APPENDIX

A. Additional Details on Flow Matching in SE(3)

In Section III, we introduced Flow Matching in the Lie group SE(3). For completeness, we provide the pseudo-code for both training (Algorithm 1) and sampling (Algorithm 2) with flow-based models.

Algorithm 1: Flow Matching Training for Action Poses

repeat
  $\boldsymbol{r}_a, \boldsymbol{p}_a, \boldsymbol{O} \sim \mathcal{D}$ `# Sample a batch of actions and observations from the dataset`
  $\boldsymbol{r}_0, \boldsymbol{p}_0, \sim q_0$ `# Sample random pose from the initial distribution`
  $t \sim$ Uniform($\{0, \dots, 1\}$) `# Sample timestep`
  $\boldsymbol{r}_t, \boldsymbol{p}_t \leftarrow \boldsymbol{\phi}(\boldsymbol{r}_0, \boldsymbol{p}_0, \boldsymbol{r}_a, \boldsymbol{p}_a, t)$`# Compute flow given Equation (2)`
  $\dot{\boldsymbol{r}}_t, \dot{\boldsymbol{p}}_t \leftarrow \boldsymbol{u}(\boldsymbol{r}_t, \boldsymbol{p}_t, \boldsymbol{r}_a, \boldsymbol{p}_a, t)$ `# Compute target vector given Equation (3)`
  $\boldsymbol{v}_p, \boldsymbol{v}_r \leftarrow \boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{r}_t, \boldsymbol{p}_t, t, \boldsymbol{O})$ `# Predict vector given the learnable model`
  $\mathcal{L} = ||\boldsymbol{v}_p - \dot{\boldsymbol{p}}_t||^2 + ||\boldsymbol{v}_r - \dot{\boldsymbol{r}}_t||^2$ `# Compute L2 loss`
  $\boldsymbol{\theta} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}$ `# Take gradient step`
until converged

Algorithm 2: Sampling Action Poses from flow-based models

Require: An observation $\boldsymbol{O}$
$\boldsymbol{r}_t, \boldsymbol{p}_t, \sim q_0$ `# Sample random pose from initial distribution`
for $t = 0, \dots, 1$ **do**
  $\boldsymbol{v}_p, \boldsymbol{v}_r \leftarrow \boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{r}_t, \boldsymbol{p}_t, t, \boldsymbol{O})$ `# Predict vector given the learned model`
  $\boldsymbol{r}_t, \boldsymbol{p}_t \leftarrow$ EulerStep$(\boldsymbol{r}_t, \boldsymbol{p}_t, \boldsymbol{v}_p, \boldsymbol{v}_r, \Delta t)$ `# Update the pose given Equation (4)`
end for
return $\boldsymbol{r}_t, \boldsymbol{p}_t$

B. Flow-based Policies in Euclidean Space

In this section, we describe how to represent a Flow Matching based policy in the Euclidean space. We provide the details as we actually evaluated the performance of Euclidean Flow Matching policies in Section IV-A. We decided to use Flow Matching in the Euclidean space in Section IV-A, as it allowed for a fairer comparison with the other baselines. We propose modeling a policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{o})$ as a *Continuous Normalizing Flow* (CNF) [30] trained via CFM (Equation (1)). Flow-based policies are ***expressive***, able to represent multimodal action distributions yet ***simple***. Additionally, the training is stable, and the sampling method is simple and deterministic. Similar to previous works [2], [4], we represent the action space as a trajectory of future actions.

The problem in flow matching boils down to designing a conditioned flow that drives randomly sampled points to the dataset. In the following, we present a popular flow (Rectified Linear Flow) and showcase how it can be used to generate robot actions.

Rectified Linear Flow [29], [24], [26] propose representing the data point conditioned flow $\boldsymbol{\phi}_t(\boldsymbol{a}|\boldsymbol{a}_1)$ with a straight line from a noisy sample $\boldsymbol{a}_0 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ at $t = 0$ to the datapoint $\boldsymbol{a}_1 \in \mathcal{D}$ at $t = 1$

$$\boldsymbol{a}_t = \boldsymbol{\phi}_t(\boldsymbol{a}_0|\boldsymbol{a}_1) = t\boldsymbol{a}_1 + (1-t)\boldsymbol{a}_0. \tag{5}$$

Then, sampling from the conditioned probability $\rho_t(\boldsymbol{a}|\boldsymbol{a}_1)$ can be easily done by sampling $\boldsymbol{a}_0 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and computing the flow at time t with Equation (5). By differentiating Equation (5), the conditional vector field is $\boldsymbol{u}_t(\boldsymbol{a}|\boldsymbol{a}_1) = \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\phi}_t(\boldsymbol{a}|\boldsymbol{a}_1) = \frac{\boldsymbol{a}-\boldsymbol{a}_1}{1-t}$. Notice that $\boldsymbol{u}_t$ is a constant velocity for any time t defined by $\boldsymbol{u}_t(\boldsymbol{a}|\boldsymbol{a}_1) = \boldsymbol{a}_1 - \boldsymbol{a}_0$. An interesting property of the straight path given by the Rectified Linear Flow is that it will incur small errors with numerical solvers, an essential property if we aim to sample with very few iterations.

