

Adaptive Mean Flow for Responsive Closed-Loop Robot Control

Aksel Vaaler¹, Marco Job¹, Christian Holden¹, and Olav Egeland¹

Abstract—Diffusion- and flow-based robot policies have recently become widespread in robotic Imitation Learning (IL) due to their high performance and ability to model continuous and multimodal distributions. However, the iterative denoising procedure used by these models introduces significant prediction latency, hindering high-frequency closed-loop robot control and leading to jittery, unstable motion when frequent updates to the robot’s action predictions are used. Therefore, it is common practice to train models to predict chunks of actions that can be executed sequentially without feedback, even when this reduces responsiveness and may mean the most recent state information is not used. In this article, we present Adaptive Mean Flow (AMF), a flow-based IL method that enables smooth and responsive, fully closed-loop robot control. AMF uses Mean Flow, which is an accelerated form of Flow Matching (FM), to minimize prediction latency. To ensure smoothness and consistency across predictions, AMF uses a corrupted version of the trajectory from the previous step when predicting new robot actions, with the signal-to-noise ratio increasing over the time parameter of the trajectory. This discourages large changes in the prediction from one step to the next, while allowing freedom to adapt the predictions for future steps. We evaluate AMF across a wide range of simulated and real robot tasks and demonstrate significantly improved performance compared with baselines. Code: github.com/akselva/Adaptive-mean-flow-RoboticIL.

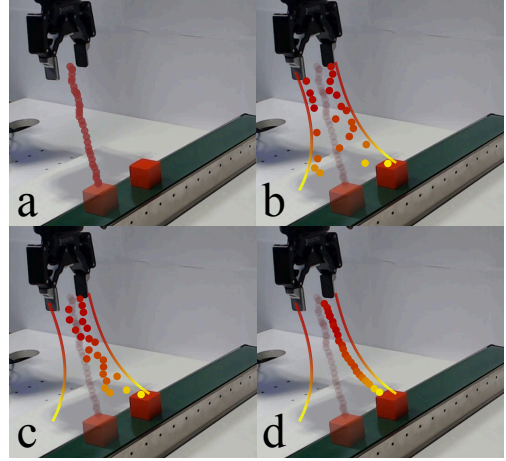


Fig. 1: Trajectory generation with Adaptive Mean Flow visualized for a real-world conveyor belt picking task. **a-b)** Noisy points are sampled in the vicinity of the trajectory computed at the previous step. The noise increases for points along the path, corresponding to predictions farther into the future. **c-d)** The AMF denoising network computes a new trajectory by modifying the noisy sequence.

Diffusion models [1], [2] and Flow Matching (FM) models [3] have recently become widespread in robotic Imitation Learning (IL) [4]. These models capture complex, high-dimensional distributions and are well-suited to model diverse, human-collected robot demonstration trajectories. Diffusion- and flow-based methods work by gradually transforming (typically Gaussian) noise into approximate samples from the data distribution in a process referred to as denoising [1]. This iterative procedure yields high sample quality at the cost of slower inference speed. To take full advantage of the capacity afforded by these models while minimizing the impact of slow inference speed, it is proposed in [5] to predict sequences of future robot actions rather than just the action at the current timestep. This allows for faster execution of robot actions than the prediction speed of the policy, since action predictions from the previous control step can be sequentially passed to the robot while the policy calculates the next action sequence. This technique is called *action chunking* [6], where a “chunk” refers to the actions in a predicted sequence that the robot executes before replanning.

Another benefit of action chunking is improved temporal consistency during deployment [5]. While the demonstration data may exhibit diverse behavior and multimodality on a high level, each individual trajectory sample that the model

is trained on will typically be smooth and self-consistent. This means that subsequent robot actions within a single predicted action sequence are likely to be consistent, while actions from different predicted sequences might diverge. For this reason, it is not uncommon to use a longer action chunk length than what is strictly required, given the inference latency of the diffusion/flow model. On the other hand, using longer chunk lengths decreases the policy’s responsiveness and means that actions are not always taken based on the most recent information, which can be detrimental, especially in the face of environmental disturbances [7]. This can be viewed as a fundamental trade-off between reactivity and consistency. Bridging this gap in diffusion and flow-based robotic IL requires policies that 1) minimize prediction latency to allow faster feedback and 2) maintain smooth and consistent behavior between control steps, while remaining responsive to changes in the environment.

To address these challenges, we propose *Adaptive Mean Flow* (AMF). AMF utilizes Mean Flow (MF) [8], which allows sampling high-quality action sequences from noise in a single denoising step, reducing inference latency. This is crucial for allowing rapid feedback control and high responsiveness. To maintain consistency with previous robot actions while remaining adaptive to the environment’s state, we draw inspiration from [9], [10] and let the noise level vary dynamically across robot actions in a sequence. During inference, instead of predicting new action sequences from pure noise, AMF reuses the previously predicted sequence as a starting point and corrupts this sequence with progressively increasing noise. This encourages near-future action predictions to remain close

¹ Department of Mechanical and Industrial Engineering, Norwegian University of Science and Technology (NTNU) {aksel.vaaler, marco.job, christian.holden, olav.egeland}@ntnu.no

Code: github.com/akselva/Adaptive-mean-flow-RoboticIL

to previous predictions, while allowing actions farther in the future more freedom to adjust in response to the environmental state, as visualized in Fig. 1. We evaluate AMF on a comprehensive set of simulated and real-world robotic manipulation tasks, and show that it exhibits smooth and responsive robot control, outperforming the baselines in both static and dynamic environments.

The contributions of this paper are:

- We present Adaptive Mean Flow (AMF), a novel robotic Imitation Learning policy that enables smooth, responsive closed-loop robot control.
- We empirically validate the proposed method on a diverse set of simulated and real-world robot tasks that require precision, consistency, and responsiveness to be solved reliably.
- We investigate key algorithmic and architectural design choices to optimize AMF performance.

I. RELATED WORK

A. Speeding up diffusion and flow inference

Diffusion and flow models rely on iterative refinement (denoising) to produce high-quality output samples, which means that multiple forward passes of the network are needed per model prediction. Considerable research has been conducted to accelerate the inference process while maintaining quality. For applications with real-time demands, such as robotics, the deterministic Denoising Diffusion Implicit Models (DDIM) sampler proposed in [11] has largely been preferred in favor of the stochastic Denoising Diffusion Probabilistic Models (DDPM) sampler originally proposed in [1], as it generally requires fewer denoising steps to produce good samples. Consistency Policy [12] can produce high-quality robot actions with very few denoising steps by utilizing the diffusion distillation framework developed in [13]. Similarly, One-Step Diffusion Policy [14] proposes a distillation scheme that enables training a student model in a single denoising step to mimic the action predictions of a pre-trained Diffusion Policy [5] network. Streaming Diffusion Policy [9] and Diffusion Forcing [10] train the networks using varying noise levels across the action sequence. This allows inference using a “stair-case” noise structure, where action predictions for near-future time steps are initialized with less noise and thus require fewer denoising steps.

