Streaming Flow Policy

Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories

Website: https://streaming-flow-policy.github.io

Sunshine Jiang, **Xiaolin Fang**, **Nicholas Roy**,
Tomás Lozano-Pérez, **Leslie Kaelbling**, **Siddharth Ancha**
Massachusetts Institute of Technology

Abstract: Recent advances in diffusion/flow-matching policies have enabled imitation learning of complex, multi-modal action trajectories. However, they are computationally expensive because they sample a *trajectory of trajectories*—a diffusion/flow trajectory of action trajectories. They discard intermediate action trajectories, and must wait for the sampling process to complete before any actions can be executed on the robot. We simplify diffusion/flow policies by *treating action trajectories as flow trajectories*. Instead of starting from pure noise, our algorithm samples from a narrow Gaussian around the last action. Then, it incrementally integrates a velocity field learned via flow matching to produce a sequence of actions that constitute a *single* trajectory. This enables actions to be streamed to the robot on-the-fly *during* the flow sampling process, and is well-suited for receding horizon policy execution. Despite streaming, our method retains the ability to model multi-modal behavior. We train flows that *stabilize* around demonstration trajectories to reduce distribution shift and improve imitation learning performance. Streaming flow policy outperforms prior methods while enabling faster policy execution and tighter sensorimotor loops for learning-based robot control.

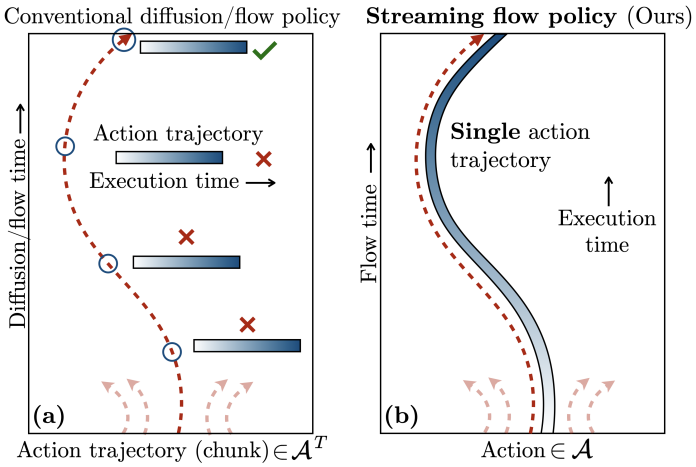

Figure 1: **(a)** Diffusion policy [1] and flow-matching policy [2] input a history of observations (not shown) to predict a "chunk" of future robot actions. The x-axis represents the action space, and the $+y$-axis represents increasing diffusion/flow timesteps. Conventional diffusion/flow policies sample a "trajectory of trajectories" — a diffusion/flow trajectory of action trajectories. They discard intermediate trajectories, and must wait for the diffusion/flow process to complete before the first actions can be executed on the robot. **(b)** We simplify diffusion/flow policies by *treating action trajectories as flow trajectories*. Our flow-matching algorithm operates in action space. Starting from a noised version of the last executed action, it incrementally generates a sequence of actions that constitutes a *single* trajectory. This aligns the "time" of the flow sampling process with the "execution time" of the action trajectory. Importantly, actions can be streamed to the robot's controller on the fly *during* the flow sampling process, while retaining the ability to model multi-modal trajectory distributions.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

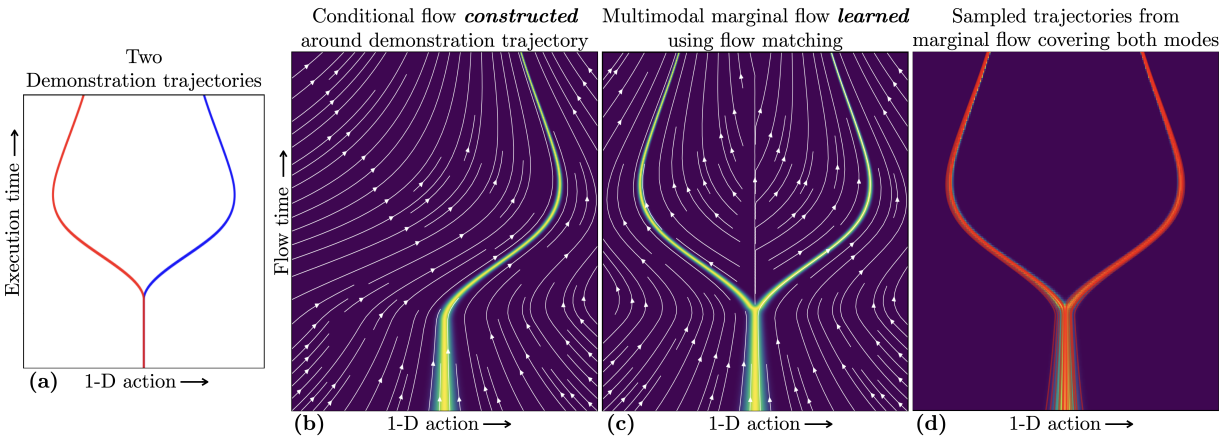

Figure 2: **(a)** To illustrate our method, we consider a toy example of 1-D robot actions with two demonstration trajectories shown in blue and red. **(b)** Given a demonstration trajectory sampled from the training set (*e.g.* the blue one), we first analytically *construct* a conditional flow *i.e.* an initial action distribution and a velocity field. The constructed flow samples trajectories from a thin Gaussian tube around the demonstration trajectory. Using the constructed velocity field as targets, we *learn* a marginal velocity field via flow matching [3], shown in **(c)**. The learned velocity field has the property that its induced marginal distribution over actions at each horizontal time slice matches the training distribution. **(d)** The initial action at $t = 0$ is sampled from a narrow Gaussian centered at the most recently executed action. Then, we iteratively integrate the learned velocity field to generate an action trajectory. Sampled trajectories (shown in red) cover both behavior modes in the training data. **(b)** We find that constructing conditional flows that *stabilize* around demonstration trajectories reduces distribution shift and improves imitation learning performance. The main takeaway is that our method is able to both represent multi-modal distributions over action trajectories like diffusion/flow policies, while also iteratively generating actions that can be streamed during the flow sampling process, enabling fast and reactive policy execution.

1 Introduction

Recent advances in robotic imitation learning, such as diffusion policy [1, 4] and flow-matching policy [2, 5, 6] have enabled robots to learn complex, multi-modal action distributions for challenging real-world tasks such as cooking, laundry folding, robot assembly and navigation [7]. They take a history of observations as input, and output a sequence of actions (also called an "action chunk"). Conventional diffusion/flow policies represent a direct application of diffusion models [8, 9] and flow-matching [3] to robot action sequences — they formulate the generative process as probabilistic transport in the space of action *sequences*, starting from pure Gaussian noise. Therefore, diffusion/flow policies represent a "*trajectory of trajectories*" — a diffusion/flow trajectory of action trajectories (Fig. 1a). This approach has several drawbacks. The sampling process discards all intermediate action trajectories, making diffusion/flow policies computationally inefficient. Importantly, the robot must wait for the diffusion/flow process to complete before executing any actions. Thus, diffusion/flow policies often require careful hyper-parameter tunning to admit tight control loops.

In this work, we propose a novel imitation learning framework that harnesses the temporal structure of action trajectories. We simplify diffusion/flow policies by treating *action trajectories as flow trajectories* (Fig. 1b). Our aim is to learn a flow transport in the *action space* $\mathcal{A}$, as opposed to trajectory space $\mathcal{A}^T$. Unlike diffusion/flow policies that start the sampling process from pure Gaussian noise (in $\mathcal{A}^T$), our initial sample comes from a *narrow* Gaussian centered around the most recently generated action (in $\mathcal{A}$). Then, we iteratively integrate a learned velocity field to generate a sequence of future actions that forms a *single* trajectory. The "flow time" — indicating progress of the flow process — coincides with execution time of the sampled trajectory. Iteratively generating the sequence of actions allows the actions to be *streamed* to the robot's controller on-the-fly *during* the flow generation process, significantly improving the policy's speed and reactivity.

We show how a streaming flow policy with the above desiderata can be learned using flow matching [3]. Given an action trajectory from the training set (Fig. 2a), we *construct* a velocity field conditioned on this example that samples paths in a narrow Gaussian "tube" around the demonstration (Fig. 2b).

Our training procedure is remarkably simple — we regress a neural network $v_\theta(a, t \mid h)$ that takes as input (i) an observation history h, (ii) flow timestep $t \in [0, 1]$, and (iii) action a, to match the constructed velocity field. We are able to re-use existing architectures for diffusion/flow policy while only modifying the input and output dimension of the network from $\mathcal{A}^T$ to $\mathcal{A}$. Flow matching guarantees that the *marginal* flow learned over all training trajectories, as shown in Fig. 2(c, d), is multi-modal. Specifically, the marginal distribution of actions at *each timestep* t matches that of the training distribution. Our approach thus retains diffusion/flow policy's ability to represent multi-modal trajectories while allowing for streaming trajectory generation.

How should we construct the target velocity field? Prior work [10] has shown that low-level stabilizing controllers can reduce distribution shift and improve theoretical imitation learning guarantees. We leverage the flexibility of the flow matching framework to construct velocity fields that *stabilize* around a given demonstration trajectory, by adding velocity components that guide the flow back to the demonstration. In our experiments, we find that stabilizing flow significantly improves performance.

Our method can leverage two key properties specific to robotics applications: (i) robot actions are often represented as position setpoints of the robot's joints or end-effector pose that are tracked by a low-level controller, (ii) the robot's joint positions/end-effector poses can be accurately measured via proprioceptive sensors (*e.g.* joint encoders) and forward kinematics. Streaming flow policy can not only imitate action trajectories, but is especially suited to imitate *state trajectories* when a stiff controller is available that can closely track state trajectories. In this case, the flow sampling process can be initialized from the known ground truth robot state instead of the state predicted from the previous chunk. This reduces uncertainty and error in the generated trajectory.

