REINFORCEMENT LEARNING FOR FLOW-MATCHING POLICIES WITH DENSITY TRANSPORT

Anonymous authors
Paper under double-blind review

ABSTRACT

We present an online reinforcement learning (RL) algorithm for fine-tuning flow-matching policies in continuous-control problems. Our key insight is to view RL-based policy improvement as a transport of action densities towards regions of high reward, which naturally aligns with the transport formulation of flow matching models. Prior methods either approximate the current or optimal policy distribution or resort to distillation, which introduces biased gradients or sacrifices multimodal modeling capacity. In contrast, our approach for RL with Density Transport, which we name *RLDT*, constructs a transport field from a maximum-entropy RL objective using Stein Variational Gradient Descent (SVGD). Then, it fine-tunes a pretrained flow matching policy to align with this field. Training with this alignment objective is nontrivial because flow-matching policies generate actions via a multi-step process, making direct gradient-based optimization challenging. To overcome this challenge and stabilize training, we approximate policy actions from intermediate denoising steps via expected-target estimation. This allows the transport-field update to propagate into the network parameters without unstable backpropagation through time. Experimental results demonstrate that RLDT matches or outperforms competitive baselines in reward quality and convergence speed. This performance holds across diverse continuous-control tasks, encompassing both dense and sparse rewards, as well as state- and vision-based long-horizon robot manipulation.

1 INTRODUCTION

Vision-Language-Action (VLA) models Black et al. (2024a); Wu et al. (2026); Shukor et al. (2025); Kim et al. (2024) have demonstrated strong capacity to encode rich priors over state-action distributions, making them well-suited as foundations for downstream task adaptation. However, the learned prior, generally referred to as *base policy*, may not be optimal for a specific use case. For example, the trajectories in the training dataset could be suboptimal, or the policy may fail on scenarios outside its training distribution. When a tractable approximation of the task-specific optimal policy is available, it can be used to steer action generation without modifying the model parameters Ye (2025); Du & Song (2025); Fang et al. (2024); Psenka et al. (2024). When no such approximation is accessible, however, reinforcement learning offers a principled way to fine-tune the base policy on the target task.

The base policy admits a multitude of instantiations, e.g., diffusion Barreiros et al. (2026), flow-matching Lipman et al. (2024), or advanced tokenization Pertsch et al. (2025). In this work, we focus on flow-matching policies, following their successful adoption in state-of-the-art models for robotics Wu et al. (2026); Intelligence et al. (2025); Shukor et al. (2025). Training a flow-matching policy with reinforcement learning raises two main challenges: (1) we don't have samples from the task-specific optimal policy; (2) estimating the policy log-likelihood is very expensive due to the multi-step denoising process required to generate actions.

Prior work has addressed these issues in a variety of ways. One popular approach is optimizing lower bounds on the policy log-likelihood Yi et al. (2026); McAllister et al. (2026); Black et al. (2024b), which, however, leads to biased policy gradients. Other methods ignore the gradients at intermediate denoising steps Ren et al. (2025), or solve the reverse ODE to obtain a per-step target Li & Levine (2026). Other works introduce changes to the policy structure by distilling a multi-step pol-

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.

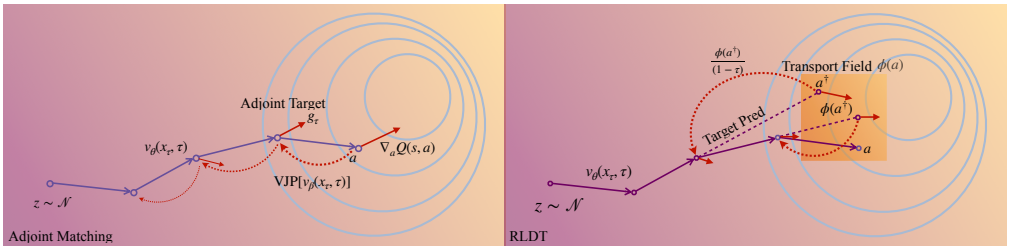

Figure 1: **RLDT Pipeline** Left: QAM Li & Levine (2026) uses the Q gradient at the final sample and the Vector Jacobian Product (VJP) of a reference policy π_β to compute intermediate adjoint targets g_τ to update π_θ. Right: RLDT constructs a transport field $\phi(a)$ on the action manifold and uses the expected posterior $a^\dagger$ to compute targets for intermediate denoising steps of π_θ.

icy into an actor requiring only a few steps Park et al. (2025); Zhang et al. (2025), which simplifies the RL problem and enables backpropagation through the denoising process. Similarly, other works model flow-matching as a recurrent neural network and gate the velocity to control the gradient magnitude Zhang et al. (2026). While effective in mitigating the problem of vanishing or exploding gradients, these methods constrain the expressivity of the trained policy, potentially leading to suboptimal performance.

In this paper, we propose Reinforcement Learning with Density Transport (**RLDT**), which recasts the RL policy improvement step as a probability transport problem *on the action manifold*. As illustrated in Fig. 1, RLDT aligns a flow-matching field to a transport field $\phi(a) : \mathbb{R}^d \to \mathbb{R}^d$ that drives action probabilities toward regions of higher expected reward. This formulation offers two key advantages over prior work: (i) it circumvents the exact computation of action log-likelihoods, and (ii) it enables direct update propagation through intermediate denoising steps without requiring structural policy changes, solving a reverse ODE, or proxy objectives.

We employ Stein Variational Gradient Descent (SVGD) Liu & Wang (2016) to compute the transport vector field $\phi(a)$. This field is designed to move samples across the action manifold toward the density of the optimal policy, as defined by a maximum-entropy RL objective. To mitigate training instabilities inherent in multi-step generation, we estimate the policy's actions from intermediate steps of the ODE using expected target prediction Lipman et al. (2024). Because these predicted targets reside on the action manifold, we can compute the transport update directly on them and propagate the resulting update into the flow-matching backbone.

We evaluate RLDT across three benchmark settings of increasing complexity: continuous control with dense rewards (OpenAI Gym), long-horizon multi-stage robot manipulation with sparse rewards and state-based information (Furniture-Bench Heo et al. (2023)), and vision-based robot manipulation with sparse rewards (Robomimic Mandlekar et al. (2021)). RLDT matches or outperforms baselines with large gains on Gym and FurnitureBench Lamp, and is comparable on Square and FurnitureBench One-Leg. We further show that, in contrast to prior methods that generate targets for intermediate denoising steps, our method produces well-conditioned update magnitudes over denoising steps.

Overall, our contributions can be summarized as follows:

- We formulate a novel RL algorithm for finetuning flow-matching policies by modeling policy improvement as a transport of action probabilities towards regions of high reward. Our formulation bypasses density estimation and backpropagation through time, leading to better performance than baselines and stable training.
- We tackle the problem of backpropagation through the denoising process of flow-matching policies by estimating the expected policy action from intermediate denoising steps. Since the transport field is defined on the action manifold, this prediction enables update propagation through the flow-matching backbone over the entire denoising process.

2 Related Work

On-Policy RL for Diffusion/Flow Policies. Methods in this category generally consist of on-policy algorithms inspired by PPO Schulman et al. (2017). The primary challenge in adapting these for

flow policies lies in estimating the policy log-likelihood to derive gradient updates. Because exact computation is computationally expensive for flow models, existing works typically utilize lower-bound approximations. For instance, DPPO Ren et al. (2025) computes the log-likelihood of all denoising latent variables as a lower bound of the marginal probability. To add stochasticity to the sampling process for exploration and gradient estimation, prior work transforms the flow ODE to an equivalent SDE Liu et al. (2025) or predicts learnable noise Sun et al. (2025b). This strategy has been applied to continuous control problems, which is the focus of this work, in ReinFlow Zhang et al. (2025) and SACFlow Zhang et al. (2026). These methods, however, remain dependent on density-based objectives. FPO McAllister et al. (2026) utilizes the Evidence Lower Bound (ELBO) Kingma & Gao (2023) to optimize the policy; however, when the advantage is negative, minimizing a lower bound does not strictly guarantee a reduction in the true policy log-likelihood. To mitigate this, FPO++ Yi et al. (2026) employs a particle ensemble and an asymmetric trust region to dampen gradients from negative advantages. In contrast to these methods, our work uses a value-based framework that bypasses the need to approximate the policy's log-likelihood entirely.

