EFFICIENT IMPLICIT BOLTZMANN POLICY OPTIMIZATION VIA MIXTURE LIKELIHOOD MAXIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Online reinforcement learning often requires representing complex, potentially multimodal action distributions induced by the value function. Recent diffusion and flow-based policies have substantially improved policy expressivity by enabling flexible approximations to complex action distributions. However, it remains challenging to achieve both high sampling fidelity and low inference latency with expressive generative policies, as they typically generate actions iteratively. While recent one-step approaches reduce inference cost, training these expressive policies has remained computationally expensive. We introduce **Efficient Implicit Boltzmann Policy Optimization via Mixture Likelihood Maximization (iBOLT)**, which efficiently trains a one-step policy to approximate the value-induced Boltzmann distribution. iBOLT represents the policy as a continuous mixture of truncated Gaussians, with component parameters generated from the state and a latent variable by a single network. Then, we use a tractable finite-mixture approximation to reweight sampled actions toward the Boltzmann target and train the policy by maximizing the weighted mixture likelihood. During deployment, the learned policy generates actions with a single network evaluation, without iterative refinement. Experiments on MuJoCo benchmarks and multimodal tasks demonstrate that iBOLT achieves performance competitive with state-of-the-art generative policy methods, while substantially reducing both training cost and inference latency.

1 INTRODUCTION

Reinforcement learning (RL) has emerged as a powerful paradigm for learning decision-making policies through interactions with the environment (Sutton and Barto, 2018), achieving strong performance across a wide range of continuous-control tasks such as robotics (Wu et al., 2023; Chi et al., 2025; Ke et al., 2024). Many widely adopted deep RL algorithms for continuous control employ deterministic or simple Gaussian policies, enabling efficient sampling and optimization (Haarnoja et al., 2018; Fujimoto et al., 2018; Schulman et al., 2017). However, their limited expressivity may restrict the diversity of behaviors explored during learning, potentially leading to suboptimal solutions (Wang et al., 2022; Yang et al., 2023). For instance, when a state admits several distinct high-value action regions, a single Gaussian may struggle to cover them without also assigning probability to low-value actions in between. Earlier online RL methods addressed this representational limitation using implicit (Tang and Agrawal, 2019) or semi-implicit (Yue et al., 2020) policies that can model multimodal action distributions, but their reverse-KL-based policy updates tend to concentrate probability mass on a subset of modes, leaving this representational capacity underutilized.

Recent works have adopted diffusion and flow models (Ho et al., 2020; Song and Ermon, 2019) as expressive policy representations to address the limited expressivity of conventional policy classes in online RL (Yang et al., 2023; Psenka et al., 2025; Ding et al., 2024; Wang et al., 2024; Zhang et al., 2026). These expressive generative policies can capture complex and multi-modal action distributions, enabling more diverse exploration and strong empirical performance. However, it remains challenging to achieve high sampling fidelity while maintaining low inference latency, as they typically rely on iterative action sampling that requires multiple sequential network evaluations.

To overcome these limitations, several approaches have sought to reduce inference cost through one-step or few-step action generation (Chao et al., 2024; Dong et al., 2026). Still, training these

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.

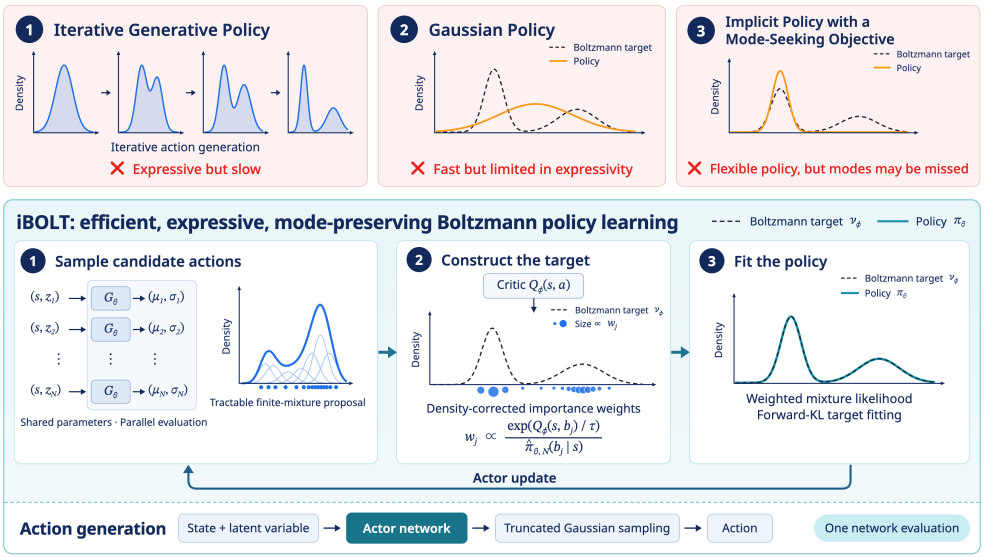

Figure 1: Overview of iBOLT. **(Top)** Iterative generative policies are expressive but slow, Gaussian policies are fast but unimodal, and implicit policies with mode-seeking objectives can collapse onto a single mode. **(Bottom)** iBOLT weights actions sampled from a latent-conditioned truncated Gaussian mixture by density-corrected critic values and fits the mixture to them by weighted likelihood, generating actions with a single network evaluation.

expressive policies has remained time-consuming due to the computational overhead of their specialized architectures and complex training procedures. This raises a more fundamental question: can we learn an expressive one-step policy without the substantial training overhead of existing generative approaches?

In this work, we answer this question by revisiting implicit policy parametrizations (Yue et al., 2020; Tang and Agrawal, 2019). We introduce **Efficient Implicit Boltzmann Policy Optimization via Mixture Likelihood Maximization (iBOLT)**, an efficient method that trains an implicit policy to approximate the value-induced Boltzmann distribution. iBOLT represents the policy as a continuous mixture of truncated Gaussians, with component parameters generated from the state and a latent variable by a single network. For each state, we sample candidate actions from a tractable finite-mixture approximation of the policy and weight them by exponentiated critic values, correcting for the proposal density to approximate the Boltzmann target. We then train the policy by maximizing the weighted mixture likelihood on these candidates. This objective provides a sample-based surrogate for forward KL minimization, encouraging coverage of high-value regions represented in the candidate samples. The resulting policy generates actions with a single network evaluation, without requiring a score model, a velocity field, or an invertible policy architecture.

Our main contributions are summarized as follows:

- We introduce **iBOLT**, an efficient method for learning expressive semi-implicit policy with one-step action generation. By combining a tractable finite-mixture approximation with density-corrected importance weight and weighted mixture likelihood maximization, iBOLT fits the policy to the value-induced Boltzmann distribution.
- We interpret the proposed objective as a sample-based surrogate for forward KL minimization and examine its behavior under the semi-implicit policy representation and importance sampling approximation. Comparisons with reverse KL training using the same policy architecture demonstrate improved mode coverage on multimodal targets.
- We empirically show that iBOLT accurately covers multimodal target distributions and achieve competitive control performance on MuJoCo benchmarks, with substantially lower training cost than other expressive policies and achieve inference latency matching that of Gaussian policies.

2 Related Work

Implicit and Semi-Implicit Policies for Online RL. Early work on expressive online RL policies explored implicit action generators beyond conventional Gaussian parameterizations. SQL (Haarnoja et al., 2017) learns an implicit sampler for a value-induced Boltzmann policy through amortized Stein variational gradient descent. IDAC (Yue et al., 2020) combines implicit return-distribution critics with a semi-implicit actor whose marginal policy mixes latent-conditioned Gaussian distributions. It optimizes an entropy-regularized objective using a finite-sample entropy bound to accommodate the intractable marginal policy density. These methods support expressive, potentially multimodal action distributions.

However, their implicit parameterizations complicate direct likelihood-based policy fitting, with policy optimization instead relying on kernel-based Stein updates, autoregressive quantile regression, or finite-sample entropy bounds. In contrast, iBOLT constructs a tractable finite-sample density for its implicit mixture policy, enabling density-corrected self-normalized importance sampling and direct likelihood-based fitting to the value-induced Boltzmann distribution.

Generative Policies for Online RL. Diffusion-based policies provide expressive alternatives to Gaussian policies and differ primarily in how value information is incorporated into learning. Gradient-based methods such as DIPO (Yang et al., 2023), QSM (Psenka et al., 2025), and DACER (Wang et al., 2024) use critic gradients to guide action refinement, score learning, or denoising-based policy optimization. In contrast, QVPO (Ding et al., 2024), DPMD, and SDAC (Ma et al., 2025) incorporate value information through Q-weighted or reweighted generative objectives without directly differentiating the critic through the policy. Beyond diffusion, SAC-Flow (Zhang et al., 2026) employs a flow-based policy with velocity parameterizations to stabilize multi-step generative policy optimization within soft actor-critic.

Another line of work aims to reduce the iterative generation required for expressive policy sampling. MEow (Chao et al., 2024) uses a state-conditioned energy-based normalizing flow to jointly parameterize the policy and its associated soft value functions, enabling non-iterative action sampling through an invertible architecture. MFPO (Dong et al., 2026) adopts MeanFlow policies that learn average velocity fields for few-step generation, together with an auxiliary average-divergence network for tractable likelihood estimation. These approaches alleviate iterative generation, but still rely on specialized generative architectures with substantial training overhead. In contrast, iBOLT adopts a sample-based policy optimization framework with a tractable value-weighted mixture likelihood, retaining the simplicity of value-weighted policy optimization and while enabling expressive, multimodal one-step action generation without unnecessary training and inference overhead.

3 Preliminaries

Reinforcement Learning. We consider a discounted Markov decision process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A} \subseteq \mathbb{R}^{d_a}$ denote the state and continuous action spaces, respectively. The transition kernel $P(\cdot \mid s, a)$ characterizes the distribution of the next state given the current state-action pair, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function, and $\gamma \in (0, 1)$ is the discount factor. The goal is to learn a policy $\pi(a|s)$ that maximizes the expected discounted return. We denote the action-value function of policy π by $Q^\pi(s, a)$, which satisfies the Bellman equation: $Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a), a' \sim \pi(\cdot|s')} [Q^\pi(s', a')]$. We parameterize the policy as π_θ and approximate its action-value function with a critic Q_ϕ, trained using transitions stored in a replay buffer $\mathcal{D} = \{(s, a, s')_i\}_{i=1}^{|\mathcal{D}|}$.

Boltzmann policies. Given a critic $Q(s, a)$ and a temperature $\tau > 0$, the value-induced Boltzmann distribution is defined as

$$\nu_Q(a \mid s) = \frac{\exp(Q(s, a)/\tau)}{Z(s)}, \qquad Z(s) = \int_{\mathcal{A}} \exp(Q(s, a)/\tau)\, da, \tag{1}$$

The temperature τ controls how strongly the distribution favors high-value actions: smaller τ concentrates probability near the maximizers of Q, while larger τ spreads it over all high-value regions, allowing the policy to explore several promising alternatives rather than commit to a single action (Jain et al., 2024).

