Appendix

Appendix A: Equivariance Proof of Rotation Matrix Construction via Schmidt Orthogonalization

A.1 Theoretical Foundation and Notation

Let a point cloud observation $x \in \mathcal{G}_i$ belong to a certain equivariant group (Assumption 1), with its decentralized form given by $x_{\text{de}} = x - x_{\text{mean}}$. Processing x_{de} through an SO(3)-equivariant network Φ, we obtain two rotation-equivariant vectors:

$$\Phi(x_{\text{de}}) = [r_{x1}, r_{x2}] \in \mathbb{R}^{3\times 2},$$

where r_{x1} and r_{x2} satisfy SO(3)-equivariance:

$$\Phi(R \cdot x_{\text{de}}) = R \cdot \Phi(x_{\text{de}}), \quad \forall R \in \text{SO}(3).$$

A.2 Rotation Matrix Construction via Schmidt Orthogonalization

The rotation matrix $R_x \in \text{SO}(3)$ is constructed through the following steps:

1) **Normalize the first vector:**
$$u_{x1} = \frac{r_{x1}}{\|r_{x1}\|}.$$
2) **Orthogonalize the second vector:**
$$v_{x2} = r_{x2} - (r_{x2} \cdot u_{x1})u_{x1}, \quad u_{x2} = \frac{v_{x2}}{\|v_{x2}\|}.$$
3) **Construct the third orthogonal basis vector via cross product:**
$$u_{x3} = u_{x1} \times u_{x2}.$$
4) **Assemble the rotation matrix:**
$$R_x = [u_{x1}, u_{x2}, u_{x3}] \in \mathbb{R}^{3\times 3}.$$

A.3 Equivariance Proof

Proposition: If $y_{\text{de}} = R \cdot x_{\text{de}}$ for some $R \in \text{SO}(3)$, then the rotation matrix R_y constructed via the above procedure satisfies:

$$R_y = R \cdot R_x.$$

Proof:
By the equivariance assumption:

$$\Phi(y_{\text{de}}) = \Phi(R \cdot x_{\text{de}}) = R \cdot \Phi(x_{\text{de}}) = [R \cdot r_{x1}, R \cdot r_{x2}].$$

1) **Normalization step:**
$$u_{y1} = \frac{R \cdot r_{x1}}{\|R \cdot r_{x1}\|} = \frac{R \cdot r_{x1}}{\|r_{x1}\|} = R \cdot u_{x1},$$
where the norm preservation holds since R is an orthogonal matrix.
2) **Orthogonalization step:**
$$\begin{aligned} v_{y2} &= R \cdot r_{x2} - ((R \cdot r_{x2}) \cdot (R \cdot u_{x1}))\,(R \cdot u_{x1}) \\ &= R \cdot r_{x2} - (r_{x2} \cdot u_{x1})(R \cdot u_{x1}) \\ &= R \cdot [r_{x2} - (r_{x2} \cdot u_{x1})u_{x1}] = R \cdot v_{x2}, \\ u_{y2} &= \frac{v_{y2}}{\|v_{y2}\|} = \frac{R \cdot v_{x2}}{\|v_{x2}\|} = R \cdot u_{x2}. \end{aligned}$$
3) **Cross product step:**
$$\begin{aligned} u_{y3} = u_{y1} \times u_{y2} &= (R \cdot u_{x1}) \times (R \cdot u_{x2}) \\ &= R \cdot (u_{x1} \times u_{x2}) = R \cdot u_{x3}, \end{aligned}$$
where the cross-product relation is preserved under SO(3) transformation.
4) **Matrix assembly:**
$$\begin{aligned} R_y = [u_{y1}, u_{y2}, u_{y3}] &= [R \cdot u_{x1}, R \cdot u_{x2}, R \cdot u_{x3}] \\ &= R \cdot [u_{x1}, u_{x2}, u_{x3}] = R \cdot R_x. \end{aligned}$$

A.4 Uniqueness of the Canonical Representation

From the above equivariance, for any x, y in the same equivariant group, their canonical representations satisfy:

$$x_{\text{cn}} = R_x^T \cdot x_{\text{de}} = (R_y^T R) \cdot (R^T y_{\text{de}}) = R_y^T \cdot y_{\text{de}} = y_{\text{cn}}.$$

This indicates that all point clouds within the same equivariant group are mapped to the same canonical representation, which validates the rationality of Assumption 1.