Off-Policy RL for Diffusion/Flow Policies. Methods in this category are primarily distinguished by how they leverage the critic to guide policy optimization. One approach employs the critic for rejection sampling, refining the policy by filtering for high-value actions Hansen-Estruch et al. (2023); Dong et al. (2025); Pfrommer et al. (2025); Sun et al. (2025a), potentially learning a reward surrogate to improve sample efficiency Pfrommer et al. (2025). Notably, Sun et al. (2025a) views the Q-value as a transport cost to build the optimal transport map from states to actions by selecting higher Q-value actions from sampled actions. However, this is generally restricted to minor deviations from the base policy, especially given a high-dimensional action space. Alternatively, some methods backpropagate the critic's gradient through the entire multi-step denoising chain Ding & Jin (2023); Zhang et al. (2024). This approach often suffers from instability, particularly as the number of denoising steps increases Park et al. (2025). To mitigate this, distillation techniques have been proposed to compress flow-matching policies into few-step models Park et al. (2025); Espinosa-Dice et al. (2025); Chen et al. (2025), though this frequently comes at the cost of the model's expressive power. Most relevant to our work are methods that bypass backpropagation through the denoising process by constructing step-wise supervision targets. While this introduces potential bias, the central challenge lies in designing targets that maintain stability without sacrificing optimality. Existing strategies address this by approximating the optimal policy Lu et al. (2023); Fang et al. (2024); Khan et al. (2026); Alles et al. (2025) or substituting the critic's gradient with its scalar value Ma et al. (2025b); Koirala & Fleming (2025); Zhang et al. (2024); Ma et al. (2025a); Psenka et al. (2024), potentially with additional losses to stabilize learning, e.g., entropy Gao et al. (2026). The method most closely related to ours utilizes adjoint matching to create step-wise objectives Li & Levine (2026). However, we observe that its updates are less well-balanced over intermediate steps than ours.

Optimal Transport for Flow Models Prior work has shown that optimal transport can be used to match noise and data pairs to improve training efficiency Mousavi-Hosseini et al. (2025). More related to our work, Kong et al. (2025) uses optimal transport for finetuning a flow policy with RL when the dynamics of the target task are different from those used to train the base policy. Specifically, they use OT for filtering rollouts and biasing exploration towards high-uncertainty states. Our transport perspective is used to build the RL objective and not only for filtering.

RL via Residuals or Guidance. These methods preserve the pretrained base policy while leveraging a few learnable parameters to modulate the action distribution on the target task. One approach involves training a lightweight residual actor, trained with RL, operating on the base policy's output Intelligence et al. (2026); Ankile et al. (2024); Wang et al. (2026). Alternatively, some methods optimize the generation process itself by using a Q-network to learn a steering noise or latent shift, which biases the denoising trajectory toward high-reward regions of the action space Ye (2025); Su et al. (2026). Guidance can also be achieved by using sampling-based optimizers like MPPI Wang et al. (2024). While these strategies maintain the robustness of the foundation model, their reliance on a frozen base can be a bottleneck when the base policy's initial coverage of the target task is poor.

3 METHOD

Our core insight is to cast RL as a transport map $T = \mathrm{id} + \epsilon\phi$, where id is the identity mapping and ϕ is the transport velocity field. This framework naturally fits flow-matching policies, which learn a time-dependent transport field from a simple prior distribution, e.g., white noise, to the target action distribution.

Following standard convention, we model reinforcement learning as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, p_s, r)$, where the state space $\mathcal{S}$ and action space $\mathcal{A}$ are continuous. The environment's (unknown) dynamics are characterized by the state transition probability density $p_s(s_{t+1}|s_t, a_t)$, where $p_s : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$. This function represents the probability density of the next state $s_{t+1} \in \mathcal{S}$ given the current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$. At each time step, the environment yields a bounded scalar reward $r(s_t, a_t) \in [r_{\min}, r_{\max}]$.

3.1 FLOW MATCHING POLICIES

Flow Matching Lipman et al. (2022) learns a velocity field $v_\theta(x_\tau, \tau)$ that generates a probability path $p_\tau(x)$, $\tau \in [0, 1]$, between an initial distribution $q(x)$ and a target distribution $p(x)$, such that $p_0(x) = q(x)$ and $p_1(x) = p(x)$. The velocity field $v_\theta(x_\tau, \tau)$ is used to construct a time-dependent diffeomorphic map $\psi_\tau(x) : [0, 1] \times \mathbb{R}^n \to \mathbb{R}^n$ defined via the ordinary differential equation:

$$\begin{aligned} \frac{d\psi_\tau(x)}{d\tau} &= v_\theta(\psi_\tau(x), \tau) \\ \psi_0(x) &= x \end{aligned} \tag{1}$$

The solution of this ODE maps a sample from the starting distribution $q(x)$ to the target distribution $p(x)$. Following Lipman et al. (2024), we train the velocity field $v_\theta(\psi_\tau(x), \tau)$ with the objective:

$$\min_\theta \mathbb{E}_{\tau \sim \mathcal{U}[0,1], x_1 \sim p(x), x_0 \sim q(x)} \|v_\theta(\psi_\tau(x), \tau) - (x_1 - x_0)\|^2. \tag{2}$$

In practice, at inference time we sample from $x_0 \sim q(x)$ and iteratively apply the velocity field for a fixed number of steps to obtain a sample $x_1 \sim p(x)$.

For a flow-matching policy, the random variables x are actions a_t, and the velocity field is conditioned on the state s_t. The initial distribution $q(a_t) = \mathcal{N}(0, I)$ and the target distribution $p(a_t) = \pi^*(a_t|s_t)$ is the optimal policy. To distinguish between the time t in the MDP and τ in the ODE, we refer to an intermediate step at τ of a flow matching policy as $a_{t,\tau}$. An important advantage of the optimization in Eq. 2 is that it enables training a policy only from samples of $\pi^*(a_t|s_t)$ (generally referred to as expert demonstrations).

From Eq. 1, we can see that a flow-matching policy effectively learns a vector field over the action manifold. In the following section, we demonstrate how to align this field to an optimal policy field for a maximum-entropy RL objective. This alignment yields a tractable, off-policy reinforcement learning algorithm.

3.2 MAXIMUM ENTROPY REINFORCEMENT LEARNING FOR FLOW MATCHING POLICIES

Following the maximum-entropy formulation Ziebart (2010), we define the RL optimization objective as:

$$\pi^* = \arg\max_\pi \mathbb{E}_{(s_0, a_0, \ldots, s_{T-1}, a_{T-1}) \sim \rho_\pi} \left[\sum_{t=0}^{T-1} \gamma^t \left(r(s_t, a_t) + \kappa \mathcal{H}(\pi(\cdot \mid s_t)) \right) \right] \tag{3}$$

where ρ_π is trajectory distribution induced by a policy π and κ is a constant to control entropy regularization. The solution to Eq. 3 can be achieved by iterative soft Q iteration Haarnoja et al. (2017):

$$\begin{aligned} Q(s_t, a_t) &\leftarrow r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}}[V(s_{t+1})] \\ V(s_t) &\leftarrow \kappa \log \int_{\mathcal{A}} \exp\left(\frac{1}{\kappa} Q(s_t, a') \right) da' \\ \pi^*(a_t|s_t) &= \frac{\exp(Q(s_t, a_t)/\kappa)}{Z}, \end{aligned} \tag{4}$$

where Z is the normalization term. Similarly to Haarnoja et al. (2017), we train a stochastic policy $\pi_\theta = \pi(a_t|s_t;\theta)$ by minimizing the cost $J_\pi(\theta)$, which consists of the KL divergence to the optimal policy $\pi^*(a_t|s_t)$:

$$J_\pi(\theta) = \mathbb{E}_{\rho_\pi}[J_\pi(\theta; s_\tau)] = \mathbb{E}_{\rho_\pi}\left[\mathrm{KL}\left(\pi_\theta(a_\tau|s_\tau)\,\middle\|\,\frac{\exp(Q(s_\tau,a_\tau)/\kappa)}{Z}\right)\right]. \tag{5}$$

This KL objective cannot be analytically computed because π_θ and Z are intractable. Therefore, similarly to Haarnoja et al. (2017), we optimize Eq. 5 via approximate sampling, and Stein Variational Gradient Descent (SVGD) Liu & Wang (2016). The Q network is trained using the target in van Seijen et al. (2016).