4 METHOD

We introduce **One-Step Boltzmann Policy Optimization via Mixture Likelihood Maximization (iBOLT)**, an efficient method for learning expressive semi-implicit policies. Our goal is to fit the policy to the value-induced Boltzmann distribution by minimizing the forward KL divergence. This requires evaluating the policy's marginal density and estimating expectations under the Boltzmann target, both of which are generally intractable. We first define the semi-implicit policy representation, then derive a tractable learning objective using a finite-mixture approximation and self-normalized importance sampling.

4.1 ONE-STEP POLICY REPRESENTATION

We parameterize the policy using a network $G_\theta(s, z)$ that maps a state s and a latent variable z to the location and scale parameters of a truncated Gaussian on $\mathcal{A} = [-1, 1]^{d_a}$:

$$\begin{aligned}(\mu_\theta(s, z), \sigma_\theta(s, z)) &= G_\theta(s, z), \\ k_\theta(a \mid s, z) &= \mathcal{N}_{\mathrm{Tr}}\big(a; \tanh(\mu_\theta(s, z)), \operatorname{diag} \sigma_\theta^2(s, z); \mathcal{A}\big),\end{aligned} \tag{2}$$

where $z \sim p_z = \mathcal{N}(0, I_{d_z})$ is a latent variable, $\sigma_\theta(s, z) > 0$, and $\mathcal{N}_{\mathrm{Tr}}(a; m, \Sigma; \mathcal{A})$ denotes the density obtained by truncating $\mathcal{N}(m, \Sigma)$ to $\mathcal{A}$. The $\tanh$ maps the location into the action bounds. Marginalizing out the latent variable yields the policy as a continuous mixture of these conditional truncated Gaussians:

$$\pi_\theta(a \mid s) = \int p_z(z)\, k_\theta(a \mid s, z)\, dz. \tag{3}$$

This defines a semi-implicit policy $\pi_\theta(\cdot \mid s)$ with a tractable conditional density $k_\theta(a \mid s, z)$ but a generally intractable marginal density. This continuous mixture provides an expressive policy representation without restricting the policy to a fixed number of Gaussian components. Sampling an action from $\pi_\theta(\cdot \mid s)$ requires only a single evaluation of $G_\theta(s, z)$ with $z \sim p_z$, followed by elementwise truncated Gaussian sampling, enabling efficient action generation without iterative refinement.

4.2 BOLTZMANN POLICY LEARNING VIA FORWARD KL MINIMIZATION

We first learn a critic Q_ϕ that approximates the action-value function of the current policy by minimizing the temporal-difference loss:

$$\mathcal{L}_{\mathrm{critic}}(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\phi(s, a) - r - \gamma\, \mathbb{E}_{a'\sim\pi_\theta(\cdot|s')}\Big[\min_{\ell=1,2} Q_{\bar{\phi}_\ell}(s', a')\Big]\right)^2\right], \tag{4}$$

where $\bar{\phi}_1$ and $\bar{\phi}_2$ denote target-critic parameters. The learned critic defines the value-induced Boltzmann target for policy learning

$$\nu_\phi(a \mid s) = \frac{\exp(Q_\phi(s, a)/\tau)}{Z_\phi(s)}, \qquad Z_\phi(s) = \int_{\mathcal{A}} \exp(Q_\phi(s, a)/\tau)\, da, \tag{5}$$

where the temperature $\tau > 0$ controls the concentration on high-value actions. Then, we fit the policy π_θ to this target ν_ϕ by minimizing the forward KL divergence:

$$D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)) = \mathbb{E}_{a\sim\nu_\phi(\cdot|s)}\left[\log \frac{\nu_\phi(a \mid s)}{\pi_\theta(a \mid s)}\right]. \tag{6}$$

During policy optimization, we treat the critic parameters ϕ as constants. Prior online RL methods with implicit or semi-implicit policies use reverse-KL-based policy updates (Tang and Agrawal, 2019; Yue et al., 2020). Such objectives have a *mode-seeking* tendency and can concentrate probability mass on a subset of target modes (Minka et al., 2005). We instead choose forward KL to encourage coverage of high-value action regions, including those assigned little probability by the current policy. Our controlled comparison in Section 5.1 supports this choice: with the same iBOLT architecture, reverse-KL training misses target modes that our forward-KL training recovers.

Still, the nonlinear forms of $G_\theta(s, z)$ and $Q_\phi(s, a)$ generally prevent closed-form evaluation of the marginal policy density and the Boltzmann expectation. To address these challenges, we first

construct a Monte Carlo approximation of the marginal policy density using independent latent samples $z_1, \dots, z_N \overset{\text{i.i.d.}}{\sim} p_z$:

$$\pi_\theta(a \mid s) = \mathbb{E}_{z\sim p_z}[k_\theta(a \mid s, z)] \approx \frac{1}{N}\sum_{i=1}^{N} k_\theta(a \mid s, z_i) =: \hat{\pi}_{\theta,N}(a \mid s; z_{1:N}). \tag{7}$$

This finite mixture $\hat{\pi}_{\theta,N}$ allows exact density evaluation conditional on the latent samples $z_{1:N}$ and is an unbiased estimator of $\pi_\theta(a \mid s)$ for each a. Replacing π_θ with $\hat{\pi}_{\theta,N}$ yields an upper bound on the forward KL in expectation over the latent samples, as formalized below.

Proposition 1 (Finite-mixture upper bound on forward KL)**.** *For a fixed state s and parameters θ, ϕ, let $z_1, \dots, z_N \overset{\text{i.i.d.}}{\sim} p_z$ and define $U_N(s) := \mathbb{E}_{z_{1:N}}[D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \hat{\pi}_{\theta,N}(\cdot \mid s; z_{1:N}))]$. Under the assumptions stated in Appendix A.1, the following holds for any integer $N \geq 1$:*

$$D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)) \leq U_{N+1}(s) \leq U_N(s). \tag{8}$$

Moreover, the bound is asymptotically tight: $\lim_{N\to\infty} U_N(s) = D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s))$.

This result follows from Lemma 1 of Yin and Zhou (2019); the proof is provided in Appendix A.1. We therefore adopt $U_N(s)$ as an upper-bound objective for forward KL minimization.

4.3 Importance-Weighted Mixture Likelihood Maximization

We now derive a sample-based actor objective from the finite-mixture upper bound established in Proposition 1. Expanding this bound gives

$$D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)) \leq U_N(s) = \mathbb{E}_{z_{1:N}\sim p_z}\left[\mathbb{E}_{a\sim\nu_\phi(\cdot|s)}\left[\log \frac{\nu_\phi(a \mid s)}{\hat{\pi}_{\theta,N}(a \mid s; z_{1:N})}\right]\right]. \tag{9}$$

Still, the inner expectation under the Boltzmann target in Eq. (9) is generally intractable. To this end, we rewrite this expectation under the tractable proposal $q(\cdot \mid s; z_{1:N}) = \hat{\pi}_{\theta,N}(\cdot \mid s; z_{1:N})$. Since π_θ is trained to approximate ν_ϕ, its finite-mixture approximation provides a natural proposal that allows direct sampling and tractable density evaluation. For candidate actions $b_1, \dots, b_M \overset{\text{i.i.d.}}{\sim} q(\cdot \mid s; z_{1:N})$, the corresponding importance-sampling approximation is

$$\begin{aligned}\mathbb{E}_{a\sim\nu_\phi(\cdot|s)}\left[\log \frac{\nu_\phi(a|s)}{\hat{\pi}_{\theta,N}(a|s; z_{1:N})}\right] &= \mathbb{E}_{a\sim q(\cdot|s)}\left[\frac{\nu_\phi(a|s)}{q(a|s; z_{1:N})}\big(\log \nu_\phi(a|s) - \log \hat{\pi}_{\theta,N}(a|s; z_{1:N})\big)\right] \\ &\approx \frac{1}{M}\sum_{m=1}^{M} \frac{\nu_\phi(b_m|s)}{q(b_m|s; z_{1:N})}\big(\log \nu_\phi(b_m|s) - \log \hat{\pi}_{\theta,N}(b_m|s; z_{1:N})\big),\end{aligned} \tag{10}$$

where the importance weight $\frac{\nu_\phi(a|s)}{q(a|s;z_{1:N})}$ corrects for sampling from q instead of ν_ϕ. Still, the target normalizing constant $Z_\phi(s)$ is unknown, so we can evaluate only the unnormalized importance weights $\tilde{w}(a \mid s) := \frac{\exp(Q_\phi(s,a)/\tau)}{q(a|s;z_{1:N})}$, which satisfy $\frac{\nu_\phi(a|s)}{q(a|s;z_{1:N})} = \frac{\tilde{w}(a|s)}{Z_\phi(s)}$. We therefore use self-normalized importance sampling (SNIS), normalizing the weights of these candidate actions as

$$w_j = \frac{\tilde{w}(b_j \mid s)}{\sum_{m=1}^{M} \tilde{w}(b_m \mid s)} = \frac{\exp(Q_\phi(s, b_j)/\tau)/q(b_j \mid s; z_{1:N})}{\sum_{m=1}^{M} \exp(Q_\phi(s, b_m)/\tau)/q(b_m \mid s; z_{1:N})}. \tag{11}$$

These normalized weights can be evaluated without knowing $Z_\phi(s)$. Using these normalized weights, we approximate the inner expectation as

$$D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \hat{\pi}_{\theta,N}(\cdot \mid s; z_{1:N})) \approx \sum_{j=1}^{M} w_j \Big[\underbrace{\log \nu_\phi(b_j \mid s)}_{\text{independent of } \theta} - \log \hat{\pi}_{\theta,N}(b_j \mid s; z_{1:N})\Big]. \tag{12}$$

For each fixed state and set of latent samples, SNIS provides a biased but consistent estimator of the inner expectation, converging in probability as $M \to \infty$ (Owen, 2013).

The first term in Eq. (12) is independent of θ, so minimizing this sample-based surrogate reduces to maximizing the weighted mixture log-likelihood. Finally, averaging over replay-buffer states, latent

samples, and candidate actions gives the policy loss:

$$\mathcal{L}_{\text{actor}}(\theta) = -\mathbb{E}_{s\sim\mathcal{D},\, z_{1:N}\sim p_z,\, b_{1:M}\sim q(\cdot|s;z_{1:N})}\left[\sum_{j=1}^{M} w_j \log \hat{\pi}_{\theta,N}(b_j \mid s; z_{1:N})\right] \tag{13}$$

$$= -\mathbb{E}_{s\sim\mathcal{D},\, z_{1:N}\sim p_z,\, b_{1:M}\sim q(\cdot|s;z_{1:N})}\left[\sum_{j=1}^{M} w_j \log\left(\frac{1}{N}\sum_{i=1}^{N} k_\theta(b_j \mid s, z_i)\right)\right]. \tag{14}$$

When optimizing this loss, we treat the candidate actions b_j and importance weights w_j as constants, differentiating only through the mixture log-likelihood. The finite-mixture density thus enables both proposal correction and direct likelihood-based policy learning. Here, N controls the marginal-density approximation, while M controls the approximation of the expectation under the Boltzmann target. The finite-mixture bound approaches the forward KL objective as $N \to \infty$, and the SNIS estimate is consistent as $M \to \infty$. In practice, we optimize a finite-sample surrogate in Eq. (14) using finite N and M.