A flow model can itself be viewed as a faster alternative to diffusion, typically requiring far fewer integration steps compared with diffusion and DDPM sampling [3]. However, induced curvature in the probability flow field used to generate samples means that several ODE integration steps are still required [8]. Analogous to Consistency Policy [12], Consistency Flow Policy [15] trains the network with additional consistency constraints on the flow ODE [16] to enable action generation with fewer integration steps. Streaming Flow Policy (SFP) [17] takes a different approach by formulating action trajectories as flows. Starting from the last robot action, SFP treats a step along the flow ODE as a step along the forward robot trajectory, meaning the next robot action is “ready” after a single integration step.

Our AMF method utilizes Mean Flow [8], which was also used in [18], and enables single-step action sequence prediction of high quality without the need for distillation, additional consistency losses, or complex inference algorithms.

B. Consistency-reactivity trade-off in robotic Imitation Learning

The aforementioned works all address the problem of speeding up diffusion/flow inference, which offers immediate benefits in terms of smoother policy execution [17], [9]. However, they still use action chunking to ensure consistent robot motion and thus do not fully utilize the responsiveness enabled by fast inference. Comparatively few works directly address the consistency-reactivity trade-off between action chunking and fully closed-loop, single-step action execution. However, there is a growing interest, particularly in developing test-time algorithms to improve the performance of pre-trained chunking-based policies during deployment. Bidirectional Decoding [7] and Past-Token Prediction [19] introduce algorithms that select the next action sequence from a set of candidates, with selection criteria that aim to balance consistency with the previous prediction and responsiveness to the most recent observation. Self-Guided Action Diffusion [20] and Real-Time Chunking [21] use weighted distances from the previous prediction to the current prediction to steer a diffusion/flow model during the denoising process, producing action sequences that more closely align with the trajectory prediction from the previous step. Temporal Action Selection [22] also employs a pre-trained IL policy, but uses online RL fine-tuning to train an action selector network to adaptively update the action trajectory based on current and previous predictions from the pre-trained policy.

In contrast, our AMF method uses no fine-tuning or additional test-time compute. By training the model on action sequences with dynamic per-step signal-to-noise ratio, flexibility is retained with respect to controlling the behavior of the policy. During deployment, AMF predicts actions rapidly in closed-loop, maintaining consistency simply by initializing the noisy samples around the previous action trajectory.

II. BACKGROUND

Flow Matching: Flow Matching [3], [23] is a framework for learning velocity fields that transport samples from a known (typically Gaussian) prior distribution p_{prior} into approximate samples from an unknown data distribution p_{data} . Given $x \sim p_{\text{data}}$ and $e \sim p_{\text{prior}}$, a linear transport schedule is defined as $z_s = sx + (1 - s)e$ for $s \in [0, 1]$. Taking the derivative with respect to s yields the *conditional velocity* $v_c = x - e$. Flow Matching involves regressing a neural network v_θ towards the conditional velocity:

$$\mathcal{L} = \mathbb{E}_{s,x,e} \|v_\theta(z_s, s) - (x - e)\|^2. \quad (1)$$

It can be shown that the minimization of Eq. 1 is equivalent to the approximation of the *marginal velocity* $v(z_s, s) = \mathbb{E}[v_c|z_s]$ [3]. With a trained network $v_\theta(z_s, s)$, synthetic samples $z_1 \sim q \approx p_{\text{data}}$ can be generated by integrating the ODE $\frac{d}{ds} z_s = v_\theta(z_s, s)$, from $s = 0$ to $s = 1$, starting from $z_0 \sim p_{\text{prior}}$.

Mean Flow: The marginal velocity field $v(z_s, s)$ (and thus v_θ) typically contains curvature due to the expectation over z_s , even though any given conditional velocity $v_c = (x - e)$ is straight [8]. This means that several ODE integration steps utilizing v_θ are needed to obtain high-quality samples z_1 . Mean Flow [8] circumvents this problem by instead modeling the *average velocity* $u(z_s, s, r)$ over the interval $s \rightarrow r$:

$$u(z_s, s, r) = \frac{1}{r - s} \int_s^r v(z_\tau, \tau) d\tau. \quad (2)$$

The estimation of the right-hand side of Eq. 2 is intractable. Instead, a tractable training objective is obtained via the *Mean Flow Identity* [8]:

$$u(z_s, s, r) = v(z_s, s) - (r - s) \frac{d}{ds} u(z_s, s, r). \quad (3)$$

By approximating $v(z_s, s)$ by the conditional velocity $(x - e)$ and u with the neural network u_θ , we obtain the training target $u_{\text{tgt}} = (x - e) - (r - s) \frac{d}{ds} u_\theta(z_s, s, r)$, and the Mean Flow loss:

$$\mathcal{L} = \mathbb{E}_{s,r,x,e} \|u_\theta(z_s, s, r) - \text{sg}(u_{\text{tgt}})\|^2, \quad (4)$$

where $\text{sg}(\cdot)$ indicates the stop-gradient operator, and the derivative $\frac{d}{ds} u_\theta$ in the regression target is computed online using Jacobian-vector products. With a trained approximator u_θ , direct sample generation is possible using $z_1 = z_0 + u_\theta(z_0, s = 0, r = 1)$, where $z_0 \sim p_{\text{prior}}$.

