

LieSpline-DP: Geometry-Aware B-Spline Diffusion Policy for Smooth SE(3) Robot Manipulation

Erxuan Xie

2nd Given Name Surname

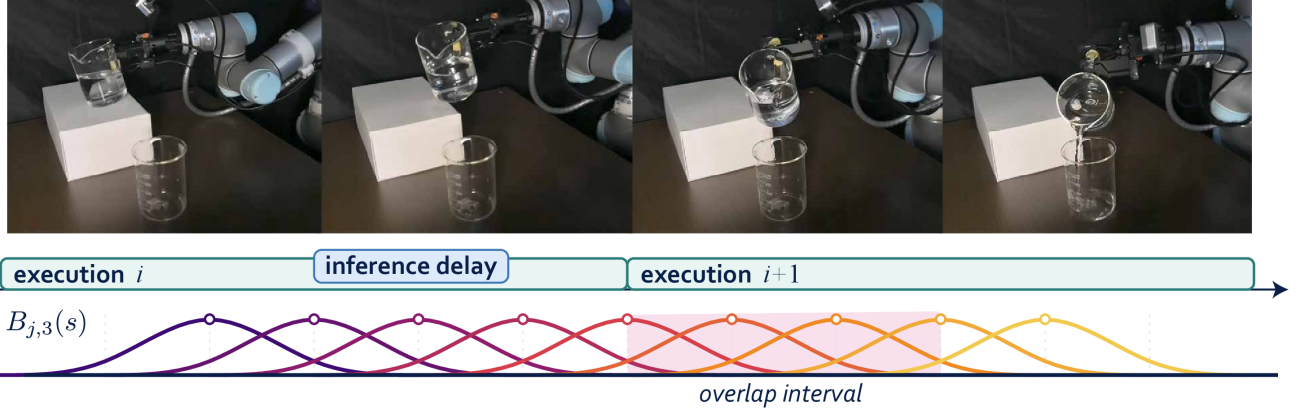


Fig. 1. **Asynchronous execution timeline of LieSpline-DP.** While waypoints from the current spline are executed (execution i), inference for the next observation-conditioned spline runs before the queue is exhausted (blue interval). The overlapping cubic B-spline basis functions have local support; reusing the boundary control poses across consecutive plans preserves the continuous reference pose, velocity, and acceleration at the handover.

Abstract—Diffusion Policy (DP) is a powerful Learning from Demonstration (LfD) method for robotic manipulation, yet it suffers from discontinuous and non-smooth trajectories. Spline-based action representations can smooth per-inference action sequences, but existing methods neither ensure smooth cross-chunk transitions nor respect SE(3) end-effector kinematics. We therefore propose LieSpline-DP, a geometry-aware DP that generates end-effector B-spline trajectories directly on SE(3) and guarantees globally smooth motion throughout the entire task execution. Experiments in simulation and on three real-robot tasks show that LieSpline-DP preserves DP’s multimodal action-generation capability while substantially improving motion smoothness. The gains are particularly pronounced in real-world tasks involving liquids and flexible objects: in our real-robot experiments, LieSpline-DP achieved a 100% success rate on both pouring and bucket hooking, whereas the DP baseline achieved only 0% and 60%, respectively. More details are available on our project homepage: **TBD**

I. INTRODUCTION

Learning visuomotor policies from demonstrations is an effective paradigm for robotic manipulation. Diffusion Policy (DP), in particular, models multimodal action distributions through conditional denoising and has achieved strong performance in simulated and real-world tasks [1]. DP typically predicts a fixed-horizon action chunk, executes part of it, and then replans. On a physical robot, non-negligible inference latency motivates predicting the next chunk while the current one is still being executed. Switching between independently generated chunks, however, can introduce discontinuities,

jerky movements, controller tracking errors, and even cause task failure. Output-level temporal ensembling mitigates this problem by taking a weighted average of the actions predicted by overlapping chunks, but such averaging may mixes and blurs distinct action modes [2]. Stateful diffusion instead conditions each new prediction on prior actions [3]. Although this strategy improves cross-chunk coherence, its coupling is learned and does not explicitly preserve the action prefix committed during asynchronous inference. For asynchronous execution, Real-Time Chunking (RTC) freezes the committed prefix and inpaints the remaining actions at test time [4]. Later approaches internalize prefix conditioning during training or reshape the generative flow to learn continuation dynamics [5], [6]. However, these methods all operate on discrete actions and do not explicitly guarantee cross-chunk velocity or acceleration continuity.

The intrinsic smoothness of splines has recently motivated their use as compact action representations [7]–[11]. A cubic B-spline is C^2 -continuous within a predicted chunk, but this property does not automatically extend across independently generated chunks. To ensure inter-chunk continuity, BEAST anchors the first control point of a new clamped B-spline to the final action of the previous chunk, guaranteeing positional (C^0) continuity but not matching boundary velocity or acceleration [8]. ABPolicy instead fits a spline over past and future actions and refines its leading control points against the executed history; this substantially improves boundary smoothness, but the least-squares refitting only approximately couples the new spline to the trajectory actually executed [11].

TABLE I
COMPARISON OF ACTION REPRESENTATIONS AND EXECUTION MECHANISMS. EXISTING METHODS PARAMETERIZE END-EFFECTOR OR JOINT ACTIONS IN EUCLIDEAN VECTOR SPACES, WHEREAS LIESPLINE-DP CONSTRUCTS END-EFFECTOR TRAJECTORIES DIRECTLY ON $SE(3)$.