Training. Given a dataset $\mathcal{D} : \{\boldsymbol{a}_n, \boldsymbol{o}_n\}_{n=0}^N$, we train a context and time-conditioned vector field $\boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{a}, \boldsymbol{o}, t)$ by regressing a designed vector field $\boldsymbol{u}_t$

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{a}_1, \boldsymbol{c}) \sim \mathcal{D}, t, \rho_t(\boldsymbol{a}|\boldsymbol{a}_1)} \left\| \boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{a}, \boldsymbol{c}, t) - \boldsymbol{u}_t(\boldsymbol{a}|\boldsymbol{a}_1) \right\|^2, \tag{6}$$

with $t \sim \mathcal{U}[0, 1]$ and ρ_t and $\boldsymbol{u}_t$ the probability path and vector field given by the Rectified Linear Flow.

Sampling. In our work, we propose sampling from $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{c})$ by naively applying Euler discretization. We first sample from $\boldsymbol{a}_0 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, and iteratively apply Euler discretization for K steps

$$\boldsymbol{a}_{k+1} = \boldsymbol{a}_k + \boldsymbol{v}_{\boldsymbol{\theta}}(\boldsymbol{a}_k, \boldsymbol{c}, k\Delta t)\Delta t, \tag{7}$$

with $\Delta t = 1/K$. Notice that we can naively play with the iterations and the Δt to find the optimal sampling for our task. In particular, in Section IV-A, we observe that we can obtain highly accurate samples with very few iterations.

C. Equivariant Generation with an Invariant Model

This section provides additional details explaining how exactly we obtain SE(3) equivariant action generation, given that the underlying transformer model is SE(3) invariant.
Given a policy $\pi(\boldsymbol{T}_a|\boldsymbol{F}_o, \boldsymbol{T}_o)$ that generates action poses $\boldsymbol{T}_a$, given the observation poses $\boldsymbol{T}_o$, the policy is SE(3) equivariant if under a transformation $T_\delta \in SE(3)$ over the observations, the distribution over the actions is similarly transformed, i.e., $\pi(\boldsymbol{T}_a|\boldsymbol{F}_o, \boldsymbol{T}_o) = \pi(T_\delta \boldsymbol{T}_a|\boldsymbol{F}_o, T_\delta \boldsymbol{T}_o)$.

Our proposed ActionFlow achieves equivariance by updating the action poses w.r.t. their own local frame. This results in equivariant action generation as long as the underlying model is invariant, as we will show in the following. This property has been previously exploited in protein folding problems [19], [21], [20].

Let us consider the update rule represented in Equation (4)

$$\begin{aligned} \boldsymbol{p}_{k+1} &= \boldsymbol{p}_k + \boldsymbol{r}_k \boldsymbol{v}_{\boldsymbol{\theta}}(T_k, \boldsymbol{T}_o, \boldsymbol{F}_o, t)\Delta t \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k \mathrm{Exp}(\Delta t \boldsymbol{v}_{\boldsymbol{\theta}}(T_k, \boldsymbol{T}_o, \boldsymbol{F}_o, t)) \end{aligned} \tag{8}$$

with $\boldsymbol{p}$ being the translation in the world frame, $\boldsymbol{r}$ the rotation matrix in the world frame, and the step length Δt. Importantly, the current pose's rotation matrix $\boldsymbol{r}_k$ is

premultiplied to the predicted update vectors $\boldsymbol{v}_{\boldsymbol{\theta}}$. Therefore, the predicted update vector operates in the local frame. We aim for equivariant generation, such that if we apply a transformation $T_\delta = (\boldsymbol{p}_\delta, \boldsymbol{r}_\delta) \in SE(3)$ over the current pose $T_k = (\boldsymbol{p}_k, \boldsymbol{r}_k)$, i.e.,

$$T'_k = (\boldsymbol{p}'_k, \boldsymbol{r}'_k) = (\boldsymbol{r}_\delta \boldsymbol{p}_k + \boldsymbol{p}_\delta, \boldsymbol{r}_\delta \boldsymbol{r}_k), \tag{9}$$

and observations $\boldsymbol{T}'_o = T_\delta \boldsymbol{T}_o$, the updated pose T'_{k+1} is by definition similarly transformed, i.e., $T'_{k+1} = (\boldsymbol{p}'_{k+1}, \boldsymbol{r}'_{k+1}) = (\boldsymbol{r}_\delta \boldsymbol{p}_{k+1} + \boldsymbol{p}_\delta, \boldsymbol{r}_\delta \boldsymbol{r}_{k+1})$.

