CoVE-PPO: Change-of-Variables Exact-Density PPO for Flow Policies

Anonymous authors
Paper under double-blind review

Abstract

Flow models transport probability density through deterministic maps, creating expressive multi-modal distributions attractive for robotic manipulation. Computing the standard policy gradient requires differentiating through action log-density, which is available through change of variables. Prior methods avoid the exact density citing Jacobian-trace estimation noise, cost, or numerical instability. We revisit these objections, and show that change of variables is tractable and effective. Sharing trace estimation probes across the ratio makes the estimator's error scale with the Jacobian difference rather than the Jacobian, and as few as 1 probe matches the full trace's learning performance. Much of the instability comes from a stiff forward solve, and can be fixed by placing integration nodes within the solver's stability region. We introduce CoVE-PPO, which optimizes the change-of-variables density of the executed action, with no surrogate or augmented variable: in a general form for any velocity field, and in EDC, a parameterization whose divergence is a bounded forward network output giving stable solves and free divergence. Across state- and vision-based robot manipulation, we show our method matches or exceeds on-policy baselines at matched or higher sample efficiency.

1 Introduction

Behavior Cloning (BC) using a policy expressed as a continuous normalizing flow has emerged as an important paradigm for learning challenging robotic behaviors (Black et al., 2026). However, the performance of this type of model (both single task and large multitask models) often plateaus short of 100% reliability. Reinforcement learning (RL) can be used to improve performance beyond what would be possible with BC alone.

However, this introduces a key challenge. While BC benefits from complex policy classes like continuous normalizing flows, standard RL algorithms can be unstable when applied to them naively. The problem is that many of these algorithms require calculating the log-density of the policy which can be computationally expensive for continuous normalizing flows. Several methods have been proposed for improving policies represented as continuous normalizing flows including using the flow matching loss gradient as a surrogate for the log density gradient, and adapting variational objectives used to train diffusion models (McAllister et al., 2025; Yi et al., 2026). Unfortunately, these approaches introduce bias and instability hurting overall performance.

One approach to calculating the log density of a continuous normalizing flow is to use the change-of-variables formula (Grathwohl et al., 2018). Unfortunately, using the exact change-of-variables formula requires velocity network Jacobian trace computation which can be expensive for high-dimensional network output spaces like chunked policies that output actions for multiple timesteps at once. Past work training flows using the change-of-variables formula used traces estimated with Hutchinson trace estimator (Skilling, 1989; Hutchinson, 1989), which adds noise to training. Additionally ODE solvers can introduce discretization error. Because of these noise sources, previous work introduced surrogates and alternatives to the exact density. However, these surrogates come at a cost. Chain based methods extend the MDP horizon, increasing credit assignment difficulty. Surrogate ratios from the flow-matching loss are biased because of the variational gaps (Hu et al., 2026), and thus do not exactly track policy change.

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.

This paper proposes an approach to using PPO with a continuous normalizing flow policy which avoids these issues by applying the change-of-variables formula. We make two contributions:

First, we show that because PPO estimates a policy ratio rather than policy log density, noise can be dramatically reduced by sharing trace probe vectors, reducing the number of probes required for stable training to a single probe. This enables dramatic computation time reductions from the exact log density or previous work's Hutchinson trace estimator approximation methods (Hutchinson, 1989). Further we find that the gradient instability noted by previous work (Yang et al., 2026) is caused by standard flow matching training and can be solved by ensuring that flow sampling and reverse solves for log density calculation both fit within the solver's linear stability region. We show that Hutchinson trace estimation with a shared single probe is effective for training, and we show the exact trace and a single probe match performance. We show by using higher order solvers with nodes placed according to the pre-trained flow's stiffness we maintain stable reverse solves with fewer network evaluations.

Second, we introduce explicit-diagonal coupling (EDC) flow PPO, building off work that seeks to provide cheap network Jacobians and invertible networks. We instead focus on the network's flow diagonal, a vector needed for the trace computation required by the CoV formula. Our network outputs a parameterized per-dimension affine velocity, and uses a 2-group structure to ensure the network's Jacobian diagonal is available without differentiation. We apply a tanh bound on the output diagonal, which ensures the flow cannot contract onto itself per dimension faster than the bound allows and stabilizes the solver. **We show that our EDC flow with an explicit output bounded Jacobian diagonal provides effective and sample efficient RL finetuning performance**, and the bound helps ODE integration remain within stiffness bounds. We also show that this parameterization can be sampled with a split-ODE solver and closed-form integration to align the discrete and continuous density computations on the flow. For this split closed-form integration, split-EDC, the density is exact for the executed discrete policy.

We call the method Change-of-Variables Exact PPO (CoVE-PPO) because its objective targets the exact marginal density of the executed action, rather than a surrogate such as the flow-matching ELBO or the density of an augmented variable. Our computation of that density is approximate only numerically: the trace estimator is unbiased, and ODE discretization error vanishes as solver steps increase, whereas a surrogate's variational gap does not shrink with compute. We evaluate on Robomimic, Robomimic-Pixel, and MimicGen. We show competitive performance in all environments with fewer samples.

2 Related Work

On-Policy RL for Diffusion/Flow Policies: Likelihood-ratio methods such as PPO (Schulman et al., 2017) require the log-density of executed actions under the current policy. For flow policies this density is given by the instantaneous change of variables (Chen et al., 2018; Grathwohl et al., 2018; Kidger, 2022), but several prior works identify it and set it aside, citing trace-estimation cost or noise, discretization error, or numerical instability (Pfrommer et al., 2025; Ding et al., 2026; Zhang et al., 2025; Yang et al., 2026). On-policy methods instead obtain the ratio in one of four ways. The first treats the denoising or integration steps as additional steps of the MDP. DPPO (Ren et al., 2024) optimizes the Gaussian likelihood of each denoising transition, while ReinFlow, $\pi_{\texttt{RL}}$, and DMPO (Zhang et al., 2025; Chen et al., 2025; Zou et al., 2026) inject noise into a deterministic flow to obtain such a chain. The resulting ratio is taken over the denoising path rather than the action, the effective horizon grows with the number of denoising steps, and in practice DPPO fine-tunes only the final few steps. The second replaces the likelihood ratio with a surrogate. FPO (McAllister et al., 2025) forms the ratio from conditional flow-matching losses, motivated by their relation to an ELBO; because the variational gaps of the old and new policies need not cancel, the surrogate ratio is biased (Hu et al., 2026), and FPO++ (Yi et al., 2026) and follow-up work (Wang et al., 2026a) add stabilization mechanisms on top of it. The third approximates the ratio without evaluating the density along the full flow: PolicyFlow (Yang et al., 2026) adds Gaussian noise at the flow endpoint and estimates the ratio from velocity-field variations. The fourth computes an exact density, but over an augmented variable under a volume-preserving map. GenPO (Ding et al., 2026) couples two copies of the action through alternating additive updates and fixed mixing steps and executes their average, while GenPO++ (Hu et al., 2026) pairs each flow state with its previous solver state. In both, the

Jacobian determinant of the generative map is a fixed constant, so the PPO ratio reduces to a ratio of base densities at the inverted latent, and the entropy of the augmented variable cannot change during training. CoVE-PPO instead optimizes the change-of-variables density over the executed action itself.

Off-Policy RL for Diffusion/Flow Policies: A separate line trains flow and diffusion policies against a learned critic, through Q-gradient guidance, Q-weighted or score-matching objectives, or adjoint-based fine-tuning (Wang et al., 2023; Hansen-Estruch et al., 2023; Psenka et al., 2024; Yang et al., 2023; Chen et al., 2024; Li & Levine, 2026; Dong et al., 2026; Patil et al., 2026; Dodeja et al., 2026; Wang et al., 2026b). RLDT (Lei et al., 2026) optimizes a maximum-entropy objective in action space and aligns the flow using intermediate target estimation and Stein variational gradient descent. These methods can be more sample efficient than on-policy alternatives, but they depend on critic accuracy, need mechanisms that stop the actor from exploiting value overestimation on out-of-distribution actions, introduce additional hyperparameters, and can be computationally inefficient. We include RLDT as a representative off-policy baseline.