Method	Action Space	Representation	Cross-Chunk Mechanism	Continuity Property
DP [1]	$\mathbb{R}^d$ (end-effector/joint)	Dense waypoints	None	No explicit guarantee
ACT [2]	$\mathbb{R}^d$ (joint)	Dense waypoints	Temporal ensembling	No explicit guarantee
RTC [4]	$\mathbb{R}^d$ (end-effector/joint)	Dense waypoints	Prefix inpainting	No explicit guarantee
BEAST [8]	$\mathbb{R}^d$ (end-effector/joint)	Euclidean B-spline	Endpoint anchoring	Within-chunk C^2 ; cross-chunk C^0
Spline Policy [9]	$\mathbb{R}^d$ (end-effector/joint)	Bernstein spline	Continuity constraints	Within-chunk C^2 ; cross-chunk C^0/C^1
B-spline Policy [10]	$\mathbb{R}^d$ (end-effector/joint)	Euclidean B-spline	Segment alignment	Within-chunk C^2
ABPolicy [11]	$\mathbb{R}^d$ (joint)	Euclidean B-spline	Constrained refitting	Approximately Cross-chunk C^2
LieSpline-DP	$SE(3)$ (end-effector)	Lie-group B-spline	Control-pose prefix inpainting	Cross-chunk C^2

Existing spline-based methods do not reliably guarantee cross-chunk acceleration continuity, which is important for robotic control. Cross-chunk high-order continuity therefore remains unresolved by the spline representation alone.

The methods mentioned above also share a geometric limitation: they parameterize joint actions or end-effector poses in Euclidean vector spaces. For rigid-body motion, Euclidean denoising and interpolation do not intrinsically respect the group structure of $SE(3)$, making rotational trajectories and smoothness varied with the chosen rotation representation. Recent work has begun to incorporate rigid-motion geometry into diffusion-based manipulation, first through diffusion-derived cost fields and bi-equivariant pose distributions on $SE(3)$ for grasp and target-pose generation [12], [13], and more recently through closed-loop action-sequence policies equivariant to $SE(3)$ or the broader $SIM(3)$ group [14]–[16]. Across these formulations, geometric equivariance improves data efficiency and spatial generalization by ensuring that predicted poses or waypoint sequences transform consistently with the scene. Their focus, however, is equivariant generation rather than temporal trajectory geometry: they neither construct the continuous action trajectory as a Lie-group spline nor guarantee high-order continuity across asynchronously replanned chunks.

Motivated by these observations, we propose **LieSpline-DP**, a geometry-aware diffusion policy that generates a stateful B-spline directly on $SE(3)$. The policy predicts compact six-dimensional local coordinates of $SE(3)$ control poses, which a cumulative Lie-group B-spline decodes through group composition into a continuous rigid-body trajectory [17]. Crucially, at each replanning cycle, LieSpline-DP preserves the boundary control-pose prefix from the previous spline and inpaints only future control poses. With a cubic B-spline, this shared-prefix construction guarantees that the generated reference trajectory is C^2 -continuous across chunk boundaries, without post-hoc blending or refitting.

To the best of our knowledge, LieSpline-DP is the first closed-loop visuomotor diffusion policy for general-purpose manipulation to use a B-spline defined directly on $SE(3)$ as its end-effector action representation. Table I compares LieSpline-DP with prior spline-based generative policies and other related baseline policies in action geometry, cross-chunk

coupling, and continuity guarantees.

Our main contributions are summarized as follows:

- **Geometry-aware spline action generation.** We formulate visuomotor action generation as conditional diffusion over compact local coordinates of $SE(3)$ control poses and decode them with a cumulative Lie-group B-spline, yielding continuous rigid-body trajectories that respect end-effector geometry.
- **Stateful replanning with guaranteed cross-chunk smoothness.** We introduce execution-conditioned control-pose prefix inpainting to couple consecutive plans during asynchronous execution. For cubic splines, preserving the boundary control-pose prefix guarantees cross-chunk C^2 continuity without post-hoc blending or refitting.
- **An end-to-end learning pipeline from waypoint demonstrations.** We develop a phase-aligned conversion and training procedure that combines event-weighted intrinsic spline fitting with latency-aligned training windows. We validate the resulting policy in simulation and on a real robot in terms of task performance and kinematic smoothness.

II. PRELIMINARIES

A. Diffusion Policy

Diffusion Policy models a conditional distribution over action sequences by reversing a gradual noising process [1]. Let $\mathbf{x}^{(0)}$ denote a clean action representation and let ℓ index the diffusion step. The forward process admits the closed form

$$\mathbf{x}^{(\ell)} = \sqrt{\bar{\alpha}_\ell} \mathbf{x}^{(0)} + \sqrt{1 - \bar{\alpha}_\ell} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I), \quad (1)$$

where $\bar{\alpha}_\ell$ is determined by a prescribed noise schedule. A denoising network $\boldsymbol{\epsilon}_\theta(\mathbf{x}^{(\ell)}, \ell, \mathcal{O})$ is trained to recover the injected noise from the noisy action and observation context $\mathcal{O}$:

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{\mathbf{x}^{(0)}, \ell, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}^{(\ell)}, \ell, \mathcal{O}) \right\|_2^2 \right]. \quad (2)$$

At inference time, the learned noise estimate parameterizes successive reverse transitions from Gaussian noise to an observation-conditioned action sequence. Known action entries can be imposed throughout this reverse process by replacing them after each denoising step, yielding conditional generation by inpainting.

