A. Implementation Details

A.1. Perceiver-Actor

The high-level Perceiver-Actor (PerAct) agent is a language-conditioned multi-task behaviour-cloning agent $\pi_{\text{high}}(a \mid o, l)$, where $a_{\text{high}} = (a_{\text{pose}}, a_{\text{grip}})$, o consists of calibrated RGB-D multi-view images, and l is a language description of the task. First, PerAct encodes the language description using the frozen pretrained CLIP [30] language encoder. For the RGB-D images, PerAct computes the 3D position utilising camera intrinsics and extrinsics to obtain a 3D voxel grid representation of the current scene. Next, PerAct employs PerceiverIO to encode both language and voxel tokens with a fixed set of latent vectors. Finally, these vectors are decoded into a 3D action-value attention map. Specifically, we follow the official implementation and use 100^3 voxels to represent the scene, and encode it 2048 fixed latent vectors with dimension 512. During training, we perform data augmentations by offsetting the voxels with random translations and rotations. During training, we minimise the following loss function

$$\begin{aligned}
\mathcal{L}_{\text{high}} &= -\mathbb{E}_{k\sim\xi,\xi\sim\mathcal{D}}\left[\log \pi_{\text{high}}(a_{\text{demo}}(k) \mid o, l)\right] \\
&= -\mathbb{E}_{k\sim\xi,\xi\sim\mathcal{D}}\left[\log \pi_{\text{trans}} + \log \pi_{\text{rot}} + \log \pi_{\text{grip}}\right] \\
\pi_{\text{trans}} &= \text{softmax}\left[Q_{\text{trans}}((x, y, z) \mid o, l)\right] \\
\pi_{\text{rot}} &= \text{softmax}\left[Q_{\text{rot}}((\psi_{\text{rot}}, \theta_{\text{rot}}, \phi_{\text{rot}}) \mid o, l)\right] \\
\pi_{\text{grip}} &= \text{softmax}\left[Q_{\text{grip}}(\omega \mid o, l)\right], \qquad (12)
\end{aligned}$$

where Q_{trans}, Q_{rot}, and Q_{grip} are the discrete Q functions for the voxel-based translation policy π_{trans}, the discrete rotation policy π_{rot}, and the discrete binary gripper opening / closing policy π_{grip}. Here, (x, y, z) are the target voxel indices, $(\psi_{\text{rot}}, \theta_{\text{rot}}, \phi_{\text{rot}})$ are the yaw, pitch, and roll parameters for a rotation, and ω is a binary value for gripper opening / closing. In particular, unlike the original PerAct with an additional action head $\pi_{\text{collision}}$ to predict whether to ignore collision during RRT planning, we use RK-Diffuser as our low-level agent which handles collision checking automatically. Thus, we ignore $\pi_{\text{collision}}$ for the high-level agent. During inference, PerAct directly takes the argmax of the discrete policy and reconstruct the continuous actions by indexing with the discrete indices. We directly take the hyper-parameters from the original PerAct [34].

A.2. RK-Diffuser

As discussed in the main text, in RK-Diffuser, we learn two separate diffusion models π_{joint} and π_{pose}. As discussed in Sect. 4.3, takes as the input the same conditions C_{pose}, including the known start pose $a(0)^0_{\text{pose}}$, the predicted next-best pose by the high-level agent $\hat{a}^0_{\text{pose}}(T)$, the low-dimensional states s of the robot, the gripper open amount g, and the point cloud of the environment v. Different from the voxel-based representation used in the high-level agent, we use point cloud for the low-level agent given the fact that low-level RK-Diffuser only requires understanding the 3D configuration of the environment, without predicting the action-values over the empty voxels. Besides, using voxels are computationally more expensive than the point cloud. We use only the front camera. During inference, to perform conditional sampling, we use classifier-free guidance following Ho and Salimans [9].

For each vector input, we use a MLP with 2 hidden layers of sizes 128 and 512 with GELU activations [8]. For the point cloud, we use a standard PointNet++ [29] as the encoder. We follow Janner et al. [21] and use a Conv1D UNet [33] as the temporal feature extractor. We use 3 repeated down-sampling residual blocks and 3 additional up-sampling residual blocks for the UNet. We train the goal-conditioned low-level setup in a multi-task setup, with 100 demonstrations per task. We optimise the networks for 100K steps with AdamW optimiser [25]. In addition, to compensate for the imperfect next-best pose predictions of the high-level agent, we perform data augmentations to the start pose and end pose by additionally taking $a(0 + k)$ and $a(T - k)$ as start / end pose along a sub-trajectory ξ, where $k \sim \mathcal{U}[0, 5]$. We determine the optimal hyper-parameters through initial validation on three challenging RLBench tasks: open oven, open microwave, and open grill. These parameters are then applied across all tasks in both simulation and real-world settings. Our search covered the following hyper-parameters and training setups, with the selected parameters highlighted in bold: 1) Denoising steps: *10, 50,* ***100***; 2) Noise initialisation method: ***normal distribution*** or *uniform distribution*; 3) Approach: *predicting the noise* ϵ or ***directly predicting the observation*** x_0.