Computing flow likelihoods: Normalizing flows are often trained by maximum likelihood, which requires the log-density of the model at every training step. Coupling and autoregressive flows are carefully designed to make log-density computations efficient, updating coordinates in masked blocks so that the Jacobian is triangular and its log-determinant reduces to a sum over the diagonal (Dinh et al., 2015; 2017; Papamakarios et al., 2017; Kingma et al., 2017). Continuous normalizing flows (Chen et al., 2018) need only the divergence of the velocity field, the Jacobian trace, which imposes no structural requirement on the network. The trace is also a linear function of the Jacobian, so it admits an unbiased estimator where the determinant does not, dropping the per-step cost to one matrix-vector product per probe or per dimension (Skilling, 1989; Hutchinson, 1989; Grathwohl et al., 2018). While the variance introduced in the density calculation is acceptable for some applications (Song et al., 2021), the estimator's variance can become a binding constraint. Hollow networks route each coordinate's velocity around its own input, splitting the Jacobian into an explicit diagonal and a dense hollow remainder so that the exact trace costs one backward pass instead of d (Chen & Duvenaud, 2019); the construction has since been scaled to graph and attention architectures (Gloy & Olsson, 2025). Verlet flows instead make the discrete map exactly invertible, composing substeps on an augmented state space inspired by symplectic integrators and generalizing coupling architectures to continuous time (Erives et al., 2024). Our EDC flow is similarly a diagonal-plus-hollow decomposition, however it forms the diagonal as a network output rather than recovering it by differentiation, which keeps the density's gradient first-order and allows the diagonal to be constrained directly (Section 3.5) which helps to give stable reverse integration; our split-EDC shares Verlet's substep construction but partitions the action coordinates already present instead of augmenting the state, so the density remains over the executed action (Appendix D), and we connect the discrete and continuous objectives. Forward solves have been analyzed in some works, including for accuracy with offline integration node placement (Gupta et al., 2026), and for stable fast sampling (Tan et al., 2025), though not through the lens of reverse integration for density computation.

3 METHOD

3.1 PROBLEM FORMULATION

We model the problem as a Markov decision process, defined as the tuple $(\mathcal{S}, \mathcal{A}, r, \gamma, p_s)$. At each timestep, the agent takes an action a and receives a reward $r(\cdot)$ from the resultant state $p_s(s'|s, a)$, $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Each future reward is discounted by γ^t. For manipulation environments like those in Robomimic, typically the reward is sparse, meaning a positive reward only on task completion.

3.2 EXACT CHANGE-OF-VARIABLES PPO ON FLOW POLICIES

The PPO objective (Schulman et al., 2017) requires a ratio of probabilities between old and new policies, with the log probability typically calculated via the exact log probability of the gaussian distribution used for the policy.

$$J = \min(r_\theta A, \text{clip}\{r_\theta, 1-\epsilon, 1+\epsilon\}A) \tag{1}$$

$$r_\theta = \exp(\log \pi_\theta(a|s) - \log \pi_{\theta_{\text{old}}}(a|s)) \tag{2}$$

However, modern imitation learning typically models policies as continuous normalizing flows (Chen et al., 2018). Continuous normalizing flows model probability density as an ODE parameterized by a network trained to predict velocity. Recent success has been found by modeling the straight line probability path (Lipman et al., 2023; Liu et al., 2022; Albergo & Vanden-Eijnden, 2023).

$$\mathcal{L} = ||v_\theta(z_t, t, s) \ - (a - z_0)||^2 \qquad z_0 \sim \mathcal{N}(0; \mathbf{I}) \tag{3}$$

Where $z_t = ta + (1-t)z_0$, and t is sampled from some distribution, usually the uniform distribution. Actions are then generated by integrating this neural ODE forward in time. We notate the flow map from noise to actions as $a = \Phi_\theta(z_0)$:

$$a = z_0 + \int_0^1 v_\theta(z_t, t, s)\ dt \qquad z_0 \sim \mathcal{N}(0; \mathbf{I}). \tag{4}$$

The density of flow policies is then computable through the instantaneous change of variables, or CoV, formula $\frac{\partial \log p(z_t)}{\partial t} = -\operatorname{tr}\left(\frac{\partial v_\theta}{\partial z}\right)$ (Chen et al., 2018; Grathwohl et al., 2018). It describes the instantaneous change in density for the flow ODE, and can be integrated across the flow's time to compute changes in the flow's density. Using that along with the known log probability of the prior distribution and the prior sample computed with the reverse solve we get the log density of the flow,

$$\log \pi_\theta(a \mid s) \ = \ \log p(z_0) - \int_0^1 \operatorname{tr}\left(\frac{\partial v_\theta}{\partial z}\right) dt \qquad z_0 = \Phi_\theta^{-1}(a). \tag{5}$$

Past work that trained flows from scratch using this density used adaptive higher order solvers, like Dormand–Prince or Runge–Kutta–Fehlberg. This leads to accurate density estimation, but many evaluations of the velocity network, or a high number of function evaluations (NFEs). In recent practice, generation following the path of the flow ODE is discretized. Because of the straight flow path, a few-step Euler integration focusing on speed of execution rather than the accuracy of the underlying ODE is used. The density of this few-step integration is similarly defined through change of variables on the flow,

$$\log \pi_\theta^K(a \mid s) = \log p(z_0) - \sum_{k=0}^{K-1} \log \left| \det \tfrac{\partial T_k}{\partial z}(z_k) \right|, \qquad a = T_{K-1} \circ \cdots \circ T_0(z_0), \tag{6}$$

where T_k is the solver's one-step map induced by the numerical ODE solver. For Euler integration, we have $T_k(z) = z + h_k v_\theta(z, t_k, s)$ and $\frac{\partial T_k}{\partial z} = I + h_k \frac{\partial v_\theta}{\partial z}$ where h_k is the step size. This density is exact with respect to the discretized flow, however it is expensive to compute, requiring computation of the full Jacobian determinant and implicit integration per-step to solve exactly. We focus on the continuous time CoV formula, but note that with a split solver, closed-form per-step solution, and block-triangular Jacobian the discrete few step density can be made to agree with the continuous one.

Obtaining the exact trace of the d-dimensional flow Jacobian requires evaluating d forward mode Jacobian vector products (JVPs), each requiring a separate differentiation pass. To avoid this cost, past work used the Hutchinson trace estimator (Skilling, 1989; Hutchinson, 1989), which uses a double product of the Jacobian and a noise vector to estimate the trace. If $\varepsilon \in \mathbb{R}^D$ is a random vector satisfying $\mathbb{E}[\varepsilon\varepsilon^\top] = I$ then for the matrix A,

$$\mathbb{E}[\varepsilon^\top A \varepsilon] = \operatorname{Tr}(A). \tag{7}$$

This typically takes many samples to get an accurate trace estimate, and introduces additional variance into the PPO objective, biasing the exponentiated ratio upward. This worry motivated past work to avoid the exact objective (Yang et al., 2026; Zhang et al., 2025). In this work, we use random

vectors sampled from the Rademacher distribution which never has higher variance than Gaussian probes (Avron & Toledo, 2011). To reduce this source of error, we note that PPO optimizes the density ratio and not exactly the density. We use shared probes between each old and new trace estimate within the PPO ratio. Since $\log \pi_{\text{old}}(a|s)$ is constant across PPO epochs, we reuse probes used in each action's old log prob estimate for each new estimate per-epoch. For each random vector, where ε_k^p, is the pth random vector for the estimation at the kth step with P total probes, we hold $\varepsilon_{k,\text{old}}^p = \varepsilon_k^p$. This is in contrast to Grathwohl et al. (2018), where probes are shared within a solve $\varepsilon_0^p = \varepsilon_k^p$.

This choice is crucial to training: the ratio's estimator noise then scales with the change in the Jacobian rather than the Jacobian itself (Section 3.3), so a single probe suffices (Section 4.2).

To prevent the sensitivity noted in Yang et al. (2026), we analyze network stiffness in Section 3.4.

3.3 Trace Estimate Noise Reduction

Any ODE solver approximates the divergence integral through a finite set of intermediate states during each integration step. Let $\mathcal{G}_k$ denote this set of function evaluations for step k, with weights dictated by the solver w_n. Then the step's trace is estimated as $\sum_n w_n \operatorname{tr} J_n$ over the intermediate states n it evaluates. We draw fresh probes ε_k^p for each step k, shared by the intermediate states of that step and by both terms of the ratio. With $J_n^{\text{new}} = \partial_z v_\theta(z_n^{\text{new}}, t_n, s)$ and J_n^{old} defined on the old policy's reverse trajectory from the same action, the estimated log-ratio is

$$\hat{\ell} = \log p(z_0^{\text{new}}) - \log p(z_0^{\text{old}}) - \sum_k \frac{1}{P} \sum_p \varepsilon_k^{p\top} \bar{\Delta}_k \, \varepsilon_k^p, \qquad \bar{\Delta}_k = \sum_{n \in \mathcal{G}_k} w_n \big(J_n^{\text{new}} - J_n^{\text{old}}\big). \tag{8}$$

The prior terms are deterministic. For Rademacher probes, (Appendix F),

$$\operatorname{Var}[\hat{\ell}] = \frac{1}{P} \sum_k V(\bar{\Delta}_k), \qquad V(A) \le 2\Big(\|A\|_F^2 - \textstyle\sum_i A_{ii}^2\Big), \tag{9}$$

where V(A) is the variance of the single-probe estimator $\varepsilon^\top A \varepsilon$, compared with $\frac{1}{P}\sum_k \big[V(\bar{J}_k^{\text{new}}) + V(\bar{J}_k^{\text{old}})\big]$ for independent probes, where $\bar{J}_k = \sum_{n \in \mathcal{G}_k} w_n J_n$. $\bar{\Delta}_k$ is zero at $\theta = \theta_{\text{old}}$ and first order in the policy change, through both the parameters and the shift of the reverse trajectory; the latter is amplified when the flow is stiff (Section 3.4). While $\hat{\ell}$ is unbiased, the ratio is not: $\mathbb{E}[e^{\hat{\ell}}] \approx e^{\ell}\big(1 + \frac{1}{2}\operatorname{Var}[\hat{\ell}]\big)$, so probe sharing reduces the ratio's bias and variance together. The gradient of the new log-density remains an unbiased few-probe estimate whose noise averages over the minibatch.