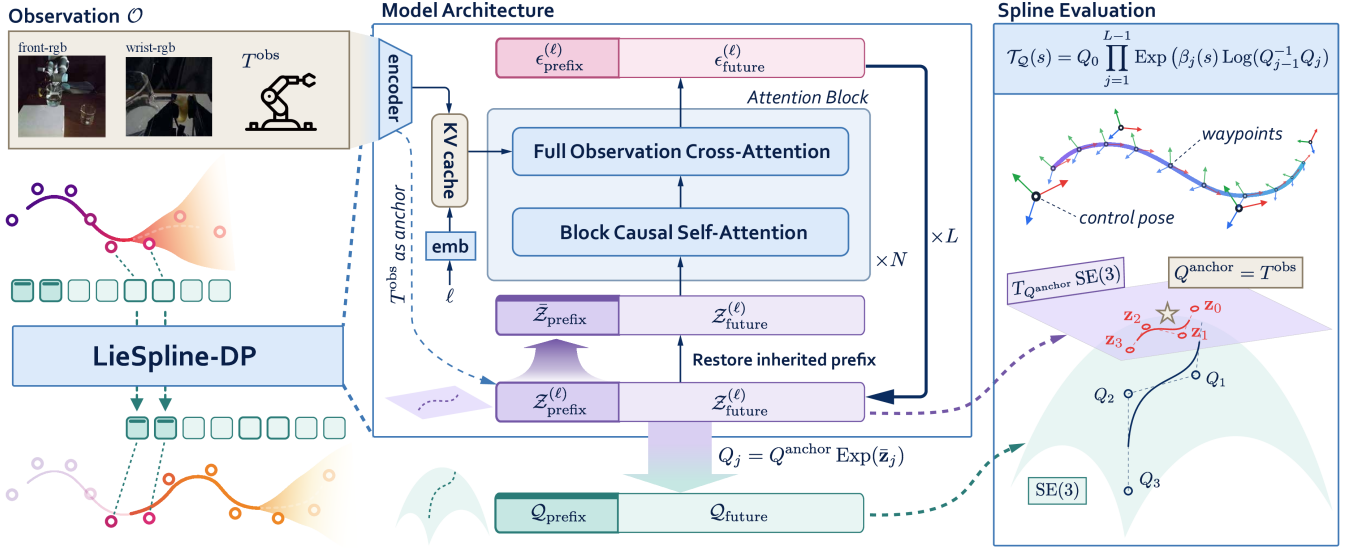


Fig. 2. **LieSpline-DP architecture.** An observation encoder conditions a diffusion model over local control-pose coordinates. Block-causal self-attention separates the fixed boundary prefix from the denoised future tokens. The resulting coordinates are lifted to SE(3) and decoded by a cumulative Lie-group B-spline into dense end-effector waypoints.

B. Geometry of SE(3) B-spline Trajectories

An end-effector pose is represented by

$$T = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \text{SE}(3), \quad R \in \text{SO}(3), \quad \mathbf{t} \in \mathbb{R}^3. \quad (3)$$

The exponential and logarithm maps relate a pose increment to a twist $\xi = [\rho^\top, \phi^\top]^\top \in \mathbb{R}^6$ through

$$T_b = T_a \text{Exp}(\xi), \quad \xi = \text{Log}(T_a^{-1}T_b). \quad (4)$$

Throughout this work, we adopt this right-multiplicative increment convention: the pose increment acts on the right of the reference pose. These maps provide local Euclidean coordinates while preserving group composition in the decoded trajectory.

Let $Q = (Q_0, \dots, Q_{H-1})$ be a sequence of control poses and define the relative twists $\Omega_j = \text{Log}(Q_{j-1}^{-1}Q_j)$. The corresponding cumulative Lie-group B-spline is [17]

$$\mathcal{T}_Q(s) = Q_0 \prod_{j=1}^{H-1} \text{Exp}(\beta_{j,p}(s)\Omega_j), \quad (5)$$

where factors are composed in increasing order of j . Here, $\beta_{j,p}(s) = \sum_{i=j}^{H-1} B_{i,p}(s)$ is the cumulative basis function constructed from the degree- p B-spline basis functions $B_{i,p}(s)$. Throughout this work, we use the uniform knot vector $\tau = \{\tau_i\}$ in the normalized spline phase s , with unit-spaced knots $\tau_i = i$. Under this uniform construction, the B-spline basis functions $B_{j,p}(s)$ has support on $s \in [j, j+p+1)$. Fig. 1 illustrates this structure for the cubic case.

III. METHOD: LIESPLINE-DP

A. Overview

The core contribution of LieSpline-DP lies in its action representation and cross-plan coupling mechanism, and is

therefore compatible with different DP backbones. In this work, we instantiate it with a Transformer denoiser.

At replanning cycle n , LieSpline-DP constructs a degree- p end-effector spline from $H = K + F$ control poses $Q_n = (Q_{n,0}, \dots, Q_{n,H-1})$. As shown in Fig. 2, the observation encoder first maps the observation history $\mathcal{O}_n$ into the observation memory. The observed end-effector pose T_n^{obs} , which is contained in the robot-state observation, defines the anchor pose $Q_n^{\text{anchor}} = T_n^{\text{obs}}$ of a local tangent-space chart. The control poses are represented in this chart by $Z_n = (\mathbf{z}_{n,0}, \dots, \mathbf{z}_{n,H-1})$. Its first $K = p$ entries form a fixed prefix inherited from the tail control poses of the preceding execution, shown with an overbar on the prefix tokens in Fig. 2; the remaining F entries represent the future controls to be generated. The diffusion model therefore operates as a control-point inpainting model: block-causal self-attention allows the noisy future tokens to attend to the fixed prefix, cross-attention supplies the observation memory.

The completed tangent-vector sequence is then lifted back to absolute control poses on SE(3) through

$$Q_{n,j} = Q_n^{\text{anchor}} \text{Exp}(\mathbf{z}_{n,j}), \quad (6)$$

for $j = 0, \dots, H-1$. These poses define the cumulative Lie-group B-spline shown in the rightmost panel of Fig. 2. The lower-right illustration summarizes the geometric relationship among the local coordinates $Z_n \in T_{Q_n^{\text{anchor}}} \text{SE}(3)$, the control poses $Q_n \in \text{SE}(3)$, and the resulting continuous trajectory.