Appendix B: Network Architectures

B.1. Canonical Feature Encoder

This module is responsible for extracting local geometric features while strictly maintaining SE(3)-equivariance. We leverage the Vector Neuron framework [1] to construct the encoder.

1) **Local Graph Construction:** In the canonical representation module, the rotation estimation network uses a k-nearest neighbor approach with $k = 10$ to capture local geometric features. The input features consist of relative coordinates normalized by the centroid of the local neighborhood.
2) **Equivariant Feature Extraction:** The encoder comprises three stacked VN-MLP layers. To effectively decouple geometric structure from semantic information, the hidden features are split into invariant scalar channels (C_s) and equivariant vector channels (C_v). We employ VN-Leaky ReLU as the activation function to preserve vector directionality during non-linear transformations.

B.2. SE(3)-Equivariant Temporal Transformer

To aggregate spatiotemporal information without breaking symmetry, we design a custom Transformer that operates on rotation-invariant geometric embeddings.

- **Invariant Geometric Encoding with RBF:** Instead of standard positional encodings, we explicitly model the relative geometry between point pairs. The pairwise Euclidean distance $d_{ij} = \|x_i - x_j\|$ is strictly invariant to rigid transformations. To effectively capture high-frequency spatial details, we map these continuous distances into a high-dimensional feature space using a learnable Radial Basis Function kernel, denoted as $\phi(d_{ij})$.
- **Dual-Stream Attention Mechanism:** The network consists of $L = 4$ stacked attention blocks with a hidden dimension of $D = 128$. Unlike canonical transformers, the attention weights are derived from the RBF-encoded edge features rather than coordinate dot products. We employ a dual-stream architecture:
  - *Spatial Heads*: Focus on intra-frame edges to model local object topology and inter-part relationships.
  - *Temporal Heads*: Integrate motion dynamics by modulating the attention weights with relative time embeddings, allowing the network to track geometric evolution across the observation horizon T_{obs}.
- **Equivariant Update Structure:** The aggregated features from both streams are fused via an element-wise sum and passed through a Vector Neuron Feed-Forward Network This guarantees that the output features retain the vector-scalar decoupled structure necessary for downstream equivariant processing.

B.3. Curvature-Guided Action Head

We utilize a 1D Conditional U-Net as the backbone for flow matching, enhanced with a curvature regularization module.

Backbone Configuration: The U-Net consists of symmetrical down-sampling and up-sampling paths with channel dimensions of [128, 256, 384]. The canonical condition features are injected into each residual block via Feature-wise Linear Modulation (FiLM). Time embeddings are projected to 128 dimensions.

Trajectory Coefficient Parameterization: Instead of predicting the coefficients directly, which would require hard constraints to satisfy boundary conditions, we propose a *residual parameterization* scheme. The coefficients $a_\phi(t)$ and $b_\psi(t)$ are modeled as a linear baseline modulated by a non-linear perturbation term:

$$\begin{aligned} a_\phi(t) &= (1-t) + \delta \cdot t(1-t) \cdot \mathcal{M}_\phi(t), \\ b_\psi(t) &= t + \delta \cdot t(1-t) \cdot \mathcal{M}_\psi(t). \end{aligned} \tag{1}$$

Here, $(1-t)$ and t represent the standard Rectified Flow linear path. The term $t(1-t)$ acts as a *boundary gate*, naturally forcing the perturbation to zero at $t = 0$ and $t = 1$, thereby guaranteeing exact boundary satisfaction ($a(0) = 1, b(1) = 1$, etc.) by construction. δ is a scaling factor set to 0.5.

Network Architecture: The mapping functions $\mathcal{M}_\phi$ and $\mathcal{M}_\psi$ are parameterized by two lightweight MLPs. Each MLP consists of three linear layers with a hidden dimension of 64. We employ SiLU activations for hidden layers to ensure smooth gradients and a Tanh activation at the output layer to bound the perturbation magnitude within $[-1, 1]$.

Appendix C: Implementation Details and Hyperparameters

C.1. Data Processing and Training Protocol

For point cloud processing, we employ a farthest point sampling strategy to sample $N = 256$ points per frame. We set the batch size to 128 and fix the observation horizon $T_{obs} = 2$ and prediction horizon $T_{pred} = 16$ for all tasks.