3.4 Stiffness

When computing the log-probability of an action, we reverse integrate the action and integrate the flow's Jacobian trace. While the probe sharing in the Jacobian trace term is stable, ill-conditioned inverses through the network $z_0 = \Phi_\theta^{-1}(a)$ can produce extreme densities (Behrmann et al., 2021). This is due to stiffness in the underlying flow model ODE. Forward solves are unstable and reverse solves then report correct extreme probability density. This gives extreme negative values for $\log r$ which impairs learning.

This instability is caused by stiffness in the flow matching objective. Under the linear path $z_t = (1-t)z_0 + ta$, the flow-matching target is

$$v^\star(z,t) \;=\; \frac{\hat{a}(z,t) - z}{1-t}, \qquad \hat{a}(z,t) = \mathbb{E}[\, a \mid z_t = z \,], \tag{10}$$

(Albergo & Vanden-Eijnden, 2023; Liu et al., 2022) the residual prediction towards data divided by remaining time. If we approximate a dimension of the behavior distribution as a gaussian with width w, $\mathcal{N}(a; w^2)$ the contraction rate of the flow model in that direction grows as $\frac{1}{1-t}$, and saturates at $\frac{1}{2w}$ within $1 - t \lesssim w$. For many conditional action distributions, this will be near deterministic. Gripper dimensions, for example, are often trained to match discrete values representing open or closed. For these narrow distributions, the optimum becomes stiff.

Typical ODE solvers have stability constraints, where the step size h and maximum velocity Jacobian eigenvalue μ product must be less than a constant, $h\mu < C$. Even if we use a higher-order

solver such as the classic Runge–Kutta method (RK4) (Hairer, 2022), a uniform 5-step grid (20 NFEs) exceeds this constraint in many of our pretrained flows. This varies with training and task, with different action distribution widths and with flows closer to or further from the flow matching optimum.

With the unstable forward sampler, actions can land off of the network's flow map. Then, CoV scoring correctly assigns an extreme flow density through a low $p(z_0)$. More accurate reverse solves don't eliminate this instability, only more accurate forward solves.

If we compute the empirical μ from a trained policy, we can set the solver and step placement to be within the stability constraint and ensure our forward solves are stable. We want to minimize the number of NFEs for training speed, so we can use a timestep schedule, $0 = t_0 < t_1 < \cdots < t_K = 1$, adapted to the stability of the flow, regularize the flow, or truncate the flow. We build a timestep schedule by measuring μ through power iteration across timesteps, states, and base noise samples. We take the 99^{th} percentile of μ at each timestep, and place nodes such that $h\mu \leq \tau < C$ where τ is a max step size. Because the stability measurement isn't perfect, we also apply a clamp in both EDC and general fields $|\log r| > 3$ to the log ratio per-sample, which zeros out any gradient from a ratio that is too far from the old policy in either direction. This handles a small remaining tail of extreme samples (Table 2). Further details in Appendix E.

3.5 PPO on Explicit-Diagonal Coupling Flows

For further speed and stability improvements, we can restrict and re-parameterize the flow to explicitly output a usable flow-velocity Jacobian diagonal, avoiding the need for JVPs entirely. Following affine coupling flows (Dinh et al., 2017; Papamakarios et al., 2017; Dinh et al., 2015), we restrict the velocity so that each dimension's self-dependence enters only through an explicit scale, making the divergence a direct network output. We call this an explicit-diagonal coupling flow, or EDC flow.

For each action dimension, if we restrict the model's input to not depend on the state of the flow at that dimension we can construct a flow with a direct Jacobian diagonal output. In particular we parameterize the flow into a scale j_θ and shift m_θ output from the network N_θ, and partition the input and output dimensions into two blocks $b \in \{A, B\}$. Each block takes as input only the other block's dimensions, giving us an affine velocity in each step (Dinh et al., 2017). This can be done with more than two blocks, but for simplicity we focus on the two-block case.

$$(m_\theta^b, u_\theta^b) = \mathbf{1}_b \odot N_\theta(\mathbf{1}_{\bar{b}} \odot z_t, e_b, t, s) \qquad v_\theta = m_\theta + j_\theta \odot z_t \tag{11}$$

Where $j_\theta = \alpha \tanh(u_\theta)$, e_b is a one-hot vector indicating which block to output, and each block's outputs are summed, $(m_\theta, j_\theta) = \sum_{b\in\{A,B\}}(m_\theta^b, j_\theta^b)$. The density of this flow is faster to compute, requiring no additional backward passes, just the forward passes needed to reverse integrate. Because m_θ^i and j_θ^i are not dependent on z_t^i, the Jacobian diagonal is a simple network output.

$$\operatorname{tr}\left(\frac{\partial v_\theta}{\partial z}\right) = \sum_i \frac{\partial v_\theta^i}{\partial z^i} = \sum_i j_\theta^i \tag{12}$$

$$\log \pi_\theta(a \mid s) = \log p_0(z_0) - \int_0^1 \textstyle\sum_i j_\theta^i(z_t, t, s)\, dt \qquad z_0 = \Phi_\theta^{-1}(a) \tag{13}$$

The bound constrains each term contributing to the divergence of the flow, which prevents individual co-ordinates from contracting faster than α. We find this consistently enables forward and reverse solves that train effectively. In practice, this removed the need for stiffness-graded meshes in EDC, though the bound constrains only the Jacobian diagonal. Because $j \geq -\alpha$, the EDC log-density is bounded above by $\alpha D - (D/2)\log 2\pi$ so the policy cannot collapse to near-determinism, with α trading off this floor against fitting narrow optimal action distributions.

3.6 Invertible Sampling with Split EDC

There is still error introduced by discretization and non-invertibility with EDC flows. Reverse integration is inexact, solving the same ODE from a different initial condition and in a different direction. However, if we allow for a controlled error with respect to the underlying flow, we can unify

the exact log-determinant and trace integral computation. Combining split integration, closed-form integration, and the EDC network, we get a coupling flow whose discrete flow density generated by a set number of steps and the continuous ODE approximation at that same number of steps are computed identically.

$$\log \left| \det \frac{\partial T_k}{\partial z} \right| = \Delta t \left[\frac{1}{2} \sum_i j_{\theta,k}^{A,i} + \sum_i j_{\theta,k}^{B,i} + \frac{1}{2} \sum_i j_{\theta,k}^{A,i\prime} \right] = \int_{t_k}^{t_{k+1}} \operatorname{tr} \left(\frac{\partial \tilde{v}_\theta}{\partial z} \right) dt, \tag{14}$$

Where $\tilde{v}_\theta$ is the split time-frozen field. We call this split-EDC, with details in Appendix D. We show that performance matches EDC showing that the discretization of our continuous objective is not a limiting factor on performance in Figure 8.

4 Experiments

We study the performance of CoVE-PPO in popular RL and manipulation benchmark environments. We compare our performance to other on-policy methods in Section 4.1, we ablate sharing probes across the ratio in Section 4.2, we demonstrate the value of stiffness placed nodes in Section 4.3, and evaluate the time cost of CoVE-PPO in Appendix B. We focus on fine-tuning BC policies on sparse reward tasks.

We also consider environments from Robomimic (Mandlekar et al., 2021) and MimicGen (Mandlekar et al., 2023), focusing on robotic manipulation where flow policies are most popular. We primarily use the noisier Multi-Human (MH) data with 300 demos, but use Proficient-Human (PH) data for tool hang with 200 demos, where Multi-Human is not available. Similar to our baselines, we use an action chunk size of 4 for Can, Lift, Square, and Tool Hang, and 8 for Transport.