Unlike diffusion/flow policies, streaming flow policy is only guaranteed to match the marginal distribution of actions at each timestep, but not necessarily the joint distribution. Consequently, our method can produce trajectories that are compositions of segments of training trajectories, even if the composition was not part of the training dataset. While this may be seen as a limitation of our method, we argue that for most robotics tasks, compositionality is not only valid, but a desirable property that requires fewer demonstrations. Furthermore, while streaming flow policy is unable to capture global constraints that can only be represented in the joint distribution, it can learn local constraints such as joint constraints, and convex velocity constraints; see Sec. 9 for more details. In practice, we find that streaming flow policy performs comparably to diffusion policy while being significantly faster.

2 Background and problem formulation

We consider the problem of imitating sequences of *future* actions $a \in \mathcal{A}$ from histories of observations $h \in \mathcal{H}$ as input, where a history $h = \{o_i\}_{i=1}^K$ is a finite sequence of observations $o_i \in \mathcal{O}$. The time horizon $T_{\text{pred}} \in \mathbb{R}^+$ of the trajectory to be predicted can be an arbitrary hyperparameter. For simplicity, we re-scale the time interval to $[0, 1]$ by dividing by T_{pred}. Therefore, we represent an action trajectory as $\xi : [0, 1] \to \mathcal{A}$. We assume an unknown data generating distribution $p_{\mathcal{D}}(h, \xi)$ of inputs and outputs, from which a finite training dataset $\mathcal{D} = \{(h_i, \xi_i)\}_{i=1}^N$ of N tuples is sampled. See Table 3 for a complete list of notation. Our aim is to learn a policy that outputs a potentially multi-modal distribution over future trajectories ξ given a history of observations h.

Assumptions: We make two key assumptions: *(i)* the action space $\mathcal{A}$ is continuous (more generally, a smooth manifold), and *(ii)* action trajectories $\xi : [0, 1] \to \mathcal{A}$ are continuous and differentiable with respect to time. These assumptions are necessary to compute time-derivatives of action trajectories when treating action trajectories as differentiable flow trajectories. Most physical action spaces used in robotics, such as joint angles or end-effector poses, indeed satisfy these assumptions.

Velocity fields: We formulate *streaming flow policy*, with model parameters θ, as a history-conditioned velocity field $v_\theta(a, t \mid h)$. For a given history $h \in \mathcal{H}$, $t \in [0, 1]$, and action $a \in \mathcal{A}$, the model outputs a velocity in the tangent bundle $\mathcal{TA}$ of $\mathcal{A}$. The velocity field is a neural ordinary differential equation (ODE) [11]. Given an initial action $a(0)$, the velocity field induces trajectories $a(t)$ in action space by specifying the instantaneous time-derivative of the trajectory $da/dt = v_\theta(a, t \mid h)$.

Flows: The pairing of $v_\theta\,(a, t \,|\, h)$ with an initial probability distribution over $a(0)$ is called a *continuous normalizing flow* [11, 12] (simply referred to as "*flow*"). A flow transforms the initial action distribution to a new distribution $p_\theta\,(a \,|\, t, h)$, for every $t \in [0, 1]$, in a deterministic and invertible manner. We want streaming flow policy to start sampling close to the action a_{prev} that was most recently executed. This is the final action that was computed in the previous action chunk. When imitating state trajectories instead of action trajectories, we set a_{prev} to the current known robot state. Invertible flows require the initial probability distribution over $a(0)$ to have non-zero probability density on the domain $\mathcal{A}$ to be well-behaved. Therefore, we chose a narrow Gaussian distribution centered at a_{prev} with a small variance σ_0^2. A trajectory is generated by sampling from the initial distribution and integrating the velocity field as:

$$a(t) = a_0 + \int_0^t v_\theta\big(a(\tau), \tau \,\big|\, h\big)\, d\tau \quad \text{where} \quad a_0 \sim \mathcal{N}\big(a_{\text{prev}}, \sigma_0^2\big) \tag{1}$$

Importantly, standard ODE solvers can perform forward finite-difference integration auto-regressively, where integration at time t depends only on previously computed actions $a(\tau), \tau \leq t$. This property allows us to stream actions *during* the integration process, without needing to wait for the full trajectory to be computed. Next, we describe how we analytically construct conditional velocity fields given a trajectory ξ. Then, we will use them as targets to learn $v_\theta\,(a, t \,|\, h)$ using flow matching [3].

3 Analytically constructing conditional velocity fields

Given an action trajectory ξ, we first analytically construct a stabilizing *conditional* flow that travels closely along ξ. This will be used as a target to train a neural network velocity field. In particular, we construct a velocity field $v_\xi(a, t)$ and an initial distribution $p_\xi^0(a)$ such that the induced marginal probability distributions $p_\xi(a \,|\, t)$ form a thin Gaussian "tube" around ξ. By "Gaussian tube", we mean that $p_\xi(a \,|\, t)$ is a narrow Gaussian distribution centered at $\xi(t)$ for every $t \in [0, 1]$. This is illustrated in Fig. 2(a,b). We construct the stabilizing conditional flow as:

$$v_\xi(a, t) \;=\; \underbrace{\dot{\xi}(t)}_{\text{Trajectory velocity}} - \underbrace{k(a - \xi(t))}_{\text{Stabilization term}} \quad \text{and} \quad p_\xi^0(a) \;=\; \mathcal{N}\left(a \mid \xi(0)\,,\, \sigma_0^2\right) \tag{2}$$

The initial distribution $p_\xi^0(a)$ is a narrow Gaussian centered at the initial action $\xi(0)$ with a small standard deviation σ_0. The velocity has two components. The *trajectory velocity* is the time-derivative of the action trajectory ξ at time t, and does not depend on a. This term serves to move along the direction of the trajectory. The *stabilization term* is a negative proportional error feedback that corrects deviations from the trajectory. Controllers that stabilize around demonstration trajectories are known to reduce distribution shift and improve theoretical imitation learning guarantees [10]. We empirically observe that the stabilizing term produces significantly more robust and performant policies, compared to setting $k = 0$. We note that our framework leverages time derivatives of action trajectories $\dot{\xi}(t)$ during training, which are easily accessible, in addition to $\xi(t)$. This is in contrast to conventional diffusion/flow policies that only use $\xi(t)$ but not $\dot{\xi}(t)$. Throughout this paper, the term 'velocity' refers to $\dot{\xi}(t)$, and not the physical velocity of the robot. While they may coincide for certain choices of the action space $\mathcal{A}$, $\dot{\xi}(t)$ may not represent any physical velocity.

Theorem 1: The stabilizing conditional flow given by Eq. 2 induces the following per-timestep marginal distributions over the action space:

$$p_\xi(a \,|\, t) = \mathcal{N}\left(a \,\middle|\, \xi(t),\ \sigma_0^2 e^{-2kt}\right) \tag{3}$$

Proof: See App. B. The distribution of states sampled at any timestep $t \in [0, 1]$ is a Gaussian centered at the trajectory $\xi(t)$. Furthermore, the standard deviation $\sigma_t = \sigma_0 e^{-kt}$ starts from σ_0 at $t = 0$ and decays exponentially with time at rate k.

4 Learning objective for velocity fields to match marginal action distributions

Let $p_{\mathcal{D}}(h, \xi)$ denote the unknown data generating distribution from which the training dataset is sampled. The conditional velocity field $v_\xi(a, t)$ defined in Sec. 3 models a single action trajectory. If

Algorithm 1 Training algorithm

Input: Training set $\mathcal{D} = \{(h_i, \xi_i)\}_{i=1}^{N}, T_{\text{pred}}$
• *ξ has time horizon T_{pred} rescaled to* $[0, 1]$

while not converged **do**
  $(h, \xi) \sim \mathcal{D}$
  $t \sim \text{Uniform}(0, 1)$
  $a \sim p_\xi(a \mid t)$ (defined in Eq. 3)
  $\theta \leftarrow \theta - \lambda \nabla_\theta \underbrace{\|v_\xi(a, t) - v_\theta(a, t \mid h)\|^2}_{\text{Conditional flow matching loss}}$
return v_θ

Algorithm 2 Inference algorithm

Input: $v_\theta(a, t \mid h), T_{\text{pred}}, T_{\text{chunk}}, \Delta t$
$h, a \leftarrow \{\}$, q_{curr} (current robot configuration)
while True **do**
  $t, h_{\text{chunk}} \leftarrow 0, h$
  if imitating state: $a \leftarrow q_{\text{curr}}$
  while $t \leq T_{\text{chunk}}/T_{\text{pred}}$ **do** *// open loop*
    $o \leftarrow \texttt{Execute}(a)$ *// stream action during flow*
    $h \leftarrow h \cup \{o\}$
    $a \leftarrow a + v_\theta(a, t \mid h_{\text{chunk}})\Delta t$ *// integration step*
    $t \leftarrow t + \Delta t$

multiple behaviors ξ are valid for the same input history h, how can we learn a velocity field $v(a, t \mid h)$ that represents multi-modal trajectory distributions? We use flow matching with $v_\xi(a, t)$ the target. The conditional flow matching loss [3] for a history-conditioned velocity field $v(a, t \mid h)$ is defined as:

$$\mathcal{L}_{\text{CFM}}(v, p_{\mathcal{D}}) = \mathbb{E}_{(h,\xi)\sim p_{\mathcal{D}}} \, \mathbb{E}_{t\sim U[0,1]} \, \mathbb{E}_{a\sim p_\xi(a \mid t)} \left\| v(a, t \mid h) - v_\xi(a, t) \right\|_2^2 \tag{4}$$

