3]$ from the same initialization and select the one with the lowest final KL divergence. We then refine the search around the best tested weight, obtaining $\beta \approx 17.2624$. The weight remains fixed throughout each run. KL is used only for selecting and evaluating the weight; the actor is trained with the regression and MMD losses.

Results. Figure 7 shows the final KL divergence and the variance in each coordinate relative to its target variance, $R_i = \sigma_i^2/(\sigma_i^*)^2$. Sequential reaches a KL divergence of 0.000422, whereas the selected Joint run reaches 0.512165. For Joint, the variance ratios are approximately $(0.80, 3.15)$: the first coordinate is too narrow, while the second is too wide. Changing the weight adjusts this tradeoff but does not bring both ratios to one.

Figure 8 shows the fitted distributions and the effect of reducing the step size. Keeping $\tau = 40$ and the selected Joint weight fixed, we reduce η from 0.01 to 0.000625. Sequential's final KL decreases from 0.001788 to 6.29×10^{-6}, while Joint's remains near 0.51. These results agree with the different small-step behavior in proposition C.1.

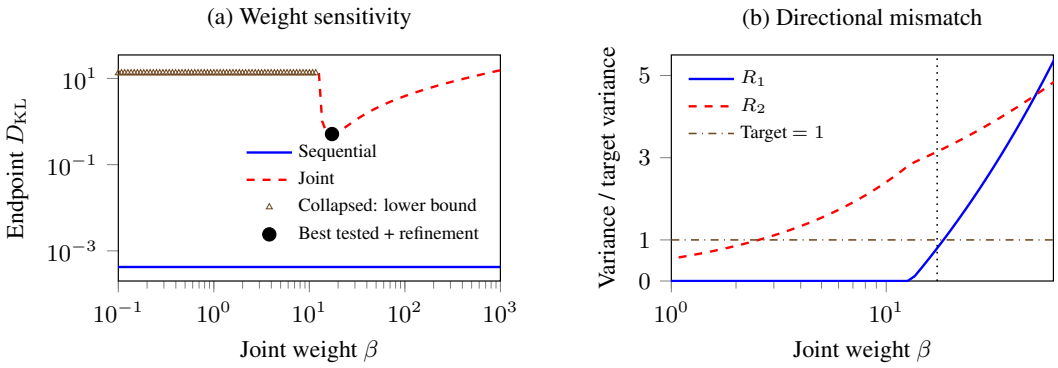


Figure 7: **Selecting the joint-loss weight in the Gaussian example.** (a) Final KL divergence after 8,000 cycles. (b) Learned variances divided by target variances; a ratio of one indicates an exact match. The dot in (a) and dotted line in (b) mark the weight selected by the grid search and refinement. The 63 runs with $\min_i \sigma_i/\sigma_i^* < 10^{-6}$ are shown in (a) at the KL lower bound 13.3155 rather than their actual KL values.

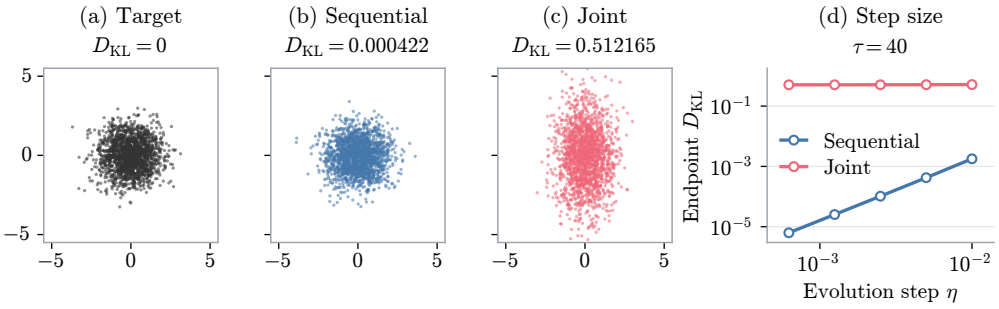


Figure 8: **Sequential and joint fitting of the Gaussian target.** (a) Boltzmann target. (b,c) Samples from the fitted actors. Scatter plots use normalized coordinates a_i/σ_i^*, so the target has unit variance in both coordinates. (d) Final KL divergence for different step sizes at the same evolution time $\tau = 40$. Joint uses the selected weight $\beta \approx 17.2624$.