To showcase that for equivariant action generation, the model should be invariant, we start by considering the update equations for the transformed poses. They equate to

$$\begin{aligned} \boldsymbol{p}'_{k+1} &= \boldsymbol{p}'_k + \boldsymbol{r}'_k \boldsymbol{v}_{\boldsymbol{\theta}}(T'_k, \boldsymbol{T}'_o, \boldsymbol{F}_o, t)\Delta t \\ \boldsymbol{r}'_{k+1} &= \boldsymbol{r}'_k \mathrm{Exp}(\Delta t \boldsymbol{v}_{\boldsymbol{\theta}}(T'_k, \boldsymbol{T}'_o, \boldsymbol{F}_o, t)). \end{aligned} \tag{10}$$

By inserting the definitions for $(\boldsymbol{p}'_k, \boldsymbol{r}'_k)$ and $(\boldsymbol{p}'_{k+1}, \boldsymbol{r}'_{k+1})$ in Equation (10), we obtain

$$\begin{aligned} \boldsymbol{r}_\delta \boldsymbol{p}_{k+1} + \boldsymbol{p}_\delta &= \boldsymbol{r}_\delta \boldsymbol{p}_k + \boldsymbol{r}_\delta \boldsymbol{r}_k \boldsymbol{v}_{\boldsymbol{\theta}}(T'_k, \boldsymbol{T}'_o, \boldsymbol{F}_o, t)\Delta t + \boldsymbol{p}_\delta \\ \boldsymbol{r}_\delta \boldsymbol{r}_{k+1} &= \boldsymbol{r}_\delta \boldsymbol{r}_k \mathrm{Exp}(\Delta t \boldsymbol{v}_{\boldsymbol{\theta}}(T'_k, \boldsymbol{T}'_o, \boldsymbol{F}_o, t)). \end{aligned} \tag{11}$$

We observe that we can cancel $\boldsymbol{p}_\delta$ and $\boldsymbol{r}_\delta$ on each side of the equations and obtain

$$\begin{aligned} \boldsymbol{p}_{k+1} &= \boldsymbol{p}_k + \boldsymbol{r}_k \boldsymbol{v}_{\boldsymbol{\theta}}(T'_k, \boldsymbol{T}'_o, \boldsymbol{F}_o, t)\Delta t \\ \boldsymbol{r}_{k+1} &= \boldsymbol{r}_k \mathrm{Exp}(\Delta t \boldsymbol{v}_{\boldsymbol{\theta}}(T'_k, \boldsymbol{T}'_o, \boldsymbol{F}_o, t)). \end{aligned} \tag{12}$$

Since Equation (12) has to hold true for the model to generate equivariant actions, it follows (from Equation (8)) that the model has to be invariant, i.e., $\boldsymbol{v}_{\boldsymbol{\theta}}(T'_k, \boldsymbol{T}'_o, \boldsymbol{F}_o, t) = \boldsymbol{v}_{\boldsymbol{\theta}}(T_k, \boldsymbol{T}_o, \boldsymbol{F}_o, t)$.

D. Invariant Point Attention

This section provides pseudo-code for the Invariant Point Attention mechanism, the key element for our SE(3) Invariant Transformer. Since Invariant Point Attention was originally proposed in the context of protein folding [19], our pseudo-code in Algorithm 3 aims to provide additional context from a robotics perspective. The algorithm receives the set of features $\boldsymbol{F}$ and their associated poses $\boldsymbol{T}$ as input. The other inputs are hyper parameters. The algorithm describes the update for one token with associated feature vector $\boldsymbol{f}_i$ and pose T_i. Implementation-wise, we built on top of [57].

Algorithm 3: Invariant Point Attention ($\boldsymbol{F}=\{\boldsymbol{f}_1, \cdots, \boldsymbol{f}_i, \cdots\}, \boldsymbol{T}=\{T_1, \cdots, T_i, \cdots\}$, $N_{head}, c, N_{\text{query points}}, N_{\text{point values}}$)

$\boldsymbol{q}_i^h, \boldsymbol{k}_i^h, \boldsymbol{v}_i^h = \text{Linear}(\boldsymbol{f}_i)$, $\boldsymbol{q}_i^h, \boldsymbol{k}_i^h, \boldsymbol{v}_i^h \in \mathbb{R}^c, h \in \{1, ..., N_{head}\}$ `# Create global query, key and velocity points. c is the dimension of the embedding space.`

$\vec{\boldsymbol{q}}_i^{h,p}, \vec{\boldsymbol{k}}_i^{h,p} = \text{Linear}(\boldsymbol{f}_i)$, $\vec{\boldsymbol{q}}_i^{h,p}, \vec{\boldsymbol{k}}_i^{h,p} \in \mathbb{R}^3, p \in \{1, ..., N_{\text{query points}}\}$ `# Create a set of three dimensional query and key points for compatibility`