This is simply an expected L_2 loss between a candidate velocity field $v(a, t \mid h)$ and the the analytically constructed conditional velocity field $v_\xi(a, t)$ as target. The expectation is over histories and trajectories under the probability distribution $p_{\mathcal{D}}(h, \xi)$, time t sampled uniformly from $[0, 1]$, and action a sampled from the constructed conditional flow known in closed-form in Eq. 3. The following theorem characterizes the per-timestep marginal distributions induced by the minimizer of this loss:

Theorem 2: The minimizer $v^* = \arg\min_v \mathcal{L}_{\text{CFM}}(v, p_{\mathcal{D}})$ induces the following per-timestep marginal distribution for each $t \in [0, 1]$ and observation history h:

$$p^*(a \mid t, h) = \int_\xi p_\xi(a \mid t) \, p_{\mathcal{D}}(\xi \mid h) \, \mathrm{d}\xi = \int_\xi \mathcal{N}\left(a \mid \xi(t), \sigma_t^2\right) \, p_{\mathcal{D}}(\xi \mid h) \, \mathrm{d}\xi \text{ where } \sigma_t = \sigma_0 e^{-kt} \tag{5}$$

Proof: This is a direct consequence of the flow matching theorems (Thms. 1 and 2) in Lipman et al. [3]. Intuitively, the per-timestep marginal distribution induced by the minimizer of $\mathcal{L}_{\text{CFM}}$ is the *average* of per-timestep marginal distributions of constructed conditional flows $p_\xi(a \mid t)$, over the distribution of future trajectories in $p_{\mathcal{D}}(\xi \mid h)$ that share the same observation history h. Since our conditional flows are constructed to sample narrow Gaussian tubes around demonstrated action sequences, the optimal v^* will produce a *multi-modal* mixture distribution over future actions from the data-generating distribution consistent with observation history h, convolved with small noise σ_t.

Matching the per-timestep marginal distributions is desirable and necessary for representing multi-modal distributions. Consider the example in Fig. 2 that constructs two conditional flows, one that samples actions to the right ($a > 0$), and the other that samples actions to the left ($a < 0$). In order for a learned model to sample both modes with probability 0.5 each, its per-timestep marginal distribution must match the averaged per-timestep marginal distributions of conditional flows. Unlike flow policies [2, 5, 6] that only require matching the target distributions at $t = 1$, our method leverages the fact that flow matching [3] matches the marginal distributions at *all* timesteps $t \in [0, 1]$.

5 Training and inference algorithms for streaming flow policy

Training: While we do not have access to the underlying data generating distribution $p_{\mathcal{D}}(h, \xi)$, we do have access to a training set $\mathcal{D} = \{(h_i, \xi_i)\} \sim p_{\mathcal{D}}(h, \xi)$ that contains N samples from this distribution. Therefore, we train a neural network velocity field $v_\theta(a, t \mid h)$ using a finite-sample estimate of Eq. 4: $\widehat{\mathcal{L}}_{\text{CFM}}(\theta, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{t\sim U[0,1]} \mathbb{E}_{a\sim p_{\xi_i}(a \mid t)} \left\| v_\theta(a, t \mid h_i) - v_{\xi_i}(a, t) \right\|_2^2$ as shown in Alg. 1.

Inference: While behavior policies are trained to predict sequences of horizon T_{pred}, they are usually run in a receding horizon fashion with a potentially different action chunk horizon $T_{\text{chunk}} \leq T_{\text{pred}}$ [1]. The integration timestep Δt is another hyperparameter that controls the granularity of

		Push-T with state input			Push-T with image input	
		State imitation Avg/Max scores	Action imitation Avg/Max scores	Latency	State imitation Avg/Max scores	Latency
		↑	↑	↓	↑	↓
1	DP [1]: 100 DDPM steps	92.9% / 94.4%	90.7% / 92.8%	40.2 ms	**87.0%** / 90.1%	127.2 ms
2	DP [1]: 10 DDIM steps	87.0% / 89.0%	81.4% / 85.3%	04.4 ms	85.3% / **91.5%**	10.4 ms
3	Flow matching policy [5]	80.6% / 82.6%	80.6% / 82.6%	05.8 ms	71.0% / 72.0%	12.9 ms
4	Streaming DP [17]	87.5% / 91.4%	84.2% / 87.0%	26.7 ms	84.7% / 87.1%	77.7 ms
5	SFP *without* stabilization	84.0% / 86.4%	81.8% / 93.2%	**03.5 ms**	73.9% / 77.5%	**08.8 ms**
6	**SFP (Ours)**	**95.1% / 96.0%**	**91.7% / 93.7%**	**03.5 ms**	83.9% / 84.8%	**08.8 ms**

Table 1: Imitation learning accuracy on the Push-T [1] dataset.
Our method (in green) compared against baselines (in red) / and ablations (in blue). See text for details.

the action sequence. Therefore, to generate an action chunk, we integrate the velocity field in $t \in [0, T_{\text{chunk}}/T_{\text{pred}}]$, producing $T_{\text{chunk}}/(T_{\text{pred}}\Delta t)$ many actions. The action chunk is computed and executed open-loop *i.e.* the neural network v_θ inputs the same observation history h_{chunk} for all integration steps. Importantly, we are able to stream and execute actions on the robot as soon as they are computed (see Alg. 2, line 6). In contrast, diffusion/flow policies must wait for the inner loop to complete before executing any actions.

Deterministic execution at test time: Our learning framework suggests the initial action be sampled from $a_0 \sim \mathcal{N}\left(a_0 \mid a_{\text{prev}}, \sigma_0^2\right)$ (see Eqs. 1 and 2). However, during inference time, we avoid adding noise to actions by setting $\sigma_0 = 0$ to produce deterministic behavior. We do so because the ability to represent multi-modal distributions is primarily motivated by the need to prevent "averaging" distinct but valid behaviors of the same task [1]. While representing multi-modality is crucial during training, the learned policy can be run deterministically at test time without loss in performance. For example, ACT [13] sets its variance parameter to zero at test time to produce deterministic behavior. In App. C, we present a variant of streaming flow policy in an extended state space that decouples stochasticity into additional latent variables. This variant allows us to sample multiple modes of the trajectory distribution at test time without adding noise to actions. However, we found that simply setting $\sigma_0 = 0$ at test time works better in practice; therefore we follow this strategy in all our experiments.

Imitating actions vs. states: When training trajectories correspond to *actions*, we start integration of the current action chunk from the most recently generated action in the previous chunk. Streaming flow policy can also be used to imitate robot *state* trajectories when a controller is available that can closely track desired states. It is especially suited for state imitation because we can start integration of the current state chunk from the *current robot state* that is accurately measured by proprioceptive sensors. Therefore, streaming flow policy is able to leverage state feedback in two ways: in the history h and the initialization a_0 for flow integration. This reduces error in the generated trajectory.

6 Experiments

We evaluate streaming flow policy on two imitation learning benchmarks: the Push-T environment [1, 14], and RoboMimic [15], and perform real-world experiments on a Franka Panda robot arm. We compare our method (in green) against 4 baselines (in red): Row 1 (DP): standard diffusion policy [1] that uses 100 DDPM [9] steps, Row 2 (DP): a faster version of diffusion policy that uses 10 DDIM [16] steps, Row 3: conventional flow matching-based policy [5] and Row 4: Streaming Diffusion Policy [17], a recent method that runs diffusion policy in a streaming manner (see Sec. 7). We also compare against streaming flow policy that does not construct stabilizing flows during training, *i.e.* uses $k = 0$ (in blue). This ablation is designed to measure the contribution of stabilization to task performance. Following Chi et al. [1], we report the average score for the 5 best checkpoints, the best score across all checkpoints, and the average latency per action for each method.

In Table 5, we report results on the Push-T environment: a simulated 2D planar pushing task where the robot state is the 2-D position of the cylindrical pusher in global coordinates, and actions are 2-D setpoints tracked by a PD controller. Push-T contains 200 training demonstrations and 50 evaluation

	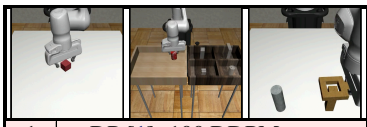	**RoboMimic Lift** Action imitation Avg/Max scores	**RoboMimic Can** Action imitation Avg/Max scores	**RoboMimic Square** Action imitation Avg/Max scores	**Latency**
		↑	↑	↑	↓
1	DP [1]: 100 DDPM steps	**100.0%** / **100.0%**	94.0% / 98.0%	77.2% / **84.0%**	53.4 ms
2	DP [1]: 10 DDIM steps	**100.0%** / **100.0%**	94.8% / 98.0%	76.0% / 82.0%	5.8 ms
3	Flow matching policy [5]	99.2% / **100.0%**	66.0% / 80.0%	54.0% / 56.0%	4.8 ms
4	Streaming DP [17]	98.8% / **100.0%**	96.8% / 98.0%	77.6% / 82.0%	30.3 ms
5	SFP *without* stabilization	99.6% / **100.0%**	90.0% / 92.0%	53.2% / 60.0%	**4.5 ms**
6	**SFP (Ours)**	**100.0%** / **100.0%**	**98.4%** / **100.0%**	**78.0%** / **84.0%**	**4.5 ms**

Table 2: Imitation learning accuracy on RoboMimic [15] environment.
Our method (in green) compared against baselines (in red) / and ablations (in blue). See text for details.