To understand how the weighted likelihood updates individual mixture components, we examine its gradient:

$$\nabla_\theta \mathcal{L}_{\text{actor}} = -\mathbb{E}_{s,z_{1:N},b_{1:M}}\left[\sum_{j=1}^{M}\sum_{i=1}^{N} w_j r_{ij} \nabla_\theta \log k_\theta(b_j|s,z_i)\right],\ r_{ij} = \frac{k_\theta(b_j|s,z_i)}{\sum_{l=1}^{N} k_\theta(b_j|s,z_l)}. \tag{15}$$

Here, r_{ij} is the responsibility of component i for reference action b_j. The importance weight w_j determines the contribution of action b_j to the update, while r_{ij} distributes this contribution across mixture components according to their relative densities at b_j. Thus, a high-weight action provides a larger learning signal to components that already assign it relatively high density, allowing components to respond differently to the same candidate actions $b_{1:M}$. Further, we show that the number of latent samples used in the finite-mixture surrogate affects component specialization: increasing N weakens a leading-order penalty against it (Theorem 1 in Appendix A.2).

In summary, iBOLT performs two steps: reweighting candidate actions $b_{1:M}$ to approximate the Boltzmann target ν_ϕ and fitting the policy π_θ to this target through importance-weighted mixture likelihood maximization. This formulation enables direct likelihood-based training of an expressive policy without auxiliary score or velocity-field objectives, while generating actions in a single step without iterative refinement. Pseudocode for iBOLT is provided in Appendix B. Our experiments in Section 5.1 further show that this forward-KL-based training mitigates mode dropping when fitting multimodal targets, compared with reverse-KL training using the same policy architecture.

5 Experiments

Our experiments aim to answer three questions: (i) Does Forward KL capture target modes more reliably than Reverse KL under the same iBOLT architecture? (ii) How well does iBOLT recover multimodal target distributions compared with existing expressive policy classes, and does this translate into discovering multiple behavioral modes in control tasks? (iii) How does iBOLT compare with existing methods in returns, training time, and inference latency across MuJoCo continuous-control benchmarks?

5.1 Multimodal Distribution Learning and Behavioral Diversity

We evaluate the effect of the learning objective, the ability to recover complex multimodal targets, and behavioral diversity in control tasks.

Effect of the learning objective. Although it is well known that forward KL encourages mode coverage (Minka et al., 2005; Jerfel et al., 2021), its optimization can still encounter local optima that miss target modes (Jin et al., 2016; Chen et al., 2024). It is therefore important to test whether this benefit persists with iBOLT's semi-implicit parameterization and finite-sample SNIS approximation. To this end, we fit a semi-implicit model to five predefined one-dimensional multimodal distributions using forward- and reverse-KL training. Both methods use the same two-layer MLP architecture to generate the conditional truncated Gaussian parameters, with matched initializations. We use $N =$

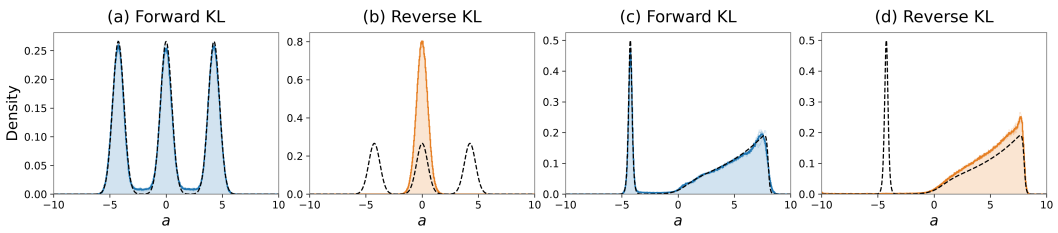

Figure 2: **Forward- versus reverse-KL fitting with the same iBOLT architecture.** Black dashed lines denote the target densities, and colored curves show the learned densities. Forward-KL training captures all target modes, whereas reverse-KL training consistently misses at least one mode.

128 latent samples and $M = 128$ candidate actions per sampling group for both methods. Forward-KL training uses iBOLT's SNIS-weighted mixture likelihood objective, while reverse-KL training minimizes the reverse KL divergence using a reparameterized gradient estimator. Figure 2a-b shows a three-mode Gaussian target: forward-KL training captures all three modes, whereas reverse-KL training concentrates on the central mode. Figure 2c-d considers a non-Gaussian target combining a narrow peak with a broad rising region. Forward-KL training captures both regions, whereas reverse-KL training fits the broad region but misses the narrow peak in practice. These results demonstrate that iBOLT's forward-KL training achieves broader mode coverage than reverse-KL training under the same semi-implicit parameterization, capturing target modes that the reverse-KL baseline misses. Detailed experimental settings are provided in Appendix C.2.

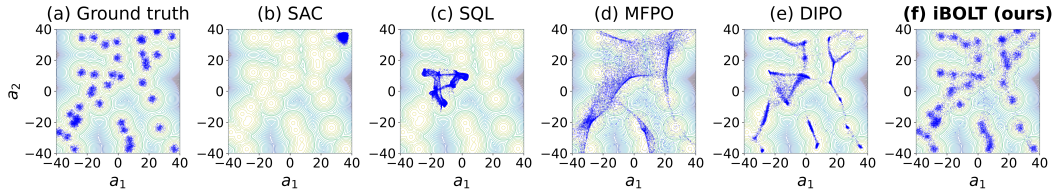

Figure 3: GMM40 final-policy samples with 10,000 samples. We compare the ground-truth target with SAC (Haarnoja et al., 2018) (Gaussian policy), SQL (Haarnoja et al., 2017) (implicit Stein sampler), MFPO (Dong et al., 2026) (MeanFlow policy), DIPO (Yang et al., 2023) (diffusion policy), and iBOLT (semi-implicit mixture policy). The experimental setup is detailed in Appendix C.3.

Table 1: GMM40 sampling quality after 100,000 actor updates. Mean ± sample standard deviation over 4 random seeds.

Algorithm	Ground truth	SAC	SQL	MFPO	DIPO	**iBOLT (ours)**
MMD ↓	–	0.8112 ± 0.0519	0.3627 ± 0.0554	0.0949 ± 0.0164	0.1276 ± 0.0064	$\mathbf{0.0260 \pm 0.0050}$
Mode coverage (/40) ↑	40.00	1.75 ± 0.96	14.00 ± 5.35	36.00 ± 0.82	36.50 ± 1.73	$\mathbf{40.00 \pm 0.00}$

Multimodal target recovery. We next compare iBOLT with baseline methods on a more complex multimodal target distribution. We use a two-dimensional Gaussian mixture with 40 components (GMM40) (Jain et al., 2024) as the target distribution. This task can be viewed as Boltzmann policy learning with a fixed value function $Q(a) = \log p_{\mathrm{GMM40}}(a)$ and temperature $\tau = 1$, for which the Boltzmann target is the GMM40 distribution restricted and renormalized to the bounded action domain. As shown in Figure 3 and Table 1, iBOLT covers all 40 target components in every seed under our coverage criterion and achieves the lowest MMD (0.0260 ± 0.0050) among the compared methods. While the preceding comparison examines the learning objective with the same model architecture, this experiment compares iBOLT with expressive policy learning methods and a Gaussian policy baseline on a richer target. Experimental details are provided in Appendix C.3.

Behavior diversity in control tasks. We then examine whether iBOLT learns diverse behaviors when the critic is learned through environment interaction rather than specified by a fixed target. We evaluate navigation environments with 4, 8, and 16 goals and multi-goal PointMaze tasks (Fu et al., 2020), training for approximately 10^6 environment interactions. In the navigation environments, iBOLT generates trajectories toward multiple goals, alongside learned value surfaces with

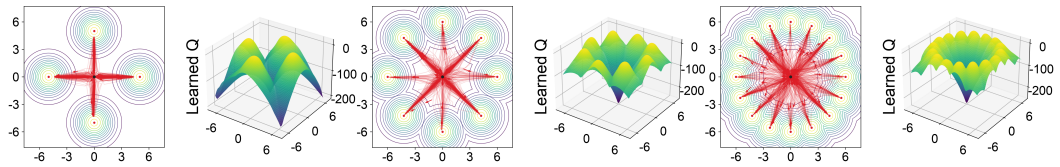

Figure 4: Policy trajectories and learned value surfaces of iBOLT ($\tau = 10$) in the 4, 8, and 16-goal environments after approximately 10^6 environment interactions. Each pair shows 1,024 rollout trajectories in red over goal-proximity contours (left) and the learned Q-value surface (right).

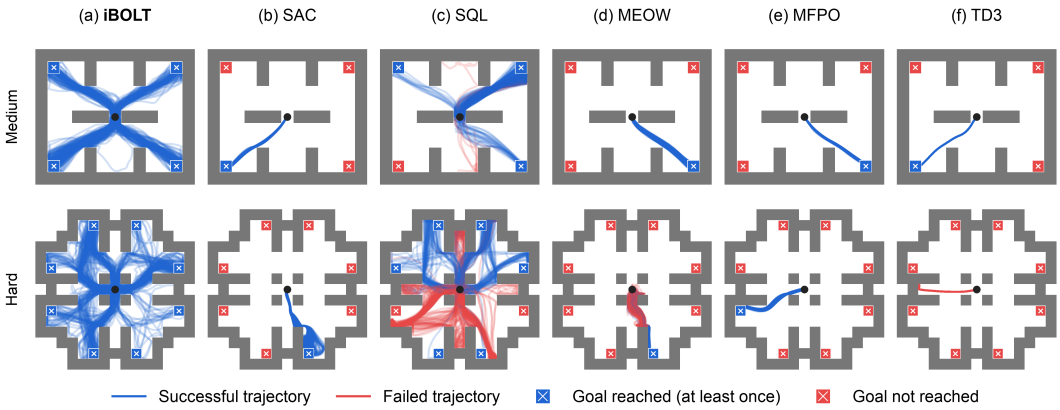

Figure 5: Policy trajectories on multi-goal PointMaze tasks. The top and bottom rows show the Medium and Hard mazes, respectively. Each panel contains 500 evaluation rollouts from the same initial state, marked by a black dot. Blue and red trajectories indicate successful and failed rollouts. Goal squares are blue if reached at least once and red otherwise. Among the six methods shown, iBOLT is the only one whose rollouts reach all four goals in Medium and all eight goals in Hard.

multiple high-value regions (Figure 4). In PointMaze, we evaluate 500 rollouts from the same initial state for each method. As shown in Figure 5, iBOLT reaches all four goals in Medium and all eight goals in Hard, while every compared method leaves some goals unreached. These evaluations provide qualitative evidence that iBOLT learns diverse goal-reaching behaviors with a learned critic. Experimental details are provided in Appendix C.5.

5.2 Benchmark Performance and Computational Efficiency

We compare iBOLT with SAC (Haarnoja et al., 2018), TD3 (Fujimoto et al., 2018), SQL (Haarnoja et al., 2017), MEow (Chao et al., 2024), DIPO (Yang et al., 2023), and MFPO (Dong et al., 2026) on five MuJoCo (Todorov et al., 2012) continuous-control benchmarks. We measure control performance, training time, and action-generation latency to assess whether efficient computation is accompanied by competitive returns. During data collection, iBOLT adds Gaussian exploration noise with a scale adapted as in DACER (Wang et al., 2024); implementation and evaluation details are provided in Appendices C.4 and C.6.