SVGD is a particle-based algorithm used to approximate a target probability distribution $p(x)$ by moving a set of particles $x_i \in \mathbb{R}^d \sim q(x)$ through a smooth transformation $T(x_i) = x_i + \epsilon\phi(x_i)$ to match that distribution. When the transformation $\phi(x)$ belongs to a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$, the optimal direction $\phi^*(x)$ minimizing the KL divergence between $p(x)$ and $q(x)$ is Liu et al. (2016):

$$\phi^*(x) = \frac{\phi^*_{q,p}(x)}{\|\phi^*_{q,p}(x)\|_{\mathcal{H}^d}} \quad \phi^*_{q,p}(x) = \mathbb{E}_{x'\sim q}[k(x',x)\nabla_{x'}\log p(x') + \nabla_{x'}k(x',x)], \tag{6}$$

where $k(x',x)$ is the positive definite kernel function and $\mathcal{H}^d$ is the d-dimensional norm that uniquely determine $\mathcal{H}$. We can use Eq. 6 to optimize the policy loss $J_\pi(\theta)$ in Eq. 5 by using as random variables the actions a_t, as sampling distribution $q(a_t) = \pi_\theta(a_t|s_t)$, and as target distribution $p(a_t) = \exp(Q(s_t,a_t)/\kappa)/Z$.

From a practical standpoint, this formulation is particularly advantageous for flow-matching policies. It bypasses the need to compute the normalization constant Z and the computationally expensive log-likelihood of the flow-matching policy. Instead, the algorithm requires only samples from $\pi_\theta(a_t|s_t)$. However, as outlined in Sec. 3.1, obtaining action samples requires multiple iterations of the flow model v_θ. Direct propagation over iterations, in particular for large number of steps, could dilute the learning signal. In the next section, we present how our algorithm can naturally overcome this issue thanks to its transport formulation.

3.3 Reinforcement Learning with Density Transport

To resolve the propagation dilution problem, we observe that any intermediate prediction $a_{t,\tau}$ (where $\tau < 1$) can be mapped to a corresponding sample $a^\dagger_{t,\tau}$ on the policy's action manifold. This allows us to evaluate the transport field ϕ^* in Eq. 6 across all intermediate steps, bypassing the need for backpropagation through the denoising process.

We define $a^\dagger_{t,\tau} = \mathbb{E}_{p_{\tau:1}}[a_{t,1} \mid a_{t,\tau}]$ as the expected posterior of $a_{t,\tau}$ Lipman et al. (2024). Tianhong Li & He (2025) show:

$$a^\dagger_{t,\tau} = a_{t,\tau} + (1-\tau)v(a_{t,\tau},\tau;\theta). \tag{7}$$

Let's denote the trajectory of $a^\dagger_{t,\tau}$ moving along the transport field $T(a) = id + \epsilon\phi^*(a)$ as $a^\dagger_{t,\tau}(\epsilon)$, where $\epsilon \in \mathbb{R}$, and the corresponding parameter change required in the network's backbone to predict $a^\dagger_{t,\tau}(\epsilon)$ as $\theta = \theta(\epsilon)$. Since $a_{t,\tau}, t, \tau$ are fixed, we can omit these symbols and write $a^\dagger_{t,\tau} = a^\dagger(\epsilon)$, $v(a_{t,\tau},\tau;\theta) = v(\theta(\epsilon)) = v(\epsilon) = (a^\dagger(\epsilon) - a)/(1-\tau)$. We can take the derivative on both sides w.r.t ϵ of the last equation at $\epsilon = 0$ and get

$$\left.\frac{dv(\epsilon)}{d\epsilon}\right|_{\epsilon=0} = \frac{1}{1-\tau}\left.\frac{da^\dagger(\epsilon)}{d\epsilon}\right|_{\epsilon=0} = \frac{1}{1-\tau}\phi^*(a^\dagger), \tag{8}$$

where the latest step is justified by the definition of the transport field $T(a)$.

We now need to find which update in the parameters $\xi = \frac{\delta\theta}{\delta\epsilon}$ will align the flow field to T at $\epsilon = 0$. To do so, we can use the chain rule to write:

$$\frac{dv(\epsilon)}{d\epsilon} = \frac{\delta v}{\delta\theta}\frac{\delta\theta}{\delta\epsilon} = \frac{\delta v}{\delta\theta}\xi. \tag{9}$$

From Eq. 8 and Eq. 9, one can see that the corresponding ξ should minimize:

$$\xi^* = \arg\min_{\xi} \left\| \frac{\delta v}{\delta \theta} \xi - \frac{1}{1-\tau} \phi^*(a^\dagger) \right\|^2 \tag{10}$$

Even though computing the exact update requires matrix inversion, we can still do a single gradient descent step Wang & Liu (2016) and compute the approximation $\hat{\xi}$ as:

$$\hat{\xi} = \frac{1}{(1-\tau)} \frac{\delta v}{\delta \theta}^\top \phi^*(a^\dagger) = \left[\partial_\theta \left(\frac{v_\theta^\top \phi^*(\mathrm{sg}(a^\dagger))}{(1-\tau)} \right) \right] \tag{11}$$

where $\mathrm{sg}(\cdot)$ is the stop gradient operator. In summary, this justifies the following RL loss:

$$\mathcal{L}_{\mathrm{RL}} = \mathbb{E}_{\tau \sim \mathcal{U}[0,1], a_{t,\tau} \sim p_\tau(\cdot|s_t)} \left[-\frac{v_\theta(a_{t,\tau}, \tau)^\top \phi^*(\mathrm{sg}(a^\dagger_{t,\tau}))}{T(1-\tau)} \right] \tag{12}$$

where T is the total number of ODE steps. We add two regularization terms to the training process. To encourage straight sampling trajectories, so that the one-step prediction $a^\dagger_{t,\tau}$ remains an accurate estimate of the clean action, we add a consistency loss to the model:

$$\mathcal{L}_{\mathrm{consist}} = \mathbb{E}_{a_{t,\tau} \sim p_\tau(\cdot|s_t), a_{t,1} \sim p_1(\cdot|s_t) = \pi_\theta(\cdot|s_t)} \left[\|a^\dagger_{t,\tau} - a_{t,1}\|^2 / (T(1-\tau))^2 \right]. \tag{13}$$

In addition, when fine-tuning a pretrained flow-matching model, we add a constraint loss $\mathcal{L}_{\mathrm{constraint}}$ to keep the fine-tuned policy close to the weights $\bar{\theta}$ from the previous iteration.

$$\mathcal{L}_{\mathrm{constraint}} = \mathbb{E}_{\tau \sim \mathcal{U}[0,1], a_{t,\tau} \sim p_\tau(\cdot|s_t)} \|v_\theta(a_{t,\tau}, \tau) - v_{\bar{\theta}}(a_{t,\tau}, \tau)\|^2 \tag{14}$$

In summary, the complete loss used in flow policy optimization is:

$$\mathcal{L}_{\mathrm{actor}} = \mathcal{L}_{\mathrm{RL}} + \lambda_1 \mathcal{L}_{\mathrm{consist}} + \lambda_2 \mathcal{L}_{\mathrm{constraint}}. \tag{15}$$

The full RLDT algorithm is summarized in the appendix Algorithm 1.

4 Experiments

We conduct experiments on a variety of environment settings, including dense reward (OpenAI Gym), long-horizon robot manipulation with sparse rewards (Furniture Bench), and vision-based robot manipulation tasks (Robomimic). For benchmarking, we adopt the same environment setting below for all the methods we test:

OpenAI Gym. We finetune the flow matching policy using RLDT on the "Walker2d", "Hopper", and "HalfCheetah" tasks. We use the original dense reward setting during the finetuning stage. The policy takes the state as input and predicts action with chunk size $T_a = 4$.