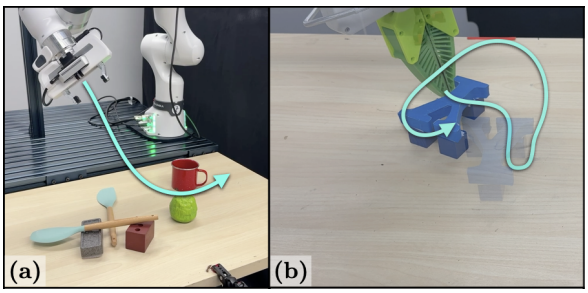

Figure 3: Real-world experiments on a 7-DOF Franka Emika Panda robot arm on the tasks of **(a)** reaching and picking an apple, and **(b)** reorienting a block. We compare our method against diffusion policy, and find that our method is noticeably faster and produces smoother motion; see comparison videos on our project website.

episodes. We perform experiments in two settings: when simulator state is used as observations, and when images are used as observations. "Action imitation" is the standard practice of imitating action sequences provided in the benchmark training set. We also perform experiments with "state imitation" (see Sec. 5), where we directly imitate the measured 2D positions of the robot. Here, we use the known ground-truth robot position at the beginning of each action chunk to as a starting point for velocity field integration. In Table 2, we report results on the RoboMimic environment, specifically the "lift" and "can" tasks, with state inputs. Both tasks involve predicting sequences of 6-DOF end-effector poses with respect to a global frame that are tracked by a PD controller. Each task contains 300 training demonstrations, and 50 evaluation episodes. The tasks involve picking objects and placing them at specific locations, including picking a square nut and placing it on a rod.

The neural network for streaming flow policy $v_\theta : \mathcal{A} \times [0,1] \times \mathcal{H} \to T\mathcal{A}$ is structurally similar to diffusion/flow policies (*e.g.* $\epsilon_\theta : \mathcal{A}^T \times [0,1] \times \mathcal{H} \to \mathcal{A}^T$) with the only change being the input and output spaces (action space $\mathcal{A}$ vs. action-trajectory space $\mathcal{A}^T$). Therefore, we are able to re-use existing diffusion/flow policy architectures [1] by changing the input and output dimension of the network and replacing 1-D temporal convolution/attention layers over action sequences [1, 18] with a fully connected layer. Furthermore, due to the reduced dimensionality of the flow sampling space, we found that streaming flow policy is faster to train and has a smaller GPU memory footprint compared to diffusion/flow policies.

We conduct real-world experiments on a Franka Emika Panda robot arm with a RealSense D435f depth camera on two tasks: (a) reaching and picking an object, and (b) reorienting a block to a vertical goal orientation (see Fig. 3). We find that streaming flow policy is noticeably faster and produces significantly smoother motion than diffusion policy, as shown in videos on the project website.

Conclusions: Stabilizing flow policy performs comparably to diffusion policy and other baselines in terms of performance on most tasks, while being significantly faster per action. Furthermore, the reported latency does not even take into account the fact that streaming flow policy can asynchronously run action generation and robot execution in parallel. In practice, this can avoid delays and jerky robot movements. Diffusion policy can be sped up by running fewer diffusion steps via DDIM [16]. And flow-matching policy is also faster than diffusion policy. However, their speed seems to come at

the cost of sometimes significant reduction in accuracy. In App. D, we analyze the performance of streaming flow policy as a function of the action chunk horizon T_{chunk}.

7 Related work

Learning dynamical systems: Our work is closely related to prior work on learning dynamical systems [19–23]. A key difference is that most prior works assume a single, deterministic future trajectory given an input state. However, we focus on learning multi-modal *distributions* over trajectories, which is known to be essential for behavior cloning in robotics [1]. For example, Sochopoulos et al. [21] learn a neural ODE [11] by minimizing the distance between the predicted and demonstrated trajectories. This is susceptible to undesirable "averaging" of distinct behaviors that achieve the same task. Singh et al. [24] learn a neural CDE [25], motivated by asynchronous multiscale sensor fusion. Our approach learns a neural ODE endowed with an initial noise distribution that induces a distribution over future trajectories, known as a continuous normalizing flow [11, 12]. We use the recently proposed flow matching framework [3] to fit the predicted trajectory distribution to training data. An important consequence is that our method aggregates "noised" demonstration trajectories as additional training samples, whereas prior works train only on states directly present in the demonstrations. Furthermore, most prior works assume a time-invariant velocity field [19–21]. Our velocity field depends on time and allows learning non-Markovian behaviors for tasks like spreading sauce on pizza dough, where the end-effector must rotate a fixed number of times. Finally, most works on learning dynamical systems focus on *point-stability* around goal states [19–21]; we don't assume goal states and construct flows that stabilize around demonstration trajectories.

Flow matching: Flow matching [3] is a recent technique for learning complex, multi-modal distributions that has been used to model images [3, 26, 27], videos [28, 29], molecular structures [30–32], and robot action sequences [2, 5, 6, 33, 34]. However, the flow sampling process starts from Gaussian noise, and the distribution of interest is only modeled at the final timestep $t = 1$. Our insight is to treat the entire flow trajectory as a sample from the target distribution over action sequences.

Streaming Diffusion Policy: The work most closely related to ours is Streaming Diffusion Policy [17], which is an adaptation of of discrete time diffusion policies [1]. Instead of maintaining all actions in the sequence at the same noise level, Streaming Diffusion Policy maintains a rolling buffer of actions with increasing noise levels. Every diffusion step reduces the noise level of all actions by one, fully de-noising the oldest action in the buffer that can be streamed to the robot. However, this method requires maintaining an action buffer of the length of the prediction horizon, even if the action horizon is much shorter. Furthermore, there is an up-front cost to initialize the buffer with increasing noise levels. The rolling buffer approach has been applied to video prediction [35] and character motion generation [36]. Our method is more economical since it computes only as many actions are streamed to the robot, without requiring a buffer. We evaluate our method against Streaming Diffusion Policy in Sec. 6.

8 Conclusion

In this work, we have presented a novel approach to imitation learning that addresses the computational limitations of existing diffusion and flow-matching policies. Our key contribution is a simplified approach that treats *action trajectories as flow trajectories*. This enables incremental integration of a learned velocity field that allows actions to be streamed to the robot during the flow sampling process. The streaming capability makes our method particularly well-suited for receding horizon policy execution. Despite the streaming nature of our approach, the flow matching framework guarantees the ability to model multi-modal action trajectories. By constructing flows that stabilize around demonstration trajectories, we reduce distribution shift and improve imitation learning performance. Our experimental results demonstrate that streaming flow policy performs comparably to prior imitation learning approaches on benchmark tasks, but enables faster policy execution and tighter sensorimotor loops, making it more practical for reactive robot control.

9 Limitations

In this section, we discuss some limitations of our approach.

9.1 SFP does not match joint distribution, only per-timestep marginal distributions

Our flow matching framework ensures that the learned distribution over trajectories conditioned on the history matches the training distribution in terms of marginal distributions of actions at each timestep $t \in [0, 1]$. We however, **do not guarantee that the *joint* distribution of actions across a trajectory matches the training distribution**. This is in contrast to diffusion policy, that is able to match the joint distribution since the diffusion model operates in trajectory space $\mathcal{A}^T$.

Figs. 4 and 5 illustrate a toy example where streaming flow policy matches marginal distributions but not the joint distribution. The x-axis represents 1-D robot actions, and the y-axis represents flow time ($t \in [0, 1]$). Fig. 4a shows two trajectories in blue and red, of shapes "S" and "Ƨ" respectively. The trajectories intersect at $t = 0.5$. The learned flow field is shown in Fig. 4c, and the induced marginal distribution over actions is shown in Fig. 4d. The marginal distribution of actions matches the training distribution at each $t \in [0, 1]$. Trajectories sampled from the flow field are shown in Fig. 4d. The trajectory distribution contains two modes of equal probability: trajectories that always lie either in $a < 0$ (shown in blue), or in $a > 0$ (shown in red). The shapes formed by sampled trajectories — "Ɛ" and "3" respectively — do not match the shapes of trajectories in the training data.

A similar phenomenon is illustrated in Fig. 5 using the latent-space variant of streaming flow policy (see App. C) trained on the same dataset of intersecting trajectories. While the marginal distribution of actions again matches with the training distribution, the trajectories contain *four* modes, with shapes "S", "Ƨ", "Ɛ" and "3". Note that the per-timestep marginal distributions over actions still match the training data.

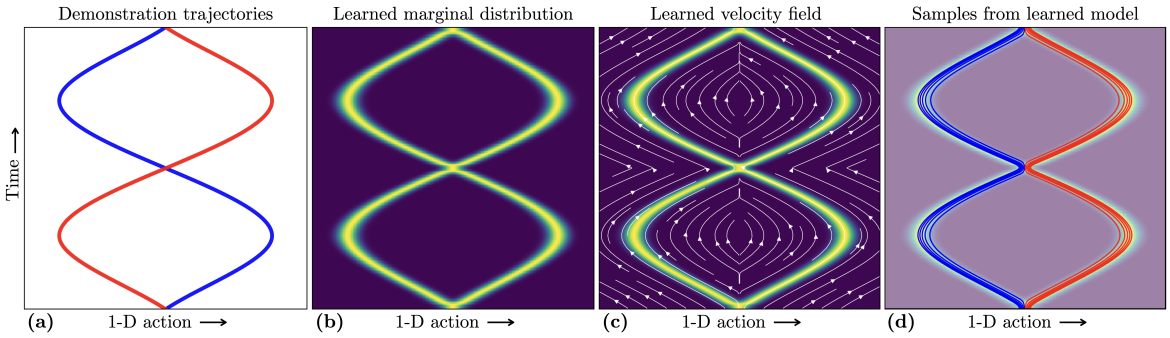

Figure 4: A toy example illustrating how streaming flow policy matches marginal distribution of actions in the trajectory at all time steps, but not necessarily their joint distribution. The x-axis represents a 1-D action space, and the y-axis represents both trajectory time and flow time. **(a)** The bi-modal training set contains two intersecting demonstration trajectories, illustrated in blue and red, with shapes "S" and "Ƨ" respectively. **(b)** The marginal distribution of actions at each time step learned by our streaming flow policy. The marginal distributions perfectly match the training data. **(c)** The learned velocity flow field $v_\theta(a, t \mid h)$ that yeilds the marginal distributions in (b). **(d)** Trajectories sampled from the learned velocity field. Trajectories that start from $a < 0$ are shown in blue, and those starting from $a > 0$ are shown in red. The sampled trajectories have shapes "Ɛ" and "3", with equal probability. These shapes are different from the shapes "S" and "Ƨ" in the training distribution, although their margin distributions are identical.