Figure 6a-e shows that iBOLT achieves competitive returns across the five tasks, with particularly strong performance on Hopper and Walker2d. It achieves the highest average normalized return across tasks (Figure 6f). This performance is obtained with approximately one hour of training, substantially less than MFPO and DIPO in the reported comparison. At inference time, iBOLT generates actions with a single network evaluation followed by truncated Gaussian sampling, achieving latency comparable to the simple Gaussian and deterministic policies used by SAC and TD3. Specifically, iBOLT achieve a latency of 0.20 ms under the same evaluation setup (Table 2). MFPO and the 100-step DIPO sampler require approximately six times and more than 100 times this latency, respectively. Thus, iBOLT combines competitive returns with lower training cost and inference latency than the evaluated diffusion and MeanFlow baselines.

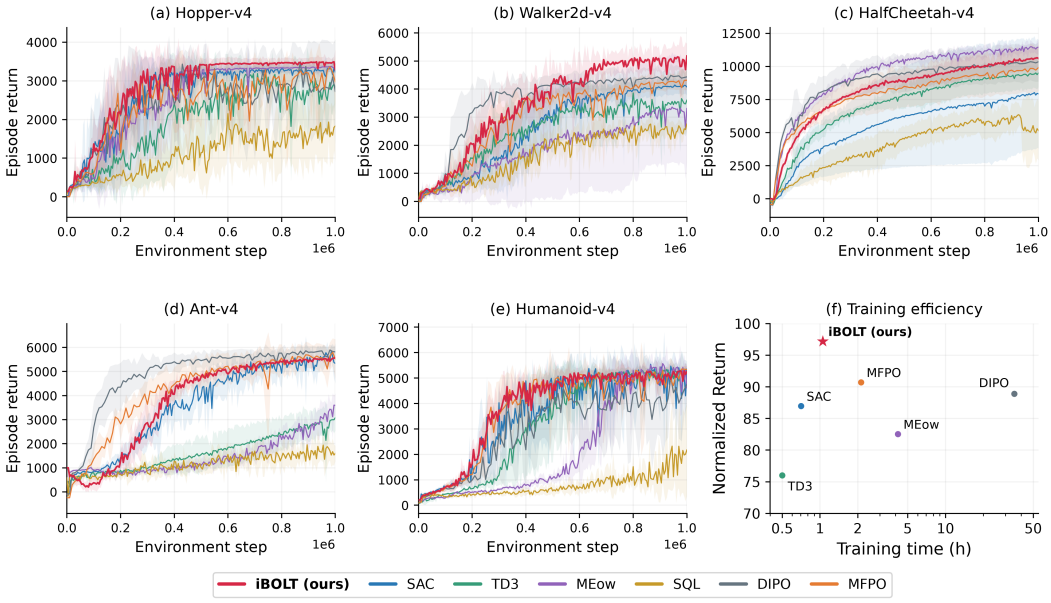

Figure 6: Comparison on 5 MuJoCo benchmarks. (a–e) Learning curves (mean $\pm$ one standard deviation over 5 runs). (f) Average normalized return versus training time (log scale). Returns are averaged over the final 100K steps and normalized so that the best method on each task scores 100. SQL is omitted from (f), as its score falls below the displayed range.

Table 2: Sampling steps and average inference latency per action on Humanoid-v4 using a single RTX 5090 (batch size 1), averaged over 5 random seeds.

Algorithm	**iBOLT (ours)**	SAC	TD3	SQL	MEow	DIPO	MFPO
Sampling steps	1	1	1	1	1	100	2
Inference time (ms)	**0.20**	0.20	0.20	0.69	1.33	33.35	1.17

5.3 Ablation Studies

We study three factors affecting iBOLT's behavior, with full results in Appendix D: the Boltzmann temperature τ, the exploration noise used during data collection, and the numbers of latent and reference samples (N, M), which control the finite-mixture and SNIS approximations.

6 Conclusion

We presented iBOLT, an efficient method for learning expressive semi-implicit policies with one-step action generation in online reinforcement learning. iBOLT represents the policy as a continuous mixture of truncated Gaussians and fits it to the value-induced Boltzmann distribution through forward-KL-based learning. To train this policy efficiently, we use a tractable finite-mixture approximation to optimize an importance-weighted mixture likelihood objective, while retaining action generation with a single network evaluation. Our experiments demonstrate improved mode coverage over reverse-KL training with the same semi-implicit model, accurate fitting of complex multimodal targets, and diverse behaviors in control tasks. On MuJoCo benchmarks, iBOLT achieves competitive returns while reducing training time and inference latency relative to the evaluated expressive policy baselines, with inference latency matching conventional Gaussian and deterministic policies in our evaluation. These results show that expressive policies can be trained through direct likelihood-based optimization while retaining efficient one-step action generation.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language polishing, plot preparation, and code modification. Generative AI tools did not contribute to the mathematical derivations, algorithmic formulation, or core research ideas presented in this work. All AI-assisted text, plots, and code modifications were reviewed and cross-checked by the authors for correctness, consistency with the underlying methodology and experiments, and faithfulness to the authors' intended content. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We describe the iBOLT policy parameterization, the importance-weighted construction of the Boltzmann target, and the mixture likelihood objective in Section 4. The relation of the objective to forward KL minimization is formalized in Proposition 1, whose assumptions and proof are given in Appendix A.1, and the consistency of the self-normalized importance sampling estimate follows from a standard result that we cite (Owen, 2013). Appendix C specifies the hardware, the implementation and computational framework of every baseline together with the official code repositories we use, the network architectures, and all shared and iBOLT-specific hyperparameters, including the per-task temperatures and the settings of the exploration noise used during data collection. The same appendix details the setups of the controlled forward-versus-reverse KL experiment, GMM40, the maze tasks, and the MuJoCo benchmarks, including the numbers of seeds, training budgets, and evaluation protocols, and Appendix D describes the ablation studies. We provide anonymized source code for iBOLT and scripts to reproduce the main experiments in the supplementary material.

REFERENCES

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on robot learning*, pages 2226–2240. PMLR, 2023.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

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

Long Yang, Zhixiong Huang, Fenghao Lei, Yucun Zhong, Yiming Yang, Cong Fang, Shiting Wen, Binbin Zhou, and Zhouchen Lin. Policy representation via diffusion probability model for reinforcement learning. *arXiv preprint arXiv:2305.13122*, 2023.

Yunhao Tang and Shipra Agrawal. Implicit policy for reinforcement learning, 2019. URL https://arxiv.org/abs/1806.06798.

Yuguang Yue, Zhendong Wang, and Mingyuan Zhou. Implicit distributional reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020. URL http://arxiv.org/abs/2007.06159.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. URL https://arxiv.org/abs/2006.11239.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via q-score matching, 2025. URL https://arxiv.org/abs/2312.11752.

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. *Advances in Neural Information Processing Systems*, 37:53945–53968, 2024.

Yinuo Wang, Likun Wang, Yuxuan Jiang, Wenjun Zou, Tong Liu, Xujie Song, Wenxuan Wang, Liming Xiao, Jiang Wu, Jingliang Duan, et al. Diffusion actor-critic with entropy regulator. *Advances in Neural Information Processing Systems*, 37:54183–54204, 2024.

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*, volume 2026, pages 121983–122016, 2026.

Chen-Hao Chao, Chien Feng, Wei-Fang Sun, Cheng-Kuang Lee, Simon See, and Chun-Yi Lee. Maximum entropy reinforcement learning via energy-based normalizing flow. *Advances in Neural Information Processing Systems*, 37:56136–56165, 2024.

Xiaoyi Dong, Xi Sheryl Zhang, and Jian Cheng. Mean flow policy optimization. In *Forty-third International Conference on Machine Learning*, 2026. URL https://arxiv.org/pdf/2604.14698.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

Haitong Ma, Tianyi Chen, Kai Wang, Na Li, and Bo Dai. Efficient online reinforcement learning for diffusion policy. *arXiv preprint arXiv:2502.00361*, 2025.

Vineet Jain, Tara Akhound-Sadegh, and Siamak Ravanbakhsh. Sampling from energy-based policies using diffusion. *arXiv preprint arXiv:2410.01312*, 2024.

Tom Minka et al. Divergence measures and message passing. 2005.

Mingzhang Yin and Mingyuan Zhou. Semi-implicit generative model, 2019. URL https://arxiv.org/abs/1905.12659.

Art B. Owen. *Monte Carlo theory, methods and examples*. https://artowen.su.domains/mc/, 2013.

Ghassen Jerfel, Serena Wang, Clara Wong-Fannjiang, Katherine A Heller, Yian Ma, and Michael I Jordan. Variational refinement for importance sampling using the forward kullback-leibler divergence. In *Uncertainty in Artificial Intelligence*, pages 1819–1829. PMLR, 2021.

Chi Jin, Yuchen Zhang, Sivaraman Balakrishnan, Martin J Wainwright, and Michael Jordan. Local maxima in the likelihood of gaussian mixture models: Structural results and algorithmic consequences. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/3875115bacc48cca24ac51ee4b0e7975-Paper.pdf.

Yudong Chen, Dogyoon Song, Xumei Xi, and Yuqian Zhang. Local minima structures in gaussian mixture models. *IEEE Transactions on Information Theory*, 70(6):4218–4257, 2024.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `https://jmlr.org/papers/v22/20-1364.html`.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning, 2016. URL `https://arxiv.org/abs/1605.08695`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL `https://arxiv.org/abs/1912.01703`.

Jiajun He, Wenlin Chen, Mingtian Zhang, David Barber, and José Miguel Hernández-Lobato. Training neural samplers with reverse diffusive kl divergence. *arXiv preprint arXiv:2410.12456*, 2024.

A Theoretical Analysis

A.1 Proof of Proposition 1

Proposition 1 (Finite-mixture upper bound on forward KL)**.** *For a fixed state s and parameters θ, ϕ, let $z_1, \ldots, z_N \overset{\text{i.i.d.}}{\sim} p_z$ and define $U_N(s) := \mathbb{E}_{z_{1:N}}[D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \hat{\pi}_{\theta,N}(\cdot \mid s; z_{1:N}))]$. Under the assumptions stated in Appendix A.1, the following holds for any integer $N \geq 1$:*

$$D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)) \leq U_{N+1}(s) \leq U_N(s). \tag{8}$$

Moreover, the bound is asymptotically tight: $\lim_{N\to\infty} U_N(s) = D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s))$.

Assume that $D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s))$ and $U_N(s)$ are finite for all $N \geq 1$. For the asymptotic-tightness claim, additionally assume that $\{\log \hat{\pi}_{\theta,N}(a \mid s)\}_{N\geq 1}$ is uniformly integrable under the joint law of $a \sim \nu_\phi(\cdot \mid s)$ and an independent i.i.d. latent sequence $z_1, z_2, \ldots \sim p_z$.