B. Asynchronous Inference Pipeline

Local SE(3) control poses. The observation history contains the measured end-effector pose, which defines the anchor $Q_n^{\text{anchor}} = T_n^{\text{obs}} \in \text{SE}(3)$. We use this pose both as policy input and as the origin of the local chart in which diffusion

is performed. For $j = 0, \dots, H - 1$, the local and absolute representations are related by

$$\mathbf{z}_{n,j} = \text{Log}((Q_n^{\text{anchor}})^{-1} Q_{n,j}). \quad (7)$$

Here $\mathbf{z}_{n,j} = [\boldsymbol{\rho}_{n,j}^\top, \boldsymbol{\phi}_{n,j}^\top]^\top \in \mathbb{R}^6$ follows the right-increment convention in (4). Because Q_n^{anchor} is close to the trajectory being predicted, these coordinates provide a well-conditioned local linearization for Gaussian noising and denoising. At each inference, the inherited control-pose prefix is stored as absolute poses on $\text{SE}(3)$ and re-expressed relative to the newly observed anchor at each cycle. Consequently, changing the anchor alters only the network input coordinates, leaving the physical boundary trajectory unchanged.

Fixed-prefix inpainting diffusion. At cycle n , the inherited poses are first mapped to local coordinates using the current anchor Q_n^{anchor} . Let $\tilde{\mathbf{z}}_{n,j}$ denote their normalized coordinates. They occupy the first K tokens of the diffusion sequence, while the future tokens are initialized with Gaussian noise. The Transformer denoiser first applies block-causal self-attention with an asymmetric mask: prefix tokens attend only to the prefix, whereas future tokens attend to both the prefix and the future block. It then uses cross-attention to condition the resulting token representations on the visual-proprioceptive memory produced by the observation encoder; the corresponding key and value projections are stored in a KV cache and reused across reverse-denoising steps.

The inherited prefix is restored after every forward-noising or reverse-denoising update:

$$\tilde{\mathbf{z}}_{n,j}^{(\ell)} \leftarrow \tilde{\mathbf{z}}_{n,j}^{(0)}, \quad j = 0, \dots, K - 1. \quad (8)$$

Consequently, sampling cannot modify the boundary controls committed by the previous plan.

Asynchronous execution with C^2 -continuity. As illustrated by the basis functions in Fig. 1, the smoothness guarantee relies on two coupled design choices: a simple uniform knot lattice shared by all replanning cycles and the reuse of boundary control poses. With no repeated interior knots, a degree- p spline is C^{p-1} within each plan. Across two consecutive plans n and $n + 1$, the common knot spacing and phase convention align the handover at phase $s = p + E$ of plan n with phase $s = p$ of plan $n + 1$. The corresponding local B-spline bases are therefore exact translates over equally long physical knot spans. After plan n advances the reference trajectory by E control intervals, plan $n + 1$ reuses the $K = p$ control poses adjacent to the handover:

$$Q_{n+1,j} = Q_{n,E+j}, \quad j = 0, \dots, K - 1, \quad K = p. \quad (9)$$

These overlapping poses form the fixed prefix of the new control lattice. The remaining F control poses are generated from the latest available observation. For the initial plan, where no preceding lattice exists, we use the stationary prefix

$$Q_{0,j} = Q_0^{\text{anchor}}, \quad j = 0, \dots, K - 1. \quad (10)$$

After denoising, the predicted future coordinates are lifted to absolute control poses using (6). The resulting poses are

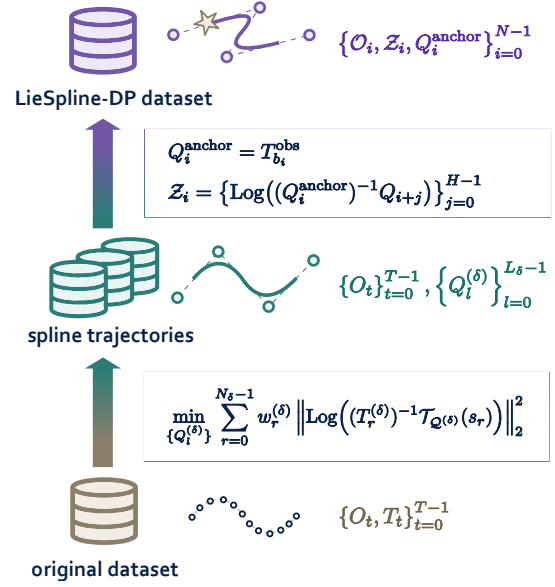


Fig. 3. **Offline dataset conversion pipeline.** From bottom to top, each dense demonstration is converted into phase-offset Lie-group spline trajectories, which are then sliced into observation-conditioned windows of local control-pose coordinates. The displayed anchor corresponds to the synchronous case $L = 0$; latency-aligned real-robot windows use the generalized anchor in (17).

concatenated with the inherited absolute prefix and decoded by the cumulative Lie-group spline in (5). We sample this continuous curve at the controller rate and append the resulting W reference poses to the execution queue:

$$T_{n,k} = \mathcal{T}_{Q_n} \left(p + \frac{k}{S} \right), \quad k = 1, \dots, W, \quad W = ES. \quad (11)$$

Here S is the number of controller steps per control interval. As shown in Fig. 1, inference for next plan starts a few steps before handover so that computation overlaps execution.