$\vec{\boldsymbol{v}}_i^{h,p} = \text{Linear}(\boldsymbol{f}_i)$, $\vec{\boldsymbol{v}}_i^{h,p} \in \mathbb{R}^3, p \in \{1, ..., N_{\text{point values}}\}$ `# Create a set of three dimensional feature points`

$w_L = \frac{1}{\sqrt{3c}}$ $\quad w_c = 0.5\sqrt{\frac{2}{(3 \cdot 9) N_{\text{query points}}}}$ `# Initialize regularisation constants`

$a_{i,j}^h = \text{softmax}_j(w_L \boldsymbol{q}_i^{h\,T} \boldsymbol{k}_j^h - w_c \sum_p \left\| \boldsymbol{r}_i \vec{\boldsymbol{q}}_i^{h,p} - \boldsymbol{r}_j \vec{\boldsymbol{k}}_j^{h,p} \right\|^2)$ `#Compute Compatibility`

$\boldsymbol{o}_i^h = \sum_j a_{i,j}^h \boldsymbol{v}_j^h$ `#Compute how the Neighbors contribute to the update of the current Node considering the global feature which does not take into account their relative transform`

$\vec{\boldsymbol{o}}_i^{h,p} = \boldsymbol{r}_i^T (\sum_j a_{i,j}^h \boldsymbol{r}_j \vec{\boldsymbol{v}}_j^{h,p})$ `#Compute how the Neighbors contribute to the update of the current Node considering the local ''point'' features`

$\tilde{\boldsymbol{f}}_i = \text{Linear}(\text{concat}_{h,p}(\boldsymbol{o}_i^h, \vec{\boldsymbol{o}}_i^{h,p}, \left\| \vec{\boldsymbol{o}}_i^{h,p} \right\|))$ `#Compute the updated feature value for token i`

E. Robomimic Experiments

In this section, we present more detailed results obtained in four Robomimic tasks (cf. Fig. 10) [36] using both state and image-based observations. Robomimic contains human demonstrations of several robotic manipulation tasks in simulated environments.

As mentioned in the main text, in these experiments, the diffusion process in Diffusion Policy [2] is replaced with a flow matching process in Euclidean space (App. B) – we refer to this policy as *Flow Matching*. We use the transformer architecture from [2] to model both the flow matching vector field and the denoising diffusion model.

Fig. 10: **Robomimic tasks** used to compare diffusion policy and flow matching in simulation.

We chose this architecture over the convolutional neural network since, in this work, we heavily use this type of architecture. For a fair comparison, we use the same hyperparameters in Flow Matching as the ones provided by the Diffusion Policy authors and build our Flow Matching code on top of the authors' code from `https://github.com/real-stanford/diffusion_policy`.

Training. Diffusion Policy is trained using DDPM [33] and cosine schedule with $K = 100$ steps. Flow Matching uses the rectified linear flow (Equation (5)) and also $K = 100$ steps. Due to limited computing resources, both policies are trained to a maximum of 3 days or 4000 epochs for the dataset with state-based observations. For the dataset with image-based observations, both policies are trained to a maximum of 3 days or 3500 epochs. Checkpoints are evaluated every 100 epochs. We use a machine with the following components (CPU, RAM, GPU): AMD EPYC 7453 28-Core; 512 GB RAM; RTX 3090 Turbo (24 GB).

Inference. For testing the policies, we choose the (epoch) checkpoint that performs best during training, i.e., with the highest average success rate. We report the average success rate from policy rollouts from 50 different initial configurations (from the test set) across 3 seeds. During inference, we use fewer steps than during training since it allows policies to be run at higher frequencies. For Diffusion