For MimicGen tasks we use an identical recipe, with action chunk size of 4. In all manipulation environments, we use a sparse reward of 1 on task completion. For data we use the core D0/D1 files, 1000 machine-generated demos across each task. We evaluate four easy-to-medium MimicGen tasks: Coffee, Nut Assembly, Threading, and Mug Cleanup.

4.1 Sample Efficiency Results

We show our method matches or exceeds baselines introduced to avoid it at matched or higher sample efficiency. For baselines, we select DPPO (Ren et al., 2024), ReinFlow (Zhang et al., 2025), FPO/FPO++ (McAllister et al., 2025; Yi et al., 2026). We also include RLDT (Lei et al., 2026) as an off-policy comparison.

Typical on-policy methods have some stochastic rollout collection mode and a deterministic eval mode. Gaussian and chain-of-gaussian policies can simply evaluate the mode of each gaussian and get the mode of the policy. However, flow based methods like ours and FPO/FPO++ (McAllister et al., 2025; Yi et al., 2026) don't have mid-flow added stochastic noise, and the flow's mode is not necessarily the mean of the base distribution. For our main results, we present the policy actually optimized by RL, with the full random base distribution. We show results with a fixed base distribution in Figure 5, Appendix A, where we outperform on sample efficiency compared to FPO and FPO++. For DPPO and ReinFlow, we use their work's eval setting.

For all Robomimic and MimicGen tasks, we match the BC performance by checkpoint selection, to account for differences between different BC training losses. We match architecture size to DPPO's architecture size, across all arms. FPO and FPO++ are run on our EDC flow parameterization, to ensure fair comparison. Sample efficiency curves are shown in Figures 1 and 2.

We note that FPO in particular is unstable on these environments, with occasional NaN collapses. In all tasks, our method matches or exceeds baselines, at matched or higher sample efficiency.

4.2 Probe Sharing

In this experiment, we demonstrate the effectiveness of shared Hutchinson estimator probes, showing in Figure 3 that shared probes enables stable training at low probe counts comparably to the full exact trace estimator, while single independent probes fail catastrophically.

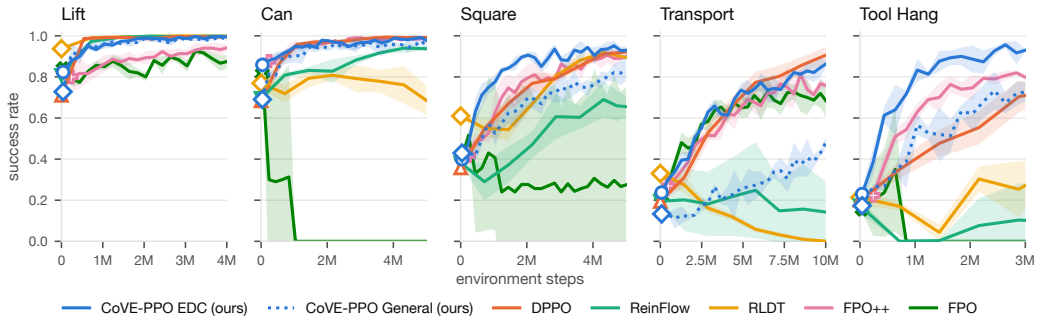


Figure 1: CoVE-PPO comparison results on Robomimic environments with state observation. Curves are a mean of three seeds for each method, with shaded areas the standard deviation.

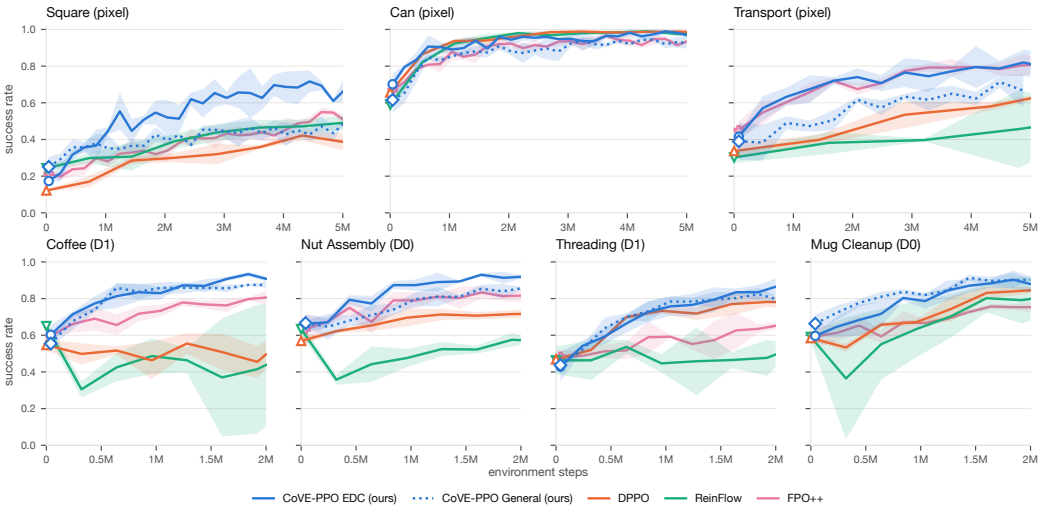


Figure 2: CoVE-PPO comparison results on Robomimic pixel envs (top), and MimicGen state envs (bottom). Curves are a mean of three seeds for each method, with shaded areas the standard deviation.

4.3 Discretization

In this section we study how accurately the ODE must be discretized for effective learning, and show that stiffness placed nodes reduce cost for general CNF fine-tuning.

In Figure 4 we compare forward and reverse solve stiffness. It makes clear that much of the solver instability lies in the forward map. In the stiff flows used for robotic manipulation, high K with a uniform grid is required for effective learning. These results show that with more precise generation, fewer reverse steps are required. With stiffness placed nodes we can reduce NFEs even farther, and gain the same benefit. With 6 RK4 steps, we train comparably to 50 uniformly placed steps.

Further, in Figure 8 we demonstrate results for EDC with our split closed-form solve which matches performance of normal EDC. Because of the exact alignment between the discrete and continuous objective computation, we use this result to show that discretization is not a limiting factor for EDC.

5 Limitations and Future Work

CoVE PPO is limited mainly through memory scalability, and the tradeoff between the standard flow parameterization and EDC parameterization. Reverse integration requires more memory and

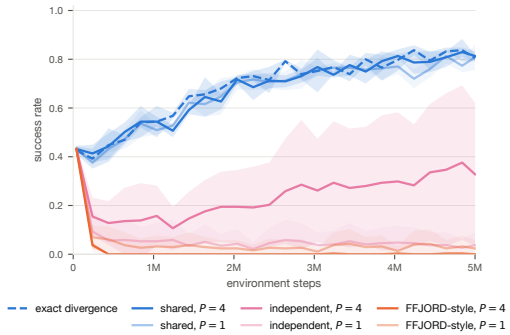


Figure 3: Training using CoVE-PPO objective with Hutchinson estimator probes shared across the PPO ratio, compared against independent draws and FFJORD style sharing. We compare four and single probe runs, and show that exact trace performs comparably. Curves are mean success rates over 3 seeds with shaded standard deviation, on Robomimic Square with state observations.

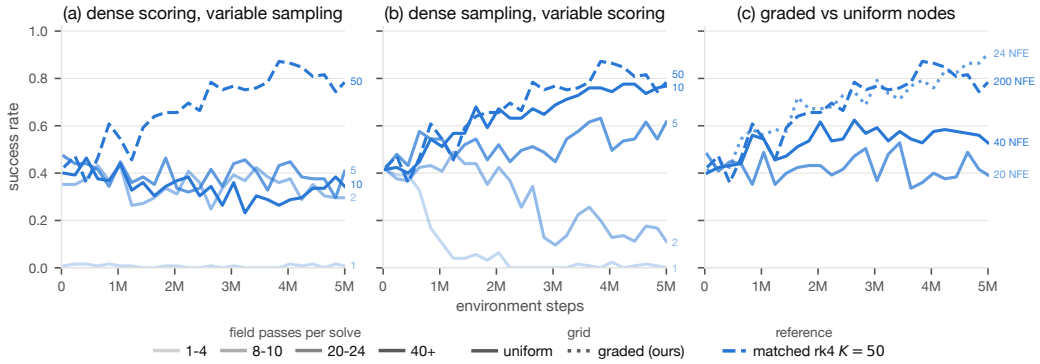


Figure 4: Training using CoVE-PPO objective with varying integration resolution. In (a), we vary the integration resolution used to produce actions, and score with a fixed higher-resolution integration. In (b), we vary the integration resolution used to score actions, and generate actions with high resolution. In (c) we show our graded stiffness mesh compared with uniform node placement. Curves are success rate over 1 seed each, using RK4, on Robomimic Square with state observations.

time than other techniques, and the requirement to accurately integrate the underlying flow can in some cases cost performance. Further, EDC is a non-standard flow model, and restricts within-block interactions. Comparing the expressivity in detail as model training scales, and the RL/BC tradeoff between different flow parameterizations is an interesting direction for future work. This work focuses on manipulation fine-tuning, however exact density computations also enable entropy bonuses and KL regularization.