With aligned uniform knots, the p shared controls in (9) determine the same boundary value and first $p - 1$ derivatives for both plans. Hence, for any smooth local chart φ around the handover pose,

$$\begin{aligned} \left. \frac{d^r}{ds^r} (\varphi \circ \mathcal{T}_{Q_n})(s) \right|_{s=p+E^-} \\ = \left. \frac{d^r}{ds^r} (\varphi \circ \mathcal{T}_{Q_{n+1}})(s) \right|_{s=p^+}, \quad r = 0, \dots, p - 1. \end{aligned} \quad (12)$$

For the cubic spline ($p = 3$), this gives cross-chunk continuity of pose, body velocity, and body acceleration. Equal knot-span durations extend the result from phase to time derivatives; hard-prefix inpainting and waypoint sampling preserve it.

C. Spline Dataset Conversion and Training

As illustrated in Fig. 3, the offline conversion proceeds in three stages, from bottom to top: the original dense demonstration, the fitted $\text{SE}(3)$ spline trajectories, and the resulting LieSpline-DP training dataset. For clarity, the figure depicts

only the pose stream; gripper commands are processed using the same temporal alignment.

Dense demonstrations and start-offset augmentation.

The pose portion of an original episode in the bottom row of Fig. 3 is $\{(\mathcal{O}_t, T_t^*)\}_{t=0}^{T-1}$, with an associated gripper sequence $\{g_t^*\}_{t=0}^{T-1}$. Because adjacent spline controls are separated by S dense steps, using only one control-lattice phase would retain only approximately one out of every S possible window alignments. We therefore create one sub-episode for each start offset $\delta \in \{0, \dots, S-1\}$. For offset δ , define

$$(\mathcal{O}_r^{(\delta)}, T_r^{(\delta)}, g_r^{(\delta)}) = (\mathcal{O}_{\delta+r}, T_{\delta+r}^*, g_{\delta+r}^*), \quad r = 0, \dots, N_\delta - 1. \quad (13)$$

To make the phase lattice align exactly with the discrete timing of the original waypoints trajectory, each sub-episode is padded by repeating its terminal sample until its length is a multiple of S ; the repeated samples define a stationary terminal hold.

Phase-offset spline trajectories. The middle row of Fig. 3 represents each sub-episode by an absolute control-pose sequence $\mathcal{Q}^{(\delta)} = \{Q_l^{(\delta)}\}_{l=0}^{L_\delta-1}$. At dense index r , the corresponding spline phase is $s_r = p + (r+1)/S$. The control poses are obtained by the key-instance-weighted intrinsic fit displayed in the figure,

$$\min_{\{Q_l^{(\delta)}\}} \sum_{r=0}^{N_\delta-1} w_r^{(\delta)} \left\| \text{Log} \left((T_r^{(\delta)})^{-1} \mathcal{T}_{\mathcal{Q}^{(\delta)}}(s_r) \right) \right\|_2^2, \quad (14)$$

Here, $\mathcal{T}_{\mathcal{Q}^{(\delta)}}(s_r)$ is evaluated using the cumulative Lie-group B-spline defined in (5). The six components of the logarithmic residual are standardized by the dataset translation and rotation scales before evaluating the norm. The first p controls are fixed to the measured pose preceding the first action, and the last p controls are fixed to the terminal hold. The waypoint weights $w_r^{(\delta)}$ can be determined from gripper-state transitions, tactile signals, or other task events. Larger weights encourage the spline to fit waypoints more accurately at critical moments, while smaller weights elsewhere place greater emphasis on smoothness.

Training windows for LieSpline-DP. The top arrow and boxed equations in Fig. 3 show how the fitted trajectories are converted into training samples. After pooling the phase-offset trajectories, we suppress δ on the selected control sequence for compactness. Let n index a control-boundary window and let $b_n = nS$ denote its first dense action index. In the synchronous case depicted in the figure, the anchor and the H local control-pose labels are

$$Q_n^{\text{anchor}} = T_{b_n}^{\text{obs}}, \quad \mathcal{Z}_n = \{ \text{Log}((Q_n^{\text{anchor}})^{-1} Q_{n+j}) \}_{j=0}^{H-1}. \quad (15)$$

Thus the final dataset shown in the top row is

$$\mathcal{D}_{\text{LieSpline}} = \{(\mathcal{O}_n, \mathcal{Z}_n, Q_n^{\text{anchor}})\}_{n=0}^{N-1}, \quad (16)$$

where $\mathcal{O}_n$ denotes the observation history ending at b_n . The first K elements of $\mathcal{Z}_n$ are the inherited prefix and the remaining F elements are the future prediction target, with $H = K + F$.

For asynchronous execution, inference is triggered L dense steps before the same action boundary. Specifically,

$$a_n = \max(b_n - L, 0), \quad Q_n^{\text{anchor}} = T_{a_n}^{\text{obs}}, \quad (17)$$

and $\mathcal{O}_n$ now ends at a_n . Substituting this anchor into (15) gives the real-robot training label, while $L = 0$ reduces exactly to the boxed formula in Fig. 3. Gripper labels remain aligned to the execution boundary b_n ; their platform-specific parameterization does not affect the SE(3) continuity result.

The denoising loss is evaluated only on valid, unconditioned entries,

$$\mathcal{L} = \mathcal{L}_{\text{pose}} + \lambda_g \mathcal{L}_{\text{grip}} \quad (18)$$

where padded entries are masked and λ_g balances the auxiliary gripper objective.

IV. EXPERIMENTS

We evaluate LieSpline-DP with two questions in mind. First, we ask whether replacing dense action chunks with Lie-group spline controls preserves manipulation performance while improving the smoothness of the executed motion. Second, we examine whether these kinematic improvements translate into more reliable behavior on physical tasks. We study the first question through a controlled comparison on four Robomimic benchmarks and the second through three real-robot manipulation tasks.

A. Experimental Setup and Metrics

Simulation benchmarks. We consider the Robomimic Proficient-Human low-dimensional of Lift, Can, Square, and Tool Hang tasks. We compare LieSpline-DP against a dense-action DP baseline. Both policies use absolute end-effector actions, two observation steps, the same Transformer denoiser, and the same demonstrations, data split, normalization, diffusion schedule, optimizer, and exponential moving average (EMA) policy. Both policies operate at 20 Hz and execute eight actions before replanning.