Method	IS	Can		Lift		Square		ToolHang
		ph	mh	ph	mh	ph	mh	ph
Diffusion Policy	**2**	$\mathbf{0.93 \pm 0.03}$	$\mathbf{0.94 \pm 0.03}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.72 \pm 0.11}$	$\mathbf{0.65 \pm 0.01}$	$\mathbf{0.51 \pm 0.15}$
	5	0.99 ± 0.02	0.96 ± 0.03	1.00 ± 0.00	1.00 ± 0.00	0.90 ± 0.03	0.77 ± 0.03	0.81 ± 0.08
	10	0.98 ± 0.03	0.97 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	0.91 ± 0.02	0.79 ± 0.06	0.90 ± 0.02
	20	0.99 ± 0.02	0.96 ± 0.02	1.00 ± 0.00	1.00 ± 0.00	0.86 ± 0.06	0.74 ± 0.02	0.90 ± 0.03
	100	0.98 ± 0.03	0.95 ± 0.03	1.00 ± 0.00	1.00 ± 0.00	0.91 ± 0.02	0.67 ± 0.10	0.85 ± 0.05
Flow Matching	**2**	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.97 \pm 0.02}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.83 \pm 0.05}$	$\mathbf{0.73 \pm 0.06}$	$\mathbf{0.73 \pm 0.11}$
	5	0.98 ± 0.00	0.95 ± 0.06	1.00 ± 0.00	1.00 ± 0.00	0.91 ± 0.07	0.73 ± 0.08	0.85 ± 0.03
	10	0.99 ± 0.01	0.96 ± 0.02	0.99 ± 0.01	1.00 ± 0.00	0.87 ± 0.10	0.65 ± 0.03	0.81 ± 0.09
	20	0.99 ± 0.01	0.96 ± 0.03	0.99 ± 0.01	1.00 ± 0.00	0.85 ± 0.06	0.65 ± 0.07	0.85 ± 0.05
	100	0.99 ± 0.01	0.96 ± 0.03	0.99 ± 0.01	1.00 ± 0.00	0.88 ± 0.05	0.67 ± 0.06	0.87 ± 0.03

TABLE I: **Robomimic results for policies using state-based observations.** Success rate (mean $\pm$ std) evaluation on Robomimic tasks with proprioception observations averaged over 3 seeds and 50 environments initializations (in the test set). The models are trained with 100 steps and tested with different inference steps (IS). Diffusion Policy models are trained with DDPM, and inference is done with DDIM, except for IS=100, for which we use DDPM since these are the number of steps the model was trained on.

Method	IS	Can		Lift		Square		ToolHang
		ph	mh	ph	mh	ph	mh	ph
Diffusion Policy	**2**	$\mathbf{0.93 \pm 0.06}$	$\mathbf{0.88 \pm 0.04}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.85 \pm 0.03}$	$\mathbf{0.67 \pm 0.10}$	$\mathbf{0.15 \pm 0.07}$
	5	0.95 ± 0.05	0.93 ± 0.01	1.00 ± 0.00	0.99 ± 0.01	0.89 ± 0.02	0.73 ± 0.10	0.54 ± 0.14
	10	0.95 ± 0.04	0.91 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	0.88 ± 0.03	0.81 ± 0.08	0.64 ± 0.12
	20	0.96 ± 0.03	0.92 ± 0.02	1.00 ± 0.00	0.99 ± 0.01	0.87 ± 0.04	0.76 ± 0.07	0.68 ± 0.13
	100	0.97 ± 0.04	0.92 ± 0.03	1.00 ± 0.00	0.99 ± 0.01	0.90 ± 0.04	0.75 ± 0.07	0.64 ± 0.09
Flow Matching	**2**	$\mathbf{0.95 \pm 0.01}$	$\mathbf{0.95 \pm 0.01}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.93 \pm 0.03}$	$\mathbf{0.73 \pm 0.01}$	$\mathbf{0.43 \pm 0.05}$
	5	0.96 ± 0.02	0.93 ± 0.02	1.00 ± 0.00	1.00 ± 0.00	0.94 ± 0.04	0.71 ± 0.08	0.47 ± 0.03
	10	0.96 ± 0.02	0.95 ± 0.01	1.00 ± 0.00	1.00 ± 0.00	0.93 ± 0.02	0.72 ± 0.14	0.51 ± 0.11
	20	0.95 ± 0.03	0.91 ± 0.04	1.00 ± 0.00	1.00 ± 0.00	0.95 ± 0.02	0.73 ± 0.13	0.57 ± 0.13
	100	0.97 ± 0.01	0.93 ± 0.03	1.00 ± 0.00	1.00 ± 0.00	0.92 ± 0.05	0.74 ± 0.09	0.58 ± 0.12

TABLE II: **Robomimic results for policies using image-based observations.** Success rate (mean $\pm$ std) evaluation on Robomimic tasks with image observations averaged over 3 seeds and 50 environments initializations (in the test set). The models are trained with 100 steps and tested with different inference steps (IS). Diffusion Policy models are trained with DDPM, and inference is done with DDIM, except for IS=100, for which we use DDPM since these are the number of steps the model was trained on.

Policy, we use DDIM for faster sampling [37]. For Flow Matching, we use exponentially spaced time steps (from the linear training schedule). This has the benefit of using larger Euler integration steps Δt when t is closer to 0 and smaller ones when it is closer to 1. We report the success rates obtained by both methods when using the same number of calls to the vector field or the denoising model (number of inference steps in Tables I and II).

Results. Tables I and II underline the findings from the main text: flow matching-based policies are capable of obtaining comparable results with diffusion-based policies with both state and image-based observations. In the tables, we highlight the rows in bold, which correspond to using a very low number of inference steps (i.e., 2 steps). Low inference steps allow for higher policy frequencies, e.g., 2 steps correspond to approximately 100 Hz. We observe that in this low inference steps regime, the success rate from flow matching policies surpasses that from diffusion policies, particularly in tasks such as Square and Tool Hang, where precise actions are needed to complete the tasks.