6 Conclusion

Flow policies define a density over action, and prior work avoids using this density. We find it is a practical density computation, even with a single probe. We show flow stiffness can be mitigated, and that an alternative flow parameterization can provide a faster and more sample efficient method. We show that the CoV density computation is both practical and matches or exceeds on-policy baselines at matched or higher sample efficiency.

AI USE STATEMENT

In this work, we used generative AI tools to design or provide feedback on research methodology and experiments and assist in the writing of proofs. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, or propose or refine hypotheses. Assisting with translation, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, interpreting results are not applicable to this work. Additionally, we used generative AI tools to identify relevant literature. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Upon acceptance, we will release our implementation publicly via a GitHub repository. Training details and hyperparameters in Appendix G.

REFERENCES

Michael Samuel Albergo and Eric Vanden-Eijnden. Building Normalizing Flows with Stochastic Interpolants. In *International Conference on Learning Representations*, 2023. URL https://mlanthology.org/iclr/2023/albergo2023iclr-building/.

Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58:8:1–8:34, 2011. URL https://api.semanticscholar.org/CorpusID:5827717.

Jens Behrmann, Paul Vicol, Kuan-Chieh Wang, Roger Grosse, and Jörn-Henrik Jacobsen. Understanding and Mitigating Exploding Inverses in Invertible Neural Networks, 2021. URL https://arxiv.org/abs/2006.09347.

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. \pi_0: A Vision-Language-Action Flow Model for General Robot Control, 2026. URL https://arxiv.org/abs/2410.24164.

Boyuan Chen, Diego Marti Monso, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion Forcing: Next-token Prediction Meets Full-Sequence Diffusion, 2024. URL https://arxiv.org/abs/2407.01392.

Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, Hao Lin, Hongzhi Zang, Quanlu Zhang, Zhaofei Yu, Guoliang Fan, and others. $\backslashpi_\backslashtexttt {RL} $: Online RL Fine-tuning for Flow-based Vision-Language-Action Models. *arXiv preprint arXiv:2510.25889*, 2025.

Ricky T. Q. Chen and David K Duvenaud. Neural Networks with Cheap Differential Operators. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/770f8e448d07586afbf77bb59f698587-Paper.pdf.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Shutong Ding, Ke Hu, Shan Zhong, Haoyang Luo, Weinan Zhang, Jingya Wang, Jun Wang, and Ye Shi. GenPO: Generative Diffusion Models Meet On-Policy Reinforcement Learning, 2026. URL https://arxiv.org/abs/2505.18763. _eprint: 2505.18763.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation, 2015. URL https://arxiv.org/abs/1410.8516.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. In *International Conference on Learning Representations*. OpenReview.net, 2017. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2017.html#DinhSB17.

Lakshita Dodeja, Ondrej Biza, Shivam Vats, Stephen Hart, Stefanie Tellex, Robin Walters, Karl Schmeckpeper, and Thomas Weng. When Life Gives You BC, Make Q-functions: Extracting Q-values from Behavior Cloning for On-Robot Reinforcement Learning. In *Robotics: Science and Systems (RSS)*, 2026.

Xiaoyi Dong, Xi Sheryl Zhang, and Jian Cheng. Mean Flow Policy Optimization, 2026. URL https://arxiv.org/abs/2604.14698.

Ezra Erives, Bowen Jing, and Tommi Jaakkola. Verlet Flows: Exact-Likelihood Integrators for Flow-Based Generative Models, 2024. URL https://arxiv.org/abs/2405.02805.

Johann Flemming Gloy and Simon Olsson. HollowFlow: Efficient Sample Likelihood Evaluation using Hollow Message Passing, 2025. URL https://arxiv.org/abs/2510.21542.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models, 2018. URL https://arxiv.org/abs/1810.01367. _eprint: 1810.01367.

Aditi Gupta, Soon Hoe Lim, Annan Yu, and N. Benjamin Erichson. Sharpen Your Flow: Sharpness-Aware Sampling for Flow Matching, 2026. URL https://arxiv.org/abs/2605.11547.

Ernst Hairer. *Solving ordinary differential equations II: Stiff and differential-algebraic problems*. 2022.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31 of *Springer Series in Computational Mathematics*. Springer, 2nd edition, 2006.

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, 2023. URL https://arxiv.org/abs/2304.10573.

Marlis Hochbruck and Alexander Ostermann. Exponential Integrators. *Acta Numerica*, 19:209–286, 2010.

Ke Hu, Shutong Ding, Panxin Tao, Jingya Wang, and Ye Shi. GenPO++: Generative Policy Optimization with Jacobian-free Likelihood Ratios, 2026. URL https://arxiv.org/abs/2606.06967. _eprint: 2606.06967.

M. F. Hutchinson. A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines. *Communications in Statistics - Simulation and Computation*, 18(3):1059–1076, 1989. doi: 10.1080/03610918908812806. URL https://doi.org/10.1080/03610918908812806. _eprint: https://doi.org/10.1080/03610918908812806.

Patrick Kidger. On Neural Differential Equations, 2022. URL https://arxiv.org/abs/2202.02435.

Diederik P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving Variational Inference with Inverse Autoregressive Flow, 2017. URL https://arxiv.org/abs/1606.04934.

Boshu Lei, Kostas Daniilidis, and Antonio Loquercio. Reinforcement Learning for Flow-Matching Policies with Density Transport, 2026. URL https://arxiv.org/abs/2606.08602. _eprint: 2606.08602.

Qiyang Li and Sergey Levine. Q-learning with Adjoint Matching. 2026. URL http://arxiv.org/abs/2601.14234.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling, February 2023. URL http://arxiv.org/abs/2210.02747. arXiv:2210.02747 [cs].

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

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.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations. In *7th Annual Conference on Robot Learning*, 2023.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow Matching Policy Gradients, 2025. URL https://arxiv.org/abs/2507.21053. _eprint: 2507.21053.

Robert I. McLachlan and G. Reinout W. Quispel. Splitting Methods. *Acta Numerica*, 11:341–434, 2002.

Avik Pal, Yingbo Ma, Viral Shah, and Christopher V Rackauckas. Opening the Blackbox: Accelerating Neural Differential Equations by Regularizing Internal Solver Heuristics. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8325–8335. PMLR, July 2021. URL https://proceedings.mlr.press/v139/pal21a.html.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/6c1da886822c67822bcf3679d04369fa-Paper.pdf.

Sarvesh Patil, Mitsuhiko Nakamoto, Manan Agarwal, Shashwat Saxena, Jesse Zhang, Giri Anantharaman, Cleah Winston, Chaoyi Pan, Douglas Chen, Nai-Chieh Huang, Zeynep Temel, Oliver Kroemer, Sergey Levine, Abhishek Gupta, Hongkai Dai, Paarth Shah, and Max Simchowitz. OGPO: Sample Efficient Full-Finetuning of Generative Control Policies, 2026. URL https://arxiv.org/abs/2605.03065.

Samuel Pfrommer, Yixiao Huang, and Somayeh Sojoudi. Reinforcement Learning for Flow-Matching Policies, 2025. URL https://arxiv.org/abs/2507.15073.

Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a Diffusion Model Policy from Rewards via Q-Score Matching. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 41163–41182. PMLR, July 2024. URL https://proceedings.mlr.press/v235/psenka24a.html.

Allen Z. Ren, Justin Lidard, Lars L. Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion Policy Policy Optimization. In *arXiv preprint arXiv:2409.00588*, 2024.

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. arXiv: 1707.06347.

John Skilling. The Eigenvalues of Mega-dimensional Matrices. In J. Skilling (ed.), *Maximum Entropy and Bayesian Methods: Cambridge, England, 1988*, pp. 455–466. Springer Netherlands, Dordrecht, 1989. ISBN 978-94-015-7860-8. doi: 10.1007/978-94-015-7860-8_48. URL https://doi.org/10.1007/978-94-015-7860-8_48.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations, 2021. URL https://arxiv.org/abs/2011.13456.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. ISSN 1533-7928. URL http://jmlr.org/papers/v15/srivastava14a.html.

Gilbert Strang. On the Construction and Comparison of Difference Schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, 1968.

Zheng Tan, Weizhen Wang, Andrea L. Bertozzi, and Ernest K. Ryu. STORK: Faster Diffusion And Flow Matching Sampling By Resolving Both Stiffness And Structure-Dependence, 2025. URL https://arxiv.org/abs/2505.24210.