D Additional Experimental Results

We provide additional experiments to examine how Spread changes the learned distribution and how its implementation affects policy learning. We first extend the ring-target experiment to more temperatures, then compare spreading-noise choices, joint-loss weights, and frequency scales on MuJoCo. The corresponding experimental settings are given in Appendix E.

D.1 Effect of Temperature on the Learned Distribution

We extend the ring-target experiment in section 5.1 to $\lambda \in \{0, 0.05, 0.1, 0.25, 0.5, 0.75, 1\}$. The critic is unchanged, so increasing λ changes the spread of the Boltzmann target without changing the location of the high-value ring. This experiment tests whether the actor learns the effect of the entropy coefficient on its output distribution.

Figure 9 shows that the Boltzmann reference becomes wider around the ring as λ increases. Drift-only remains concentrated near the ring, whereas DriftSpread learns a broader distribution that follows this change. At $\lambda = 0$, Spread is omitted, and the reference is the uniform-circle limit of the Boltzmann targets as λ approaches zero.

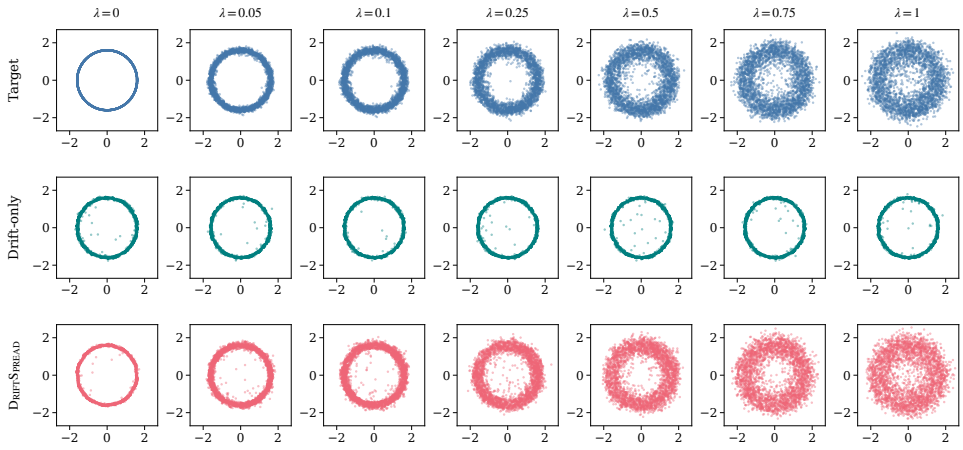


Figure 9: **Learning the ring-shaped target at different temperatures.** Columns vary the entropy coefficient λ. Rows show target samples, Drift-only samples, and samples from DRIFTSPREAD. At $\lambda = 0$, the reference is uniform on the circle and both training procedures omit Spread.

D.2 SEQUENTIAL FITTING VERSUS JOINT-LOSS WEIGHTING

Appendix C.1 compares sequential and joint fitting in a Gaussian example. Here, we compare them in online policy learning on Humanoid-v3 using Gaussian Spread noise. Joint minimizes $\mathcal{L}_{\text{Drift}} + \beta \mathcal{L}^{\text{old}}_{\text{Spread}}$ with $\beta \in \{0, 0.1, 1, 10\}$ fixed within each run. Both targets are constructed from the pre-Drift actor; the case $\beta = 0$ removes the Spread loss. Sequential instead constructs its Spread target after the Drift update. All other training settings are kept the same.

Figure 10 shows that Sequential achieves higher mean returns than Joint for all tested weights. Positive weights improve Joint over the zero-weight control, but none of the tested weights closes the gap to Sequential.