For each task and method, we train three models with three seeds. The EMA policy is evaluated at 80 checkpoints distributed uniformly over training, with 50 rollouts per checkpoint. The same environment seeds and initial states are used for the two methods at corresponding checkpoints. We report both the best success rate attained during training (Max) and the average over the final ten evaluations (Last-10). Max selects the checkpoint independently for each training seed using success rate alone, with the earliest checkpoint used to break ties; its jerk values are measured at that same checkpoint. Last-10 provides a complementary measure of late-training stability. All results are reported as the mean across the three training seeds, with each table entry ordered as Max / Last-10.

LieSpline-DP uses cubic splines with $p = K = 3$, $F = 8$, and $H = 11$ control poses. Consecutive controls are separated by two simulator steps, and each replanning cycle advances four control intervals, yielding the same eight executed actions as the dense-action baseline.

Real-robot tasks. We evaluate on a UR5e arm equipped with a Unitree Dex1 gripper. Policy waypoints are issued at

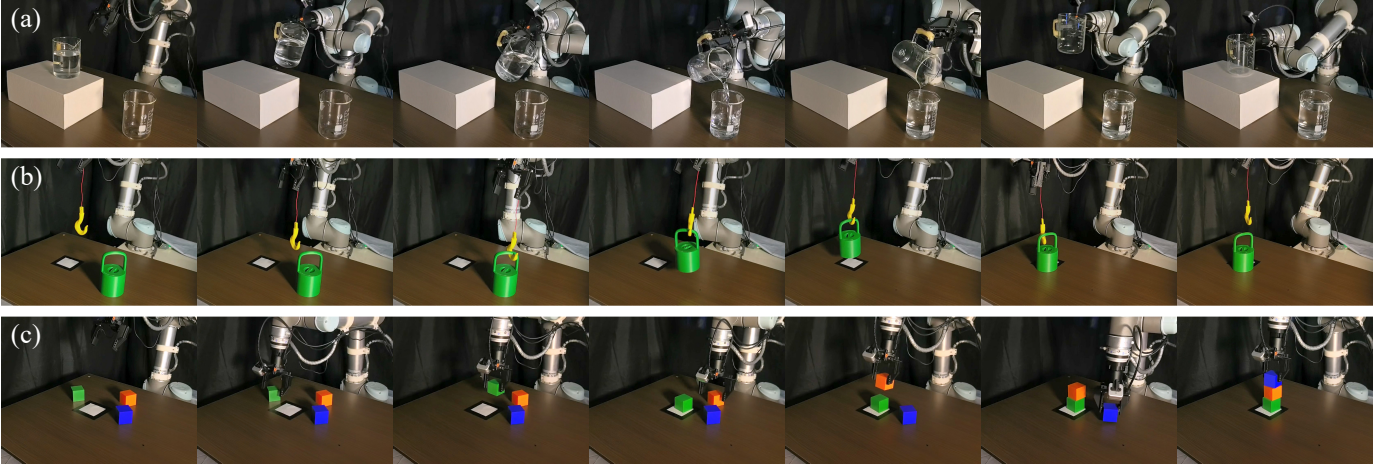


Fig. 4. Keyframe snapshot sequences for the three real-robot tasks. (a) Pouring. (b) Bucket hooking. (c) Cube stacking.

TABLE II
ROBOMIMIC PH LOW-DIMENSIONAL RESULTS. EACH CELL REPORTS MAX / LAST-10, AVERAGED OVER THREE TRAINING SEEDS. FOR MAX, JERK IS EVALUATED AT THE CHECKPOINT SELECTED BY SUCCESS; THE MEAN AND JERK COLUMNS ARE FOUR-TASK MACRO AVERAGES.

Method	Success rate (%)				Mean	Executed-trajectory jerk	
	Lift	Can	Square	Tool Hang		$\bar{J}_{\text{trans}}$ (m/s ³) ↓	$\bar{J}_{\text{rot}}$ (rad/s ³) ↓
DP	TBD / TBD	TBD / TBD	TBD / TBD	TBD / TBD	TBD / TBD	TBD / TBD	TBD / TBD
LieSpline-DP	TBD / TBD	TBD / TBD	TBD / TBD	TBD / TBD	TBD / TBD	TBD / TBD	TBD / TBD

15 Hz. The real-robot model uses $p = K = 3$, $F = 8$, $H = 11$, $S = 2$, $E = 4$, $W = 8$, and a nominal latency of $n_{\text{latency_steps}} = 2$. We consider the following three tasks:

- **Pouring**: grasp a water-filled beaker and pour its contents into a target beaker. A trial succeeds if all water is transferred without spilling.
- **Bucket hooking**: use a tethered hook to engage a bucket handle and transport the bucket to a designated target. A trial succeeds if the hook remains engaged and the bucket reaches the target.
- **Cube stacking**: stack three randomly scattered blocks at a predefined location in a prescribed color order, with occasional human interventions that disturb a partial stack. A trial succeeds if a stable, correctly ordered three-block stack is completed at the target within one minute.

For each task, DP and LieSpline-DP are trained on the same set of demonstrations: 50 for pouring, 50 for bucket hooking, and 100 for cube stacking. We conduct TBD evaluation trials per method on each task.

Simulation metrics. Task success is the fraction of rollouts that complete the environment objective within the episode horizon. To measure motion quality, we use the complete end-effector trajectory actually realized in the simulator, rather than spline controls or commanded waypoints. Poses are sampled at $\Delta t = 0.05$ s from the initial post-reset state until termination. We include both successful and failed rollouts and apply no padding, interpolation, or temporal filtering.