Ge Wang, Yibo Peng, Fan Feng, Shenhao Yan, Chengsi Yao, Jiahao Yang, Honghao Cai, Yiming Zhao, Xi Li, Jinke Ren, Shuguang Cui, Yatong Han, and Zhen Li. ReFPO: Reflow Regularization for Flow Matching Policy Gradients, 2026a. URL https://arxiv.org/abs/2606.21086. _eprint: 2606.21086.

Zhendong Wang, Jonathan J. Hunt, and Mingyuan Zhou. Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning, 2023. URL https://arxiv.org/abs/2208.06193. _eprint: 2208.06193.

Ziqian Wang, Yitian Liu, Xingjian Mao, Minqian Wang, and Yao Mu. Q-VGM: Q-Value-Gradient Matching for Off-Policy Reinforcement Learning of Flow-Matching VLA, 2026b. URL https://arxiv.org/abs/2606.08015.

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, 2023. URL https://arxiv.org/abs/2305.13122.

Shunpeng Yang, Ben Liu, and Hua Chen. PolicyFlow: Policy Optimization with Continuous Normalizing Flow in Reinforcement Learning, 2026. URL https://arxiv.org/abs/2602.01156. _eprint: 2602.01156.

Brent Yi, Hongsuk Choi, Himanshu Gaurav Singh, Xiaoyu Huang, Takara E. Truong, Carmelo Sferrazza, Yi Ma, Rocky Duan, Pieter Abbeel, Guanya Shi, Karen Liu, and Angjoo Kanazawa. Flow Policy Gradients for Robot Control. *arXiv preprint arXiv:2602.02481*, 2026.

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. _eprint: 2505.22094.

Guowei Zou, Haitao Wang, Hejun Wu, Yukun Qian, Yuhang Wang, and Weibing Li. One Step Is Enough: Dispersive MeanFlow Policy Optimization, 2026. URL https://arxiv.org/abs/2601.20701. _eprint: 2601.20701.

A Base Distribution Noise

For a small environment selection, we show performance of our method and FPO++ with a base noise of 0. We note that this policy evaluation mode starts far higher on transport, and reaches higher success rates faster for both methods. However, on tool hang, it starts much lower.

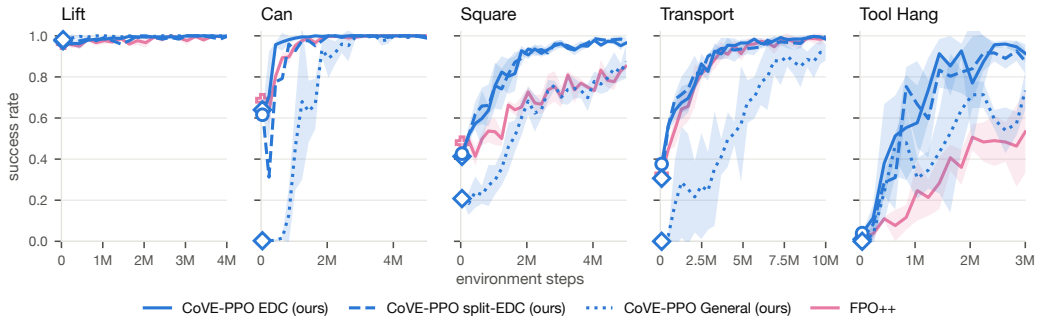


Figure 5: CoVE-PPO compared against FPO++ when using zero as the base noise draw, instead of the full Gaussian draw. EDC uses 20 NFEs with 5 step RK4, and split-EDC uses k=7 giving 21 NFEs. Curves are means 3 seeds with the shaded area of standard deviation.

B Computational/Time Cost

B.1 Computational Cost

When training primarily in fast simulations, the computation time cost of updates can dominate. However, in many situations the environment step cost is the limiting factor. For many manipulation environments, the update time cost adds only marginally to the speed of training. In wall clock time, this is less expensive than DPPO's official implementation, but slower than ReinFlow.

With EDC, we carry no additional trace computation burden because of the diagonal output, and the only additional cost above chain based methods is additional forward passes on the network proportional to the number of integration steps. EDC requires 2 net evaluations per coupled flow step, but because they are not split coupled flows they can be run in parallel. We show that in both cases, this cost is not prohibitive for training, and only marginally adds to the speed of training. For the coupling flow, each full step under Strang solving requires three NFEs. To match cost, we reduce the number of full steps evaluated. The same number of steps again is required for the inverse.

Here we show the time cost of each method on Robomimic square. We ran 10 iterations at each method's tuned hyperparameters, and calculated the rollout and policy update cost in seconds.

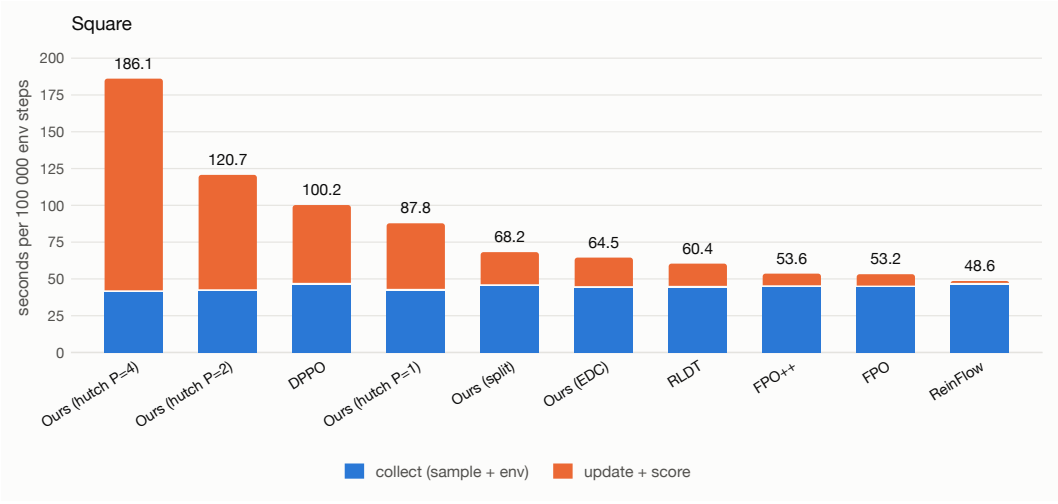


Figure 6: Time cost per 100k steps of each method, on Robomimic square. Measured running on a single machine on an RTX 4090. Using matched network sizes. Batch sizes differ. We measure our method with a varying number of probes, P.

C Flow Parameterization Ablation

In this section, we ablate FPO and FPO++'s choice of base policy. Because FPO and FPO++'s surrogate objective uses the flow matching loss, it is compatible with both standard CNFs and our EDC parameterization. In our main figures, FPO and FPO++ use the EDC flow. We show FPO++ and FPO on both our EDC base flow, and on a normal flow model and in Figure 7 we show our method outperforms in both cases. For our method, we change only how the trace is obtained, as EDC provides the trace without estimation. In both cases we use the same sampler, RK4 with the stiffness graded mesh for the general field, and RK4 with a uniform 5-step mesh for EDC.

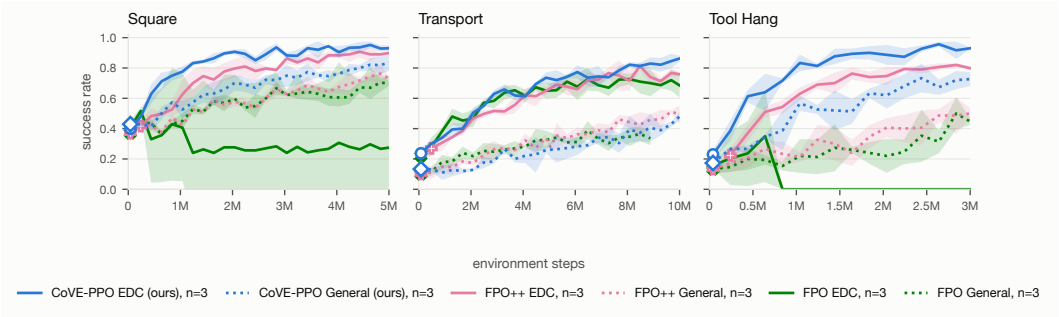


Figure 7: Success rate contrasting the FPO, FPO++ and our objective across the EDC and general flow base policies. Curves are mean of 3 seeds with shaded area as the standard deviation.

D Split EDC

Following Strang (1968); McLachlan & Quispel (2002); Hairer et al. (2006), we can use a Strang split sampling procedure which is second order accurate to the underlying ODE. Instead of both networks using the current flow state and time as input, we split the step into reversible substeps, $\psi^b_\tau(z_t, e_b, t, s)$, where τ is the time variable for the substep, and $T_k(z_t, t, s)$ is the full composed step.