F. Mimicgen Experiments

In this section, we provide a more detailed description of the experiment introduced in Section IV-B.

1) Observations and Actions Representation: ActionFlow represents both the observations and actions with a tuple of poses $\boldsymbol{T}$ and features $\boldsymbol{F}$. Each pair of pose and feature represents a different entity in the space. For the experiments in Section IV-B, the pose represents the location of the different relevant objects in the space, while the feature is a fixed identifier for each object. The considered objects in each task are:

Three Piece Assembly: robot's end effector, the base, piece 1, and piece 2.

Coffee: robot's end effector, coffee pod, coffee machine, coffee pod holder, and coffee machine lid.

Stack Three: robot's end effector, cubeA, cubeB, and cubeC.

Threading: robot's end effector, needle, and tripod.

Additionally, the action is also expressed as a pose, representing the desired target pose for the robot's end-effector.

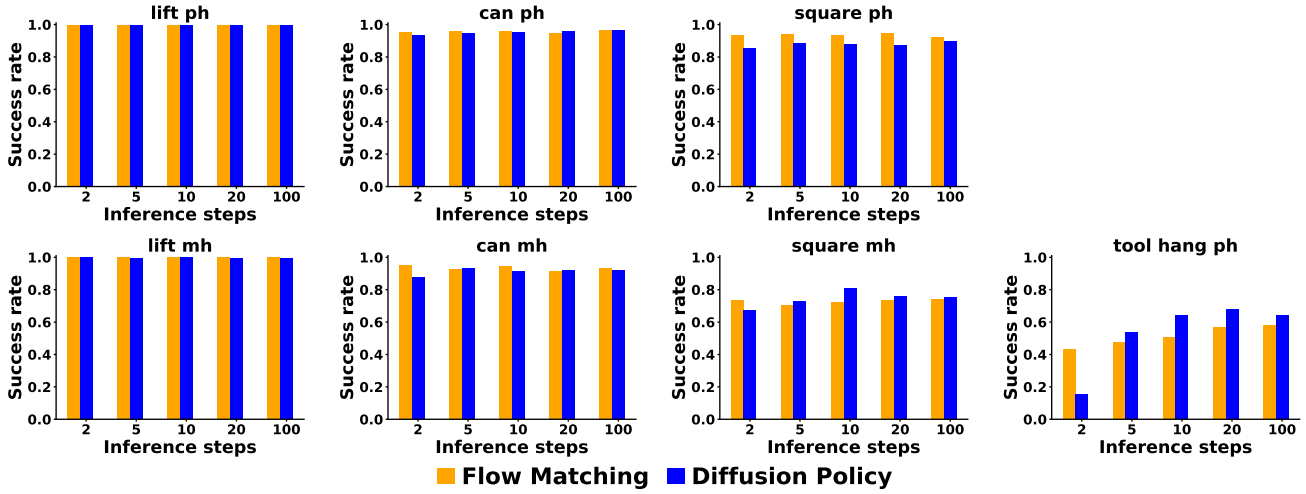

Fig. 11: **Robomimic results for policies using image-based observations**. The bar plots showcase the mean results from Table II and provide a better visualization.

2) Policy Representation: ActionFlow's network is a SE(3) Invariant Transformer as introduced in Section III-B. We additionally introduce an adaptation and normalization module, which is applied to the poses before they are further processed inside the transformer network. This is a common practice when training deep learning models.

Adaptation Module. Given a set of observation and action poses, we represent all the poses in the end-effector's frame. Then, we scale the translation vectors with a scaling factor of 10 and apply $Tanh$ to the translations to regularize the distances to a range within -1 and 1. The objective of this adaptation module is to increase the distance error between the points in the IPA module, while reducing variability when the object's are too far. We represent all the poses around the end effector to guarantee that the initial distribution of the flow is centered close to the actions.

G. Real Robot Experiments

This section provides additional details and results regarding our real robot experiments presented in Section IV-C.