Furniture-Bench. We show that our method could also improve the base policy in long-horizon and sparse reward multi-stage tasks in the Furniture-Bench environment Heo et al. (2023). We chose the *One-Leg* and *Lamp* settings with Low and Medium randomness for the experiment. The *One-Leg* task has a single stage, and the *Lamp* task has two stages. The robot will get a single reward at the time when it successfully completes one stage. The policy takes the state as input and predicts action with chunk size $T_a = 8$.

Robomimic. To prove the effectiveness of our method on high-dimensional visual input, we run experiments in the Robomimic environment Mandlekar et al. (2021). Specifically, we chose the *Square* and *Transport* tasks. Our policy takes the proprioception and RGB image as input and predicts action with chunk size $T_a = 4$ for Square and $T_a = 8$ for Transport.

4.1 Implementation Details

For the base policy, we adopt the same network architecture as DPPO. For a fair comparison, we train all base policies on the same demonstrations as DPPO, and use the same learning rate and training steps. The only difference is the training objective, where DPPO uses the DDPM objective while we

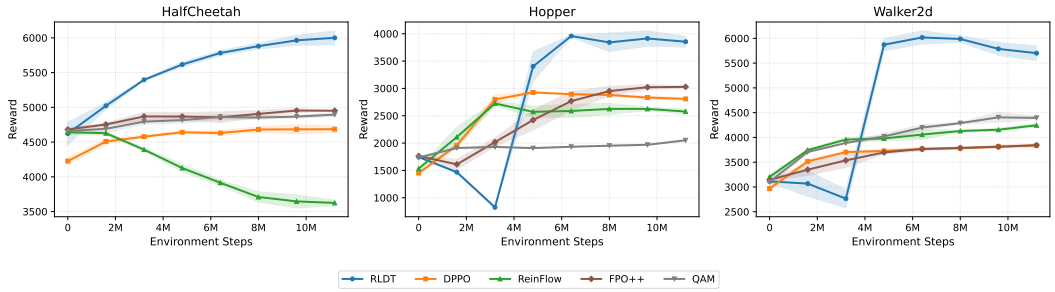

Figure 2: **Dense Reward Tasks** We run each setting for 3 different seeds and plot the mean and standard deviation.

use the Flow Matching objective. The ODE step T is 8 for the OpenAI Gym and Robomimic setup and 32 for the Furniture Bench setup.

For each state sample s_t, we generate $K = 8$ action samples for SVGD computation. We choose the temperature κ as $\kappa = \frac{\kappa_0}{K} \sum_{m=1}^{K} \|\nabla_a Q(s, a_m)\|$. We set $\kappa_0 = 0.001$. We use the K clean generated samples to construct the transport field $\phi^*(a)$. To compute the SVGD loss, for each sample in the K ensemble, we uniformly randomly select a timestep $t_i = i/T$ for $i \in \{0, 1, ..., T-1\}$ as the intermediate a_{t_i}. We choose the RBF kernel as the kernel function for $\phi^*(x)$, where $k(x, x') = \exp(-\|x - x'\|^2/(2h))$. The radius $h = \tilde{h}/(2\log(K+1))$, and $\tilde{h}$ is the median of the distances between the expected posterior and the clean samples. The loss weights are $\lambda_1 = 0.5$ and $\lambda_2 = 0.2$.

For state-based tasks, we use fully connected networks as the Q network. We choose SiLU as the activation function. For vision-based tasks, we use the Transformer as the feature encoder to extract features from the images and robot proprioception. Afterwards, we concatenate the features with the action and pass them to a fully connected network to estimate the Q value. The details for the network architecture and hyperparameters for Q-learning are presented in the appendix.

4.2 Results

We design our experimental setting to answer the following set of questions. The ablation study of hyperparameters (coefficient κ_0, number of particles K, and loss weights λ_1, λ_2) is in the appendix.

Q1: How effective is RLDT compared to state-of-the-art methods? We compare our method against prior work that approximate the log-likelihood with different lower bounds. Specifically, we select **DPPO** Ren et al. (2025), **ReinFlow** Zhang et al. (2025) and **FPO++** Yi et al. (2026). We additionally compare to **QAM** Li & Levine (2026), which generates targets for intermediate denoising steps using adjoint matching.

Baselines Implementation. DPPO and ReinFlow approximate the log-likelihood with the joint probability of all latent variables. Conversely, FPO++ approximates the log-likelihood using the ELBO lower bound. For DPPO and ReinFlow, we use the official implementations and parameters. For FPO++, we use the same pre-trained flow-matching policy as RLDT. We sample 16 time steps to compute the conditional flow matching loss for the ratio computation. We compute the ratio for each individual sample. We use the ASPO setting for clipping, where the ratios are clipped using the PPO Schulman et al. (2017) method for positive advantages, and the ratios are clipped using the SPO Xie et al. (2025) for negative advantages. We sweep the clipratio parameter for FPO++ of $\{0.01, 0.02, 0.04\}$ and find 0.01 works the best. For QAM, we keep the Q network learning parameters the same as RLDT and only change the policy update scheme. For the denoising steps, we use the same number of denoising steps as RLDT. For QAM, the slow actor π_β is fixed to the pre-trained flow matching policy. We sweep the inverse temperature parameter over $\{0.1, 0.3, 0.05\}$ for the inverse temperature in QAM and find 0.3 has the best performance, which is also recommended by the paper.

Results on the dense-reward Gym benchmarks are summarized in Fig. 2. RLDT achieves the highest rewards across all baselines by a significant margin. DPPO results match those reported in the orig-

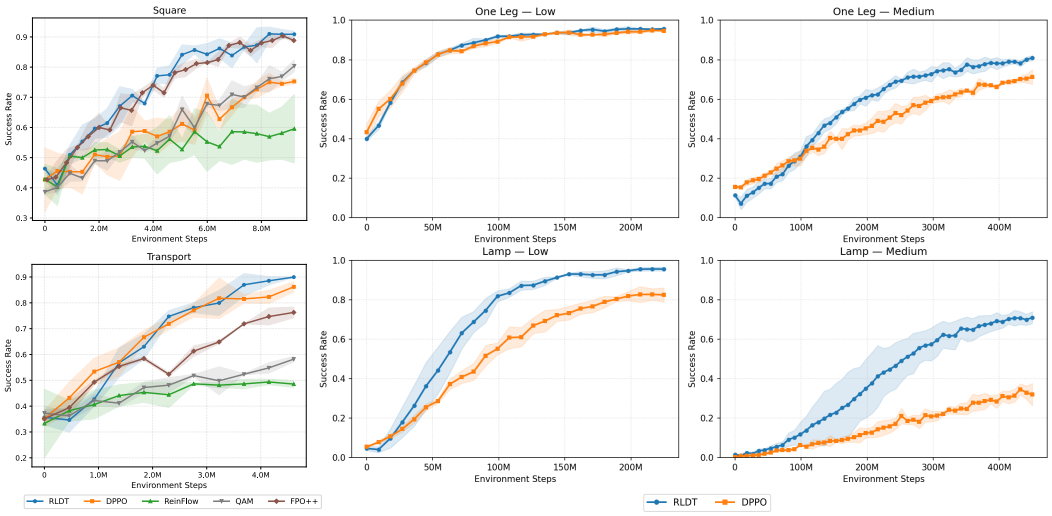

Figure 3: **Sparse Reward Tasks** The left plot presents results on the visual-based robomimic benchmark, and the right plot on state-based furniture bench tasks. Results are averaged over 3 seeds.

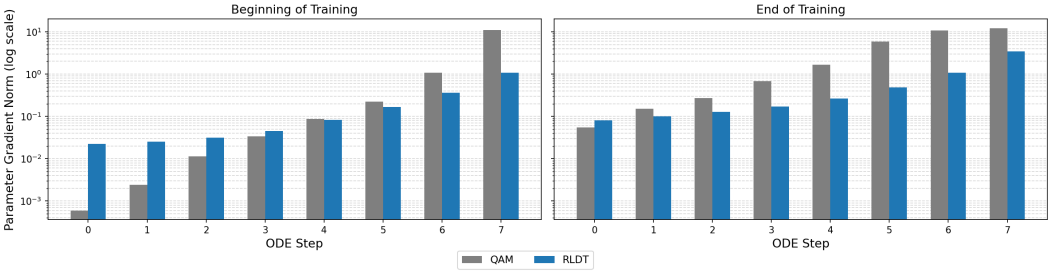

Figure 4: **Gradient Norm for ODE Timestep** Left: Parameter gradient norm computed using the pre-trained policy at the beginning of finetuning. Right: Parameter gradient norm computed using the finetuned policy at the end of finetuning. 0 is the first ODE step and 7 is the last ODE step.

inal paper, while ReinFlow underperforms on HalfCheetah and Hopper despite using their official code. QAM performance varies across tasks, in some outperforming other baselines and in others underperforming them. Nonetheless, it always underperformed our method.