Let $T_k = (R_k, \mathbf{p}_k)$ denote the measured pose at step

k . Translational jerk and its episode-level root-mean-square (RMS) magnitude are

$$\mathbf{j}_k^{\text{trans}} = \frac{\mathbf{p}_{k+3} - 3\mathbf{p}_{k+2} + 3\mathbf{p}_{k+1} - \mathbf{p}_k}{\Delta t^3}, \quad (19)$$

$$J_{\text{trans}} = \sqrt{\frac{1}{N-3} \sum_{k=0}^{N-4} \|\mathbf{j}_k^{\text{trans}}\|_2^2}. \quad (20)$$

For rotation, we map relative orientations to the Lie algebra before differentiation:

$$\boldsymbol{\omega}_k = \frac{\text{Log}(R_{k+1}R_k^\top)^\vee}{\Delta t}, \quad (21)$$

$$\boldsymbol{\alpha}_k = \frac{\boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k}{\Delta t}, \quad (22)$$

$$\mathbf{j}_k^{\text{rot}} = \frac{\boldsymbol{\alpha}_{k+1} - \boldsymbol{\alpha}_k}{\Delta t}, \quad (23)$$

$$J_{\text{rot}} = \sqrt{\frac{1}{N-3} \sum_{k=0}^{N-4} \|\mathbf{j}_k^{\text{rot}}\|_2^2}. \quad (24)$$

Here J_{trans} and J_{rot} are expressed in m/s³ and rad/s³, respectively. Rotations are converted from Robomimic's XYZW quaternions, avoiding direct finite differences in a representation-dependent coordinate system.

We first average each jerk metric over the 50 rollouts at a checkpoint. For Max, the jerk is taken at the checkpoint selected by success; for Last-10, both success and jerk are averaged over the final ten evaluations. Four-task macro aver-

ages are formed within each training seed before computing the across-seed mean.

Real-robot metrics. Task success is evaluated using the completion criteria specified above and is reported over all physical trials. To isolate the smoothness of the reference motion produced by each policy, the real-robot jerk comparison uses only the time-stamped TCP target poses actually issued to the robot; measured TCP poses and intermediate spline or policy outputs that were not sent to the controller are excluded. For each trial, samples are sorted by timestamp and duplicate or non-increasing timestamps are discarded. No interpolation, smoothing, filtering, resampling, or outlier removal is applied.

Let $(\tau_i, \mathbf{y}_i)$ denote a vector-valued sequence with strictly increasing timestamps. We recursively apply the midpoint finite-difference operator

$$(D\mathbf{y})_i = \frac{\mathbf{y}_{i+1} - \mathbf{y}_i}{\tau_{i+1} - \tau_i}, \quad \tau_i^D = \frac{\tau_{i+1} + \tau_i}{2}. \quad (25)$$

For commanded translations $\mathbf{p}_i^{\text{cmd}}$ expressed in millimeters, the translational jerk is $\mathbf{j}_i^{\text{cmd}} = D^3 \mathbf{p}_i^{\text{cmd}}$. For commanded orientations R_i^{cmd} , angular velocity and angular jerk are

$$\boldsymbol{\omega}_i^{\text{cmd}} = \frac{\text{Log}((R_i^{\text{cmd}})^\top R_{i+1}^{\text{cmd}})^\vee}{t_{i+1} - t_i}, \quad \boldsymbol{\eta}_i^{\text{cmd}} = D^2 \boldsymbol{\omega}_i^{\text{cmd}}. \quad (26)$$

where each $\boldsymbol{\omega}_i^{\text{cmd}}$ is assigned the midpoint timestamp $(t_{i+1} + t_i)/2$ before D is applied recursively. We summarize the Euclidean magnitudes $\|\mathbf{j}_i^{\text{cmd}}\|_2$ and $\|\boldsymbol{\eta}_i^{\text{cmd}}\|_2$ in mm/s^3 and rad/s^3 , respectively. For each trial, we compute the mean, median, 95th percentile (p95), and maximum. The comparison figure shows p95 with a filled marker, the maximum with an open marker, and the p95-to-maximum range with a vertical segment on a logarithmic axis.

B. Simulation Results

Table II compares the best performance reached during training with performance near the end of training. Under the Max criterion, LieSpline-DP achieves a mean success rate of **TBD%**, compared with **TBD%** for DP, while reducing translational and rotational jerk by **TBD%** and **TBD%**, respectively. The largest task-level difference in success occurs on **TBD**, where performance changes from **TBD%** to **TBD%**. These results indicate that the spline representation improves the smoothness of the realized motion without requiring a separate post-processing stage.

The Last-10 evaluation shows a mean success rate of **TBD%** for LieSpline-DP and **TBD%** for DP. Over the same checkpoints, LieSpline-DP reduces translational jerk from **TBD** to **TBD** m/s^3 and rotational jerk from **TBD** to **TBD** rad/s^3 . Comparing the Max and Last-10 results further reveals **TBD**.

C. Real-Robot Results

Fig. 4 shows keyframe sequences from the physical tasks, Table III summarizes the corresponding success rates, and Fig. 5 compares the jerk of the TCP target trajectories actually issued to the robot. LieSpline-DP outperforms the DP baseline on all three tasks. Its commanded trajectories also reduce

TABLE III
SUCCESS RATES ON THE THREE REAL-ROBOT TASKS.