Flow-based robotic IL: Assume that a dataset $\mathcal{D} = \{a_t, o_t\}$ is given, where a_t is the robot action at timestep t and o_t is the corresponding observation of images and/or robot/object states. The dataset $\mathcal{D}$ is collected from an expert policy π_E that we wish to imitate. We follow Diffusion Policy [5] and define the underlying conditional distribution $p_{\text{data}}(A_t|O_t)$ induced by the expert, with action sequences $A_t = a_{t-T_o:t+T_p} - 1$ and observation sequences $O_t = o_{t-T_o:t}$. The action horizon T_p is the number of future action steps to predict (including the current step, t), while the observation horizon T_o is the number of past observations to include as conditioning. Note that the past T_o actions are included in the action sequence A_t . The total sequence length is then $T = T_o + T_p$. Given action sequences A_t and Gaussian noise $e \sim \mathcal{N}(0, I)$, define $z_s = A_t^s = sA_t + (1-s)e$ and $v_c = (A_t - e)$. Then IL policies can be constructed using either Flow Matching or Mean Flows. The FM policy is trained by minimizing the loss (adapted from Eq. 1):

$$\mathbb{E}_{s,A_t,O_t,e} \|v_\theta(A_t^s, O_t, s) - (A_t - e)\|^2. \quad (5)$$

With a trained network, robot action sequences $A_t^{s=1}$ are sampled by integrating $\frac{d}{ds} A_t^s = v_\theta(A_t^s, O_t, s)$, starting from $A_t^0 \sim p_{\text{prior}}$, given the most recent observation sequence O_t . For the MF policy, Eq. 4 is modified to:

$$\mathcal{L} = \mathbb{E}_{s,r,A_t,O_t,e} \|u_\theta(A_t^s, O_t, s, r) - \text{sg}(u_{\text{tgt}})\|^2, \quad (6)$$

where

$$u_{\text{tgt}} = (A_t - e) - (r - s) \frac{d}{ds} u_\theta(A_t^s, O_t, s, r). \quad (7)$$

After training, robot actions can be sampled directly using $A_t^{s=1} = A_t^0 + u_\theta(A_t^0, O_t, 0, 1)$.

III. ADAPTIVE MEAN FLOW

In this section, we present our proposed Adaptive Mean Flow (AMF). We leverage Mean Flow's fast, direct action sampling to achieve tighter closed-loop control. A problem

to be addressed is that frequent replanning can introduce jittery, inconsistent behavior, as it allows the policy to rapidly switch between behavior modes. As noted in [19] and [21], the predicted action sequence at the previous step (A_{t-1}) can be used as an anchor to steer the new action sequence prediction A_t , as long as a subset of the predicted actions from each sequence corresponds to the same timesteps. We take advantage of this, and instead of predicting A_t starting from pure noise $A_t^0 = e$, we start from

$$A_{t-}^s = sA_{t-} + (1-s)e, \quad (8)$$

where

$$A_{t-} = \text{Concat}([A_{t-1}]_{1:T-1}, [A_{t-1}]_{T-1}). \quad (9)$$

A_{t-} is obtained by removing the first element and copying the last element of A_{t-1} , effectively shifting the sequence one step forward. If we write $A_t = A_t(O_t)$ as the current action sequence prediction given the most recent observation sequence, then we can view $A_{t-} \approx A_t(O_{t-1})$ as the overlapping action sequence given the observation sequence from the previous step. The new prediction A_t becomes

$$A_t^{s=1} = A_{t-}^s + (1-s)u_\theta(A_{t-}^s, O_t, s, 1) \quad (10)$$

As we increase s , the relative importance of A_{t-} and thus A_{t-1} increases, as can be seen from Eqs. 8 and 9. Consequently, A_t is likely to remain more similar with A_{t-} .

A drawback of this scheme is that it uses a single value s to specify the noise level uniformly across the sequence. Ideally, we want to keep past and current predictions $a_{t-T_o:t}$ from A_t more closely aligned with A_{t-} , while allowing future predictions $a_{t+1:t+T_p-1}$ more freedom to change with respect to A_{t-} [21]. To enable this, we follow [10], [9] and replace the uniformly perturbed action sequence A_t^s with the variable-noise sequence:

$$\begin{aligned} A_t^S &= [a_{t-T_o}^{s_0}, a_{t-T_o+1}^{s_1}, \dots, a_{t+T_p-1}^{s_{T-1}}] \\ &= [(s_0 a_{t-T_o} + (1-s_0)e_0), \\ &\quad \dots, (s_{T-1} a_{t+T_p-1} + (1-s_{T-1})e_{T-1})]. \end{aligned} \quad (11)$$

To enable training and inference with variable noise levels, the scalar start-point s and end-point r are replaced with the vectors $S = s_{0:T-1}$ and $R = r_{0:T-1}$ of length T .

Inference with varying noise levels: During inference, we wish to use a sequence S that is constant for $t - T_o : t - 1$, and decreases monotonically for $t : t + T_p - 1$ (corresponding to more noise). To this end, we define:

$$\begin{aligned} s_i &= \begin{cases} s_a, & \text{if } i < T_o \\ s_a \left(\frac{s_b}{s_a} \right)^{\lambda_\gamma(i)}, & \text{if } T_o \leq i < T, \end{cases} \\ \lambda_\gamma(i) &= \left(\frac{i - (T_o - 1)}{T - T_o} \right)^\gamma, \text{ for } i \in [T_o, T). \end{aligned} \quad (12)$$

Here, s_a defines the constant flow index for the initial steps $0 : T_o - 1$ (corresponding to $t - T_o : t - 1$), and $s_b < s_a$ defines the index at the final step $T - 1$, while γ decides how quickly it approaches s_b in the intermediate steps $T_o : T - 2$ (corresponding to $t : t + T_p - 2$). The inference procedure is summarized in Algorithm 1. In our implementation, we always set $s_b = 0.0$, meaning that the action corresponding to the last prediction step becomes pure noise. We also set $s_a = 0.5$ and $\gamma = 1.5$ for all experiments.

Training with varying noise levels: When going from con-

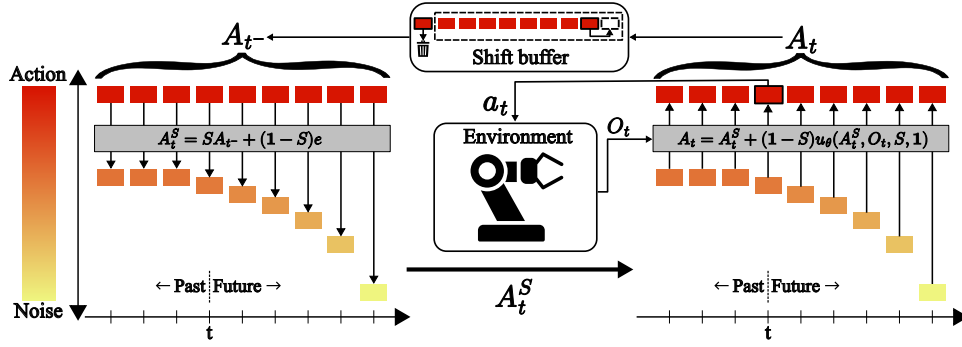


Fig. 2: AMF inference. From the left: The previous action sequence prediction A_{t-} is corrupted with Gaussian noise according to Eq. 12, producing A_t^S . Actions further in the future are more uncertain and, therefore, more corrupted. The Mean Flow network $u_\theta(\cdot)$ predicts the next action sequence A_t , using A_t^S and the most recent observation sequence O_t . The action a_t corresponds to the current time-step sent to the robot for execution, and A_t is shifted (Eq. 9) and reused as A_{t-} for the next control step.