Proof. This is a direct application of Lemma 1 of Yin and Zhou (2019) to the target $\nu_\phi(\cdot \mid s)$ and the mixture $\pi_\theta(\cdot \mid s)$. The lower bound follows from convexity of $D_{\mathrm{KL}}(\nu_\phi \| \cdot)$ and $\mathbb{E}_{z_{1:N}}[\hat{\pi}_{\theta,N}] = \pi_\theta$. For monotonicity, each mixture with $N+1$ components is the average of its $N+1$ leave-one-out mixtures, each containing N components. Applying the same convexity inequality and taking expectations gives $U_{N+1}(s) \leq U_N(s)$, following the argument of Yin and Zhou. The strong law of large numbers gives $\hat{\pi}_{\theta,N}(a \mid s) \to \pi_\theta(a \mid s)$ almost surely for each a. Positivity of the component densities and uniform integrability then imply convergence of the expected log densities, establishing the limit. □

A.2 Effect of the Number of Latent Samples

The preceding proposition shows that increasing N tightens the finite-mixture upper bound. We further characterize its gap to understand how N affects component specialization.

Theorem 1 (Finite-mixture regularization)**.** *Fix s, θ, ϕ, and let $U_N(s)$ be defined as in Proposition 1. Assume that $D_{\mathrm{KL}}(\nu_\phi(\cdot \mid s) \| \pi_\theta(\cdot \mid s)) < \infty$ and that there exist constants $0 < c \leq C < \infty$ such that*

$$c \leq \frac{k_\theta(a \mid s, z)}{\pi_\theta(a \mid s)} \leq C$$

for $\nu_\phi(\cdot \mid s)$-almost every a and p_z-almost every z. For the fixed state and target, define

$$\mathcal{R}_N(\theta) := \frac{1}{2N} \mathbb{E}_{a\sim\nu_\phi(\cdot|s)} \left[\mathrm{Var}_{z\sim p_z} \left[\frac{k_\theta(a \mid s, z)}{\pi_\theta(a \mid s)} \right] \right].$$

Then, as $N \to \infty$,

$$U_N(s) = D_{\mathrm{KL}}\left(\nu_\phi(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)\right) + \mathcal{R}_N(\theta) + O(N^{-2}).$$

Thus, the expected finite-mixture objective introduces $\mathcal{R}_N(\theta) \geq 0$ as a leading-order regularizer on the relative variability of the latent-conditioned densities, with regularization strength proportional to N^{-1}.

Proof. We suppress the common conditioning on s. Then,

$$\begin{aligned}
\mathbb{E}_{z_{1:N}}\left[\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right] &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{z_i\sim p_z}\left[\frac{k_\theta(a\mid z_i)}{\pi_\theta(a)}\right]-1\\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{\pi_\theta(a)}{\pi_\theta(a)}-1\\
&= 0,\\
\mathbb{E}_{z_{1:N}}\left[\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^2\right] &= \frac{1}{N^2}\mathbb{E}_{z_{1:N}}\left[\left(\sum_{i=1}^{N}\left(\frac{k_\theta(a\mid z_i)}{\pi_\theta(a)}-1\right)\right)^2\right]\\
&= \frac{1}{N^2}\sum_{i=1}^{N}\mathbb{E}_{z_i\sim p_z}\left[\left(\frac{k_\theta(a\mid z_i)}{\pi_\theta(a)}-1\right)^2\right] \quad (\because \text{ indep.})\\
&= \frac{1}{N}\operatorname{Var}_{z\sim p_z}\left[\frac{k_\theta(a\mid z)}{\pi_\theta(a)}\right],\\
\mathbb{E}_{z_{1:N}}\left[\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^3\right] &= \frac{1}{N^2}\mathbb{E}_{z\sim p_z}\left[\left(\frac{k_\theta(a\mid z)}{\pi_\theta(a)}-1\right)^3\right]\\
&= O(N^{-2}),\\
\mathbb{E}_{z_{1:N}}\left[\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^4\right] &= O(N^{-2}).
\end{aligned}$$

Applying a third-order Taylor expansion of $\log x$ about $x=1$, with $x=\hat{\pi}_{\theta,N}(a)/\pi_\theta(a)$, we obtain

$$\begin{aligned}
\log\hat{\pi}_{\theta,N}(a) &= \log\pi_\theta(a)+\log\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}\\
&= \log\pi_\theta(a)+\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)-\frac{1}{2}\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^2\\
&\quad+\frac{1}{3}\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^3+O\left(\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^4\right).
\end{aligned}$$

From the boundedness assumption, the moment constants are uniformly bounded in a:

$$\mathbb{E}_{z\sim p_z}\left[\left|\frac{k_\theta(a\mid z)}{\pi_\theta(a)}-1\right|^r\right]\le(C+1)^r<\infty,\qquad r\in\{2,3,4\}.$$

Moreover, $(\log)^{(4)}(x)=-6/x^4$ implies that the fourth-order Taylor remainder coefficient satisfies

$$\sup_{x\in[c,C]}\frac{|(\log)^{(4)}(x)|}{4!}=\frac{1}{4c^4}<\infty.$$

Taking the latent expectation and using the moment identities above,

$$\begin{aligned}
\mathbb{E}_{z_{1:N}}\left[\log\hat{\pi}_{\theta,N}(a)\right] &= \log\pi_\theta(a)+\cancelto{0}{\mathbb{E}_{z_{1:N}}\left[\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right]}-\frac{1}{2}\mathbb{E}_{z_{1:N}}\left[\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^2\right]\\
&\quad+\frac{1}{3}\mathbb{E}_{z_{1:N}}\left[\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^3\right]+O\left(\mathbb{E}_{z_{1:N}}\left[\left(\frac{\hat{\pi}_{\theta,N}(a)}{\pi_\theta(a)}-1\right)^4\right]\right)\\
&= \log\pi_\theta(a)-\frac{1}{2N}\operatorname{Var}_{z\sim p_z}\left[\frac{k_\theta(a\mid z)}{\pi_\theta(a)}\right]+O(N^{-2}).
\end{aligned}$$

Since $\mathcal{R}_N(\theta)\ge 0$ by definition, the claim follows by taking the target expectation:

$$\begin{aligned}
U_N(s)-D_{\mathrm{KL}}(\nu_\phi\|\pi_\theta) &= \mathbb{E}_{a\sim\nu_\phi}\left[\log\pi_\theta(a)-\mathbb{E}_{z_{1:N}}\left[\log\hat{\pi}_{\theta,N}(a)\right]\right]\\
&= \mathcal{R}_N(\theta)+O(N^{-2}).
\end{aligned}$$

□

Interpretation. The regularizer $\mathcal{R}_N(\theta)$ penalizes variability across latent-conditioned densities at each target action. For small N, actions explained by only a few latents can be poorly covered by the sampled mixture. This creates pressure toward broader, overlapping Gaussian components, potentially increasing their variances at the expense of component specialization. As N increases, this leading-order penalty weakens as $1/N$, allowing components to specialize to different target regions with less pressure to cover multiple modes individually. Consistent with this interpretation, Figure 11 shows that increasing N improves mode coverage on GMM40. Thus, larger N not only tightens the upper bound but also reduces the implicit regularization against component specialization.

B ALGORITHM

We provide the training procedure for iBOLT in Algorithm 1. Here, sg denotes the stop-gradient operator, d is the terminal indicator excluding time-limit truncations, and $Q_{\mathrm{avg}} = \frac{1}{2}(Q_{\phi_1} + Q_{\phi_2})$ denotes the mean of the two critics.

Algorithm 1 iBOLT: Efficient Implicit Boltzmann Policy Optimization.

Require: Mixture size N, reference sample size M, batch size B, temperature τ
Require: Discount γ, target update rate ρ
Require: Initial exploration coefficient α_0, update interval K_α
Initialize actor θ, critics ϕ_1, ϕ_2, and target critics $\bar{\phi}_\ell \leftarrow \phi_\ell$
Set $\alpha \leftarrow \alpha_0$ and $\mathrm{step} \leftarrow 0$; initialize $\mathcal{D}$ with random transitions (s, a, r, s')
for each iteration **do**
  for each sampling step **do**
    Sample $z \sim p_z$ and $(\mu, \sigma) \leftarrow G_\theta(s, z)$
    Sample $a \sim \mathcal{N}_{\mathrm{Tr}}\big(\tanh(\mu), \mathrm{diag}(\sigma^2); [-1, 1]^{d_a}\big)$ using Eq. (2)
    Sample $\xi \sim \mathcal{N}(0, I)$ and $a^{\mathrm{env}} \leftarrow \mathrm{clip}(a + 0.1\alpha \cdot \xi, -1, 1)$ using Eq. (31)
    Execute a^{env}, and store $(s, a^{\mathrm{env}}, r, s')$ in $\mathcal{D}$
  end for
  for each update step **do**
    Sample a minibatch $\mathcal{B} = \{(s_h, a_h, r_h, s'_h)\}_{h=1}^{B}$ from $\mathcal{D}$
    Sample $a'_h \sim \pi_\theta(\cdot \mid s'_h)$ for each h
    $y_h \leftarrow \mathrm{sg}\big[r_h + \gamma(1-d)\min_{\ell\in\{1,2\}} Q_{\bar{\phi}_\ell}(s'_h, a'_h)\big]$
    Update critics ϕ_1, ϕ_2 by minimizing
    $\widehat{\mathcal{L}}_{\mathrm{critic}}(\phi_1, \phi_2) = \frac{1}{B}\sum_{h=1}^{B}\sum_{\ell=1}^{2}\big(Q_{\phi_\ell}(s_h, a_h) - y_h\big)^2$
    $\bar{\phi}_\ell \leftarrow (1-\rho)\bar{\phi}_\ell + \rho\phi_\ell$, for $\ell \in \{1, 2\}$
    for $h = 1, \ldots, B$ **do**
      Sample $z_{h,1}, \ldots, z_{h,N} \overset{\mathrm{iid}}{\sim} p_z$
      Construct $\hat{\pi}^{(h)}_{\theta,N}(a) = \frac{1}{N}\sum_{i=1}^{N} k_\theta(a \mid s_h, z_{h,i})$ using Eq. (7)
      Sample $b_{h,1:M} \overset{\mathrm{iid}}{\sim} \hat{\pi}^{(h)}_{\theta,N}$; $b_{h,1:M} \leftarrow \mathrm{sg}(b_{h,1:M})$
      $w_{h,1:M} \leftarrow \mathrm{sg}\left[\mathrm{softmax}\left(\left(\frac{Q_{\mathrm{avg}}(s_h, b_{h,j})}{\tau} - \log\hat{\pi}^{(h)}_{\theta,N}(b_{h,j})\right)_{j=1}^{M}\right)\right]$ using Eq. (11)
    end for
    Update policy parameters θ by minimizing
    $\widehat{\mathcal{L}}_{\mathrm{actor}} = -\frac{1}{B}\sum_{h=1}^{B}\sum_{j=1}^{M} w_{h,j}\log\hat{\pi}^{(h)}_{\theta,N}(b_{h,j})$ using Eq. (14)
    $\mathrm{step} \leftarrow \mathrm{step} + 1$
    if $\mathrm{step} \bmod K_\alpha = 0$ **then**
      Update α using the entropy regulator in Appendix C.4
    end if
  end for
end for

C Experimental Details

C.1 Common Settings

All experiments are conducted on a single NVIDIA GeForce RTX 5090 GPU (32 GB) with an Intel Xeon w5-3423 CPU. Our implementations of SAC and TD3 follow SBX, the JAX (Bradbury et al., 2018) implementation of Stable-Baselines3 (Raffin et al., 2021). For SQL, MEow, DIPO, and MFPO, we use their official code (`https://github.com/haarnoja/softqlearning`, `https://github.com/ChienFeng-hub/meow`, `https://github.com/BellmanTimeHut/DIPO`, and `https://github.com/dongxiaoyi-xyz/MFPO`, respectively). The implementations use different computational frameworks: iBOLT, SAC, TD3, and MFPO use JAX, SQL uses TensorFlow (Abadi et al., 2016), and MEow and DIPO use PyTorch (Paszke et al., 2019). We retain these frameworks rather than reimplementing all methods in a single framework, so absolute training and inference times partly reflect framework-level differences.