Method	Pouring	Bucket hooking	Cube stacking
DP	0%	60%	50%
LieSpline-DP	100%	100%	80%

Reserved for real-robot commanded-trajectory jerk plot

Top: translation command jerk
Bottom: angular command jerk

Filled marker: p95; open marker: max
Logarithmic vertical axes

Fig. 5. **Jerk comparison for the TCP target trajectories actually issued to the robot.** Translational jerk (top) and angular jerk (bottom) are computed independently for each evaluation trial without interpolation or filtering. Filled and open markers denote p95 and maximum jerk, respectively, and each vertical segment spans the p95-to-maximum range. Only commanded target trajectories are included; measured TCP trajectories are excluded.

p95 translational jerk by **TBD%** and p95 angular jerk by **TBD%** relative to DP, with corresponding reductions of **TBD%** and **TBD%** in the maximum values. In the *pouring* task, the baseline’s jerky end-effector motions frequently caused water to splash outside the target container. The baseline also occasionally stalled before completing the pour or oscillated repeatedly between two poses. By contrast, LieSpline-DP generated smooth, sustained pouring motions and succeeded in every trial.

Similar benefits are observed in the *bucket hooking* and *cube stacking* tasks. In *bucket hooking*, abrupt baseline motions induced oscillations in the flexible rope, driving the object configuration away from those encountered in the demonstrations and producing out-of-distribution observations from which the policy often failed to recover. We did not observe this failure mode with LieSpline-DP, which achieved a 100% success rate. In *cube stacking*, baseline jitter sometimes struck and toppled a block during approach or placement. The displaced block could then fall outside the robot’s reachable workspace, making recovery impossible; the baseline also exhibited stalling and oscillatory behavior. LieSpline-DP reduced these disturbances and improved the success rate from 70% to 80%, although occasional failures remained.

D. Discussion

In the matched Robomimic simulations, LieSpline-DP achieves task success comparable to DP while producing smoother trajectories. Thus, in controlled simulation, improved motion smoothness does not necessarily yield higher task success. The distinction becomes substantially more pronounced on the real robot, where unmodeled dynamics, controller tracking errors, and physical interactions can amplify small action discontinuities.

This advantage is especially evident in tasks involving liquids or flexible objects, whose complex dynamics can turn abrupt motions into persistent fluid or rope oscillations. By producing substantially smoother motions, LieSpline-DP suppresses these disturbances, keeps subsequent observations closer to the demonstrated state distribution, and consequently achieves markedly better performance in *pouring* and *bucket hooking*. The smaller improvement in *cube stacking*, together with the remaining failures, suggests that trajectory continuity is most beneficial when task dynamics amplify motion irregularities, but cannot by itself eliminate errors arising from perception, contact uncertainty, or task-level planning.

V. CONCLUSION

TBD

REFERENCES

- [1] 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,” arXiv:2303.04137, 2023.
- [2] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, “Learning fine-grained bimanual manipulation with low-cost hardware,” arXiv:2304.13705, 2023.
- [3] X. Liu, Y. Zhou, F. Weigend, S. Sonawani, S. Ikemoto, and H. B. Amor, “Diff-Control: A stateful diffusion-based policy for imitation learning,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2024, pp. 7453–7460.
- [4] K. Black, M. Y. Galliker, and S. Levine, “Real-time execution of action chunking flow policies,” in *Advances in Neural Information Processing Systems*, 2025.
- [5] K. Black, A. Z. Ren, M. Equi, and S. Levine, “Training-time action conditioning for efficient real-time chunking,” arXiv:2512.05964, 2025.
- [6] Y. Liu *et al.*, “Learning native continuation for action chunking flow policies,” in *Proc. Robot.: Sci. Syst.*, 2026.
- [7] J. Carvalho, A. T. Le, P. Kicki, D. Koert, and J. Peters, “Motion planning diffusion: Learning and adapting robot motion planning with diffusion models,” arXiv:2412.19948, 2024.
- [8] H. Zhou *et al.*, “BEAST: Efficient tokenization of B-splines encoded action sequences for imitation learning,” in *Advances in Neural Information Processing Systems*, 2025.
- [9] M. Tian, Y. Li, S. Liu, A. Ijspeert, and S. Calinon, “Spline policy: A structured representation for robot policies,” arXiv:2606.07386, 2026.
- [10] X. Han, H. Xiong, H. Chen, C. Liu, A. Torralba, Y. Zhu, and Y. Du, “B-spline policy: Accelerating manipulation policies via B-spline action representations,” arXiv:2607.09648, 2026.
- [11] F. Yang, P. Jing, K. Qu, N. Zhao, and Y. Su, “ABPolicy: Asynchronous B-spline flow policy for real-time and smooth robotic manipulation,” arXiv:2602.23901, 2026.
- [12] J. Urain, N. Funk, J. Peters, and G. Chelvatzi, “SE(3)-DiffusionFields: Learning smooth cost functions for joint grasp and motion optimization through diffusion,” in *Proc. IEEE Int. Conf. Robot. Autom.*, 2023, pp. 5923–5930.
- [13] H. Ryu *et al.*, “Diffusion-EDFs: Bi-equivariant denoising generative modeling on SE(3) for visual robotic manipulation,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2024, pp. 18007–18018.
- [14] J. Yang, Z. Cao, C. Deng, R. Antonova, S. Song, and J. Bohg, “EquiBot: SIM(3)-equivariant diffusion policy for generalizable and data efficient learning,” in *Proc. Conf. Robot Learn.*, 2025, pp. 1048–1068.
- [15] C. Tie *et al.*, “ET-SEED: Efficient trajectory-level SE(3) equivariant diffusion policy,” in *Proc. Int. Conf. Learn. Represent.*, 2025.
- [16] X. Zhu, F. Wang, R. Walters, and J. Shi, “SE(3)-equivariant diffusion policy in spherical Fourier space,” in *Proc. Int. Conf. Mach. Learn.*, 2025.
- [17] C. Sommer, V. Usenko, D. Schubert, N. Demmel, and D. Cremers, “Efficient derivative computation for cumulative B-splines on Lie groups,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11148–11156.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.