All policies are trained using the AdamW optimizer with a cosine learning rate scheduler and a warm-up period of 500 steps. For the EquiBot baseline, we adjust the learning rate to 1.0×10^{-4} following its original configuration, while other methods use a default learning rate of 8.0×10^{-5}. The experiments are conducted on a workstation equipped with a single NVIDIA RTX 4090d GPU.

C.2. Hyperparameter Configuration

Table I details the specific settings used in our experiments.

TABLE I
Hyperparameter Configurations for Training

Hyperparameter	Value
Optimization & Training	
Optimizer	AdamW
Learning Rate (CurveFlow, DP3, iDP3)	8.0×10^{-5}
Learning Rate (EquiBot)	1.0×10^{-4}
Weight Decay	1.0×10^{-6}
Batch Size	128
LR Scheduler	Cosine
Warmup Steps	500
Total Training Epochs	$500 \sim 3000$
Data & Horizon	
Observation Horizon (T_{obs})	2
Prediction Horizon (T_{pred})	16
Action Execution Steps (T_{exec})	8
Point Cloud Points (N)	256
Architecture: Canonical Encoder (VN)	
Input Neighbors (k)	10
Scalar Channels (C_s)	96
Vector Channels (C_v)	32
Encoder Layers	3
Architecture: Temporal Transformer	
Transformer Blocks (L)	4
Attention Heads	4
Hidden Dimension	128
Architecture: Action Head (U-Net)	
Down-sampling Channels	$[128, 256, 384]$
Time Embedding Dim	128
Kernel Size	5
Curvature MLP Hidden Dim	64
Inference	
Inference Steps (NFE)	10
Noise Schedule	Squared Cosine

Appendix D: Additional Experimential Analysis

D.1. Robustness to Noise Intensity (ϵ)

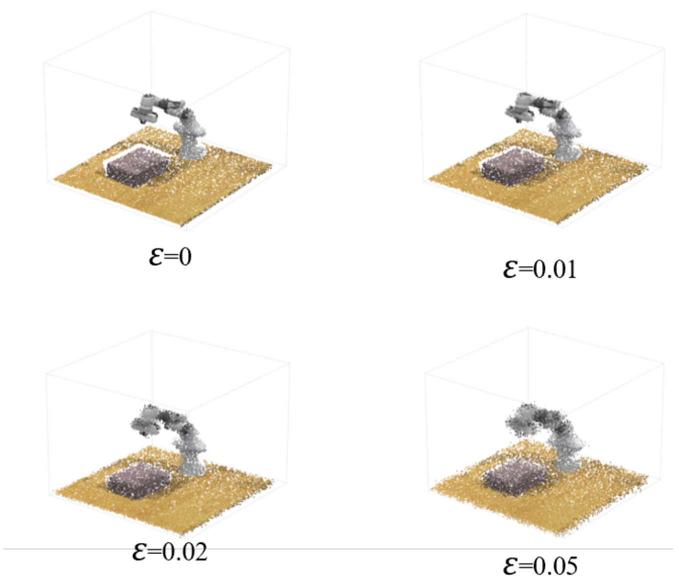

Fig. 1. Visual comparison of point clouds with different Gaussian noise levels during the 'Take-shoes-off-box' task. Note the degradation of geometric details as ϵ increases.

To quantify the limits of our framework's perceptual robustness, we extend the ablation study in Section V-D by varying the intensity of additive Gaussian noise $\mathcal{N}(0, \epsilon^2)$ injected into the observation point clouds. As Fig. 1 shows, we evaluate noise levels $\epsilon \in \{0, 0.01, 0.02, 0.05\}$ across the four representative tasks. The results are reported in Table II.

The empirical data demonstrates high resilience to sensory perturbations:For $\epsilon \leq 0.02$, the performance degradation is negligible ($\Delta < 0.7\%$). Notably, the success rate on Open oven remains statistically invariant, confirming that the Canonical Representation effectively filters high-frequency jitter before it affects the policy manifold. Even under severe noise ($\epsilon = 0.05$), the policy retains $\approx 93\%$ of its original performance. Unlike per-frame baselines that often suffer catastrophic failure under such perturbations (as seen in Table III), our SE(3)-Equivariant Temporal Transformer maintains geometric consistency by aggregating features over time, preventing transient noise from destabilizing the trajectory generation.