C.2 Forward versus Reverse KL

Targets and policy. We compare forward-KL (FKL) and reverse-KL (RKL) training on five fixed one-dimensional Boltzmann targets over $\mathcal{A} = [-10, 10]$. The state is fixed, and we omit state conditioning throughout. We use an oracle value function $Q(a) = \tau \log \nu(a)$ up to an additive constant, with $\tau = 0.25$. No critic is learned, and neither target samples nor mode labels are provided during training. These illustrative targets were selected in a preliminary screening study.

For the Gaussian targets, we normalize $\sum_k w_k \mathcal{N}(a; c_k, 0.5^2)$ over $\mathcal{A}$. For the non-Gaussian targets, let $S(x) = (1 + e^{-x})^{-1}$ and define

$$\begin{aligned} U(a; c, h) &= \frac{1}{h} S\left(\frac{a-c}{h}\right) S\left(-\frac{a-c}{h}\right), \\ V(a; l, r, e_l, e_r, t) &= S\left(\frac{a-l}{e_l}\right) S\left(\frac{r-a}{e_r}\right) \exp\left[t\left(a - \frac{l+r}{2}\right)\right]. \end{aligned} \tag{16}$$

Here, U is a logistic spike, and V is a smooth plateau when $t = 0$ and a rising ramp when $t > 0$. Each non-Gaussian shape is normalized individually before mixing:

$$\nu(a) = \sum_k w_k \frac{f_k(a)}{\int_{-10}^{10} f_k(x)\, dx}. \tag{17}$$

The five targets, in the column order of Figure 7, are:

1. **Three Gaussian modes:** centers $(-4.25, 0, 4.25)$ and weights $(1/3, 1/3, 1/3)$.
2. **Spike + ramp:** shapes $U(a; -4.25, 0.1)$ and $V(a; 0.5, 8, 0.6, 0.08, 0.2)$, with weights $(0.2, 0.8)$.
3. **Spike + plateau + ramp:** shapes $U(a; -5, 0.13)$, $V(a; -1, 1, 0.12, 0.12, 0)$, and $V(a; 3, 7, 0.25, 0.05, 0.4)$, with weights $(0.25, 0.30, 0.45)$.
4. **Two offset Gaussian modes:** centers $(0, 4.25)$ and weights $(0.5, 0.5)$.
5. **Unequal Gaussian masses:** centers $(-4.25, 0, 4.25)$ and weights $(0.2, 0.5, 0.3)$.

Both methods use the same semi-implicit policy,

$$\pi_\theta(a) = \mathbb{E}_{z \sim \mathcal{N}(0,1)}[k_\theta(a \mid z)], \tag{18}$$

where k_θ is a conditional truncated Gaussian. An MLP with two hidden layers of 256 units each and GELU activations produces a raw location $\mu_\theta(z)$ and a log-scale output $h_{\sigma,\theta}(z)$. For this action domain, we set

$$\begin{aligned} \bar{\mu}_\theta(z) &= 10 \tanh(\mu_\theta(z)), \\ \log \sigma_\theta(z) &= \mathrm{clip}(h_{\sigma,\theta}(z), -5, -1), \\ k_\theta(a \mid z) &= \mathcal{N}_{\mathrm{Tr}}\big(a; \bar{\mu}_\theta(z), \sigma_\theta^2(z); \mathcal{A}\big). \end{aligned} \tag{19}$$

The location head uses a variance-scaling initializer with scale 3, `fan_avg` mode, and a uniform distribution. The log-standard-deviation head is initialized to the constant -1.

Training setup. FKL training uses the SNIS-weighted finite-mixture likelihood described in Section 4.3, with the oracle Q in place of the learned critic. Both methods average $B = 32$ independent groups per update, each containing $N = 128$ latent samples and $M = 128$ action samples. Within each group, actions are sampled by selecting uniformly among the N conditional Gaussian components. We use Adam with learning rate 3×10^{-4} for 100,000 updates and 5 random seeds, with paired policy initialization across methods.

Reverse-KL objective and gradient. RKL minimizes

$$\mathcal{J}_{\mathrm{R}}(\theta) = D_{\mathrm{KL}}(\pi_\theta \| \nu) = \mathbb{E}_{a\sim\pi_\theta}\left[\log \pi_\theta(a) - \frac{Q(a)}{\tau}\right] + C, \tag{20}$$

where C does not depend on θ. For a reparameterized action $a_\theta = T_\theta(z, \epsilon)$, where ϵ supplies the randomness for sampling the conditional truncated Gaussian, the gradient is

$$\begin{aligned} \nabla_\theta \mathcal{J}_{\mathrm{R}} &= \mathbb{E}_{z,\epsilon}\left[\left(s_\theta(a_\theta) - \frac{1}{\tau}\nabla_a Q(a_\theta)\right)\nabla_\theta a_\theta\right], \\ s_\theta(a) &= \nabla_a \log \pi_\theta(a). \end{aligned} \tag{21}$$

This expression uses the score identity

$$\mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a)|_{a \text{ fixed}}\right] = 0. \tag{22}$$

Monte Carlo score estimation. Since $\pi_\theta(a)$ is generally intractable, we draw L i.i.d. latent samples $\tilde{z}_\ell \sim \mathcal{N}(0, 1)$, independently of the samples used to generate policy actions, and estimate

$$\hat{\pi}_{\theta,L}(a) = \frac{1}{L}\sum_{\ell=1}^{L} k_\theta(a \mid \tilde{z}_\ell), \tag{23}$$

$$\hat{s}_{\theta,L}(a) = \nabla_a \log \hat{\pi}_{\theta,L}(a) = \frac{\sum_{\ell=1}^{L} k_\theta(a \mid \tilde{z}_\ell)\nabla_a \log k_\theta(a \mid \tilde{z}_\ell)}{\sum_{\ell=1}^{L} k_\theta(a \mid \tilde{z}_\ell)}. \tag{24}$$

For our truncated Gaussian kernels, at interior actions,

$$\nabla_a \log k_\theta(a \mid \tilde{z}_\ell) = \frac{\bar{\mu}_\theta(\tilde{z}_\ell) - a}{\sigma_\theta^2(\tilde{z}_\ell)}. \tag{25}$$

We implement the per-group RKL gradient estimate by differentiating the surrogate

$$\mathcal{L}_{\mathrm{R}}^{\mathrm{sur}} = \frac{1}{M}\sum_{j=1}^{M} a_{\theta,j}\ \mathrm{sg}\left[\hat{s}_{\theta,L}(a_{\theta,j}) - \frac{1}{\tau}\nabla_a Q(a_{\theta,j})\right]. \tag{26}$$

Here, sg denotes stop-gradient, so gradients flow only through the reparameterized actions. The surrogate's numerical value is not the RKL objective. Group gradients are averaged before the optimizer update.

We use $L = 2^{20}$ MC samples per group, shared across its M actions and resampled independently for each group and update. Thus, L controls the marginal-score approximation, while N and M specify the sampled mixture size and the number of policy actions, respectively. Finite L yields an approximate gradient.

Density evaluation and comparison. Learned densities are visualized using 512-bin histograms of 2^{20} policy samples per seed, averaged across the 5 seeds without smoothing. Figure 7 extends the comparison in Figure 2 to all five targets. In these illustrative settings, FKL training recovers the separated modes and broad regions of the target densities, whereas RKL training concentrates on a subset of these regions, leaving at least one target mode poorly covered.

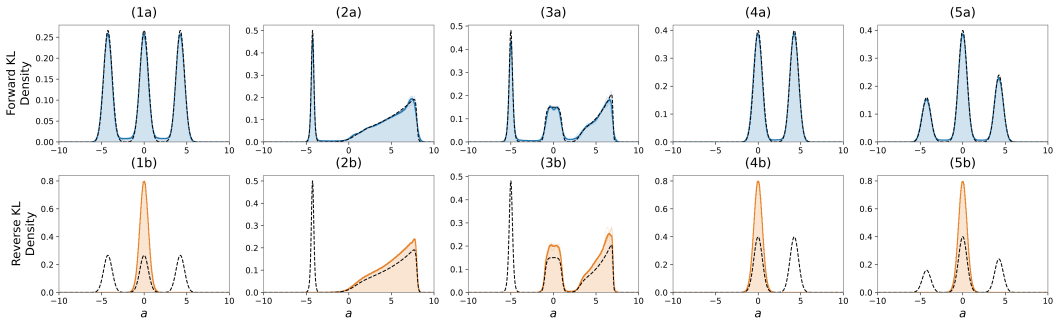

Figure 7: **Forward- and reverse-KL fitting across five target distributions.** Columns correspond to the three-Gaussian, spike-and-ramp, spike–plateau–ramp, two-offset-Gaussian, and unequal-mass Gaussian targets, respectively. The top and bottom rows show FKL and RKL policies after 100,000 updates. Colored curves show the mean sampled densities over 5 seeds; black dashed curves show the target densities. Both objectives use the same policy architecture and paired initializations.

Score-estimation convergence. To assess the score approximation, we freeze each final RKL policy and evaluate fixed actions while varying $L = 2^7, 2^8, \ldots, 2^{24}$. We perform 16 independent MC repetitions; within each repetition, increasing L uses nested prefixes of the same latent sequence. The mean of four additional independent estimates, each using 2^{24} latent samples, provides an empirical reference. Quantitative score errors are evaluated on 128 fixed policy actions for every seed.

For brevity, Figure 8 shows the seed-0 score-convergence diagnostics for the two targets presented in the main text; the remaining targets exhibit qualitatively similar convergence behavior. At the displayed actions, the estimates are close to the empirical reference and show little MC variability at the training value $L = 2^{20}$. The reference is not an exact score, and this final-checkpoint diagnostic does not establish accuracy throughout training.