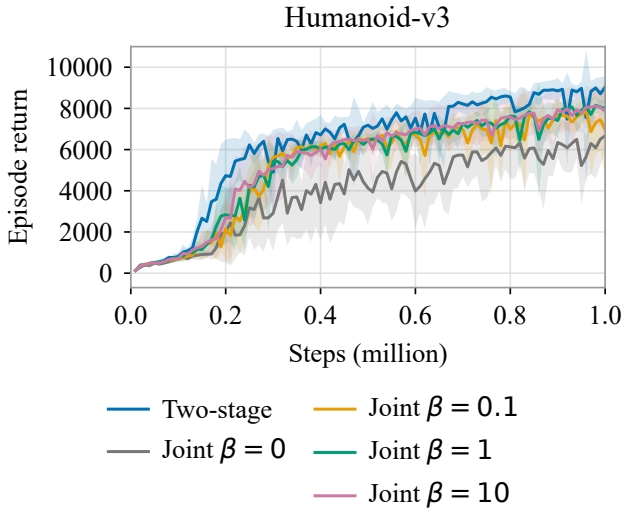


Figure 10: **Sequential and joint fitting on Humanoid-v3.** Joint weights the Spread MMD loss by β, with $\beta = 0$ using the Drift loss alone. Sequential is labeled *Two-stage* in the figure. Curves show unsmoothed means over five seeds, with sample standard deviations shaded.

D.3 Sensitivity to the Frequency Scale

The frequency distribution determines the Gaussian kernel used to compute the Spread MMD loss. We test its scale on Humanoid-v3 using $\sigma_\omega = \alpha_\omega \sigma_{\mathrm{G}}$, where $\sigma_{\mathrm{G}} = (2\lambda\eta d)^{-1/2}$ is the default and $\alpha_\omega \in \{0.1, 1, 10\}$. The Gaussian Spread target and all other training settings remain fixed.

Figure 11 shows that the default scale gives faster learning and higher late-training mean returns than either change. Thus the frequency scale affects learning even when the target distribution is unchanged. This comparison supports the default scale for the tested Humanoid-v3 setting.

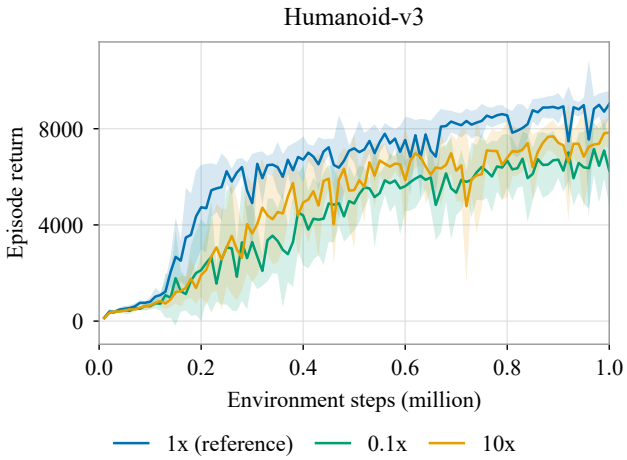


Figure 11: **Effect of the frequency scale on Humanoid-v3.** We multiply the default frequency standard deviation by 0.1, 1, or 10, keeping the spreading noise and other training settings fixed. Curves show unsmoothed means over five seeds, with sample standard deviations shaded.

E Experimental Details

We first describe the ring-target experiment. We then give the MuJoCo implementations and training settings, followed by the evaluation and runtime measurements and the frequency settings used in the noise comparison.

E.1 Ring-target Experiment

Target distribution. We use the two-dimensional critic

$$Q(a) = -\frac{(\|a\|^2 - r_0^2)^2}{4\sigma_Q^2}, \qquad r_0 = 1.6, \quad \sigma_Q = 0.8. \tag{49}$$

For $\lambda > 0$, the target is $\pi^*_{Q,\lambda}(a) \propto \exp(Q(a)/\lambda)$. To draw reference samples, sample the squared radius V from $\mathcal{N}(r_0^2, 2\sigma_Q^2\lambda)$ conditioned on $V > 0$, and independently sample $\Theta \sim \mathrm{Uniform}[0, 2\pi)$. Then set $a = \sqrt{V}(\cos\Theta, \sin\Theta)^\top$. At $\lambda = 0$, use $V = r_0^2$, giving a uniform distribution on the circle. Reference samples are used only for evaluation and visualization.