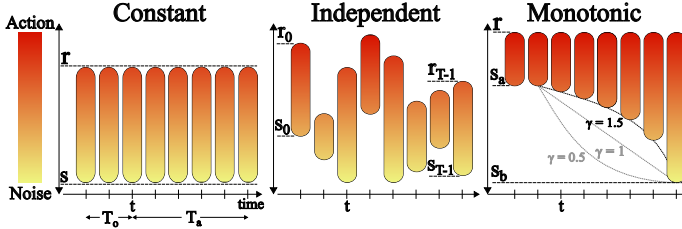


Fig. 3: Flow interval sampling for an example action sequence with $T_o = 2$ and $T_p = 6$. Instead of using the same values s, r across the action sequence (left), AMF uses independent intervals s_i, r_i . During training, we use a mix of independently sampled intervals (middle) and monotonically increasing intervals (right).

stant sequence-wide flow intervals s, r to variable intervals $S = s_{0:T-1}, R = r_{0:T-1}$, we must decide how to sample S and R during training. The most obvious choice is to sample each (s_i, r_i) independently, with the same sampling schedule as used in the constant (s, r) case. For this “independent” regime, we follow [8] and sample pairs (s_i, r_i) using the Log-Norm schedule [24], while making sure $s_i \leq r_i$. This regime ensures high diversity in the sampled noise intervals S, R . However, during inference, we intend to use the monotonically decreasing schedule in Eq. 12 for S , while setting $R = 1$. Thus, to more closely align the distribution of S, R seen during training with the inference-time distribution, with probability p_m we sample S using Eq. 12, with $s_a \sim \mathcal{U}[s_{a,\min}, s_{a,\max}]$, $s_b \sim \mathcal{U}[s_{b,\min}, s_{b,\max}]$ and $\gamma \sim \mathcal{U}[\gamma_{\min}, \gamma_{\max}]$, while R is sampled using a constant $r_{0:T-1} = r$, with r drawn from a LogNorm with expectation close to 1. The training pipeline is presented in Algorithm 2.

Conditioning on flow index intervals S and R : When going from scalar flow intervals (s, r) to interval sequences (S, R) , an important design choice is how to condition the network u_θ on S and R . We conduct experiments using three different conditioning variants: concatenation (**Concat**), addition (**Add**), and feature-wise Adaptive LayerNorm-Zero (**F-AdaLN-Z**) [25]. **Concat** simply involves appending the values (s_i, r_i) to the corresponding noisy action a_i^S . With **Add**, we first project s_i and r_i to the embedding dimension and add them to the action tokens. **F-AdaLN-Z** uses s_i and r_i to parametrize zero-initialized adaptive normalization layers applied individually to each action token (feature).

Predicting past actions: The observation horizon T_o denotes

Algorithm 1: AMF Inference

Require: Mean Flow Network $u_\theta(\cdot)$

- 1: **while** not terminated **do**
- 2: $O_t \leftarrow T_o$ most recent observations
- 3: **if** $t == 0$ **then**
- 4: $S \leftarrow 0$
- 5: $A_{t-} \leftarrow 0$
- 6: **else**
- 7: Construct S with fixed s_a, s_b and γ , (Eq. 12)
- 8: Construct A_{t-} with A_{t-1} , (Eq. 9)
- 9: **end if**
- 10: Sample $e \sim \mathcal{N}(0, I)$
- 11: Construct A_t^S with A_{t-} , e and S , (Eq. 11)
- 12: $A_t \leftarrow A_t^S + (1 - S)u_\theta(A_t^S, O_t, S, 1)$
- 13: **end while**

Algorithm 2: AMF Training

Require: Dataset $\mathcal{D}$, Mean Flow Network $u_\theta(\cdot)$

- 1: **while** not converged **do**
- 2: Sample $(A_t, O_t) \sim \mathcal{D}$
- 3: Sample $p \sim \mathcal{U}[0, 1]$
- 4: **if** $p \leq p_m$ **then**
- 5: Sample $s_a \sim \mathcal{U}[s_{a,\min}, s_{a,\max}]$,
- 6: $s_b \sim \mathcal{U}[s_{b,\min}, s_{b,\max}]$, $\gamma \sim \mathcal{U}[\gamma_{\min}, \gamma_{\max}]$
- 7: Construct S with s_a, s_b and γ , (Eq. 12)
- 8: Sample $R = r \sim \text{LogNorm}(\mu = \mu_h, \sigma = 1.0)$
- 9: **else**
- 10: Sample $s_i, r_i \sim \text{LogNorm}(\mu = \mu_l, \sigma = 1.0)$
- 11: Ensure $s_i \leq r_i$
- 12: **end if**
- 13: Sample $e \sim \mathcal{N}(0, I)$
- 14: Construct A_t^S with A_t , e and S , (Eq. 11)
- 15: $\hat{u} \leftarrow u_\theta(A_t^S, O_t, S, R)$
- 16: Construct u_{tgt} with $A_t, e, \hat{u}, S$ and R , (Eq. 7)
- 17: $\theta \leftarrow \theta - \nabla_\theta \mathcal{L}(\hat{u}, u_{\text{tgt}})$
- 18: **end while**

the number of past observation steps that is used as input when predicting an action sequence. As shown in Figure 2, the policy also predicts the *past actions* $a_{t-T_o:t-1}$. As shown in [19], predicting past robot actions and aligning these predictions with the true past robot trajectory improves temporal action consistency and overall performance. Since AMF initializes the current predicted sequence from the previous, including partially noised actions from the past implicitly encourages consistent policy behavior.

IV. SIMULATION EXPERIMENTS

In this section, we present a comprehensive set of simulated experiments to evaluate the overall efficacy of our proposed method and investigate key design choices.

A. Simulation environments

Metaworld: To assess each method’s capability to perform varied, precise, and smooth robot manipulation, we extract 8 challenging tasks from the widely used Metaworld object manipulation benchmark [26]. For each task, we collect 100 expert demonstrations using pre-trained RL policies included in the official repository of [26]. The action space consists of 3DOF end-effector velocities and gripper open/close. To test policy responsiveness and robustness, we conduct additional experiments in which correlated noise is added to the robot’s actions, requiring policies to adjust quickly to end-effector perturbations in order to solve the tasks.

Push-T: Push-T is a 2D manipulation task where the agent must carefully align a T-shaped object within a goal region. The dataset contains 90 highly multimodal demonstrations collected via human teleoperation, so smoothness and temporal mode consistency are important for solving the task.