1) Additional Information regarding the Teleoperation Interface: The teleoperation interface used in our real robot experiments consists of two main components. We leverage an off-the-shelf presenter (cf. here) for conveniently starting and stopping the recording of the individual demonstrations, as well as controlling, i.e., opening and closing the gripper. To control the pose of the robot's end effector, we rely on the OptiTrack Motion capture system. In particular, as shown in Fig. 5, the teleoperator wears a glove that has OptiTrack markers rigidly attached to it. Upon starting teleoperation, the glove's current pose is defined as the reference. Moving the glove w.r.t. this reference results in moving the robot's end-effector w.r.t. its initial pose accordingly. The teleoperation interface is set to operate at 25 Hz. Throughout all the real robot experiments, we use this teleoperation interface to control the robot end-effector's 6D pose, as well as the gripper opening width through a binary signal corresponding to gripper open / closed. Last, we want to point out that the last phase of the lightbulb mounting task solely necessitates a rotation to fix the bulb and turn it on. We found it extremely challenging to command a pure rotation around the end-effector's upward-pointing axis through the teleoperation interface. We, therefore, assigned one of the presenter's keys to trigger a rotation of 67° around the end-effector's upward-pointing axis. Thus, for the lightbulb mounting task, the teleoperator is mainly tasked with inserting the lightbulb's pins into the socket, and subsequently, the necessary rotation can be achieved by pressing the presenter's respective key.

2) Robot Control: As shown, e.g., in Fig. 1 & Fig. 5, this work uses a Franka Panda 7 DoF manipulator equipped with a parallel gripper. On the lowest level, we control the robot through the effort joint interface. This interface requires real-time control actions at 1000 Hz. For converting the current desired end-effector pose (which is either provided through the teleoperation interface or the running policy) into the effort joint commands, we build on top of the Cartesian Pose Impedance Controller provided in [41]. Since we do not have any smoothness guarantees on the output of our teleoperation interface and the policy's output, we employ exponential smoothing on the update of the desired target pose. In practice, we found this measure sufficient to stay within the Franka Panda robot's acceleration limits and yield smooth trajectories for both teleoperation and policy rollouts.

3) ActionFlow for Real Robot Manipulation - Implementation Details: The real robot experiments presented in Section IV-C are conducted using ActionFlow, i.e., the combination of the proposed SE(3) Invariant Transformer and SE(3) Flow Matching on the action space resulting in equivariant action generation.

Observations & Actions. For both experiments, we use an observation history of 5 steps and predict an action sequence containing 16 steps. In line with the teleoperation interface (which is set to collect actions at 25 Hz), we employ a time discretization of 0.04 s. While the RealSense camera returns RGB readings with a resolution of 640×480, we resize

the images to 80×80 pixels before passing them through the ResNet18 [39] for obtaining the encodings. The resizing helps to reduce the dataset's size significantly and, therefore, facilitates & speeds up policy training.

Training. We parametrize our ActionFlow policies for real robot manipulation using the SE(3) Invariant Transformer introduced in Fig. 2, and use four layers of IPA. Additionally, we consider $K = 4$ inference steps. We train the policies for 75 epochs and evaluate the last checkpoint. We use a machine with the following components (CPU, RAM, GPU): AMD EPYC 7453 28-Core; 512 GB RAM; RTX 3090 Turbo (24 GB).

Inference. As mentioned in Section IV-C, our policies are efficient and can be run in real-time. On average, it takes $0.03\,\mathrm{s}$ to generate an action sequence of 16 steps on an NVIDIA RTX 3090 GPU. On the real robot, we also account for the delay between passing the observations to the model and obtaining the action sequences. This is done by monitoring the time required for model inference and skipping the respective entries within the action sequence. In particular, we skip the actions that should have been applied at times when the model inference was still active. Moreover, we do not apply the whole remaining action sequence after each call to the model. Instead, we leverage our model's fast inference speeds and only apply the two next actions. Additionally, we employ exponential smoothing to ensure a smooth transition when updating the action sequence.

Modified Image Observations for the Mug Hanging Task. The mug hanging task can be divided into two phases, i.e., 1) reaching and grasping the mug, and subsequently, 2) hanging it. While for the first part, the view of the robot's wrist-mounted camera is essential, for the second phase of approaching the hanger, apart from the mug's pose, the camera image does not contain any information. In initial experiments, we nevertheless found that the ResNet18 extracts spurious correlations from background pixels for the phase of approaching the hanger, which harmed performance. To circumvent this issue, once the robot's gripper is closed, we set all pixels to black apart from the image's center region of size 30×30 pixels. This additional inductive bias, i.e., eliminating all the image background information once the mug is grasped, effectively improved the models' performance. We want to point out that a similar effect could be achieved by masking out background pixels based on the camera's depth information. However, we discovered that the depth readings from the Intel RealSense D435 were not accurate enough for this purpose, so we decided to employ the previously described image masking once the gripper is closed.

4) Additional Information on the Point Cloud conditioned Mug Hanging Experiment: This last section provides additional details about the mug hanging experiment presented in Section IV-C. For the point cloud conditioned mug hanging experiment, we had to switch from the RealSense D435 which is used in all other experiments to an RealSense D405. The reason for this change in RGB-D camera is that initial testing revealed that the depth readings from the RealSense D435 are insufficient to capture the details of the thin hanger. Therefore, the point cloud conditioned version of this experiment used a RealSense D405, which is mounted in the same pose as the RealSense D435 has been mounted in the previous experiments. Moreover, as mentioned in the main paper, to obtain a good point cloud of the hanger, before starting the teleoperation (for data collection) or the policy rollout, the robot visits 7 pre-defined end-effector poses, which ensure good visibility of the hanger as shown in Fig. 13.