Results on the robot manipulation tasks with sparse rewards are summarized in Fig. 3. We start from the robomimic benchmark (Fig. 3-left), where the policy has image-based inputs. In the *Square* task, RLDT starts with a higher initial success rate and continues to improve, achieving a final success rate of approximately 90%. For the *Transport* task, RLDT improves slightly more slowly than DPPO early in training but converges to comparable performance. FPO++'s performance is close to RLDT on the *Square* task, while it achieves a lower final success rate around 75% on the *Transport* task. As in the Gym benchmark, we find Reinflow to underperform other methods.

We conclude with FurnitureBench tasks (Fig. 3-right), where the policy has state and proprioception as input. Here, we only compare to the strongest baseline, DPPO. RLDT achieves performance comparable to it on the *One Leg Low* and *One Leg Med* tasks. However, on the *Lamp Med* task, RLDT reaches a success rate of approximately 70%, whereas DPPO achieves only 30%.

Q2: Does RLDT balance the learning signal across denoising timesteps? To analyze the effectiveness of the proposed optimization method, we compare parameter gradient magnitudes across ODE timesteps to QAM Li & Levine (2026). We plot the gradient norm at the beginning and end of the training for the *Hopper* environment in Fig. 4.

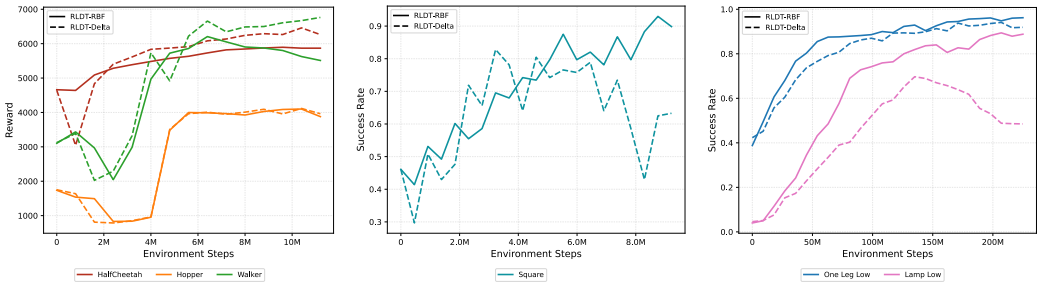

Figure 5: **Ablation on Kernel Function** We compare the RBF kernel (RLDT-RBF) against the Delta kernel (RLDT-Delta) in the SVGD computation.

At the beginning of training, QAM has a very unbalanced gradient from different ODE steps. The initial step contributes less than 0.001 to the parameter update, indicating that the early ODE step output does not contribute to the overall gradient. At the end of training, this problem is mitigated because the finetuned policy deviates from the reference policy, which contributes to a larger gradient at ODE step 0. For RLDT, the gradient contribution from each ODE is more evenly distributed, and the distribution remains steady over the training process. This empirically leads to faster and more stable training.

Q3: What is the role of the kernel function in the transport field? To analyze the impact of the RBF kernel, we compare with **RLDT-Delta**, which uses a delta function for the kernel function. Specifically, $k(x_i, x_j) = 1$ if $\|x_j - x_i\| = 0$ and $k(x_i, x_j) = 0$ if $\|x_j - x_i\| \neq 0$. In this case, we get $\phi^*(\text{sg}(a_t^\dagger)) = \nabla_a Q(s, a_t^\dagger)$, i.e., the critic's gradient. Therefore, RLDT-Delta does not have repulsive forces between particles. We run experiments on the Gym, the RoboMimic Transport, and the FurnitureBench environment.

The result are summarized in Fig. 5. On the Gym task, the two kernels behave similarly, with RLDT-Delta slightly outperforming RLDT-RBF. This is expected since exploration is simple in these dense reward tasks: following the critic's gradient is sufficient for high performance. We confirm this by checking the norm $\|\phi^*(a_t^\dagger)\|$, and find that **RLDT-Delta** always has a higher norm than **RLDT-RBF**, and the $\mathcal{L}_{\text{constraint}}$ is consistently higher, indicating a more drastic update of the actor (Figures in the Appendix). However, this behavior changes when moving to sparse reward robotics tasks. For the *Square* task and the *Lamp-Low* task, **RLDT-Delta** fails to improve the policy after half of the training. Interestingly, the performance drops near the end, potentially due to an overfitted critic. Overall, this shows that the delta kernel leads to instabilities for sparse reward tasks.

While the relative trends between these kernels were expected, we unexpectedly found that the delta function does not lead to mode collapse. Conversely, we found that the pair-wise distance between action samples is positive for all tasks. We hypothesize that this is because the Q function keeps evolving, so that the samples cannot stop in the same location. However, only relying on the Q function for exploration is insufficient for tasks with sparse rewards.

5 Conclusion

We presented RLDT, a reinforcement learning finetuning framework for flow-matching policies that formulates policy improvement as probability transport on the action manifold. Experiments across dense-reward continuous control (OpenAI Gym), sparse-reward long-horizon manipulation (Furniture-Bench), and visual manipulation (Robomimic) demonstrate that RLDT matches and outperforms state-of-the-art baselines. Despite these advantages, RLDT introduces a computational overhead during training due to the ensemble sampling when computing the transport field. However, our ablation (see Appendix) shows that only four particles are sufficient for high performance, and the wall clock time is comparable to QAM. Overall, our work demonstrates the advantages of formulating RL as the alignment of a flow-matching transport field with a target field that drives action probabilities toward the optimal policy density.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for designing or providing feedback on research methodology or experiments, and for implementing methods. We have not used generative AI tools for interpreting results, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, or supporting qualitative and thematic data analysis. Generating synthetic data sets, cleaning and reformatting datasets is not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures or images, suggesting experimental parameters, and identifying relevant literature. We have reviewed all AI-assisted work. We manually check all the AI-proposed relevant works and all the figures. LLM-generated code was verified and tested for correctness by all authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

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

Lars Ankile, Anthony Simeonov, Idan Shenfeld, Marcel Torne, and Pulkit Agrawal. From imitation to refinement – residual rl for precise assembly, 2024. URL https://arxiv.org/abs/2407.16677.

Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, Eric Cousineau, Hongkai Dai, Ching-Hsin Fang, Kunimatsu Hashimoto, Muhammad Zubair Irshad, Masha Itkina, et al. A careful examination of large behavior models for multitask dexterous manipulation. *Science Robotics*, 11 (113):eaea6201, 2026.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. *CoRR*, abs/2410.24164, 2024a. doi: 10.48550/ARXIV.2410.24164. URL https://doi.org/10.48550/arXiv.2410.24164.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *The International Conference on Learning Representations*, 2024b.

Tianyi Chen, Haitong Ma, Na Li, Kai Wang, and Bo Dai. One-step flow policy mirror descent. *arXiv preprint arXiv:2507.23675*, 2025.

Zihan Ding and Chi Jin. Consistency models as a rich and efficient policy class for reinforcement learning. *arXiv preprint arXiv:2309.16984*, 2023.

Perry Dong, Qiyang Li, Dorsa Sadigh, and Chelsea Finn. Expo: Stable reinforcement learning with expressive policies. *arXiv preprint arXiv:2507.07986*, 2025.

Maximilian Du and Shuran Song. Dynaguide: Steering diffusion policies with active dynamic guidance. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Nicolas Espinosa-Dice, Yiyi Zhang, Yiding Chen, Bradley Guo, Owen Oertell, Gokul Swamy, Kiante Brantley, and Wen Sun. Scaling offline rl via efficient and expressive shortcut models. *Neural Information Processing Symposium (NeurIPS)*, 2025.