Dynamic Block Push (Dyn-BP): This is a dynamic and multimodal 2D manipulation where the agent must push a red and a green block into either of 2 goal zones that are drifting randomly within the environment. The dataset contains 1000 episodes.

D3IL: Lastly we include the Stack-1 and Stack-2 tasks from the D3IL benchmark [27]. D3IL tasks contain multiple solution strategies and the corresponding datasets consist of diverse, human-collected demonstrations. Stack-1 and Stack-2 evaluate the agent’s ability to successfully stack 1 block or 2 blocks, respectively. Dataset size: 1000 episodes.

B. Baselines

We compare AMF against a varied set of baseline policies. All presented policies use a transformer architecture based on [5] as the policy network. Noisy robot actions a^s represent the transformer input tokens, while observations o and flow indices s (and optionally r) are projected to the token dimension and incorporated using cross-attention. The transformer network has 8 layers, 4 heads, and an embedding size of 256. All policies use the past $T_o = 9$ observation frames and predict $T_p = 10$ future actions. For the chunking variants, we set the chunk length to 8. With a control frequency of 10Hz, this gives an open-loop horizon of 0.8s, which has been shown to work well for quasi-static tasks [5].

FM-8, FM-1 and FM-RTC predict robot actions using Flow Matching [3], and are trained using Eq. 5. FM-8 uses a chunk length of $T_a = 8$ (replans every 8th step), while FM-1 uses single-step action execution. FM-RTC uses Real-Time Chunking [21] at inference time, an algorithm that leverages past action sequence predictions to improve consistency across subsequent model predictions. **AFM** is equivalent to AMF but with Flow Matching instead of Mean Flow. AFM uses the same architecture as AMF. All FM-based methods use 10 denoising iterations.

MF-8 and MF-1 use Mean Flow [8] (Eq. 6), and predict action sequences directly from Gaussian noise in a single denoising step. MF-8 and MF-1 use chunk lengths of 8 and 1, respectively. **MF-WarmStart (MF-WS)** reuses past trajectories similar to AMF, but with a constant noise-level across the trajectory instead of increasing noise. MF-WS is evaluated with $T_a = 1$ and a corruption level of $s = 0.5$.

SDP [9] uses variable-noise diffusion to reduce the number of denoising iterations needed to predict action sequences. We implemented SDP using the same transformer architecture and time-step conditioning (AdaLNZero) as AMF. **Consistency Policy (CP)** [12] distills a diffusion-based action policy into a student model using consistency-preserving objective [13], allowing inference with a single denoising step. We evaluate both SDP and CTM with $T_a = 1$.

C. Results

Table I summarizes the main simulation results. AMF performs stably across all tasks and achieves a 6% relative improvement over the best-performing baseline, AFM. The Mean Flow baselines MF-8 and MF-1 perform worse than the Flow Matching baselines, indicating that naively swapping Flow Matching for Mean Flow, while enabling much faster inference, reduces model prediction quality. Note that inference delay does not impact the simulation environments, so we defer this analysis to Section V. AMF (and AFM) performs significantly better than SDP, even though both methods use adaptive noise schedules and the same architecture. AMF fully denoises the predicted trajectory at every step and subsequently corrupts it with fresh noise (according to the adaptive schedule). SDP, by contrast, partially denoises each action in the trajectory at each step, appending fresh noise only at the end (i.e., the farthest future point) of the action sequence. We hypothesize that the added variation provided by the AMF/AFM approach leads to better performance compared with SDP. CP performs relatively poorly, and the training of the student policy was unstable for some tasks. We used the same transformer-based architecture for all baselines, which differs from the Unet architecture used in [12]. However, training instability is a known drawback of consistency models [28].

Metaworld tasks with the largest performance discrepancies, such as Basketball, Box-close, and Pick-out-of-hole, generally require very precise manipulation to succeed. Indeed, we observe that AMF produces smooth and accurate robot trajectories. Push-T, on the other hand, also requires occasional trajectory “switches” to solve reliably, e.g., switching from pushing on the side of the block to pushing from the top to align it vertically. If the policy is overly biased towards smooth motion, it is prone to getting stuck in a single task phase and eventually stalling. However, this is not the case for AMF, which we attribute to the increased corruption applied to future action predictions, providing sufficient flexibility to switch trajectories when needed.

Smoothness: In Table II, we report trajectory smoothness estimates on the Metaworld action trajectories computed using third-order finite differences. AMF exhibits considerably less

TABLE I: Main simulation results. For Push-T and Metaworld, the models are trained for 500 epochs, while for Dyn-BP and D3IL (Stack-1 and Stack-2) the models are trained for 200 and 90 epochs, respectively. We use a batch size of 256, and model checkpoints are collected every 10th epoch. For Push-T and Dyn-BP, the score is the maximum goal overlap ratio achieved during an episode, whereas for Metaworld and D3IL, the score is 1 for success and 0 otherwise. Each model rollout consists of 50 tests, and we report averaged metrics over the last three model checkpoints, across three seeds. Standard deviation is shown in parentheses.