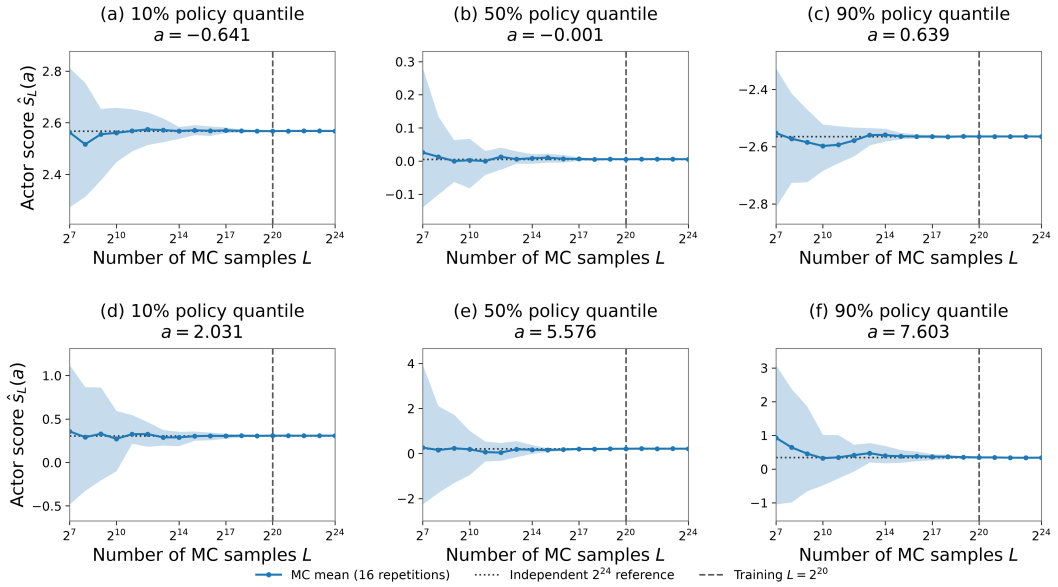

Figure 8: **Monte Carlo estimation of the reverse-KL policy's marginal action score.** The top and bottom rows correspond to the three-Gaussian and spike-and-ramp targets, respectively. Columns show actions at the 10th, 50th, and 90th percentiles of the corresponding seed-0 policy. Blue curves and shading show the MC mean and the 10th–90th percentile range across repetitions. Dotted horizontal lines mark the empirical reference scores, and dashed vertical lines mark the training value $L = 2^{20}$.

Table 3: Settings for the GMM40 comparison. All iBOLT scale parameters are specified in normalized action coordinates $a = x/40$.

Setting	Value
Common settings	
Target dimension	2
Number of target components	40
Physical action domain	$[-40, 40]^2$
Temperature	1
Actor optimizer updates	100,000
Batch size	256
Learning rate	3×10^{-4}
Evaluation samples per run	10,000
Samples per distribution for MMD	2,048
iBOLT settings	
Actor hidden layers	$(256, 256, 256)$
Activation	GELU
Latent prior	$\mathcal{N}(0, I_2)$
Latent sampling	Fresh samples at every update
Mixture components per group, N	256
Reference actions per group, M	256
Conditional distribution	Box-truncated Gaussian
Actor log-standard-deviation bounds	$[-5, -3.5]$
Initial log-standard deviation	-4
Proposal standard-deviation floor	0.05
Mean-head initialization variance scale	16
Log-scale head initial weights / bias	0 / -4
Density-correction exponent	1
Gradient clipping	None

C.3 GMM40

Target distribution. We use the two-dimensional GMM40 target from the DiKL implementation (He et al., 2024), consisting of 40 equally weighted Gaussian components:

$$p_{\mathrm{GMM40}}(x) = \frac{1}{40} \sum_{k=1}^{40} \mathcal{N}(x; \mu_k, \sigma_{\mathrm{GT}}^2 I_2), \qquad \sigma_{\mathrm{GT}} \approx 1.31326. \tag{27}$$

The component centers are generated once using target seed 0 and shared across all methods and training seeds. We restrict the action domain to $\mathcal{X} = [-40, 40]^2$ and set $Q(x) = \log p_{\mathrm{GMM40}}(x)$ with temperature $\tau = 1$. The resulting Boltzmann target is

$$\nu(x) = \frac{p_{\mathrm{GMM40}}(x)\mathbf{1}\{x \in \mathcal{X}\}}{\int_{\mathcal{X}} p_{\mathrm{GMM40}}(u)\, du}. \tag{28}$$

Thus, the reference distribution is obtained by truncating and renormalizing the entire mixture. Policies operate in normalized coordinates $a \in [-1, 1]^2$, with $x = 40a$ used for oracle queries and evaluation. No critic is learned, and ground-truth samples are used only for evaluation.

Training settings. We compare iBOLT, SAC, SQL, MFPO, and DIPO using 100,000 actor optimizer updates, batch size 256, and 5 random seeds. All methods use Adam-based optimization with learning rate 3×10^{-4}, while retaining their method-specific architectures and sampling procedures. For iBOLT, each batch contains 256 independent latent/proposal groups at the same dummy state. The common and iBOLT-specific settings are summarized in Table 3. The shared actor-update budget does not imply equal oracle-query counts or computational cost.

Mode coverage. Coverage is measured with respect to the 40 reference components, rather than the number of density local maxima. Each sample is assigned to the nearest component under squared Mahalanobis distance:

$$k^*(x) = \arg\min_k d_k^2(x), \qquad d_k^2(x) = \frac{\|x - \mu_k\|^2}{\sigma_{\mathrm{GT}}^2}. \tag{29}$$

A sample contributes to its assigned component only when $d^2_{k^*(x)}(x) \leq 9$. Let c_k and c_k^{GT} denote the resulting counts from the 10,000 policy and reference samples. We report

$$\text{Coverage} = \sum_{k=1}^{40} \mathbf{1}\{c_k \geq \max(10, 0.1c_k^{\mathrm{GT}})\}. \tag{30}$$

This minimum-occupancy criterion prevents isolated samples from being counted as component recovery, and each sample contributes to at most one component.

C.4 Implementation Details of iBOLT for Online RL

The algorithm-specific hyperparameters of iBOLT are listed in Table 4. For exploration during data collection, iBOLT adopts the entropy regulator of DACER (Wang et al., 2024) without modification or additional tuning; the baselines use their own default exploration schemes. As in DACER, an exploration coefficient α is adapted toward a target entropy using an entropy estimate obtained by fitting a Gaussian mixture model to actions sampled from the current policy, and the executed action is

$$a_{\mathrm{env}} = \mathrm{clip}(a + 0.1\,\alpha \cdot \xi,\, -1,\, 1), \qquad \xi \sim \mathcal{N}(0, I), \tag{31}$$

where a is sampled from the current policy. The executed action a_{env} is stored in the replay buffer. The coefficient α affects only this exploration noise; it does not enter the temporal-difference targets, the reference construction, or the actor objective, and no noise is added during evaluation. Although the entropy of iBOLT's finite-mixture approximation can be estimated by Monte Carlo, we retain DACER's GMM-based estimator for the exploration regulator.

Table 4: Algorithm-specific hyperparameters of iBOLT.

Hyperparameter	Value
Number of latent samples N	64
Number of reference actions M	64
Latent dimension d_z	$\dim(\mathcal{A})$
Lower bound of $\log \sigma_\theta$	-5
Upper bound of $\log \sigma_\theta$	-1
Initial $\log \sigma_\theta$	-1
DACER-based exploration	
Initial exploration coefficient α	0.27
Target entropy $\mathcal{H}_{\mathrm{target}}$	$-0.9\dim(\mathcal{A})$
Optimizer for $\log \alpha$	Adam
Learning rate for $\log \alpha$	3×10^{-2}
Exploration noise standard deviation	0.1α
Coefficient update interval	10,000 learner updates
States per entropy estimation	256
Action samples per state	200
Number of fitted GMM components	3
GMM covariance type	Full

C.5 Multimodal Navigation

Environments. We evaluate behavioral diversity in multi-goal PointMaze tasks and wall-free navigation environments. For PointMaze, we use the Medium and Hard layouts from the DrAC implementation, containing four and eight goals, respectively. The agent starts from a fixed central state and receives a reward of 100 upon reaching any goal and zero otherwise. Episodes terminate upon success or after 300 steps in Medium and 600 steps in Hard. All methods share the same maze geometry, goal locations, initial state, and reward function.

The wall-free tasks contain 4, 8, or 16 goals arranged around the origin, where each episode begins. The agent observes its two-dimensional position and selects a displacement action in $[-1, 1]^2$. The reward combines a nearest-goal distance cost, an action cost, and a success bonus:

$$r_t = -\min_{g \in \mathcal{G}} \|s_{t+1} - g\|_2^2 - 30\|a_t\|_2^2 + 10\,\mathbf{1}\{\text{success}\}.$$

The four-goal task places goals at $(\pm 5, 0)$ and $(0, \pm 5)$; the eight- and sixteen-goal tasks place goals uniformly on a circle of radius six. No intrinsic reward is used in either environment family.

Training settings. Policies are trained online for approximately 10^6 environment transitions using a batch size of 4,096 and discount factor $\gamma = 0.99$. Unlike GMM40, the value function is learned from environment interaction rather than supplied by an oracle. The PointMaze comparison includes iBOLT, SAC, SQL, MEow, MFPO, and TD3, with method-specific architectures and optimization settings. For iBOLT, the actor and twin critics use two hidden layers of 256 units, with $N = M = 64$. The reported PointMaze and wall-free results use $\tau = 5$ and $\tau = 10$, respectively. The DACER exploration controller is disabled. These experiments provide qualitative evidence of behavioral diversity.

C.6 MuJoCo

Training setup. Each method is trained for 10^6 environment steps over 5 random seeds. To reduce confounding effects from network capacity, we use actor and critic MLPs with two hidden layers of 256 units for iBOLT, SAC, TD3, SQL, DIPO, and MFPO, while retaining each method's default activation function; MEow retains its structured flow-based architecture. These settings match backbone capacity where applicable rather than total parameter counts. All critic-based methods employ twin critics with clipped double-Q targets (Fujimoto et al., 2018).

All methods use a batch size of 256 and an update-to-data (UTD) ratio of one, that is, one learner update per environment transition after warm-up, while method-specific actor update delays and internal optimization steps are retained. Shared hyperparameters are listed in Table 5.

Table 5: Hyperparameters used in the MuJoCo experiments.

Hyperparameter	**iBOLT (ours)**	SAC	TD3	MEow	SQL	DIPO	MFPO
Batch size	256	256	256	256	256	256	256
Discount factor γ	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Target smoothing coefficient	0.005	0.005	0.005	0.0005	Hard update[a]	0.005	0.005
Actor hidden layers	2	2	2	2[b]	2	2	2
Actor hidden units per layer	256	256	256	64/256[b]	256	256	256
Critic hidden layers	2	2	2	—[b]	2	2	2
Critic hidden units per layer	256	256	256	—[b]	256	256	256
Actor learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	10^{-3} [b]	3×10^{-4}	3×10^{-4}	3×10^{-4}
Critic learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	10^{-3} [b]	3×10^{-4}	3×10^{-4}	3×10^{-4}
Activation function	GELU	ReLU	ReLU	Leaky ReLU	ReLU	Mish	GELU
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam
Replay buffer capacity	10^6	10^6	10^6	10^6	10^6	10^6	10^6
Buffer warm-up size	5000	5000	5000	5000	5000	5000	5000
Update-to-data ratio	1	1	1	1	1	1	1
Action-generation steps	1	1	1	1	1	100	2

[a] SQL performs a hard target update every 1,000 updates. [b] MEow uses a structured flow/value architecture: two-layer, 64-unit flow modules and two-layer, 256-unit scale modules, trained using a joint optimizer.

The per-task temperatures of iBOLT are listed in Table 6. For each task, the temperature τ of the Boltzmann target (Section 4.3) is selected from $\{0.05, 0.1, 0.25, 0.5, 1.0, 5.0, 10.0\}$. Only the temperature varies across tasks, and all other hyperparameters are held fixed.