Actor and initialization. The actor is $f_\theta(z) = 0.8z + g_\theta(z)$, where $z \sim \mathcal{N}(0, I_2)$ and g_θ is an MLP with two hidden layers of width 64 and $\tanh$ activations. Its final linear layer is initialized to zero, so the initial output distribution is $\mathcal{N}(0, 0.8^2 I_2)$. Drift-only and DriftSpread use the same architecture and initialization.

Training targets. For this experiment, the critic gradient flow can be solved analytically. We use its exact action movement over duration $h = 0.025$ to construct the Drift target. Writing $R = r_0^2$ and

$v = \|a\|^2$, this movement is

$$D_h(a) = \frac{a}{\sqrt{e^{-2Rh/\sigma_Q^2} + (v/R)(1 - e^{-2Rh/\sigma_Q^2})}}, \qquad D_h(0) = 0. \tag{50}$$

The actor fits the fixed target $\mathrm{sg}[D_h(f_{\theta_k}(z))]$ through regression using the same input z. After Drift, we compute the updated actor's empirical characteristic function and multiply it by $e^{-\lambda h\|\omega\|^2}$. We hold this target fixed while optimizing the Spread MMD loss. The analytic Drift target is exact, while both actor fits use finite numerical optimization.

Optimization and sampling. We train for 400 Drift–Spread cycles. Each cycle draws 2,048 actor inputs. For $\lambda > 0$, it also draws 128 frequencies from $\mathcal{N}(0, \sigma_\omega^2 I_2)$ with $\sigma_\omega = (4\lambda h)^{-1/2}$. These inputs and frequencies remain fixed during that cycle's fitting steps. Each stage uses L-BFGS with at most 25 inner iterations, with learning-rate settings 1.0 for Drift and 0.5 for Spread. The optimizer history is reset before each stage, and its loss is divided by the initial loss value, with a lower bound of 10^{-12} on the divisor. Drift-only uses the same Drift settings and actor inputs, with Spread omitted. At $\lambda = 0$, DRIFTSPREAD also omits Spread and does not draw Spread frequencies.

Displayed samples. The ring experiments use seeds 23, 41, 59, 77, and 95. Each displayed distribution contains 600 samples per seed, for 3,000 points in total. Target panels use the same number of points, and all panels share the same axes. The Drift-only columns show different fixed subsets of the same per-seed control sample pools. Every generated sample is obtained with one actor forward pass.

E.2 MuJoCo Implementations and Training Settings

We use HalfCheetah-v3, Hopper-v3, Walker2d-v3, Ant-v3, and Humanoid-v3. Each method is trained for 10^6 environment interactions per task using five random seeds. The compared methods are listed in section 5.2.

Baseline implementations. We use the official baseline implementations as references for our JAX implementations. For methods originally implemented in PyTorch, we port the implementation to JAX for consistent runtime comparisons. We align batch size, network depth and width, and update-to-data ratio across methods. Other settings follow the source implementations, including learning rates, activation functions, sampling steps, and action-selection rules. For DFP, DPMD, and SDAC, we additionally adopt the same critic configuration as MFPO.

The source repositories for the baseline implementations are listed in Table 2.

Table 2: Source repositories for the baseline implementations. The SAC entry is the implementation referenced by MFPO. DPMD and SDAC share the same repository.