Linjiajie Fang, Ruoxue Liu, Jing Zhang, Wenjia Wang, and Bing-Yi Jing. Diffusion actor-critic: Formulating constrained policy iteration as diffusion noise regression for offline reinforcement learning. *arXiv preprint arXiv:2405.20555*, 2024.

Ting Gao, Stavros Orfanoudakis, Nan Lin, Elvin Isufi, Winnie Daamen, and Serge Hoogendoorn. Flow matching policy with entropy regularization. *arXiv preprint arXiv:2603.17685*, 2026.

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

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

Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J. Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. In *Robotics: Science and Systems*, 2023.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. Pi0.5: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, et al. Pi07: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Fairoz Nower Khan, Nabuat Zaman Nahim, Ruiquan Huang, Haibo Yang, and Peizhong Ju. Flow matching for offline reinforcement learning with discrete actions. *arXiv preprint arXiv:2602.06138*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Paul Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2024. URL https://proceedings.mlr.press/v270/kim25c.html.

Diederik P Kingma and Ruiqi Gao. Understanding diffusion objectives as the ELBO with simple data augmentation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=NnMEadcdyD.

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

Lingkai Kong, Haichuan Wang, Tonghan Wang, GUOJUN XIONG, and Milind Tambe. Composite flow matching for reinforcement learning with shifted-dynamics data. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Qiyang Li and Sergey Levine. Q-learning with adjoint matching. *CoRR*, abs/2601.14234, 2026. doi: 10.48550/ARXIV.2601.14234. URL https://doi.org/10.48550/arXiv.2601.14234.

Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise. *CoRR*, abs/2511.13720, 2025. doi: 10.48550/ARXIV.2511.13720. URL https://doi.org/10.48550/arXiv.2511.13720.

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

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *CoRR*, abs/2412.06264, 2024. doi: 10.48550/ARXIV.2412.06264. URL https://doi.org/10.48550/arXiv.2412.06264.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-grpo: Training flow matching models via online rl. *arXiv preprint arXiv:2505.05470*, 2025.

Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29, 2016.

Qiang Liu, Jason Lee, and Michael Jordan. A kernelized stein discrepancy for goodness-of-fit tests. In *International conference on machine learning*, pp. 276–284. PMLR, 2016.

Cheng Lu, Huayu Chen, Jianfei Chen, Hang Su, Chongxuan Li, and Jun Zhu. Contrastive energy prediction for exact energy-guided diffusion sampling in offline reinforcement learning. In *International Conference on Machine Learning*, pp. 22825–22855. PMLR, 2023.

Haitong Ma, Tianyi Chen, Kai Wang, Na Li, and Bo Dai. Efficient online reinforcement learning for diffusion policy. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025a. URL https://proceedings.mlr.press/v267/ma25d.html.

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

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=eoEmoKoQpJ.

Alireza Mousavi-Hosseini, Stephen Y Zhang, Michal Klein, and Marco Cuturi. Flow matching with semidiscrete couplings. *arXiv preprint arXiv:2509.25519*, 2025.

Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=KVf2SFL1pi.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Samuel Pfrommer, Yixiao Huang, and Somayeh Sojoudi. Reinforcement learning for flow-matching policies. *arXiv preprint arXiv:2507.15073*, 2025.

Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via q-score matching. In *International Conference on Machine Learning*, pp. 41163–41182. PMLR, 2024.

Allen Z Ren, Justin Lidard, Lars Lien Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. In *The International Conference on Learning Representations*, 2025.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Entong Su, Tyler Westenbroek, Anusha Nagabandi, and Abhishek Gupta. Rfs: Reinforcement learning with residual flow steering for dexterous manipulation. In *The Fourteenth International Conference on Learning Representations*, 2026.

Mingyang Sun, Pengxiang Ding, Weinan Zhang, and Donglin Wang. Score-based diffusion policy compatible with reinforcement learning via optimal transport. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=2dqiqST8ZJ.

Xiaohui Sun, Ruitong Xiao, Jianye Mo, Bowen Wu, Qun Yu, and Baoxun Wang. F5r-tts: Improving flow-matching based text-to-speech with group relative policy optimization. *arXiv preprint arXiv:2504.02407*, 2025b.

Harm van Seijen, Ashique Rupam Mahmood, Patrick M. Pilarski, Marlos C. Machado, and Richard S. Sutton. True online temporal-difference learning. *J. Mach. Learn. Res.*, 17:145:1–145:40, 2016. URL https://jmlr.org/papers/v17/15-599.html.

Dilin Wang and Qiang Liu. Learning to draw samples: With application to amortized mle for generative adversarial learning. *arXiv preprint arXiv:1611.01722*, 2016.

Pengcheng Wang, Chenran Li, Catherine Weaver, Kenta Kawamoto, Masayoshi Tomizuka, Chen Tang, and Wei Zhan. Residual-mppi: Online policy customization for continuous control. *arXiv preprint arXiv:2407.00898*, 2024.

Yunshen Wang, Shaohang Zhu, Peiyuan Zhi, Yuhan Li, Jiaxin Li, Yong-Lu Li, Yuchen Xiao, Xingxing Wang, Baoxiong Jia, and Siyuan Huang. Omnixtreme: Breaking the generality barrier in high-dynamic humanoid control. *arXiv preprint arXiv:2602.23843*, 2026.

Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, et al. A pragmatic vla foundation model. *arXiv preprint arXiv:2601.18692*, 2026.

Zhengpeng Xie, Qiang Zhang, Fan Yang, Marco Hutter, and Renjing Xu. Simple policy optimization. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/xie25m.html.

Hanming Ye. Steering diffusion policies with value-guided denoising. In *NeurIPS 2025 Workshop: Second Workshop on Aligning Reinforcement Learning Experimentalists and Theorists*, 2025. URL https://openreview.net/forum?id=HKG7I3iqCY.

Brent Yi, Hongsuk Choi, Himanshu Gaurav Singh, Xiaoyu Huang, Takara E. Truong, Carmelo Sferrazza, Yi Ma, Yan Duan, Pieter Abbeel, Guanya Shi, Karen Liu, and Angjoo Kanazawa. Flow policy gradients for legged robots, 2026. URL https://openreview.net/forum?id=BA6n0nmagi.

Ruoqi Zhang, Ziwei Luo, Jens Sjölund, Thomas B Schön, and Per Mattsson. Entropy-regularized diffusion policy with q-ensembles for offline reinforcement learning. *Advances in neural information processing systems*, 37:98871–98897, 2024.

Tonghe Zhang, Chao Yu, Sichang Su, and Yu Wang. Reinflow: Fine-tuning flow matching policy with online reinforcement learning, 2025. URL https://arxiv.org/abs/2505.22094.

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 *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=zZvWj4JrYj.

Brian D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, USA, 2010. URL https://doi.org/10.1184/r1/6720692.v1.

A Algorithm Summary

Our method is summarized in Algorithm 1.

Algorithm 1: RLDT

Flow matching policy parameters θ, Q-network parameters $\{\phi_i\}, i = 1, 2$;
Assign target parameters: $\bar{\theta} \leftarrow \theta,\ \bar{\phi}_i \leftarrow \phi_i$;
$\mathcal{D} \leftarrow$ empty replay buffer;
for *each training iteration* **do**
  `// Environment Rollout`
  for $m = 1$ **to** M **do**
    Sample $a_t \sim \pi_\theta(a_t \mid s_t)$ from the flow matching policy;
    Execute a_t, observe (s_t, a_t, r_t, s_{t+1}); store transition in $\mathcal{D}$;
  Sample minibatch (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$;
  `// Q-Network Update`
  for *each gradient step* **do**
    Compute the Q target using Eq.(5) from van Seijen et al. (2016);
    Update ϕ_i by minimizing $\|Q_{\phi_i}(s_t, a_t) - y_i\|^2$ for $i = 1, 2$;
    Update target networks: $\bar{\phi}_i \leftarrow \rho\,\phi_i + (1-\rho)\,\bar{\phi}_i$ for $i = 1, 2$;
  `// Policy Update via RLDT`
  for *each gradient step* **do**
    Sample $\tau \sim \mathcal{U}[0, 1]$, $a_{t,0} \sim \mathcal{N}(0, I)$;
    Run ODE in Eq.1 to get intermediate a_t;
    Compute manifold prediction: $a^\dagger_{t,\tau} \leftarrow a_{t,\tau} + (1-\tau)\,v_\theta(a_{t,\tau}, \tau \mid s_t)$;
    Compute transport direction $\phi^*(a^\dagger_{t,\tau})$ via Eq. 6;
    Update θ by minimizing Eq. 15;
  Update target $\bar{\theta}$ by $\bar{\theta} \leftarrow \theta$;

B Training Parameters

B.1 Base Policy Training

All base policies are trained via behavioral cloning using the flow matching objective in Eq. 2 with a cosine-annealing learning rate scheduler. The differences among the three environments are listed below:

OpenAI Gym. The policy is a three-layer residual MLP (hidden dim 512, ReLU activations, time embedding dim 16, no layer normalization) conditioned on a single observation frame. Training runs for 200 epochs with batch size 128, initial learning rate 10^{-3} decaying to 10^{-4}, and a single warm-up step.