9.2 Streaming flow policies exhibit compositionality

The loss of fidelity to the joint distribution is a potential weakness of our framework. Therefore, this framework may not be the right choice when learning the correct joint distributions is crucial. However, another perspective is to think of our method as providing *compositionality* over training demonstrations. The sampled trajectories can be composed of pieces across the training data.

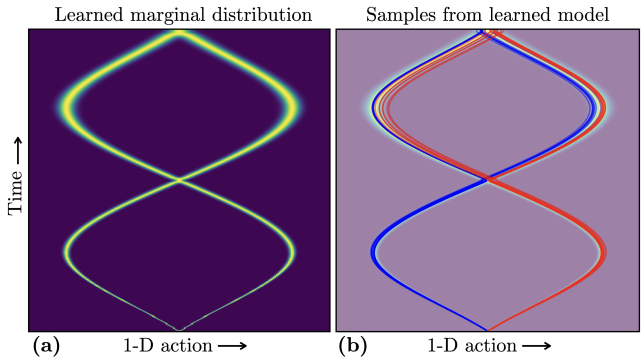

Figure 5: Different variants of streaming flow policy can produce different joint distributions of actions that are consistent with the marginal distributions in the training data. This example is produced using the latent-variable version of streaming flow policy, described in App. C. **(a)** The marginal distribution of actions at each time step learned by the streaming flow policy matches the training data. **(b)** Samples from the trained policy produces *four* modes with shapes "S", "Ƨ", "Ɛ" and "3", whereas the training data contains only two modes with shapes "S" and "Ƨ".

For many robotics tasks, compositionality might be both valid and desirable. For example, in quasi-static tasks where the robot moves slowly, if two demonstration trajectories are valid, then the compositions across these trajectories are often also valid. Under this assumption, compositionality allows the flow model to learn many valid combinations of partial trajectories with fewer demonstrations.

What constraints on trajectories reflected in the training data can streaming flow policy learn? Streaming flow policy is **unable to capture global constraints** that can only be represented in the joint distribution. However, it can learn certain local constraints.

9.3 SFPs can learn arbitrary position constraints

Robot actions $a \in Q \subseteq \mathcal{A}$ may be constrained to lie in a subset $Q \subseteq \mathcal{A}$. For example, Q may reflect joint limits of a robot arm. Then, a well-trained streaming flow policy should learn this constraint as well.

To see why, consider Eq. 5 which states that the learned marginal density of actions $p^*(a \,|\, t, h) = \int_\xi p_\xi(a \,|\, t)\, p_\mathcal{D}(\xi \,|\, h)\, \mathrm{d}\xi$ at time t is a weighted average of marginal densities of conditional flows $p_\xi(a \,|\, t)$. Recall that we construct $p_\xi(a \,|\, t)$ to be narrow Gaussian tubes around demonstration trajectories ξ. Assume that the thickness of the Gaussian tube is sufficiently small that $a \notin Q \implies p_\xi(a \,|\, t) < \epsilon$, for some small $\epsilon > 0$ and for all ξ, t. Then we have from Eq. 5 that $p_\xi(a \,|\, t) < \epsilon \implies p^*(a \,|\, t, h) < \epsilon$ for all $t \in [0, 1]$. Therefore, the probability of sampling an action a that violates the constraint Q is extremely low.

9.4 SFPs can learn convex velocity constraints

Theorem 2 of Lipman et al. [3] implies that the minimizer of the conditional flow matching loss $v^* := \arg\min_v \mathcal{L}_{\mathrm{CFM}}(v, p_\mathcal{D})$ has the following form:

$$
\begin{aligned}
v^*(a, t \,|\, h) &= \int_\xi v_\xi(a, t) \underbrace{\frac{p_\mathcal{D}(\xi \,|\, h)\, p_\xi(a \,|\, t)}{\int_{\xi'} p_\mathcal{D}(\xi' \,|\, h)\, p_{\xi'}(a \,|\, t)\, \mathrm{d}\xi'}}_{p_\mathcal{D}(\xi \,|\, a, t, h)} \mathrm{d}\xi \\
&= \int_\xi v_\xi(a, t)\, p_\mathcal{D}(\xi \,|\, a, t, h)\, \mathrm{d}\xi \qquad (6) \\
&\approx \int_\xi \dot{\xi}(t)\, p_\mathcal{D}(\xi \,|\, a, t, h)\, \mathrm{d}\xi \text{ (assuming } k \approx 0)
\end{aligned}
$$

Intuitively, the target velocity field v^* at (a, t) is a weighted average of conditional flow velocities $v_\xi(a, t)$ over demonstrations ξ. The weight for ξ is the Bayesian posterior probability of ξ, where the prior probability $p_{\mathcal{D}}(\xi \mid h)$ is the probability of ξ given h in the training distribution, and the likelihood $p_\xi(a \mid t)$ is the probability that the conditional flow around ξ generates a at time t.

Under sufficiently small values of k, we have from Eq. 2 that $v_\xi(a, t) \approx \dot{\xi}(t)$. Note that v^* is then a convex combination of demonstration velocities $\dot{\xi}(t)$. Consider convex constraints over velocities $\dot{\xi}(t) \in C$ *i.e.* $\dot{\xi}(t)$ is constrained to lie in a convex set C for all ξ with non-zero support $p_{\mathcal{D}}(\xi) > 0$ and for all $t \in [0, 1]$. This is the case, for example, when robot joint velocities lie in a closed interval $[v_{\min}, v_{\max}]$. Then, Eq. 6 implies that v^* also lies in C.

References

[1] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, Daegu, Republic of Korea, July 2023.

[2] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control, 2024. *ArXiv Preprint:2410.24164*, 2024.

[3] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

[4] Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, C. Xu, J. Luo, T. Kreiman, Y. Tan, L. Y. Chen, P. Sanketi, Q. Vuong, T. Xiao, D. Sadigh, C. Finn, and S. Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024.

[5] F. Zhang and M. Gienger. Affordance-based Robot Manipulation with Flow Matching. *arXiv preprint arXiv:2409.01083*, 2024.

[6] S. Ye and M. C. Gombolay. Efficient trajectory forecasting and generation with conditional flow matching. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2816–2823. IEEE, 2024.

[7] A. Sridhar, D. Shah, C. Glossop, and S. Levine. NoMaD: Goal masked diffusion policies for navigation and exploration. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–70. IEEE, 2024.

[8] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning (ICML)*, pages 2256–2265. PMLR, 2015.

[9] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:6840–6851, 2020.

[10] A. Block, A. Jadbabaie, D. Pfrommer, M. Simchowitz, and R. Tedrake. Provable guarantees for generative behavior cloning: Bridging low-level stability and high-level behavior. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 48534–48547, New Orleans, USA, December 2023.

[11] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[12] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, and D. Duvenaud. FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models. In *International Conference on Learning Representations (ICLR)*, 2019.

[13] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems (RSS)*, Daegu, Republic of Korea, July 2023.

[14] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on robot learning (CoRL)*, pages 158–168. PMLR, 2022.

[15] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. In *5th Annual Conference on Robot Learning (CoRL)*, London & Virtual, November 2021.

[16] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations (ICLR)*, 2021.

[17] S. H. Høeg, Y. Du, and O. Egeland. Streaming Diffusion Policy: Fast Policy Synthesis with Variable Noise Diffusion Models. In *IEEE International Conference on Robotics and Automation (ICRA)*, Atlanta, USA, May 2025.

[18] M. Janner, Y. Du, J. Tenenbaum, and S. Levine. Planning with Diffusion for Flexible Behavior Synthesis. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research (PMLR)*, pages 9902–9915, Baltimore, USA, 17–23 July 2022.

[19] S. M. Khansari-Zadeh and A. Billard. Imitation learning of globally stable non-linear point-to-point robot motions using nonlinear programming. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2676–2683. IEEE, 2010.

[20] S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics (T-RO)*, 27(5):943–957, 2011.

[21] A. Sochopoulos, M. Gienger, and S. Vijayakumar. Learning deep dynamical systems using stable neural ODEs. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11163–11170. IEEE, 2024.

[22] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[23] S. Schaal, A. Ijspeert, and A. Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358 (1431):537–547, 2003.

[24] S. Singh, F. M. Ramirez, J. Varley, A. Zeng, and V. Sindhwani. Multiscale sensor fusion and continuous control with neural CDEs. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10897–10904. IEEE, 2022.

[25] P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural controlled differential equations for irregular time series. *Advances in neural information processing systems*, 33:6696–6707, 2020.

[26] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[27] N. Ma, M. Goldstein, M. S. Albergo, N. M. Boffi, E. Vanden-Eijnden, and S. Xie. SiT: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *European Conference on Computer Vision (ECCV)*, pages 23–40. Springer, 2024.

[28] Y. Jin, Z. Sun, N. Li, K. Xu, H. Jiang, N. Zhuang, Q. Huang, Y. Song, Y. Mu, and Z. Lin. Pyramidal flow matching for efficient video generative modeling. *arXiv preprint arXiv:2410.05954*, 2024.