Method	Repository
MFPO	`dongxiaoyi-xyz/MFPO`
DIME	`ALRhub/DIME`
FlowRL	`bytedance/FlowRL`
SAC-Flow	`Elessar123/SAC-FLOW`
MaxEntDP	`diffusionyes/MaxEntDP`
DACER	`happy-yan/DACER-Diffusion-with-Online-RL`
QVPO	`wadx2019/qvpo`
TD3	`sfujim/TD3`
SAC	`toshikwa/soft-actor-critic.pytorch`
DPMD	`mahaitongdae/diffusion_policy_online_rl`
SDAC	`mahaitongdae/diffusion_policy_online_rl`
DFP	`KAIST-Visual-AI-Group/DFP`
WPPG	`Asimov-Chuang/wasserstein-proximal-policy-gradient`

Table 3 summarizes the hyperparameters used for the baseline methods.

Table 3: Hyperparameters of the baseline methods on MuJoCo. All methods use a batch size of 256, a discount factor $\gamma = 0.99$, and an update-to-data ratio of 1.

Hyperparameter	MFPO	DIME	FlowRL	SAC-Flow	MaxEntDP	DACER	QVPO
Target smoothing τ	0.005	1	0.95	1	0.005	0.005	0.005
Hidden layers	3	3	3	N/A	3	3	3
Hidden nodes per layer	256	256	256	N/A	256	256	256
Actor learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	10^{-4}	3×10^{-4}
Critic learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	10^{-3}	3×10^{-4}	10^{-4}	3×10^{-4}
Activation	GELU	GELU	ELU	GELU	Mish	Mish	Mish
Replay buffer size	10^6	10^6	10^6	10^6	10^6	10^6	10^6
Diffusion / generation steps	2	16	1	4	20	20	20
Action candidates	10	N/A	N/A	N/A	10	N/A	32

Hyperparameter	TD3	SAC	DPMD	SDAC	DFP	WPPG
Target smoothing τ	0.005	0.005	0.005	0.005	0.005	0.005
Hidden layers	3	3	3	3	3	3
Hidden nodes per layer	256	256	256	256	256	256
Actor learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Critic learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Activation	ReLU	ReLU	Mish	Mish	GELU	Tanh
Replay buffer size	10^6	10^6	10^6	10^6	2×10^6	10^6
Diffusion / generation steps	N/A	N/A	20	20	1	1
Action candidates	N/A	N/A	32	32	16	N/A

Training settings for DRIFTSPREAD**.** Table 4 lists the settings used across all five tasks. We use the same Drift learning rate on every task and the Spread learning rates in Table 5. The optimizer learning rates control parameter updates; the step size η controls the action displacement and spreading-noise variance used to construct training targets.

Table 4: Training settings for DRIFTSPREAD on MuJoCo. Entries marked † are aligned across all methods.

Parameter	Value	Parameter	Value
Batch size†	256	Drift samples per state N_{D}	4
Hidden layers†	3	Spread samples per state N_{CF}	32
Hidden width†	256	Frequencies per cycle M_ω	64
Update-to-data ratio†	1	Spreading noise	Gaussian
Discount γ	0.99	Entropy coefficient λ	0.04
Target update rate	0.005	Step size η	0.1
Critic learning rate	3×10^{-4}	Drift learning rate	2×10^{-5}
Activation	GELU	Evaluation candidates	10
Replay buffer size	10^6	Interactions per seed	10^6
Random seeds	0–4	Frequency distribution	Gaussian

Table 5: Spread learning rates for DRIFTSPREAD. The same task-specific rates are used in the spreading-noise comparison.

Task	Spread learning rate
HalfCheetah-v3	10^{-5}
Hopper-v3	10^{-5}
Walker2d-v3	2×10^{-5}
Ant-v3	1.5×10^{-5}
Humanoid-v3	10^{-5}

Bounded actions. The actor produces an unconstrained output $u = f_\theta(s, z)$, which is mapped to an environment action by $a = T(u)$. For each coordinate, $T_i(u_i) = c_i + r_i \tanh u_i$, where c_i and r_i are the center and half-width of the action range. Both actor losses are computed before this mapping. Drift uses the gradient of

$$Q^T(s, u) = \min_{i \in \{1,2\}} Q_{\phi_i}(s, T(u)) + \lambda \log J_T(u), \qquad J_T(u) = |\det DT(u)|. \tag{51}$$

The Jacobian term accounts for the change in entropy under T. Spread acts on the distribution of the updated actor outputs u. Critic learning uses mapped actions and the reward-only target in section 4.4.