First we note that each flow substep is affine with respect to its block, so we can use the exact ODE solution to the substeps's ODE. For each substep we have a closed form solution to $v^b_\theta = m^b_\theta + j^b_\theta \odot z^b_t$ (Hochbruck & Ostermann, 2010).

$$\psi^b_\tau(z) \;=\; e^{\tau j^b_\theta} \odot z \;+\; \tau\,\varphi\big(\tau j^b_\theta\big) \odot m^b_\theta, \qquad \varphi(u) = \frac{e^u - 1}{u}, \quad \varphi(0) = 1 \tag{15}$$

This sub-step is composed with other sub-steps to create a full integration step T_k, through Strang splitting. The first A substep updates z^A using the starting value of z^B, the B substep then updates z^B using the updated value of z^A, and the final A substep updates z^A again. Full steps are composed to integrate and generate actions. Let $\Delta t = 1/K$, $t_k = k\Delta t$, and we evaluate all three sub-steps' networks at the step midpoint $t_{k+1/2} = t_k + \frac{\Delta t}{2}$. Thus,

$$T_k \;=\; \psi^A_{\Delta t/2} \circ \psi^B_{\Delta t} \circ \psi^A_{\Delta t/2}, \qquad a \;=\; T_{K-1} \circ \cdots \circ T_0(z_0), \quad z_0 \sim \mathcal{N}(0, \mathbf{I}). \tag{16}$$

Strang splitting in particular gives us second order accuracy to the underlying flow, and split solving gives us exact invertibility. A sub-step does not alter the coordinates it uses as input, so re-evaluating N_θ after the sub-step returns the same m^b_θ and j^b_θ. Therefore $(\psi^b_\tau)^{-1} = \psi^b_{-\tau}$ exactly. Reversing the network then is simply the same composition in reverse.

$$(\psi^b_\tau)^{-1}(z) = \psi^b_{-\tau}(z) = e^{-\tau j^b_\theta} \odot z - \tau\,\varphi(-\tau j^b_\theta) \odot m^b_\theta \tag{17}$$

For the split solver we need to compute the density of the actions, either through the discrete map, or the continuous trace change-of-variables. Coupling network layers has been used previously

(Dinh et al., 2017) for block-triangular Jacobians, which makes the log-determinant of the Jacobian equal to the sum of log-diagonal entries. With closed-form integration at each substep, the sum of log-diagonal entries of the discrete map and the continuous Jacobian trace are identical.

$$\frac{\partial \psi_\tau^A}{\partial z} = \begin{pmatrix} \mathrm{diag}\left(e^{\tau j_\theta^A}\right) & * \\ 0 & I \end{pmatrix}, \qquad * = \frac{\partial}{\partial z^B}\left[e^{\tau j_\theta^A} \odot z^A + \tau \varphi(\tau j_\theta^A) \odot m_\theta^A\right] \tag{18}$$

$$\log\left|\det \tfrac{\partial \psi_\tau^b}{\partial z}\right| \;=\; \log\left(\prod_i e^{\tau j_\theta^b}\right) \;=\; \tau \sum_i j_\theta^{b,i} \tag{19}$$

In contrast, Euler steps give $\sum_i \log(1 + \tau j_\theta^i)$ for the discrete map. Further the flow's density is multiplicative by the chain rule, since each step is composed by function composition. Importantly, this unifies the discrete map and continuous ODE perspectives on computing the log density. Both the discrete flow generated by a fixed number of flow steps, and the continuous ODE approximation at that same fixed number of steps are computed identically.

$$\log\left|\det \tfrac{\partial \psi_\tau^b}{\partial z}\right| \;=\; \tau \sum_i j_\theta^{b,i} \qquad \log\left|\det \frac{\partial T_k}{\partial z}\right| = \Delta t\left[\tfrac{1}{2}\textstyle\sum_i j_\theta^{A,1} \;+\; \sum_i j_\theta^B \;+\; \tfrac{1}{2}\sum_i j_\theta^{A,2}\right] \tag{20}$$

The full exact log density of the actions under split sampling is computed similarly to a standard continuous normalizing flow, a prior and trace accumulation.

$$\log \pi_\theta^K(a \mid s) \;=\; \log p_0(z_0) \;-\; \Delta t \sum_{k=0}^{K-1}\left[\tfrac{1}{2}\textstyle\sum_i j_{\theta,k}^{A,i} \;+\; \sum_i j_{\theta,k}^{B,i} \;+\; \tfrac{1}{2}\sum_i j_{\theta,k}^{A,i\prime}\right], \qquad z_0 = \Phi_\theta^{-1}(a), \tag{21}$$

where $j_{\theta,k}^A$ and $j_{\theta,k}^{A\prime}$ denote the first and third sub-steps of step k. Split solving increases the NFEs, since for each Strang step it requires 3 evaluations. For fair comparison, we reduce K in our experiments. The gradient is efficient to compute, since each term is a forward output. This structure is equivalent to a RealNVP affine coupling layer (Dinh et al., 2017), but allows for flow matching training.

In terms of the stiffness general fields deal with, for split-EDC the scale factor $e^{\tau j_\theta^b}$ is positive for all real arguments, so Eq. equation 16 is a bijection unconditionally, at any Δt. The linear stability constraint above does not apply to scoring, only to the fidelity of the continuous ODE. Fidelity to the underlying ODE is often not required for task success, as few-step integration of flow matching policies has already shown.

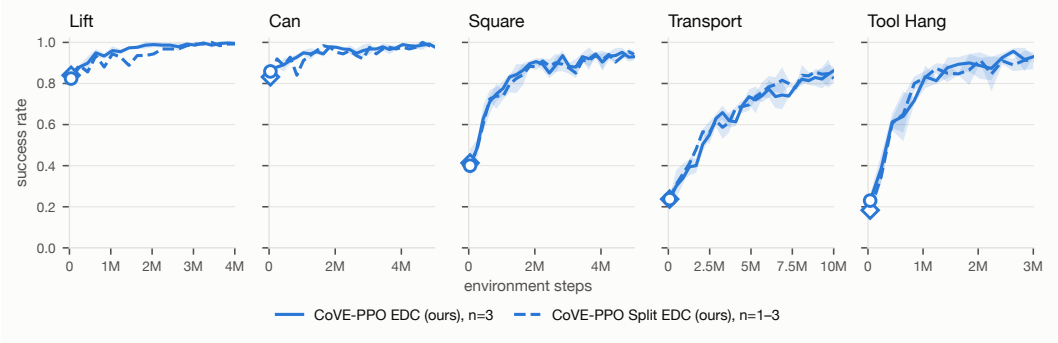


Figure 8: CoVE-PPO split solver v.s. standard EDC comparison results on Robomimic environments with state observation. Curves are a mean of three seeds for each method, with shaded areas the standard deviation.

E STIFFNESS MESH DETAILS

In this section, we detail the construction of our stiffness placed integration nodes. As we described above, typical ODE solvers have stability constraints, where the step size h and maximum velocity Jacobian eigenvalue μ product must be less than a constant, $h\mu < C$.

We take a random sample of states and initial noises. We estimate the pretrained policy's $\mu(t)$, the 99th percentile over states and initial noises of the spectral radius of $\partial v_\theta/\partial z$ along the policy's sampling trajectories, by power iteration on Jacobian-vector products. We then place nodes greedily from the data end, taking at each step the largest h_k with $h_k \max_{[t_k,t_{k+1}]} \mu \le \tau < C$. The number of steps K is an output of this procedure.

This set of timesteps then is within the solver's linear stability constraint, and can reliably reverse integrate. Because the stability measurement isn't perfect, we also apply a clamp to the log ratio per-sample, which zeros out any gradient from a ratio that is too far from the old policy in either direction. Below we report measured stiffness from our pretrained policies. These meshes concentrate nodes near the data-end of the flow. We use $\tau = 2$, to ensure we stay below RK4's limit of 2.785 (Hairer, 2022). For MimicGen the greedy mesh was placed with $\tau = 2.0$ on Threading and reused on the other tasks.

Table 1: Maximum $h_k\mu_{0.99}$ over the uniform and greedy mesh both at $K = 5$ under RK4, measured on the pretrained policies.

Mesh	Threading	Coffee	Nut Assembly	Mug Cleanup
Uniform	13.76	12.89	13.56	16.23
Greedy (ours)	2.00	2.11	2.04	2.36

It is also possible to regularize during flow matching training to reduce stiffness (Pal et al., 2021). EDC flow imposes a bound that accomplishes a similar goal, with additional speed benefits. For our general flows, we find that the gradient can be sensitive in parameter space for some MLP flows, and we train our BC policies with dropout (Srivastava et al., 2014), set at 0.2, and it is not active during RL.