Initialization	ActionFlow Success Rate
Train	9/10
Test	8/10

Fig. 12: Success Rates for the Pose-conditioned Mug hanging experiment on train and test configurations.

5) Additional Information on the Pose-based Mug Hanging Experiment: As mentioned in the main paper, for the task of picking up a mug and placing it onto a hanger, we also trained an ActionFlow policy that directly obtains the hanger's pose from OptiTrack readings. Therefore, this baseline policy is conditioned on the RGB images from the wrist camera and the hanger's pose obtained through OptiTrack. For training this baseline policy, we collect 200 demonstrations using variations, as shown in the main paper. To reiterate, the demonstrations only include slight variations of the mug poses, while the hanger always stays in the same pose. The results are presented in Fig. 12. The table's first row reveals that the ActionFlow policy achieves high success rates of 90% upon evaluating in similar scenarios as those encountered during training. We only observe one failure in which the mug is not grasped properly. Importantly, our policies run online in real-time as action generation only takes $0.03\,\mathrm{s}$ on an NVIDIA RTX 3090 GPU. We also evaluate the policy in previously unseen test scenarios, where, as mentioned in the main paper, the hanger is moved to either side of the table. The results show that our policy can still handle these novel test scenarios well, achieving 80% success. Fig. 14 shows a policy rollout in one of the testing scenarios. These high success rates, despite the previously unseen scenarios, demonstrate the equivariance property of our proposed ActionFlow, which inherently can handle these translated scene instances.

One potential explanation for the slightly increased success rates in the test configurations compared to the experiments presented in Section IV-C might be that despite the changed hanger pose in the test configurations, the waypoints visited during initialization do not change. Therefore, there might be a slight distribution shift w.r.t. the hanger's point cloud and its latent encoding from the point cloud encoder, which might negatively affect the network's generalization. This could be tackled, e.g., by adding more initial waypoints to ensure better coverage of the overall space or further refining the point cloud encoder's architecture.

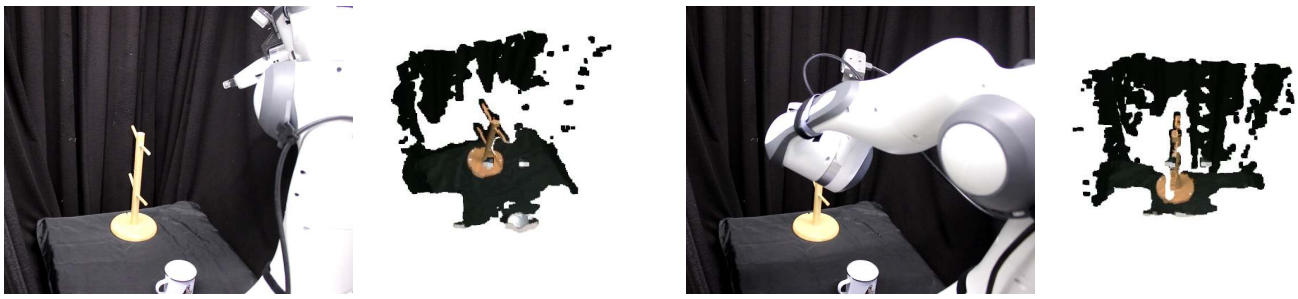

Fig. 13: Illustrating the initial phase for the point cloud-conditioned mug hanging experiment in which a couple of waypoints are visited to register the hanger. Shown are two of these waypoints and the corresponding raw pointcloud observations next to them.

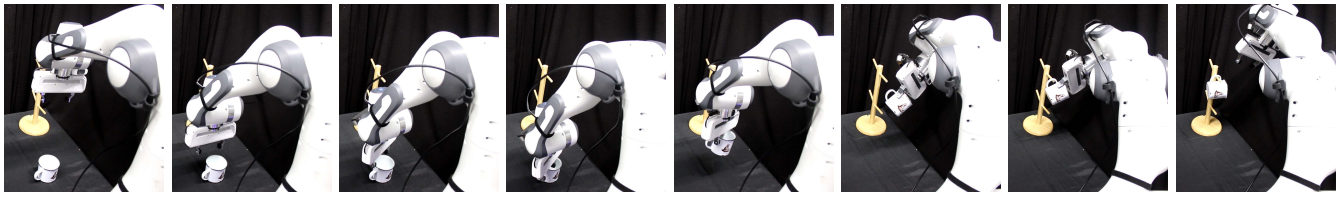

Fig. 14: Visualization of one ActionFlow policy rollout on the mug hanging task, in which the hanger is placed in a pose that was not included in the demonstrations.