E.3 Evaluation and runtime measurements

Control performance. The main MuJoCo learning curves report evaluation returns averaged over five seeds, with sample standard deviations shown as shaded bands. For DriftSpread and MFPO, evaluation generates ten candidate actions and executes the action with the highest critic value. Other methods use the action-selection rules of their reference implementations.

Aggregate normalized return. For each task and seed, we average the final ten evaluation returns. We divide each result by the corresponding task's mean return for DriftSpread, averaged over its seeds. We then average the normalized results across seeds and tasks, giving each task equal weight. Thus a normalized return of one corresponds to the DriftSpread reference level. The aggregate error bars are 95% bootstrap intervals obtained by resampling seed results while keeping the task-specific normalization constants fixed.

Training and inference time. The methods included in the efficiency comparison are timed using JAX implementations on the same NVIDIA RTX 4090 GPU. We exclude one-time compilation and warm-up from both training and inference measurements. Training measurements also exclude evaluation and experiment-management overhead.

Average training time is the total GPU time for one seed on all five tasks, each with 10^6 interactions, divided by five.

Inference time is measured per selected action and averaged across tasks. It includes candidate generation, critic scoring, and final selection when these steps are used. For DriftSpread and MFPO, this includes all ten evaluation candidates. Each candidate requires one generation step for DriftSpread and two for MFPO. The other recorded settings use one step for FlowRL, four for SAC-Flow, 16 for DIME, and 20 for DACER, QVPO, and MaxEntDP.

Policy evaluation is performed every 10,000 environment steps using 10 episodes.Each evaluation return is the arithmetic mean of the undiscounted cumulative rewards over these ten episodes.

E.4 Frequency Settings for the Noise Comparison

The spreading noise determines the target distribution, while the frequency distribution determines the MMD kernel used to compare it with the actor. For every noise choice, we sample $\omega = \sigma_\omega Z$ with $Z \sim \mathcal{N}(0, I_d)$. The resulting Gaussian kernel is $h(u, v) = \exp(-\sigma_\omega^2 \|u - v\|^2/2)$. Frequencies are resampled in each actor-update cycle; their distribution is fixed throughout a run.

For the default Gaussian noise, we use $\sigma_{\mathrm{G}} = (2\lambda\eta d)^{-1/2}$, as described in Appendix A.2.4. Let m_η denote the characteristic function of the spreading noise. At the default scale, the Gaussian noise factor has average value

$$\mu_* = \mathbb{E}_Z \left[e^{-\lambda\eta\sigma_{\mathrm{G}}^2 \|Z\|^2} \right] = (1 + 1/d)^{-d/2}. \tag{52}$$

For each alternative, we choose its frequency scale so that its noise factor has the same average value:

$$F(\sigma_\omega) = \mu_*, \qquad F(\sigma) = \mathbb{E}_Z[m_\eta(\sigma Z)]. \tag{53}$$

This provides a common rule for setting the frequency scale across noise choices.

To compute the scale, let ϵ_η be a noise sample independent of Z. Averaging over the Gaussian frequency first gives

$$F(\sigma) = \mathbb{E}_{\epsilon_\eta} \left[e^{-\sigma^2 \|\epsilon_\eta\|^2/2} \right]. \tag{54}$$

For the noise distributions used here, this function decreases continuously from one to zero. The required expectations have closed forms or reduce to one-dimensional integrals. We evaluate them numerically and solve equation (53) by bisection, without using policy returns to select the scale. The rule matches the average noise factor; it does not make the losses or their gradients identical. Because the scales differ, the noise comparison also uses different Gaussian kernel bandwidths.

For the Gamma-mixture noise used in the comparison, the shape parameter is fixed at $r = 4$.