													
Method	Overlap	Overlap	SR	SR	SR	SR	SR	SR	SR	SR	SR	SR	Avg
FM-8	0.73 (0.05)	0.21 (0.01)	0.67 (0.07)	0.24 (0.07)	0.76 (0.10)	0.85 (0.07)	0.31 (0.03)	0.86 (0.07)	0.95 (0.03)	0.59 (0.05)	0.90 (0.07)	0.92 (0.04)	0.67
FM-1	0.83 (0.02)	0.31 (0.03)	0.43 (0.05)	0.08 (0.04)	0.95 (0.05)	0.88 (0.02)	0.59 (0.12)	0.94 (0.03)	0.99 (0.01)	0.50 (0.22)	0.83 (0.06)	0.92 (0.04)	0.69
FM-RTC	0.79 (0.01)	0.30 (0.02)	0.46 (0.12)	0.08 (0.06)	0.92 (0.07)	0.90 (0.05)	0.57 (0.13)	0.98 (0.03)	0.99 (0.01)	0.50 (0.18)	0.81 (0.06)	0.90 (0.04)	0.68
AFM	0.82 (0.05)	0.44 (0.04)	0.63 (0.05)	0.24 (0.04)	0.93 (0.01)	0.79 (0.32)	0.92 (0.04)	1.00 (0.00)	1.00 (0.00)	0.63 (0.09)	0.84 (0.05)	0.93 (0.03)	0.76
MF-8	0.65 (0.04)	0.21 (0.02)	0.20 (0.24)	0.06 (0.07)	0.20 (0.12)	0.48 (0.20)	0.63 (0.14)	0.87 (0.14)	0.96 (0.06)	0.08 (0.03)	0.93 (0.04)	0.90 (0.05)	0.51
MF-1	0.75 (0.02)	0.34 (0.06)	0.03 (0.08)	0.01 (0.02)	0.41 (0.10)	0.59 (0.14)	0.51 (0.08)	0.96 (0.06)	1.00 (0.00)	0.17 (0.09)	0.78 (0.07)	0.84 (0.05)	0.53
MF-WS	0.81 (0.04)	0.30 (0.04)	0.12 (0.14)	0.01 (0.02)	0.92 (0.03)	0.86 (0.03)	0.79 (0.12)	0.97 (0.05)	1.00 (0.00)	0.33 (0.04)	0.87 (0.02)	0.95 (0.04)	0.66
SDP	0.90 (0.03)	0.39 (0.00)	0.56 (0.10)	0.21 (0.08)	0.76 (0.06)	0.67 (0.18)	0.92 (0.04)	1.00 (0.00)	0.18 (0.13)	0.90 (0.05)	0.79 (0.16)	0.53 (0.14)	0.65
CP	0.34 (0.33)	0.03 (0.00)	0.00 (0.00)	0.00 (0.00)	0.67 (0.25)	0.01 (0.01)	0.2 (0.1)	0.68 (0.44)	0.56 (0.38)	0.02 (0.00)	0.83 (0.12)	0.81 (0.10)	0.35
AMF (ours)	0.92 (0.02)	0.50 (0.05)	0.58 (0.08)	0.27 (0.08)	0.97 (0.02)	1.00 (0.00)	0.86 (0.07)	1.00 (0.00)	1.00 (0.00)	0.92 (0.05)	0.83 (0.04)	0.93 (0.04)	0.82

jerkiness compared with MF-1 and FM-1, and also compared with the constant-noise trajectory reuse baseline (MF-WS). We also see that the Mean Flow baseline MF-1 is significantly less smooth than its Flow Matching counterpart, which helps explain the performance difference between these two baselines.

Robustness to perturbation: One of the main reasons for using single-step (i.e., fully closed-loop) action execution as opposed to action chunking is to increase robustness to environmental disturbances. In the Dyn-BP task, we see a clear trend where the chunking policies FM-8 and MF-8 perform significantly worse than their single-step variants, which indicates that reactivity is indeed needed to align the blocks with the moving goal zones. To further analyze this, we augment Metaworld with dynamic robot perturbations. The perturbations are implemented as temporally correlated Ornstein-Uhlenbeck noise

$$\epsilon_t = \rho \epsilon_{t-1} + \sqrt{1 - \rho^2} \sigma w_t, \quad w_t \sim \mathcal{N}(0, I), \quad (13)$$

which is added to the robot actions as $\tilde{a}_t = a_t + \epsilon_t$. ρ is set to a fixed value, so the perturbation strength is tuned by varying σ . Figure 4 compares AMF to selected baselines on two Metaworld tasks, for a wide range of perturbation magnitudes. AMF consistently shows the greatest robustness. The chunking baseline FM-8 performs the worst on average, consistent with the intuition that open-loop execution makes policies less robust to dynamic disturbances. This is particularly evident on the *Faucet-close* task.

Design ablations: Table III compares the performance of key design variations on Push-T and Metaworld. The best performance is achieved using feature-wise Adaptive LayerNorm-Zero conditioning (**F-AdaLN-Z**) [25] for the flow intervals (S, R). With this formulation, the values for s_i and r_i are incorporated into the corresponding action tokens in every transformer block, permitting more fine-grained adaptation compared with the **Concat** and **Add** variants, which only apply the conditioning once at the input to the transformer. We also see that setting $p_m = 0.5$, i.e., training the model with a mix of sequences with independently sampled flow intervals and sequences with monotonically increasing intervals (Eq. 12) yields the best performance, indicating that exposing the policy

TABLE II: Trajectory smoothness analysis on Metaworld. We quantify action trajectory smoothness using the Euclidean norm of the third finite difference of consecutive Cartesian actions. We report its mean and maximum over evaluation trajectories, where lower values indicate smoother action sequences. The mean and maximum is computed for each task, then averaged across all tasks.

Method	Mean	Max
FM-1	1.16	26.67
MF-1	2.19	29.69
MF-WS	1.33	23.63
AMF (Ours)	0.98	15.96

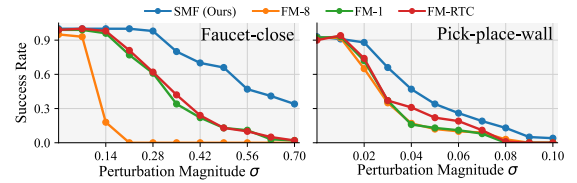


Fig. 4: Success rate on 2 Metaworld tasks with increasing perturbations added to the end-effector. Each policy obtains a close to 100% success rate without perturbation; however, the performance of AMF (blue curve) consistently degrades the least as the perturbation magnitude σ (and thus the task difficulty) increases.

to both corruption patterns during training is beneficial. We also tested a variant of AMF that removes the conditioning on the end-points R , since this sequence is always set to 1 during inference. This variant performs almost as well as the base AMF, indicating that strong performance is achievable even without explicit end-point conditioning.

Figure 5 shows performance on the Push-T task for AMF and MF-WS, with varying levels of noise corruption. Performance remains relatively stable across this range.

V. REAL-WORLD EVALUATION

We conduct real-world experiments on three challenging robot manipulation tasks, as visualized in Figure 6.

Real-world Push-T: This is a real-world implementation of the Push-T task [5]. The starting position and orientation of the T-shaped block is randomized, and the policies must use a cylindrical rod attached to the end-effector to push the block

TABLE III: Performance impact from selected design variations. The type of conditioning used for the flow intervals (S, R) is denoted in bold. p_m is the probability of sampling interval sequences (S, R) using the monotone schedule (Eq. 12), leaving $(1 - p_m)$ probability of sampling each (s_i, r_i) independently.