[29] A. P. et. al. Movie Gen: A Cast of Media Foundation Models. *arXiv preprint arXiv:2410.13720*, 2025.

[30] B. Jing, B. Berger, and T. Jaakkola. AlphaFold meets flow matching for generating protein ensembles. *arXiv preprint arXiv:2402.04845*, 2024.

[31] A. J. Bose, T. Akhound-Sadegh, G. Huguet, K. Fatras, J. Rector-Brooks, C.-H. Liu, A. C. Nica, M. Korablyov, M. Bronstein, and A. Tong. SE(3)-stochastic flow matching for protein backbone generation. *arXiv preprint arXiv:2310.02391*, 2023.

[32] L. Klein, A. Krämer, and F. Noé. Equivariant flow matching. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:59886–59910, 2023.

[33] N. Funk, J. Urain, J. Carvalho, V. Prasad, G. Chalvatzaki, and J. Peters. ActionFlow: Equivariant, Accurate, and Efficient Policies with Spatially Symmetric Flow Matching. *arXiv preprint arXiv:2409.04576*, 2024.

[34] M. Braun, N. Jaquier, L. Rozo, and T. Asfour. Riemannian flow matching policy for robot motion learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5144–5151. IEEE, 2024.

[35] D. Ruhe, J. Heek, T. Salimans, and E. Hoogeboom. Rolling Diffusion Models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 42818–42835. PMLR, 21–27 Jul 2024.

[36] Z. Zhang, R. Liu, R. Hanocka, and K. Aberman. TEDi: Temporally-entangled diffusion for long-term motion synthesis. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024.

Appendix

Streaming Flow Policy

Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories

A Mathematical notation

Symbol	Description	Domain
T_{pred}	Prediction time horizon of trajectories during training	$\mathbb{R}^+$
T_{chunk}	Time horizon of action chunk during inference	$\mathbb{R}^+$
t	Flow time = execution time rescaled from $[0, T_{\text{pred}}]$ to $[0, 1]$	$[0, 1]$
a	Robot action (often a robot configuration)	$\mathcal{A}$
v	Action velocity	$T\mathcal{A}$
o, h	Observation, Observation history	$\mathcal{O}, \mathcal{H}$
ξ	Action trajectory (chunk), where time is rescaled from $[0, T_{\text{pred}}]$ to $[0, 1]$	$[0, 1] \to \mathcal{A}$
$\dot{\xi}$	Time derivative of action trajectory: $\dot{\xi}(t) = \frac{d}{dt}\xi(t)$	$[0, 1] \to T\mathcal{A}$
$p_{\mathcal{D}}(h, \xi)$	Distribution of observation histories and future action chunks. Training set is assumed to be sampled from this distribution.	$\Delta(\mathcal{H} \times [0, 1] \to \mathcal{A})$
$v_\theta(a, t \mid h)$	Learned marginal velocity field with network parameters θ	$T\mathcal{A}$
$v_\xi(a, t)$	Conditional velocity field for demonstration ξ	$T\mathcal{A}$
$p_\xi(a \mid t)$	Marginal probability distribution over a at time t induced by v_ξ	$\Delta(\mathcal{A})$
$v^*(a, t \mid h)$	Optimal marginal velocity field under data distribution $p_{\mathcal{D}}$	$T\mathcal{A}$
$p^*(a \mid t, h)$	Marginal probability distribution over a at time t induced by v^*	$\Delta(\mathcal{A})$
k, σ_0	Stabilizing gain, Initial standard deviation	$\mathbb{R}_{\geq 0}, \mathbb{R}^+$

Table 3: Mathematical notation used consistently throughout the paper.

B Proof of Theorem 1

Integrating learned velocity fields can suffer from drift since errors accumulate during integration. We adding a stabilization term, we can correct deviations from the demonstration trajectory. The stabilizing velocity field is:

$$v_\xi(a, t) = \underbrace{-k(a - \xi(t))}_{\text{Stabilization term}} + \underbrace{\dot{\xi}(t)}_{\text{Path velocity}} \tag{7}$$

where $k > 0$ is the stabilizing gain. This results in exponential convergence to the demonstration:

$$\frac{d}{dt}(a - \xi(t)) = -k(a - \xi(t)) \tag{8}$$

$$\implies \frac{1}{a - \xi(t)} \frac{d}{dt}(a - \xi(t)) = -k \tag{9}$$

$$\implies \frac{d}{dt} \log(a - \xi(t)) = -k \tag{10}$$

$$\implies \log(a - \xi(t))\Big|_0^t = -\int_0^t k dt \tag{11}$$

$$\implies \log \frac{a(t) - \xi(t)}{a_0 - \xi(0)} = -kt \tag{12}$$

$$\implies a(t) = \xi(t) + (a_0 - \xi(0))e^{-kt} \tag{13}$$

Since $a_0 \sim \mathcal{N}\left(\xi(0), \sigma_0^2\right)$ (see Eq. 1), and $a(t)$ is linear in a_0, we have by linearity of Gaussian distributions that:

$$p_\xi(a \,|\, t) = \mathcal{N}(a \,\big|\, \xi(t), \sigma_0^2 e^{-2kt}) \tag{14}$$

□

C Decoupling stochasticity via latent variables

In order to learn multi-modal distributions during training, streaming flow policy as introduced in Sec. 3 requires a small amount of Gaussian noise added to the initial action. However, we wish to avoid adding noise to actions at test time. We now present a variant of streaming flow policy in an extended state space by introducing a latent variable $z \in \mathcal{A}$. The latent variable z decouples stochasticity from the flow trajectory, allowing us to sample multiple modes of the trajectory distribution at test time while deterministically starting the sampling process from the most recently generated action.

We now define a conditional flow in the extended state space $(a, z) \in \mathcal{A}^2$. We define the initial

σ_0	Initial standard deviation	$\mathbb{R}^+$
σ_1	Final standard deviation	$\mathbb{R}^+$
k	Stabilizing gain	$\mathbb{R}_{\geq 0}$
σ_r	Residual standard deviation = $\sqrt{\sigma_1^2 - \sigma_0^2 e^{-2k}}$	$\mathbb{R}_{\geq 0}$

Table 4: Hyperparameters used in the stochastic variant of streaming flow policy that uses stochastic latent variables.

distribution by sampling a_0 and z_0 independently. a_0 is sampled from a vanishingly narrow Gaussian distribution centered at the initial action of the demonstration trajectory $\xi(0)$, but with a extremely small variance $\sigma_0 \approx 0$. z_0 is sampled from a standard normal distribution, similar to standard diffusion models [9] and flow matching [3].

<u>Initial sample</u>

$$z_0 \sim \mathcal{N}(0, I) \tag{15}$$

$$a_0 \sim \mathcal{N}(\xi(0), \sigma_0^2) \tag{16}$$

We assume hyperparameters σ_0, σ_1 and k. They correspond to the initial and final standard deviations of the action variable a in the conditional flow. k is the stabilizing gain. Furthermore, we constrain them such that $\sigma_1 \geq \sigma_0 e^{-k}$. Then, let us define $\sigma_r \coloneqq \sqrt{\sigma_1^2 - \sigma_0^2 e^{-2k}}$. Then we construct the joint *flow trajectories* of (a, z) starting from $(a(0), z(0))$ as:

<u>Flow trajectory diffeomorphism</u>

$$\begin{aligned} a(t \,|\, \xi, a_0, z_0) &= \xi(t) + (a_0 - \xi(0))\, e^{-kt} + (\sigma_r t) z_0 \\ z(t \,|\, \xi, a_0, z_0) &= (1 - (1 - \sigma_1)t) z_0 + t\xi(t) \end{aligned} \tag{17}$$

The flow is a diffeomorphism from $\mathcal{A}^2$ to $\mathcal{A}^2$ for every $t \in [0, 1]$.

Note that $a(0 \,|\, \xi, a_0, z_0) = a_0$ and $z(0 \,|\, \xi, a_0, z_0) = z_0$, so the diffeomorphism is identity at $t = 0$. The marginal distribution at $t = 1$ for a and z is given by $a(1 \,|\, \xi) \sim \mathcal{N}(\xi(1), \sigma_1^2)$ and $z(1 \,|\, \xi) \sim \mathcal{N}(\xi(1), \sigma_1^2)$.

Intuitively, the variable a follows the shape of the action trajectory $\xi(t)$ with an error starting from $a_0 - \xi(0)$ and decreasing with an exponential factor due to the stabilizing gain. However, it uses the sampled noise variable $z_0 \sim \mathcal{N}(0, I)$ to increase the standard deviation from σ_0 around $\xi(0)$ to σ_1 around $\xi(1)$. This is done in order to sample different modes of the trajectory distribution at test time. On the other hand, the latent variable z starts from the random sample $z_0 \sim \mathcal{N}(0, I)$ but continuously moves closer to the demonstration trajectory $\xi(t)$, reducing its variance from 1 to σ_1.