FurnitureBench. The higher observation dimensionality ($d_s = 58$, $d_a = 10$) calls for a deeper network: a seven-layer residual MLP with hidden dim 1024, a two-layer conditioning MLP (dims [512, 64]), time embedding dim 32, and layer normalization. Training runs for 8 000 epochs with batch size 256, initial learning rate 10^{-4} decaying to 10^{-5}, and 100 warm-up steps.

Robomimic. The visual policy uses a Vision Transformer (ViT) image encoder (patch size 8, depth 1, embedding dim 128, 4 attention heads, spatial embedding dim 128) followed by a three-layer residual MLP (hidden dim 768, time embedding dim 32) with online data augmentation. Proprioceptive observations ($d_s = 9$ for *Square*, $d_s = 18$ for *Transport*) and RGB images ($3 \times 96 \times 96$) are fused via the spatial embedding. Training runs for 8 000 epochs with batch size 256, initial learning rate 10^{-4} decaying to 10^{-5}, and 100 warm-up steps.

A summary of the key hyperparameters is provided in Table 1.

Environment	Network	Hidden dim	T_a	Epochs	Batch size	LR
Gym (HalfCheetah, Hopper, Walker2d)	MLP $\times 3$	512	4	200	128	$10^{-3} \rightarrow 10^{-4}$
FurnitureBench	MLP $\times 7$	1024	8	8000	256	$10^{-4} \rightarrow 10^{-5}$
Robomimic (*Square*)	ViT + MLP $\times 3$	768	4	8000	256	$10^{-4} \rightarrow 10^{-5}$
Robomimic (*Transport*)	ViT + MLP $\times 3$	768	8	8000	256	$10^{-4} \rightarrow 10^{-5}$

Table 1: Base policy pretraining hyperparameters.

B.2 RL Training

All environments share the same SVGD and actor update settings: temperature $\kappa_0 = 0.001$, $K = 8$ particles, actor learning rate 2×10^{-5} (32-step warmup), and 32 policy gradient steps per iteration. The RBF kernel bandwidth is set to the median pairwise distance of the K action samples. The critic uses a double-Q ensemble ($M = 2$). Two critic warm-up iterations are performed before the first actor update.

The three environment families differ in discount factor, number of training iterations, Q-update steps per iteration, λ-Return, critic architecture, and reward handling, as detailed below.

OpenAI Gym. The critic is a three-layer MLP (hidden dim 256, SiLU, no residual) taking the concatenated state and action chunk as input. Rewards are normalized online with EMA (smoothing $\alpha = 0.01$). We smooth over the average of rewards per step. Training runs for 70 iterations with 100 parallel environments, Q-network updated for 1 024 gradient steps per iteration with learning rate 0.001, $\gamma = 0.99$, Q target momentum $\rho = 0.998$, and $\lambda = 0.0$. The training takes around 1.5 GPU hours on a single NVIDIA 2080Ti GPU.

FurnitureBench. The critic is a three-layer residual MLP (hidden dim 512, SiLU, layer normalization disabled). Rewards are sparse and fixed to $[0, 1]$ per stage; balanced sampling (equal nonzero and zero reward trajectories) is used to mitigate reward sparsity. Training runs for 500 iterations for medium randomness and 250 iterations for low randomness with 1 000 parallel environments, Q-network updated for 512 gradient steps per iteration with learning rate 0.0005, $\gamma = 0.995$, Q target momentum $\rho = 0.998$, and $\lambda = 0.5$. The training takes around 48 GPU hours on a single NVIDIA A40 GPU.

Robomimic. Both the actor and critic share the same ViT backbone architecture (patch size 8, depth 1, embedding dim 128, 4 heads) to encode RGB observations; the critic head is a three-layer MLP (hidden dim 256, SiLU). Sparse binary rewards are used with balanced sampling and Q-target clipping to $[0, 1]$. Training runs for 200 iterations for Square and 100 iterations for Transport using 64 parallel environments ($n_{\text{steps}} = 200$ per iteration), Q-network updated for 512 gradient steps per iteration with learning rate 0.001, $\gamma = 0.999$, Q target momentum $\rho = 0.995$, and $\lambda = 0.5$.The training takes around 30 GPU hours on a single NVIDIA A40 GPU.

A summary of key hyperparameters of RL training is provided in Table 2.

Environment	n_{envs}	Q steps/itr	γ	ρ	λ	Critic	Reward
Gym	100	1024	0.990	0.995	0.0	MLP $\times 3$ (256)	EMA norm.
FurnitureBench	1000	512	0.995	0.995	0.5	MLP $\times 3$ (512)	Sparse $[0, 1]$
Robomimic	64	512	0.999	0.995	0.5	ViT + MLP $\times 3$ (256)	Sparse $[0, 1]$

Table 2: RL fine-tuning hyperparameters.

C More Ablation Study

C.1 Ablation on Temperature κ_0

We analyze two hyperparameters relevant to our methods. The first one is the temperature κ_0. We choose three different settings for κ_0 as {1e-3, 1e-4, 0}. We run the experiment in the Robomimic environment. The result is summarized in Fig. 6. For the *Transport* task, all three temperature

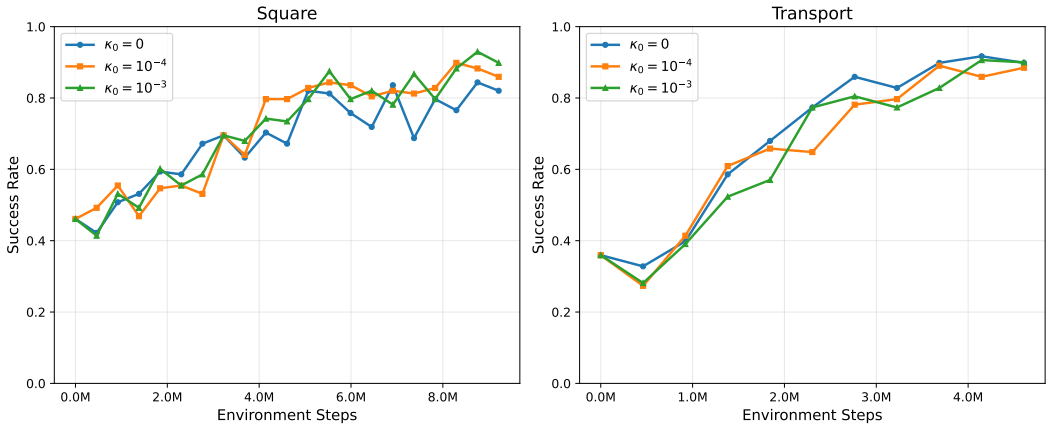

Figure 6: **Ablation Study on Temperature κ_0 on Robomimic Task**

settings have similar performance at the end. For the *Square* task, a higher temperature leads to a higher terminal success rate. We attribute this to the exploration led by a higher temperature.