D.2. Numerical Implementation of Curvature Regularization

To facilitate the optimization of the curvature regularization term $\mathcal{L}_{curvature}$ defined in Eq. (23) , we implement a discrete approximation of the trajectory derivatives. Since the scaling functions $a_\phi(t)$ and $b_\psi(t)$ are parameterized by neural networks, their derivatives with respect to the diffusion time $t \in [0, 1]$ must be calculated across a uniform temporal grid to ensure training stability and geometric consistency.

The time interval $[0, 1]$ is partitioned into M equal subintervals. We define the discrete time steps as:

$$t_i = \frac{i}{M}, \quad i \in 0, 1, \ldots, M \tag{2}$$

where the grid spacing is given by $\Delta t = 1/M$. In our implementation, we set $M = 1000$ to maintain high numerical precision. To approximate the first-order derivatives ($\dot{a}_\phi, \dot{b}_\psi$) and second-order derivatives ($\ddot{a}_\phi, \ddot{b}_\psi$) , we employ the central difference scheme for interior points to minimize truncation errors:

- **First-order approximation**:

$$\begin{aligned} \dot{a}_\phi(t_i) &\approx \frac{a_\phi(t_{i+1}) - a_\phi(t_{i-1})}{2\Delta t}, \\ \dot{b}_\psi(t_i) &\approx \frac{b_\psi(t_{i+1}) - b_\psi(t_{i-1})}{2\Delta t}. \end{aligned} \tag{3}$$

- **Second-order approximation**:

$$\begin{aligned} \ddot{a}_\phi(t_i) &\approx \frac{a_\phi(t_{i+1}) - 2a_\phi(t_i) + a_\phi(t_{i-1})}{(\Delta t)^2}, \\ \ddot{b}_\psi(t_i) &\approx \frac{b_\psi(t_{i+1}) - 2b_\psi(t_i) + b_\psi(t_{i-1})}{(\Delta t)^2}. \end{aligned} \tag{4}$$

For the boundary points $i \in \{0, M\}$, we apply one-sided forward and backward differences respectively to ensure gradient continuity across the entire horizon. The integral for the curvature penalty is then evaluated as a Riemann sum over the discrete grid:

$$\mathcal{L}_{curvature} \approx \lambda \sum_{i=1}^{M-1} \left(\dot{a}_\phi(t_i)\ddot{b}_\psi(t_i) - \dot{b}_\psi(t_i)\ddot{a}_\phi(t_i) \right)^2 \Delta t \tag{5}$$

This enables direct geometric regularization of the trajectory via backpropagation, bypassing the computational overhead of explicit ODE simulation.

D.3. Sensitivity Analysis of Curvature Regularization (λ)

To rigorously evaluate the impact of the curvature regularization weight λ (Eq. 23) on policy performance, we conducted a sensitivity analysis on two representative tasks: *Stack* (Robomimic) and *Open-box* (RLBench). We varied λ across the set $\{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ while keeping all other hyperparameters fixed. The results are summarized in Fig. 2.

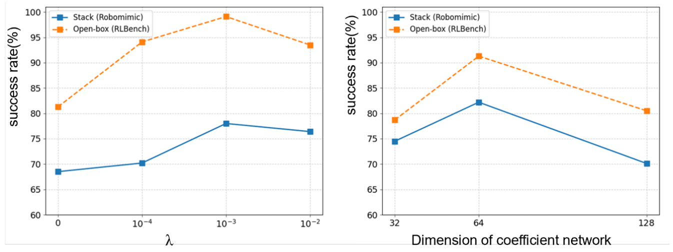

Fig. 2. Impact of λ (left) and the dimensions of coefficient networks (right) on policy performance.

Specifically, setting $\lambda \approx 10^{-3}$ yields peak performance across both tasks; at this magnitude, the regularization effectively suppresses unnecessary high-frequency oscillations and

TABLE II
PERFORMANCE STABILITY UNDER VARYING NOISE LEVELS (ϵ)