We report the fraction of per-epoch sample updates masked by the log-ratio mask in Table 2.

Table 2: Fraction of sample-updates masked by the log-ratio mask accumulated over training on MimicGen. The denominator counts each sample once per PPO epoch, so this is the fraction of per-epoch sample updates masked. Range reported over 3 seeds.

Method	Threading	Coffee	Nut Assembly	Mug Cleanup
EDC	0.62–0.75%	1.72–2.67%	0.68–0.77%	0.20–0.38%
General field, $P = 1$	1.67–2.31%	0.26–0.38%	1.59%	0.05–0.08%

E.1 STIFFNESS OF THE FLOW-MATCHING OPTIMUM

Consider one dimension with $z_0 \sim \mathcal{N}(0, 1)$, $a \sim \mathcal{N}(\bar{a}, w^2)$ independent, and $z_t = (1-t)z_0 + ta$. Then z_t is Gaussian with variance $\sigma_t^2 = (1-t)^2 + t^2w^2$, and $\hat{a}(z,t) = \mathbb{E}[a \mid z_t = z] = \bar{a} + tw^2(z - t\bar{a})/\sigma_t^2$. Differentiating Eq. equation 10,

$$\lambda(t) \;=\; \frac{\partial v^\star}{\partial z} \;=\; \frac{1}{1-t}\Big(\frac{tw^2}{\sigma_t^2} - 1\Big) \;=\; \frac{tw^2 - (1-t)}{(1-t)^2 + t^2w^2}. \tag{22}$$

With $s = 1 - t$ and $w \ll 1$: for $s \gg w$, $\lambda \approx -1/s$; the minimum satisfies $s^2 - 2w^2s - w^2 \approx 0$, i.e. $s^\star \approx w$, where $\lambda \approx -1/(2w)$; and $\lambda \to +1$ as $s \to 0$. As a check, $\int_0^1 \lambda\, dt = \log(\sigma_1/\sigma_0) = \log w$, the expected volume change. For Gaussian data with covariance $U\,\mathrm{diag}(w_i^2)U^\top$ the optimal Jacobian is symmetric and diagonal in U with eigenvalues $\lambda(t; w_i)$, so the stiffness is set by the narrowest direction. Real action distributions are not Gaussian; this model describes local width, and multi-modality can add further stiffness at intermediate t.

RK4 is stable on the negative real axis for $h|\lambda| \leq 2.785$ (Hairer, 2022). At the peak this requires $h \lesssim 5.6\,w$, so a uniform grid needs $K \gtrsim 1/(5.6\,w)$ steps; for $w = 0.01$, $K \gtrsim 18$ RK4 steps. Because the constraint binds only near $1 - t \approx w$, a graded mesh can take large steps elsewhere.

The continuous inverse flow scales the narrow direction by $\sigma_0/\sigma_1 = 1/w$. A forward error δ in that direction therefore maps to a base-space error $\approx \delta/w$, and $\log p(z_0)$ drops by $\approx \delta^2/(2w^2)$. The reverse solve reports this extreme value correctly according to the underlying ODE; improving the reverse solve cannot remove it, only improving the forward solve can.

F Probe Sharing Variance

Let $\varepsilon \in \{\pm 1\}^D$ be Rademacher and $S = (A + A^\top)/2$. Since $\varepsilon^\top A\varepsilon = \varepsilon^\top S\varepsilon$ and $\varepsilon_i^2 = 1$,

$$\varepsilon^\top A\varepsilon = \operatorname{tr}(A) + 2\sum_{i<j} S_{ij}\,\varepsilon_i\varepsilon_j. \tag{23}$$

The products $\varepsilon_i\varepsilon_j$ $(i < j)$ have zero mean, unit variance, and are pairwise uncorrelated, so the estimator is unbiased with variance $V(A) = 4\sum_{i<j} S_{ij}^2 = 2\sum_{i\neq j} S_{ij}^2$. Since $S_{ij}^2 \leq (A_{ij}^2 + A_{ji}^2)/2$, $V(A) \leq 2\sum_{i\neq j} A_{ij}^2 = 2(\|A\|_F^2 - \sum_i A_{ii}^2)$ (Avron & Toledo, 2011). Averaging P independent probes divides the variance by P, and probes are independent across steps, so step variances add. Within a step, the quadrature is linear in the Jacobians, so the shared-probe estimates of the new and old terms combine into the single quadratic form $\varepsilon_k^{p\top}\bar{\Delta}_k\varepsilon_k^p$, giving $V(\bar{\Delta}_k)/P$. With independent probes the two terms are independent and their variances add.

The Jacobian difference at a node decomposes as

$$J_n^{\text{new}} - J_n^{\text{old}} = \big[\partial_z v_\theta - \partial_z v_{\theta_{\text{old}}}\big](z_n^{\text{new}}) + \big[\partial_z v_{\theta_{\text{old}}}(z_n^{\text{new}}) - \partial_z v_{\theta_{\text{old}}}(z_n^{\text{old}})\big], \tag{24}$$

a parameter term and a trajectory term. Both vanish at $\theta = \theta_{\text{old}}$, since both reverse solves then follow the same path from a.

For the ratio, we have $\hat{\ell} = \ell + \eta$ with $\mathbb{E}\eta = 0$ and $\operatorname{Var}\eta = \sigma^2$. Then $\mathbb{E}[e^{\hat{\ell}}] = e^\ell\,\mathbb{E}[e^\eta] \geq e^\ell$ by Jensen, and a second-order expansion gives $\mathbb{E}[e^\eta] = 1 + \sigma^2/2 + O(\mathbb{E}|\eta|^3)$.

G Hyperparameters

Table 3 lists the RL recipe shared by all CoVE-PPO runs (Robomimic state, Robomimic pixel, and MimicGen). Table 4 lists the policy parameterization and solver settings.

We note that we re-use probes across epochs, using the same probes and log-density calculation for the old policy per-epoch. In our testing, re-sampling had a negligible impact on performance.

Table 3: Shared CoVE-PPO RL hyperparameters for Robomimic (state and pixel) and MimicGen.

Rollouts	
Parallel environments	25
Chunks per env. per iteration	400 (10,000 chunk transitions per iteration)
Env. steps per iteration	40,000; 80,000 for Transport (chunk length 8)
Observation history	1 step
Action bounds	Executed action clamped to $[-1, 1]$; the unclamped flow output is stored and scored
Returns and advantages	
Discount γ	0.999
GAE λ	0.95
Reward scaling	Running reward scaler
Advantage normalization	Per minibatch
PPO	
Clip ϵ	0.5
Epochs per iteration	20
Minibatch size	1,000 chunks
Dual clip	$c = 5.0$, applied to $\log r$ before exponentiation
Log-ratio mask	samples with $\lvert \log r \rvert > 3$ receive zero gradient
Value loss coefficient	0.5
Entropy coefficient	0
Optimization	
Actor learning rate	4×10^{-6} (Adam, constant)
Critic learning rate	1×10^{-3} (constant)
Gradient-norm clip	1.0
Critic warm-up	2 iterations with actor frozen
Critic	
Architecture	Residual MLP, 3×256, Mish, single value head
Evaluation	
Frequency	Every 5 iterations
Episodes	100 (Robomimic state and pixel); 64 (MimicGen)
Evaluation	Stochastic, $z_0 \sim \mathcal{N}(0, \mathbf{I})$; $z_0 = 0$ results in Appendix A

Table 4: Policy parameterization and solver settings. One field evaluation of the EDC velocity requires two trunk passes, one per block, which are batched; a split-EDC substep requires one.

EDC velocity field	
Blocks	2
α	4.0; 8.0 for Can
Block partition	Fixed partition, $\{0, \ldots, \lfloor d/2 \rfloor - 1\}$ and $\{\lfloor d/2 \rfloor, \ldots, d - 1\}$
Activation	GELU
Output initialization	Final layer zero-initialized, so $v \equiv 0$ and the flow starts at $\mathcal{N}(0, \mathbf{I})$ before BC
Time embedding	Sinusoidal $\to$ Linear $\to$ Mish $\to$ Linear
State encoder	MLP $[512, 64]$ (optional)
Solvers	
EDC	RK4, $K = 5$ uniform steps, identical for sampling and scoring; 20 field evaluations per sample and per log-probability
Split-EDC	Strang splitting, $K = 7$; 21 single-block trunk passes
General velocity field	
Solver	RK4 on the greedy stiffness mesh, $\tau = 2.0$ (Appendix E)
Trace estimator	Rademacher probes shared across the PPO ratio
P	1
BC regularization	Dropout 0.2 during BC; disabled during RL