APPENDIX

A. In-the-Wild Tasks

For In-the-Wild tasks, we consider:

1) Pull Out Chair: In the Pull Out Chair task, a black rolling chair is placed underneath a table. A trial is considered a success if the chair is moved at least 5cm horizontally from its initial position. The reason for the limited motion requirement is that in certain configurations, it is difficult for the Franka robot to push the chair.

2) Open Drawer: The Open Drawer task involves opening a partially opened drawer out to at least 90% of its full possible extension.

3) Sweep Pasta: In the Sweep Pasta task, there is a brush with a green handle placed against a wooden support structure, along with four pieces of dried pasta next to a compost bin. The objective is for the robot to grasp the handle of the brush and use the brush to push the pasta into the compost bin. If all pieces of pasta are inside of the compost bin, it is considered a success.

4) Recycle Can: In the Recycle Can task, an aluminum can is placed in between a recycling and trash bin while upright. A trial is successful if the can is inside of the recycling bin.

See Fig. 7 for rollouts for each of these in-the-wild tasks.

B. Particle Dynamics Model

The particle dynamics model used in the Push-T task takes as input feature-augmented particles $\tilde{x}_t \in \mathbb{R}^{N\times 14}$, consisting of the position, RGB value, normal vector, and push parameters and produces $\Delta\hat{x}_{t+1}$, the delta of positions of each point for the next timestep. The architecture of the particle dynamics model is a small Point Transformer V3 [89] backbone surrounded by MLPs to project between the desired input and output sizes.

To get the input set of particles from the scene, there are 4 virtual cameras around the workspace, whose RGB-D observations are combined into a single point cloud. Then, points outside of the bounds of the wooden platform that the T-shaped block is on are discarded. Finally, this point cloud is downsampled by randomly keeping only 1 particle per each 1.5cm sided cube. The same push parameter is appended to each particle's features.

To train the particle dynamics model, we collect 500 transitions of random pushing actions. In this setting, we track particle positions before and after the push by using all objects' poses from the simulator for efficiency, but in practice this can also be tracked with CoTrackerV3 [3] and depth.

C. Push-T Planning Details

To optimize the particle trajectory following cost with the learned particle based dynamics model, we use random shooting, where r push skill parameters are randomly sampled such that all pushes will make contact with the object of interest at different points and directions. Then, we select the push skill parameter out of those r ones that has the least cost according to the predicted particle positions from the dynamics model with respect to a subgoal of particle positions. Since Dream2Flow tracks particles for the object of interest instead of the entire scene and the particle dynamics model takes in downsampled particles, we perform nearest neighbor matching to find correspondences between the particles in the trajectory and the feature-augmented particles which are the input to the dynamics model. With these correspondences, and the delta position output, we compute the predicted particle positions of the object of interest as $\hat{x}_{t+1} = \hat{x}_t + \Delta\hat{x}_{t+1}$.

For determining which timestep from the video should be used in the cost function as the subgoal, we find the timestep $t^\star$ where the particles of the relevant object are closest to the observed particles, as a single push can encompass the motion of many timesteps. We then choose the particles from the timestep $\min(t^\star + L, t_{\text{end}})$ as the next subgoal for planning, where L is a fixed look ahead time amount, and t_{end} is the final timestep of the video. In our experiments, we used $L = 20$.

D. Real World Planning Details

The optimization problem with a rigid-grasp assumption in the real world follows the same formulation as in Sec. III-A, where we optimize robot joint angles $\mathbf{q} \in \mathbb{R}^7$ to minimize:

$$\sum_{t=0}^{H-1} \lambda_{\text{task}}\left(\hat{x}_t^{\text{obj}}, \tilde{P}_t\right) + \lambda_{\text{control}}(\hat{x}_t, u_t) \tag{1}$$

The control cost $\lambda_{\text{control}}(\hat{x}_t, u_t)$ is expanded into three components:

$$\lambda_{\text{control}}(\hat{x}_t, u_t) = w_r\mathcal{C}_r(\mathbf{q}_t) + w_s\mathcal{C}_s(\mathbf{q}_t, \mathbf{q}_{t-1}) + w_m\mathcal{C}_m(\mathbf{q}_t) \tag{2}$$

where:

- $\mathcal{C}_r(\mathbf{q}_t)$: Reachability cost penalizing joint configurations outside the robot's workspace
- $\mathcal{C}_s(\mathbf{q}_t, \mathbf{q}_{t-1})$: Pose smoothness cost measuring the difference between consecutive end-effector poses after running forward kinematics
- $\mathcal{C}_m(\mathbf{q}_t)$: Manipulability cost encouraging configurations with good manipulability

The weight parameters are set to $w_r = 100$, $w_s = 1$, and $w_m = 0.01$ to encourage the robot to make smooth motions within its joint limits. The task cost λ_{task} uses a weight of $w_f = 10$ and follows the same formulation as in Sec. III-A. The optimized end-effector poses after running forward kinematics are then fit with a B-spline and sampled such that each sampled pose is at least 1cm away from the previous and next pose and/or has a rotation difference of at least 20 degrees.

E. Video Generation Failure Visualization

From the generated videos, we observe that there are two common failure modes: morphing and hallucination. Object morphing occurs when an existing object in the scene dramatically changes shape to something else, such as another object or an object with significantly different geometric

Fig. 7: **In-the-Wild Task Rollouts.** The Franka successfully pulls out a chair, opens a partially opened drawer, sweeps pasta into the compost bin, and recycles an aluminum can.

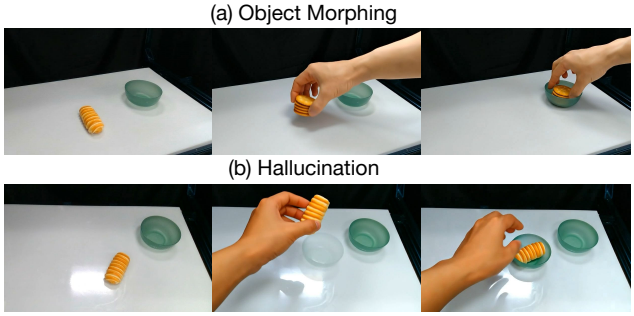

Fig. 8: **Video Generation Failure Examples.** (a) The bread undergoes object morphing, where it turns into a stack of crackers, causing the downstream tracking to fail. (b) Another green bowl appears due to model hallucination, causing a downstream execution failure where the bread is dropped onto the surface of the workspace instead of the original green bowl.

properties than what it should be. Hallucination occurs when a new object (previously non-existent) appears in the scene. We show examples of morphing and hallucination for the Put Bread task in Fig. 8.