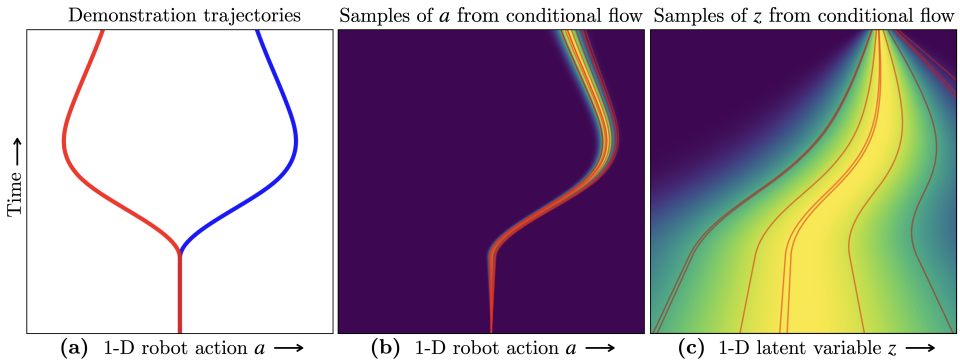

Figure 6: Constructing a conditional flow using auxiliary stochastic latent variables instead of adding noise to actions. In this toy example, the x-axis represents a 1-D action space, and the y-axis represents both trajectory time and flow time. **(a)** A toy bi-modal training set contains two trajectories shown in red and blue; the same as in Fig. 1a. Given a demonstration trajectory ξ from the training set (*e.g.* the demonstration in blue), we design a velocity field $v_\xi(a, z, t)$ that takes as input time $t \in [0, 1]$, the action a at time t, as well as an additional latent variable z. The latent variable is responsible for injecting noise into the flow sampling process, allowing the initial action $a(0)$ to be deterministically set to the initial action $\xi(0)$ of the demonstration. The latent variable $z(0) \sim \mathcal{N}(0, 1)$ is sampled from the standard normal distribution at the beginning of the flow process, similar to conventional diffusion/flow policies. The velocity field $v_\xi(a, z, t)$ generates trajectories in an extended sample space $[0, 1] \to \mathcal{A}^2$ where a and z are correlated and co-evolve with time. **(b, c)** Shows the marginal distribution of actions $a(t)$ and the latent variable $z(t)$, respectively, at each time step. Overlaid in red are the a- and z- projections, respectively, of trajectories sampled from the velocity field. The action evolves in a narrow Gaussian tube around the demonstration, while the latent variable starts from $\mathcal{N}(0, 1)$ at $t = 0$ and converges to the demonstration trajectory at $t = 1$; see App. C for a full description of the velocity field.

Since (a, z) at time t is a linear transformation of (q_0, z_0), the joint distribution of (a, z) at every timestep is a Gaussian given by:

<u>Joint distribution of (a, z) at each timestep</u>

$$\begin{bmatrix} a \\ z \end{bmatrix} = \underbrace{\begin{bmatrix} e^{-kt} & \sigma_r t \\ 0 & 1-(1-\sigma_1)t \end{bmatrix}}_{A} \begin{bmatrix} a_0 \\ z_0 \end{bmatrix} + \underbrace{\begin{bmatrix} \xi(t) - \xi(0)e^{-kt} \\ t\xi t \end{bmatrix}}_{b} \tag{18}$$

$$p_\xi(a, z \,|\, t) = \mathcal{N}\left(A\mu_0 + b\,,\; A\Sigma_0 A^T\right) \tag{19}$$

$$= \mathcal{N}\left(\begin{bmatrix} \xi(t) \\ t\xi(t) \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{12} & \Sigma_{22} \end{bmatrix}\right) \text{ where} \tag{20}$$

$$\Sigma_{11} = \sigma_0^2 e^{-2kt} + \sigma_r^2 t^2 \tag{21}$$

$$\Sigma_{12} = \sigma_r t\,(1 - (1-\sigma_1)t) \tag{22}$$

$$\Sigma_{22} = (1 - (1-\sigma_1)t)^2 \tag{23}$$

Note that $\mu_0 = \begin{bmatrix} \xi(0) \\ 0 \end{bmatrix}$ and $\Sigma_0 = \begin{bmatrix} \sigma_0^2 & 0 \\ 0 & 1 \end{bmatrix}$.

Since the flow is a diffeomorphism, we can invert it and express (a_0, z_0) as a function of $(a(t), z(t))$:

<u>Inverse of the flow diffeomorphism</u>

$$\begin{aligned} z_0 &= \frac{z - t\xi(t)}{1 - (1-\sigma_1)t} \\ a_0 &= \xi(0) + (a - \xi(t) - (\sigma_r t) z_0)\, e^{kt} \end{aligned} \tag{24}$$

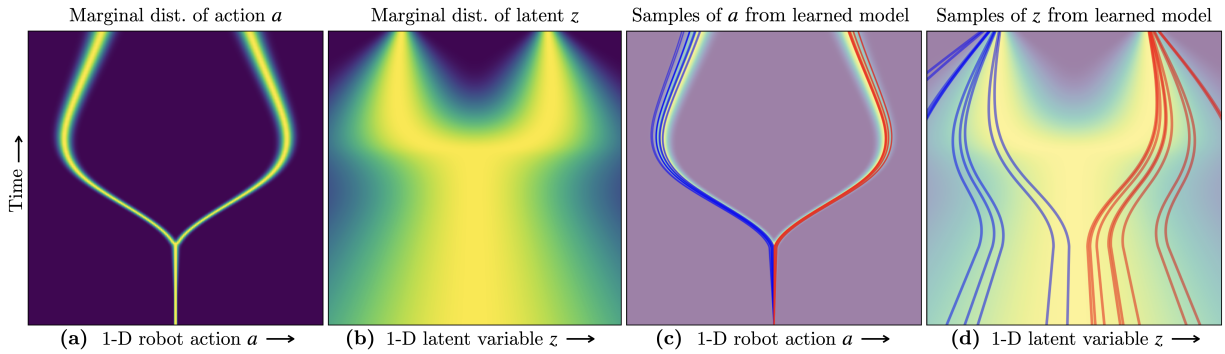

Figure 7: The marginal velocity flow field $v_\theta(a, z, t \mid h)$ learned using the flow construction in Fig. 6. **(a, b)** shows the marginal distribution of actions $a(t)$ and the latent variable $z(t)$, respectively, at each time step under the learned velocity field. **(c, d)** Shows the a- and z- projections, respectively, of trajectories sampled from the learned velocity field. By construction, $a(0)$ deterministically starts from the most recently generated action, whereas $z(0)$ is sampled from $\mathcal{N}(0, 1)$. Trajectories starting with $z(0) < 0$ are shown in blue, and those with $z(0) > 0$ are shown in red. The main takeaway is that in **(c)**, even though all samples deterministically start from the same initial action (*i.e.* the most recently generated action), they evolve in a stochastic manner that covers both modes of the training distribution. This is possible because the stochastic latent variable z is correlated with a, and the initial random sample $z(0) \sim \mathcal{N}(0, 1)$ informs the direction a evolves in.

At time t, the velocity of the trajectory starting from (a_0, z_0) can be obtained by differentiating the flow diffeomorphism in Eq. 17 with respect to t:

<u>Velocity in terms of (a_0, z_0)</u>

$$\begin{aligned} \dot{a}(t \mid \xi, a_0, z_0) &= \dot{\xi}(t) - k\,(a_0 - \xi(0))\,e^{-kt} + \sigma_r z_0 \\ \dot{z}(t \mid \xi, a_0, z_0) &= \xi(t) + t\dot{\xi}(t) - (1 - \sigma_1) z_0 \end{aligned} \tag{25}$$

The flow induces a velocity field at every (a, z, t). The conditional velocity field $v_\theta(a, z, t \mid h)$ by first inverting the flow transformation as shown in Eq. 24, and plugging that into Eq. 25, we get:

<u>Conditional velocity field</u>

$$\begin{aligned} v^a_\xi(a, z, t) &= \dot{\xi}(t) - k\,(a - \xi(t)) + \frac{\sigma_r\,(1 + kt)}{1 - (1 - \sigma_1)t}\,(z - t\xi(t)) \\ v^z_\xi(a, z, t) &= \xi(t) + t\dot{\xi}(t) - \frac{1 - \sigma_1}{1 - (1 - \sigma_1)t}\,(z - t\xi(t)) \end{aligned} \tag{26}$$

Importantly, the evolution of a and z is inter-dependent *i.e.* the sample z_0 determines the evolution of a. Furthermore, the marginal probability distribution $p^a_\xi(a, t)$ can be deduced from the joint distribution in Eq. 20 and is given by:

$$p_\xi(a \mid t) = \mathcal{N}\left(a \,\middle|\, \xi(t)\,,\, \sigma_0^2 e^{-2kt} + \sigma_r^2 t^2\right) \tag{27}$$

In other words, q evolves in a Gaussian tube centered at the demonstration trajectory $\xi(t)$ with a standard deviation that varies from σ_0 at $t = 0$ to σ_1 at $t = 1$. The fact that the marginal distribution lies close to the demonstration trajectories, from Eq. 5 ensures that the per-timestep marginal distributions over actions induced by the learned velocity field are close to training distribution. However, this formulation allows us to select extremely small values of σ_0, essentially deterministically starting from the last generated action a_{prev}. The stochasticity injected by sampling $z_0 \in \mathcal{N}(0, I)$, as well as the correlated evolution of a and z ensures that we sample a diverse distribution of actions in a starting from the same action a_{curr}. This phenomenon is illustrated via a 1-D toy example in Figs. 6 and 7, with details in captions.

D Action Horizon

In Fig. 8, we analyze the effect of action chunk size on the performance of streaming flow policy, under various benchmark environments: (1) `Robomimic: Can`, (2) `Robomimic: Square`, (3) `Push-T` with state input and (4) `Push-T` with image input. The x-axis shows the chunk size in log scale. The y-axis shows the relative decrease in performance compared to that of the best performing chunk size. All scores are less than or equal to zero, where higher is better. In 3/4 environments, the performance peaks at chunk size 8, and 1/4 environments peak at chunk size 6. The performance decreases as the chunk size deviates from the optimum. Our results match with findings from Chi et al. [1], suggesting that behavior cloning policies have a "sweet spot" in the chunk size of the action trajectories. We recommend choosing a larger chunk size (*i.e.* closer to open-loop execution) when the environment dynamics are deterministic and stable. Smaller chunk sizes should be used in stochastic environments with high uncertainty, where the policy may benefit from a tighter feedback loop.