Noise Level (ϵ)	Mug cleanup	Hammer cleanup	Open oven	Shoes box	Mean SR	Δ
0 (Clean)	41.7 ± 2.1	71.8 ± 3.3	18.6 ± 1.8	16.3 ± 2.6	**37.10**	-
0.01	41.6 ± 2.0	71.5 ± 3.1	18.8 ± 1.9	16.2 ± 2.4	**37.03**	-0.07
0.02	41.4 ± 2.3	69.5 ± 3.6	19.1 ± 2.5	15.9 ± 2.9	**36.48**	-0.62
0.05	39.8 ± 2.8	66.4 ± 4.1	17.5 ± 3.1	14.2 ± 3.4	**34.48**	-2.62

abrupt ”turning” behaviors while retaining sufficient flexibility to navigate the non-linear manifold of expert demonstrations. Conversely, under-regularization ($\lambda \to 0$) fails to constrain the trajectory curvature, leading to execution failures in contact-rich scenarios like Stack due to instability. On the other hand, excessive regularization ($\lambda \geq 10^{-2}$) forces the generated trajectories toward rigidity and linearity similar to standard Rectified Flow, which limits the policy's expressivity in modeling complex, curved manipulations required for obstacle avoidance and results in a sharp decline in success rates.

Furthermore, the analysis of the coefficient network architecture reveals that increasing the dimension from 32 to 64 significantly improves success rates on both the Stack (from $\approx 75\%$ to $\approx 83\%$) and Open-box (from $\approx 80\%$ to $\approx 92\%$) tasks, indicating that a higher-dimensional representation is necessary to effectively model the complex dynamics of the scaling functions $a_\phi(t)$ and $b_\psi(t)$. However, further increasing the dimension to 128 leads to a performance drop (falling back to $\approx 70\%$ for Stack and $\approx 80\%$ for Open-box), likely due to overfitting and optimization instability. Therefore, a dimension of 64 provides sufficient model capacity to capture intricate motion dynamics without compromising generalization, making it the optimal choice for our framework.

Based on these empirical findings, we adopt $\lambda = 10^{-3}$ and a coefficient network dimension of 64 as the default setting for all main experiments reported in Section V.

APPENDIX E: REAL-WORLD EXPERIMENTS DETAILS

E.1. Task Workspace and Randomization Settings

We evaluated the policy on four long-horizon tasks as shown in Fig. 3. To ensure safe and reachable operation, we defined specific workspace bounds for each task relative to the robot base frame. Objects were randomly initialized within these designated zones to rigorously test the policy's generalization capabilities across different spatial configurations. Detailed settings are provided in Table III.

TABLE III
REAL-WORLD TASK WORKSPACE AND RANDOMIZATION SETTINGS

Task	Workspace Range (m)	Object Randomization
Sorting	$x : [0.3, 0.7], y : [-0.3, 0.3]$	Pos: ± 10 cm, Rot: $\pm 180°$
Hanoi	$x : [0.4, 0.6], y : [-0.2, 0.2]$	Pos: ± 10 cm, Rot: $\pm 180°$
Insert Battery	$x : [0.2, 0.6], y : [-0.1, 0.1]$	Pos: ± 3 cm, Rot: $\pm 60°$
Hammer cleanup	$x : [0.2, 0.8], y : [-0.4, 0.4]$	Pos: ± 5 cm, Rot: $\pm 5°$

E.2. Detailed Failure Analysis

To provide a granular understanding of the performance disparity between **CurveFlow** and the baseline **DP3**, we conducted a frame-by-frame analysis of 80 evaluation trials (20 per task). We visualized representative success and failure frames in Fig.4.

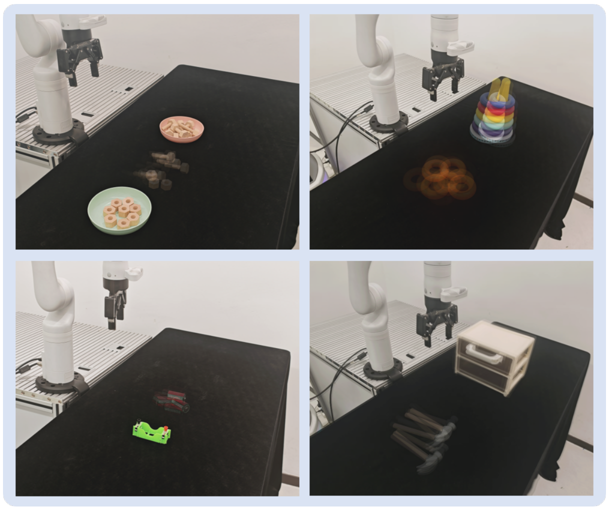
Fig. 3. Randomized initial object configurations across tasks.