Evaluation protocol. We evaluate iBOLT every 5,000 steps over 10 episodes, using the deterministic action $a = \tanh(\mu_\theta(s, 0))$. Neither conditional Gaussian noise nor additional exploration noise is used during evaluation. Learning curves report the mean episode return, with shaded regions indicating one standard deviation across 5 runs. Final performance is computed by averaging evaluation

Table 6: Per-task temperatures of iBOLT on MuJoCo.

Task	Temperature τ
Hopper-v4	0.05
Walker2d-v4	0.1
HalfCheetah-v4	0.25
Ant-v4	0.25
Humanoid-v4	0.1

returns over the last 100,000 training steps and then across runs. For cross-task aggregation, we normalize each method's final performance by the highest score achieved by any compared method on the same task, assigning the best method a score of 100, and average the normalized scores across the 5 tasks.

D Ablation Studies

In this section, we provide the settings and full results of the ablation studies summarized in Section 5.3. Each study varies a single factor while keeping all other hyperparameters at the values in Appendix C.4, isolating its effect on iBOLT's performance.

D.1 Sensitivity to the Value-Weighting Temperature

iBOLT introduces a single task-specific hyperparameter: the value-weighting temperature τ of the Boltzmann target (Section 4.3). A small τ concentrates the importance weights on the highest-valued reference actions, approaching greedy improvement but reducing the effective number of samples, whereas a large τ flattens the target toward the uniform distribution over the action space, weakening the value signal. To assess this trade-off, we sweep $\tau \in \{0.05, 0.1, 0.25, 0.5, 1.0, 5.0\}$ on each MuJoCo task while keeping all other hyperparameters fixed, and report the learning curves in Figure 9.

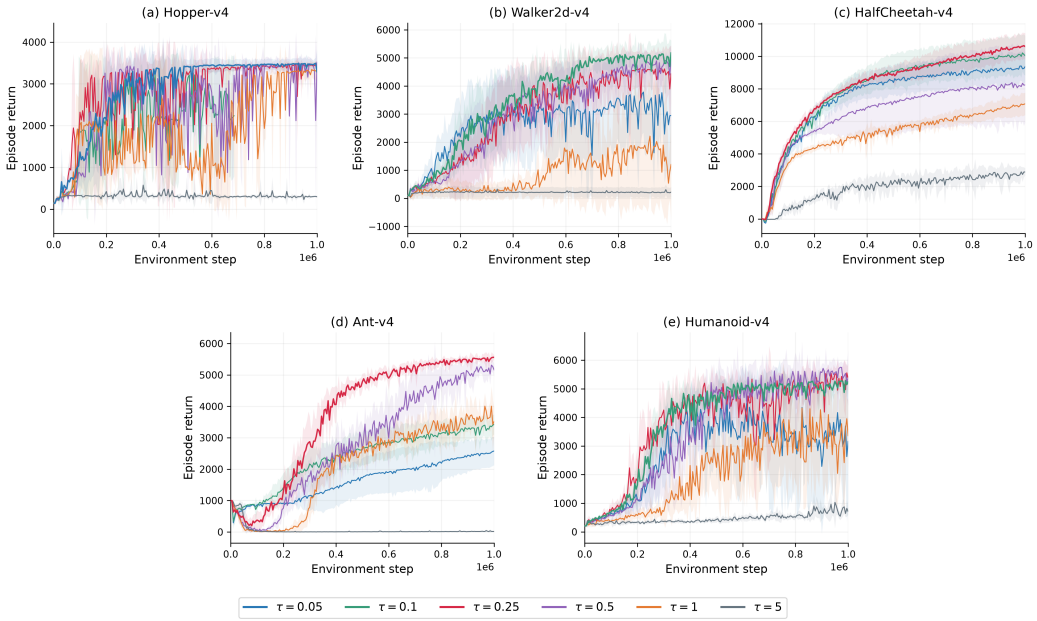

Figure 9: Sensitivity of iBOLT to the value-weighting temperature τ on 5 MuJoCo benchmarks. Curves show the mean and shaded regions one standard deviation over 5 seeds. All other hyperparameters are fixed.

Large temperatures consistently degrade performance: with $\tau = 5$, iBOLT fails to improve on every task, as the density-corrected target approaches the uniform distribution, and $\tau = 1$ lags substantially on all tasks except Hopper. Intermediate temperatures between 0.1 and 0.5 perform well across tasks, with $\tau = 0.25$ among the best on every task. The smallest temperature, $\tau = 0.05$, yields the most stable learning on Hopper but underperforms on Walker2d, Ant, and Humanoid, consistent with weights collapsing onto few reference actions. A temperature in the range $[0.1, 0.25]$ therefore provides a robust default.

D.2 Effect of Exploration Noise

During data collection, iBOLT perturbs executed actions with Gaussian noise whose scale is adapted as in DACER (Appendix C.4). To assess how much of iBOLT's performance depends on this noise rather than on the policy class and objective, we remove it so that actions are sampled directly from the policy, keeping all other hyperparameters fixed. Figure 10 shows that iBOLT learns effectively without the noise. On Ant and Humanoid, the two variants are nearly indistinguishable throughout

training. On Hopper, the final returns are comparable, although learning without noise remains less stable. The noise yields higher final returns on Walker2d and particularly on HalfCheetah. These results indicate that the policy class and objective account for most of iBOLT's performance, while the exploration noise mainly improves stability and final return on a subset of tasks.

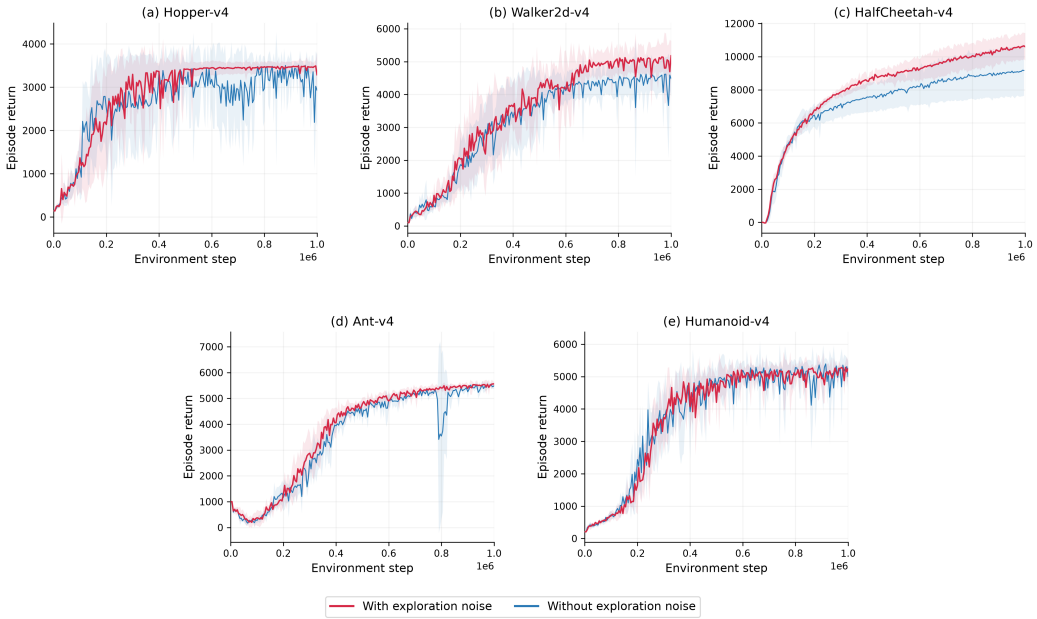

Figure 10: Effect of the exploration noise applied during data collection on 5 MuJoCo benchmarks. Curves show the mean and shaded regions one standard deviation over 5 random seeds. All other hyperparameters are fixed.

D.3 Effect of the Number of Samples

The numbers of latent samples N and reference actions M determine both the resolution of the finite mixture and the accuracy of the self-normalized weights (Section 4.3). We vary $N = M \in \{1, 4, 16, 64, 128, 256, 512\}$ on GMM40 while keeping all other settings fixed. Figure 11 and Table 7 show that mode recovery improves steadily with $N = M$. With $N = M = 1$, the objective reduces to fitting a single Gaussian to its own sample, and the policy collapses onto one mode. Small values such as 4 and 16 recover only a subset of modes, and the policy additionally places mass along paths connecting them, reflecting too few components to represent all modes separately. From $N = M = 128$ onward, iBOLT recovers all 40 modes in every seed, and MMD continues to decrease as $N = M$ grows, consistent with both approximations tightening as the numbers of samples increase.

Table 7: GMM40 sampling quality after 100,000 actor updates for different $N = M$. MMD is computed using 2,048 samples per distribution, while mode coverage is computed from 10,000 policy samples per seed. Values are mean $\pm$ sample standard deviation over 4 random seeds.

$N = M$	Ground truth	1	4	16	64	128	256	512
MMD ↓	–	0.6958 ± 0.1675	0.4867 ± 0.1096	0.1740 ± 0.0162	0.0541 ± 0.0241	0.0400 ± 0.0169	0.0250 ± 0.0131	$\mathbf{0.0184 \pm 0.0132}$
Mode coverage (/40) ↑	40.00	2.25 ± 0.96	9.25 ± 3.50	23.50 ± 1.00	38.25 ± 2.36	$\mathbf{40.00 \pm 0.00}$	$\mathbf{40.00 \pm 0.00}$	$\mathbf{40.00 \pm 0.00}$

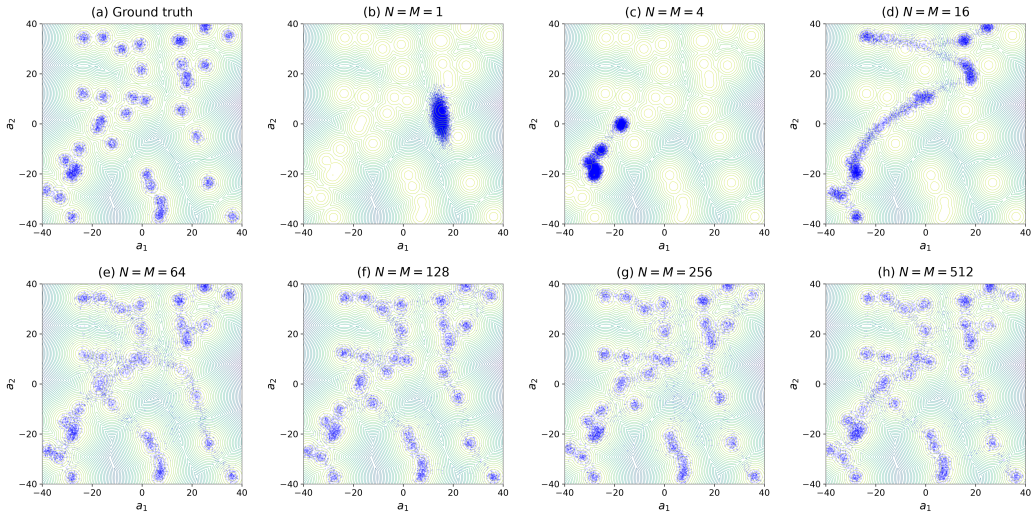

Figure 11: GMM40 samples from iBOLT for different numbers of latent samples and reference actions $N = M$, after 100,000 actor updates. Contours show the target density.