Ablation	Push-T (Overlap)	Metaworld Avg. (SR)
F-AdaLN-Z / $p_m = 0.5$	0.92 (0.02)	0.95
Concat / $p_m = 0.5$	0.86 (0.01)	0.90
Add / $p_m = 0.5$	0.85 (0.00)	0.92
F-AdaLN-Z / $p_m = 0.0$	0.89 (0.02)	0.93
F-AdaLN-Z / $p_m = 1.0$	0.88 (0.01)	0.77
S -conditioning only	0.9 (0.02)	0.93

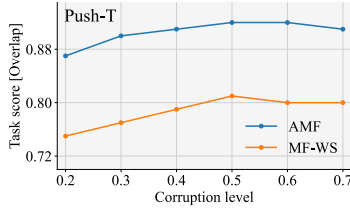


Fig. 5: Task score on Push-T for AMF and MF-WS with varying levels of noise corruption. For AMF, the value on the x-axis corresponds to the starting noise s_a , while $s_b = 1.0$ and $\gamma = 1.5$. For MF-WS, the value corresponds to the constant noise corruption level.

into the marked goal location. As in the simulated version, the dataset is highly multimodal and contains trajectories collected with different solution strategies.

Box-close: In this task, a lid with a handle must be picked up and accurately placed on the corresponding box. The starting orientations and positions of both the lid and box are randomized, and correctly placing the lid requires a high degree of precision.

Conveyor Belt: A miniature conveyor belt of length 90 cm and width 10 cm is placed in front of the robot, and the task is to intercept a red block moving along the belt and place it in the container. The belt speed varies, so the agent must adapt in real time to successfully pick up the block.

The Real-world Push-T task is implemented using a Universal Robots UR10 manipulator, and the action space consists of 2D delta position references in the xy -plane. The Box-close and Conveyor Belt tasks use a Franka Emika Panda setup with a 5-dimensional action space comprising xyz -translation, yaw rotation, and gripper open/close. For all tasks, the observations consist of end-effector poses and RGB frames from two RealSense D435 cameras, one mounted on the robot wrist and the other on a camera stand overlooking the workspace. The frames are processed by a pair of ResNet-18 image encoders [29] trained end-to-end alongside the robot policies. All demonstrations are conducted with a control frequency of 10 Hz. Model inference was conducted locally on a single Nvidia RTX 3090 GPU.

A. Results

Results for the real Push-T and Box-close tasks are shown in Table IV. AMF performs best overall and significantly outperforms the baselines on the real Push-T task. Visually, both FM-8 and FM-RTC were noticeably less smooth than AMF, and a typical failure mode for the baselines was abrupt

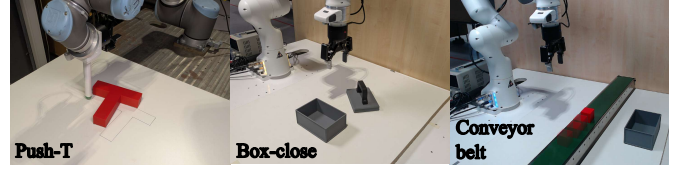


Fig. 6: Real-world manipulation tasks.

TABLE IV: Results on static real-world tasks. Each policy is trained for 600 epochs and evaluated on 20 random initial configurations for each task. For Push-T, the policies receive a score of 1 for successful alignment and 0 otherwise. For Box-close, correctly picking the lid yields 0.5 score, while correctly placing it yields an additional 0.5.

Method	Real Push-T	Real Box-close	Inference latency
FM-8	0.35	0.48	54 ms
FM-RTC	0.35	0.43	79 ms
AMF (Ours)	0.60	0.58	14 ms

movements that misaligned the T-block in the Push-T case or resulted in an incorrect grip on the Box-close task. A likely contributor is inference latency, which is reported in the rightmost column of Table IV. The RTC inference algorithm [21] is designed to increase smoothness, but comes at the cost of even slower inference compared with vanilla FM. AMF combines fast inference speed and the smoothing effect of reusing previous predictions to minimize jerkiness. AMF would, however, sometimes become “stuck” before fully completing the tasks. For example, almost all AMF failures on the Box-close task were due to not releasing the grip on the lid after aligning it with the box.

Table V shows success rates on the Conveyor Belt task for increasing belt speeds. The policies were trained on demonstrations with randomized speeds approximately in the range of 10–16 cm/s. While each method is quite competitive, AMF performs best overall. All the policies consistently tracked the end-effector relatively close to the block, and most failures were due to slight over- or undershoots, as exemplified in Figure 7. AMF demonstrated the most consistency in executing the fine adjustments needed to fully align the grippers with the moving block. Curiously, all policies performed better at 15.3 cm/s than at 13.6 cm/s. This might indicate that some speeds were underrepresented in the data, and that this factor matters as much as the inherent increase in difficulty with higher speeds.

VI. CONCLUSION

In this paper, we presented Adaptive Mean Flow, a robotic IL method that enables flexible, responsive, and smooth robot

TABLE V: Success rate on conveyor belt picking task. Each policy is trained for 600 epochs. We conduct experiments at 4 different belt speeds, corresponding to DC belt motor actuation with 10–13V. We perform 10 trials for each method at each speed setting, with slight variations in the initial block position on the belt.

Method	11.5 cm/s	12.5 cm/s	13.6 cm/s	15.3 cm/s	Avg
FM-8	0.70	0.70	0.20	0.30	0.48
FM-RTC	0.50	0.50	0.10	0.50	0.40
AMF (Ours)	0.80	0.50	0.30	0.60	0.55

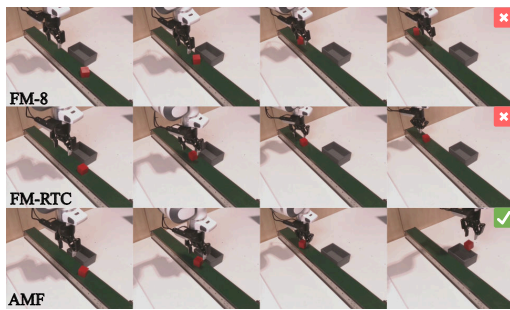


Fig. 7: Rollout comparison on real-world conveyor belt task, with the maximum belt speed of 15.3 cm/s. **Top:** FM-8 initially misjudges the trajectory needed to intercept the block, and is unable to recover, eventually ending up behind the object and failing the task. **Middle:** FM with real-time chunking adjusts more aggressively, but closes the gripper prematurely and misses the object. **Bottom:** AMF tracks slightly behind initially, but performs the necessary adjustments to catch up and successfully grasp the block.

control. We demonstrated the efficacy of AMF on a comprehensive set of simulated and real-world tasks, both with and without dynamic components. We also presented training recipes and dedicated architectural modifications to optimize AMF performance. A limitation of the current approach is that all action modalities are treated the same. The proposed scheme was primarily designed with continuous modalities in mind, such as end-effector poses, and not discrete modes such as binary gripper commands. Indeed, a common failure mode for AMF was an inability to open/close the gripper when required. Future work will investigate modality-specific strategies to improve generalizability and robustness. There is also room to explore alternative corruption schedule formulations and parametrizations, and further improve and optimize the adaptive noise corruption scheme.