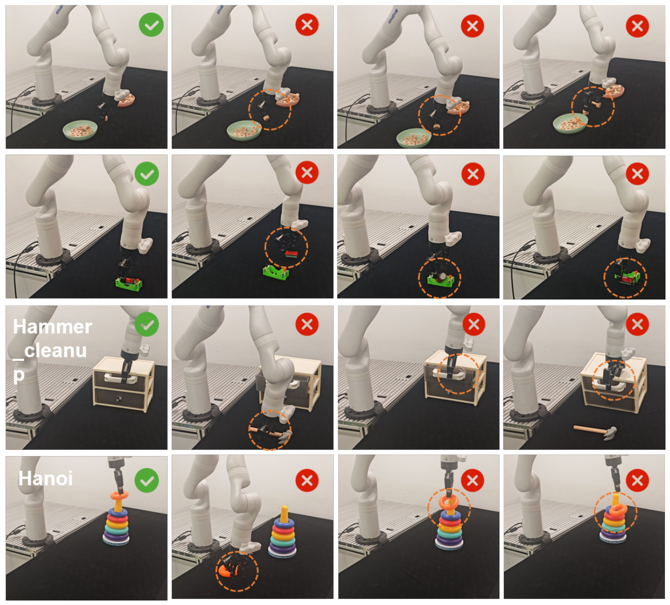

Fig. 4. Gallery of real-world experiments. The leftmost column shows successful executions, contrasted with specific failure modes shown in the three right columns.

As quantified in Table IV, the baseline methods exhibit a significantly higher failure rate in precision-critical phases. We categorize the primary failure modes as follows:

A. Detailed Failure Analysis

To provide a granular understanding of the performance disparity between CurveFlow and the baseline DP3, we conducted a frame-by-frame analysis of 80 evaluation trials (20 per task). We visualized representative success and failure sequences in Fig. 4. As quantified in Table IV, the baseline methods exhibit a significantly higher failure rate in precision-critical phases. We categorize the primary failure modes as follows:

1) Execution Failure: Trajectory Discontinuity. This is the most prevalent failure mode for DP3 (18 out of 28 failures). It is characterized by sudden jerks or "stop-and-go" motion. The linear probability transport assumption in DP3 often forces the generative process to traverse low-density regions of the action manifold. This yields physically inconsistent waypoints near object boundaries, compelling the low-level controller to make abrupt corrections (e.g., a sudden 15° wrist rotation), causing the gripper to slip or collide with the target socket.

2) Planning Failure: Kinematic Singularity. Standard flow matching policies occasionally guide the robot into configurations near kinematic singularities (6 cases for DP3, 2 cases for CurveFlow). Lacking curvature constraints, the generated path may demand infinite joint velocities to maintain the linear end-effector trajectory, causing the robot to trigger safety stops.

3) Perception Failure: Visual Occlusion/Drift. According to Table IV, a minority of failures are attributed to perception issues (4 case for DP3, 2 cases for CurveFlow). These failures typically occur during the close-range manipulation phase where the end-effector occludes the camera view, or due to slight extrinsic drift over long-term operation. Although CurveFlow has one more case in this category, the overall low numbers suggest both methods are relatively robust in static perception, but this remains a challenge under extreme occlusion.

Analysis of CurveFlow's Robustness. Compared to DP3, CurveFlow significantly reduces the aforementioned failures. By explicitly regularizing the rate of change of the trajectory coefficients ($\dot{a}_\phi, \dot{b}_\psi$), this curvature-aware modeling acts as a "geometric damper". It effectively smooths out high-frequency noise and ensures that the generated waypoints respect the manifold's intrinsic continuity. Consequently, CurveFlow reduces contact-induced failures by over 50%, maintaining stable manipulability even in narrow-tolerance tasks like battery insertion.

TABLE IV
DETAILED FAILURE MODE BREAKDOWN (REAL WORLD)

Method	Execution Failure (Oscillation/Jerk)	Planning Failure (Singularity/Safety Stop)	Perception Failure (Occlusion/Drift)	Total Failures
DP3	18	6	4	28
CurveFlow	4	2	2	8

REFERENCES

[1] C. Deng, O. Litany, Y. Duan, A. Poulenard, A. Tagliasacchi, and L. J. Guibas, "Vector neurons: A general framework for SO(3)-equivariant networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2021, pp. 12200–12209.