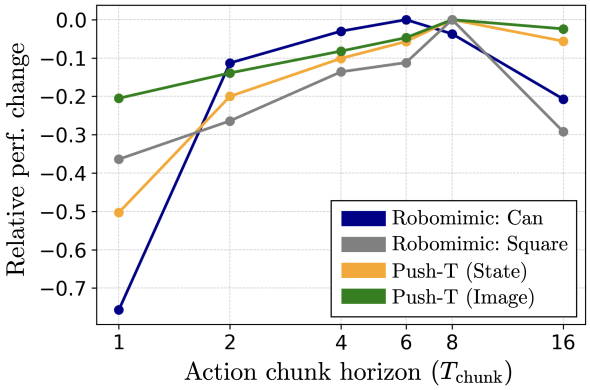

Figure 8: Analysis of the effect of action chunk size on the performance of streaming flow policy, under various benchmark environments. x-axis shows the chunk size, in log scale. y-axis shows the relative decrease in performance compared to that of the best performing chunk size. All scores are less than or equal to zero, where higher is better. In 3/4 environments, the performance peaks at chunk size 8, and the other environment peaks at chunk size 6. The performance decreases as the chunk size increases or decreases from the optimum.

E Push-T experiments with image inputs and action imitation

In this section, we perform experiments in the Push-T environment [1, 14] using images as observations, and imitating actions instead of states (see Sec. 6 for a discussion on state imitation vs. action imitation). This was missing in Table 5 of the main paper.

The conclusions from the table are essentially the same as in the main paper. Streaming flow policy performs nearly as well as the best performing baseline *i.e.* diffusion policy with 100 DDPM inference steps. However, streaming flow policy is significantly faster than diffusion policy. It is also faster than the remaining baselines, while also achieving a higher task success rate.

F CoRL rebuttal responses

In this section, we share our responses during the CoRL 2025 rebuttal process, to help clarify our work to reader who might have similar questions or concerns as our reviewers did.

➤ ***"Streaming flow policy is a special case of diffusion/flow policy with prediction horizon and denoising steps equal to 1."*** We believe that there are important differences between the two. *(i)* At training time, streaming flow policy with prediction horizon learns to forecast 16 future actions given an input observation history, whereas diffusion/flow policy with prediction horizon $T_p = 1$ only learns to predict a single action. This forces streaming flow policy to "implicitly learn a (task-relevant) dynamics model" (Sec. 4.5 in Chi et al. [1]) which is not the case when using diffusion/flow policy

		Push-T with image input	
		Action imitation Avg/Max scores	Latency
		↑	↓
1	DP [1]: 100 DDPM steps	**83.8% / 87.0%**	127.2 ms
2	DP [1]: 10 DDIM steps	80.8% / 85.5%	10.4 ms
3	Flow matching policy [5]	67.9% / 69.3%	12.9 ms
4	Streaming DP [17]	80.5% / 83.9%	77.7 ms
5	**SFP (Ours)**	82.5% / **87.0%**	**08.8 ms**

Table 5: Imitation learning accuracy on the Push-T [1] dataset with images as observation inputs, and imitating action trajectories. Our method (in green) compared against baselines (in red). See text for details.

with only. *(ii)* At test time, diffusion/flow policy with runs *closed-loop* i.e. the input observation history is updated after every single action. Whereas streaming flow policy with predicts future actions conditioned on the *same observation history* and executes this chunk *open-loop*. Executing action sequences open-loop is an important feature of action-chunking [1, 13]; open-loop action chunks are known to reduce distribution shift and improve imitation learning performance.

➤ ***"Noise dilemma: excessive noise in the first action reduces accuracy, while insufficient noisecompromises multimodality."*** Our approach attempts to get the best of both worlds by adding noise at training time but not at test time. *(i)* Representing multimodality is more critical at training time to avoid learning the "average" of multimodal demonstrations [1]. Thus we add sufficient noise at train time to make it easy to learn multimodal distributions. *(ii)* However, at test time, we make the learned policy deterministic by not adding any noise(setting $\sigma_0 = 0$) similar to [13]. This generates accurate, noise-free trajectories at test time. It is acceptable for the policy to be deterministic at test time and sample only one mode (there always exists an optimal deterministic policy).

➤ ***"Multimodality analysis has only been performed in the toy 1D environment."*** During the rebuttal period, we evaluated the multimodality of rollouts of streaming flow policy in the Push-T environment along the lines of Chi et al. [1]. We found that streaming flow policy samples multimodal trajectories in symmetric configurations Fig. 9 (a, b) as reported by Chi et al. [1]. Going a step further, we also analyzed streaming flow policy in *asymmetric* configurations Fig. 9 (c, d) where we also found that streaming flow policy samples are highly multimodal.

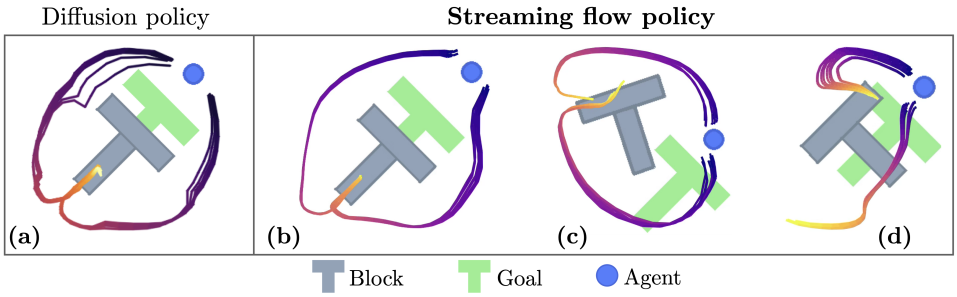

Figure 9: Streaming flow policy can sample multimodal action trajectories, similar to diffusion policy [1] trained on the Push-T dataset. **(a)** Chi et al. [1] report that a diffusion policy trained on the Push-T dataset samples multimodal action trajectories starting from initial configurations that are symmetric with respect to block, goal pose and the agent (pusher). **(b)** We find the same to hold true for a streaming flow policy trained using the same neural network architecture on the Push-T dataset. **(c, d)** Going a step further, we visualize sampled action trajectories from *asymmetric* configurations. Streaming flow policy generates multimodal trajectories, capturing diverse but valid behaviors to push the block towards the goal pose.

➤ ***"Did you compare Streaming Flow Policy with feed-forward velocity field prediction?"*** In the rebuttal period, we implemented this baseline and found that it performed significantly worse than streaming flow policy on `Push-T-State` (82.3% vs. 95.1%), `Push-T-action` (72.1% vs. 91.7%) and RoboMimic-Can (12.4% vs. 95.6%). We hypothesize that the poor performance is due to not

sampling action noise during the training process. Thus,the baseline does not inherit guarantees from Lipman et al. [3] about matching the marginal training distributions.

➤ ***"Streaming flow policy has similar latency as diffusion/flow policy with 10 denoising steps"*** While the latency is similar between streaming flow policy and few-step diffusion/flow policy, the behavior of the two policies is very different. Consider that both have a latency L over T_a actions in a chunk. diffusion/flow policy must wait for L seconds for denoising to complete before the first action can be executed; then all T_a actions can be executed in quick succession. streaming flow policy *spreads out* latency evenly across all actions — it can start executing the first action after L/T_a seconds, and every pair of consecutive actions have the same gap. This *results in much smoother and reactive motion compared to diffusion policy, as shown in our real-world videos on our* ***project website***. Furthermore, we find that diffusion policy suffers a reduction in task performance when we decrease the number of denoising steps to match streaming flow policy's speed.

➤ ***"What are the implications of needing access to time derivatives of action trajectories during training?"*** Unlike diffusion/flow policies, our framework requires the action space to be *continuous* and action trajectories to be *continuous* and *differentiable* with respect to time. This means that certain conventional action spaces, such as binary $\{0, 1\}$ gripper open/close actions, cannot be modeled by streaming flow policy. However, most real-world physical quantities are intrinsically continuous and differentiable with respect to time. For example, we had to modify discrete $\{0, 1\}$ gripper open/close actions to a continuous gripper open fraction in $[0, 1]$; the continuous action is supported by both the real robot and simulation benchmarks. We computed time-derivatives by piecewise-linear interpolation of action sequences. In practice, we found that piecewise-linear derivatives worked well for all tasks without needing more sophisticated smoothing techniques.

➤ ***"How sensitive is streaming flow policy to hyper-parameters*** $k, \sigma_0, \Delta t$***?"*** We found that streaming flow policy is not sensitive to any of k, σ_0 or Δt. The only significant performance drop occurs when the stabilizing gain k is set exactly *equal* to zero, because *stabilization mitigates distribution shift during inference*. However, we found that streaming flow policy is insensitive to a wide range of $k > 0$.

➤ ***"Why does the Streaming Diffusion Policy [17] baseline have high latency despite streaming?"*** The reason streaming diffusion policy [17] is slower than expected is that streaming diffusion policy iteratively denoises every action chunk and not every action. For $N = 100$ denoising steps, prediction horizon T_p and action horizon T_a, the latency per action for streaming diffusion policy is N/T_p, whereas it is N/T_a for diffusion policy. For the choice of hyper-parameters $T_p \approx 16, T_a \approx 8$, used in all three papers (diffusion policy [1], streaming diffusion policy [17] and ours), streaming diffusion policy is only twice as fast as diffusion policy. This can be corroborated by Fig. 4a in Høeg et al. [17]. However, streaming flow policy is an order of magnitude faster than diffusion policy with $N = 100$. Streaming diffusion policy can be sped-up by increasing T_p but only at the cost of reduced task performance, as shown in Fig. 4b in Høeg et al. [17].