REFERENCES

- [1] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020. **1, 2**
- [2] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep unsupervised learning using nonequilibrium thermodynamics,” in *Proc. of the 32nd Int. Conf. on Machine Learning*, vol. 37, 2015, pp. 2256–2265. **1**
- [3] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, “Flow matching for generative modeling,” in *The 11th Int. Conf. on Learning Representations*, 2023. **1, 2, 5**
- [4] R. Wolf, Y. Shi, S. Liu, and R. Rayyes, “Diffusion models for robotic manipulation: a survey,” *Frontiers in Robotics and AI*, vol. 12, 2025. **1**
- [5] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, “Diffusion policy: Visuomotor policy learning via action diffusion,” *The Int. Journal of Robotics Research*, 2023. **1, 2, 3, 5, 6**
- [6] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, “Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware,” in *Proc. of Robotics: Science and Systems*, 2023. **1**
- [7] Y. Liu, J. I. Hamid, A. Xie, Y. Lee, M. Du, and C. Finn, “Bidirectional decoding: Improving action chunking via guided test-time sampling,” in *The 13th Int. Conf. on Learning Representations*, 2025. **1, 2**
- [8] Z. Geng, M. Deng, X. Bai, J. Z. Kolter, and K. He, “Mean flows for one-step generative modeling,” in *The 39th Annual Conf. on Neural Information Processing Systems*, 2025. **1, 2, 3, 4, 5**
- [9] S. H. Høeg, Y. Du, and O. Egelund, “Fast policy synthesis with variable noise diffusion models,” in *IEEE Int. Conf. on Robotics and Automation*, 2025, pp. 4821–4828. **1, 2, 3, 5**
- [10] B. Chen, D. Martí Monsó, Y. Du, M. Simchowitz, R. Tedrake, and V. Sitzmann, “Diffusion forcing: Next-token prediction meets full-sequence diffusion,” in *Advances in Neural Information Processing Systems*, vol. 37, 2024, pp. 24 081–24 125. **1, 2, 3**
- [11] J. Song, C. Meng, and S. Ermon, “Denoising diffusion implicit models,” in *The 9th Int. Conf. on Learning Representations*, 2021. **2**
- [12] A. Prasad, K. Lin, J. Wu, L. Zhou, and J. Bohg, “Consistency policy: Accelerated visuomotor policies via consistency distillation,” in *Proc. of Robotics: Science and Systems*, 2024. **2, 5**
- [13] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever, “Consistency models,” in *Proc. of the 40th Int. Conf. on Machine Learning*, vol. 202, 2023, pp. 32 211–32 252. **2, 5**
- [14] Z. Wang, M. Li, A. Mandelkar, Z. Xu, J. Fan, Y. Narang, L. Fan, Y. Zhu, Y. Balaji, M. Zhou, M.-Y. Liu, and Y. Zeng, “One-step diffusion policy: Fast visuomotor policies via diffusion distillation,” in *42nd Int. Conf. on Machine Learning*, 2025. **2**
- [15] Q. Zhang, Z. Liu, H. Fan, G. Liu, B. Zeng, and S. Liu, “Flowpolicy: Enabling fast and robust 3D flow-based policy via consistency flow matching for robot manipulation,” in *Proc. of the AAAI Conf. on Artificial Intelligence*, vol. 39, 2025, pp. 14 754–14 762. **2**
- [16] L. Yang, Z. Zhang, Z. Zhang, X. Liu, M. Xu, W. Zhang, C. Meng, S. Ermon, and B. Cui, “Consistency flow matching: Defining straight flows with velocity consistency,” *arXiv preprint arXiv:2407.02398*, 2024. **2**
- [17] S. Jiang, X. Fang, N. Roy, T. Lozano-Pérez, L. P. Kaelbling, and S. Ancha, “Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories,” in *9th Annual Conf. on Robot Learning*, 2025. **2**
- [18] J. Sheng, Z. Wang, P. Li, and M. Liu, “Mp1: Meanflow tames policy learning in 1-step for robotic manipulation,” in *Proc. of the AAAI Conf. on Artificial Intelligence*, vol. 40, 2026, pp. 18 532–18 539. **2**
- [19] M. T. Villasevil, A. Tang, Y. Liu, and C. Finn, “Learning long-context diffusion policies via past-token prediction,” in *9th Annual Conf. on Robot Learning*, 2025. **2, 3, 4**
- [20] R. Malhotra, Y. Liu, and C. Finn, “Self-guided action diffusion,” *arXiv preprint arXiv:2508.12189*, 2025. **2**
- [21] K. Black, M. Y. Galliker, and S. Levine, “Real-time execution of action chunking flow policies,” in *Advances in Neural Information Processing Systems*, 2025. **2, 3, 5, 7**
- [22] Y. Weng, X. Zhang, Y. Mu, Y. Zhu, Y. Li, and Q. Liu, “Temporal action selection for action chunking,” *arXiv preprint arXiv:2511.04421*, 2025. **2**
- [23] X. Liu, C. Gong, and Q. Liu, “Flow straight and fast: Learning to generate and transfer data with rectified flow,” in *The 11th Int. Conf. on Learning Representations*, 2023. **2**
- [24] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, D. Podell, T. Dockhorn, Z. English, K. Lacey, A. Goodwin, Y. Marek, and R. Rombach, “Scaling rectified flow transformers for high-resolution image synthesis,” in *41st Int. Conf. on Machine Learning*, 2024. **4**
- [25] W. Peebles and S. Xie, “Scalable diffusion models with transformers,” in *IEEE/CVF Int. Conf. on Computer Vision*, 2023, pp. 4172–4182. **4, 6**
- [26] R. McLean, E. Chatzaroulas, L. McCutcheon, F. Röder, T. Yu, Z. He, K. Zentner, R. Julian, J. K. Terry, I. Woungang, N. Farsad, and P. S. Castro, “Meta-world+: An improved, standardized, RL benchmark,” in *The 39th Annual Conf. on Neural Information Processing Systems*, 2025. **5**
- [27] X. Jia, D. Blessing, X. Jiang, M. Reuss, A. Donat, R. Lioutikov, and G. Neumann, “Towards diverse behaviors: A benchmark for imitation learning with human demonstrations,” in *The 12th Int. Conf. on Learning Representations*, 2024. **5**
- [28] Y. Kim, D. Kim, W. Kim, and J. Park, “Stabilizing consistency training: A flow map analysis and self-distillation,” *arXiv preprint arXiv:2601.22679*, 2026. **5**
- [29] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. of the IEEE Conf. on computer vision and pattern recognition*, 2016, pp. 770–778. **7**