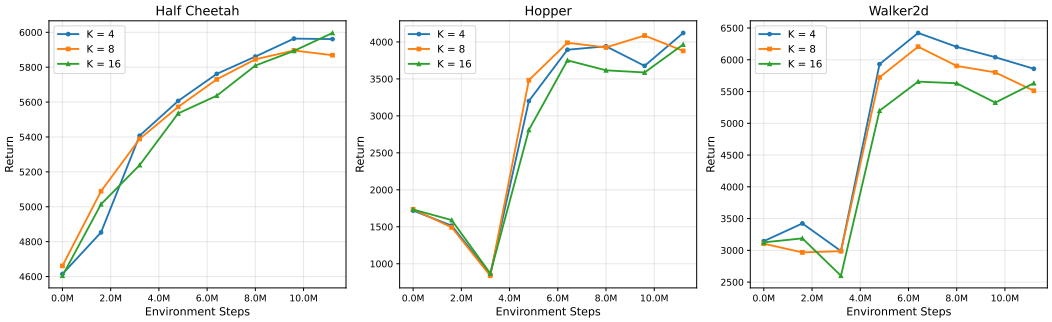

Figure 7: **Ablation Study on Ensemble Size on Gym Task**

C.2 Ablation on ensemble size K

The second one is the sampled ensemble size K to compute the transport field $\phi^*(a)$. We test three settings: $K = 4, 8, 16$. As we can see from Fig. 7, our RLDT has similar performance for different ensemble sizes, indicating our algorithm is not sensitive to this parameter in the Gym task. We do a wall-clock time comparison with QAM on an L40 GPU. We measure the time for each update step for the actor with the same batch size. We calculate the mean and standard deviation from a total of 128 update steps. For RLDT, we set $K = 8$ samples. For RLDT, the time is 75 ± 7 ms. For QAM, the time cost is 70 ± 16 ms. The slight difference is because although we spend more time sampling multiple particles, we only choose one intermediate time step per particle when computing the loss. For QAM, it needs to compute the adjoint targets for all 8 denoising steps, so it spends 8x more time during the adjoint target computation.

C.3 Ablation on loss weight λ_1, λ_2

We compare different choices of the loss weight (λ_1, λ_2) in Eq. 15 on the *Lamp-Low* task. The results are summarized in Table. 3. We use $\lambda_1 = 0.5, \lambda_2 = 0.2$ (first row) in our paper. Our algorithm is not very sensitive to the consistency loss (λ_1). The result is similar for row 1, 4, 5, 6. When we set $\lambda_1 = 0.0$ (row 4), the improvement will become slower at the beginning, but eventually it is comparable to our setting in the paper. For the constraint loss, decreasing the constraint loss will make the training unstable at the beginning (row 3), but it eventually gets similar performance

at the end. However, removing the constraint loss (row 2) will lead to training failure, getting only a 0 success rate.

Env Steps	45M	90M	135M	180M	215M
$(\lambda_1 = 0.5, \lambda_2 = 0.2)$	**0.456**	**0.937**	**0.949**	**0.969**	**0.980**
$(\lambda_1 = 0.5, \lambda_2 = 0.0)$	0.000	0.000	0.000	0.000	0.000
$(\lambda_1 = 0.5, \lambda_2 = 0.1)$	0.360	0.798	0.899	0.966	0.980
$(\lambda_1 = 0.0, \lambda_2 = 0.2)$	0.185	0.613	0.821	0.952	0.980
$(\lambda_1 = 0.1, \lambda_2 = 0.2)$	0.229	0.650	0.890	0.939	0.970
$(\lambda_1 = 1.0, \lambda_2 = 0.2)$	0.227	0.709	0.893	0.959	0.973

Table 3: Ablation on the loss weights λ_1 (consistency) and λ_2 (constraint) on the *Lamp-Low* task. We report success rate at different numbers of environment steps.

C.4 Ablation on Velocity Formulation

We do an ablation study to show that our formulation of the RL process as density transport is non-trivial. We compare with another previous study Haarnoja et al. (2017) that amortized the learning using a gradient perspective, which is denoted as G-View. For G-View, they set $\frac{\partial J_\pi}{\partial a^\dagger} \propto \phi(a^\dagger)$. Then using back-propagation, the gradient is

$$\frac{\partial J_\pi}{\partial \theta} = \frac{\partial J_\pi}{\partial a^\dagger} \frac{\partial a^\dagger}{\partial v_\theta} \frac{\partial v_\theta}{\partial \theta} \propto \mathbb{E}[(1-\tau)\phi(a^\dagger)^\top \partial_\theta v_\theta] \tag{16}$$

where the coefficient $(1-\tau)$ appears in the numerator instead of the denominator in Eq. 12. The result is shown in Fig. 8. We find that V-View (Ours) consistently outperforms the G-View implementation. By definition, the transform ϕ describes how the samples will move instead of the change rate of KL divergence w.r.t the sample location. It is important to note that $\tau = 0$ in Haarnoja et al. (2017) since they use a Gaussian policy, so there is no difference for them between the two interpretations. However, there is a difference if a policy is parametrized as a flow-matching model.

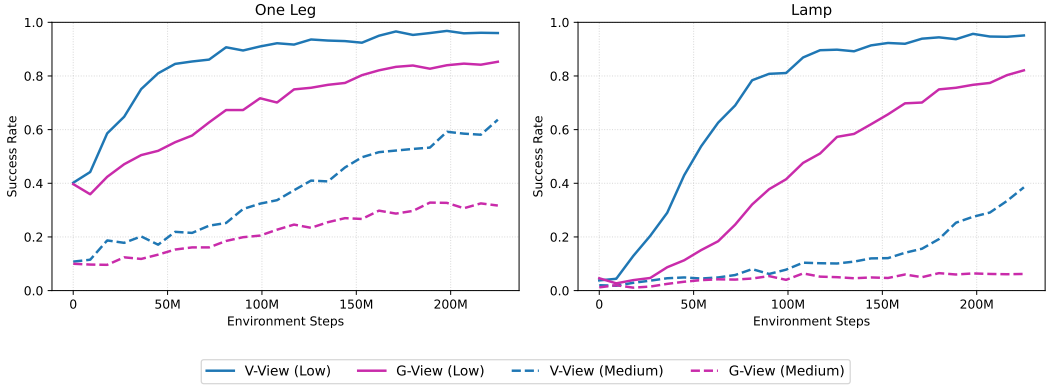

Figure 8: **Ablation on Velocity Formulation** on the FurnitureBench environment. **V-View** is our implementation and **G-View** is the gradient formulation. Solid lines are low randomness and dash lines are medium randomness.

C.5 More Results for Kernel Function

To show the difference of delta kernel and RBF kernel. We plot the constraint loss and the average norm of transport velocity $\phi(a^\dagger)$ over the training steps. The result is summarized in Fig. 9. The delta kernel incurs a higher velocity norm than the RBF kernel. This is because the RBF kernel smooths the Q gradient value at all the clean samples, while the delta kernel only uses the gradient at a single point. The higher norm leads to higher constraint loss, indicating the policy updates more drastically using the delta kernel. There are sharp spikes during training using a delta kernel, indicating the instability incurred by the delta function.

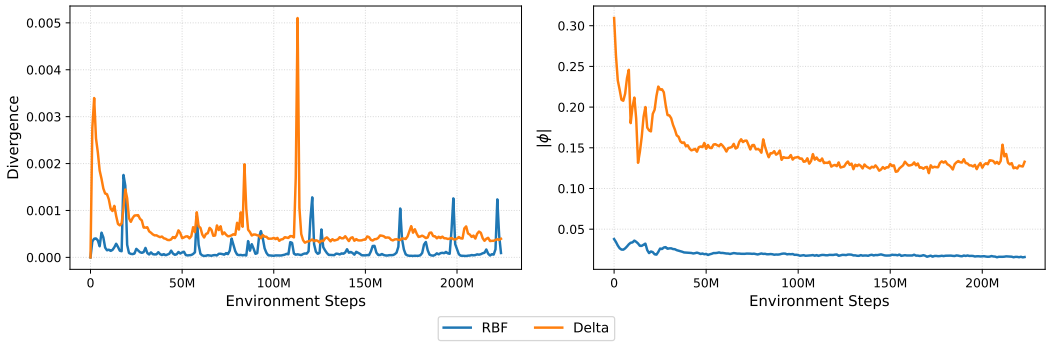

Figure 9: **Training Statistics for Different Kernel Function** The training statistics on the *Lamp-Low* task. Left: The constraint loss over the training steps. Right: The average norm of $\phi(a^\dagger)$ over the training steps