B. Additional Results

B.1. Real-Robot Experiment Results

Table 3. Real-robot experiment results for opening oven and sorting objects into drawer tasks.

	Open Oven		Sort Objects into Drawer					Overall
	reach	open	open	beetle	bear	cube	close	
%	100	100	100	85	100	85	100	95.71

B.2. PerAct + Path Generation Baseline

In the work [15], a set of trajectories are generated and ranked, reflecting the likelihood of success. Trajectories are generated using three methods including: sampled (linear + RRT), Bezier and learnt. Subsequently, over environment rollouts, an RL policy is trained to evaluate and rank these trajectories, with the highest being taken forward for execution. In the Behaviour Cloning setting, this direct approach is not applicable due to the absence of environment rollouts.

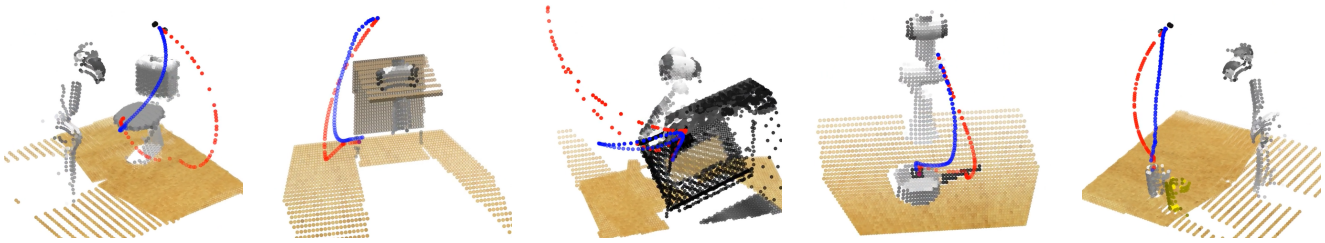

Figure 6. During training, we compute the rank of the trajectories, i.e., the optimality of the trajectory, to encourage the agent to differentiate quality of the sub-optimal trajectories generated by sample-based planners. During inference, we encourage RK-Diffuser to generate high-rank trajectories only, i.e., shorter trajectories. In this figure, red trajectories denote the ground-truth trajectories generated by planners, and blue trajectories are generated by RK-Diffuser, which has shorter lengths while satisfying the kinematics-constraints.

Nevertheless, as part of the demonstration creation in RL-Bench [19], the path generation method is recorded at each environment step and can be used as a training signal. To make use of this, an additional head is added to the PerAct backbone, which is trained to predict the optimal planning method – either sampled or Bezier – in addition to generating next-best-pose, gripper and ignore collision outputs. In this setting, the concept of collisions, as implemented in [34], is crucial to enable reliable planning. Therefore, we adapt the loss function in equation 12, incorperating the ignore collision policy, to include a path generation policy

$$\begin{aligned}\mathcal{L}_{\text{high}} &= -\mathbb{E}_{k\sim\xi,\xi\sim\mathcal{D}}\left[\log \pi_{\text{trans}} + \log \pi_{\text{rot}}\right. \\ &\quad \left. + \log \pi_{\text{grip}} + \log \pi_{\text{collision}} + \log \pi_{\text{path}}\right] \\ \pi_{\text{path}} &= \text{softmax}\left[Q_{\text{path}}(\lambda \mid o, l)\right], \end{aligned} \tag{13}$$

where Q_{path} is the discrete Q function for the voxel-based path planning policy π_{path}. Here, λ is a discrete selection for optimal planner method. During inference, if the model determines that a sampled approach is optimal, it initially attempts a linear path, followed by RRT planning if necessary. In the case of the Bezier method, the model samples a set of random curvature parameters, executing the first successful configuration, if any.

B.3. Trajectory Ranking

As discussed in Sect. 4.3, we include an additional conditional variable, trajectory rank, into RK-Diffuser. We define trajectory rank as $r_\xi = \frac{d_{\text{Euclidean}}}{d_{\text{travel}}}$, where $d_{\text{Euclidean}}$ is the Euclidean distance between the start and end pose and d_{travel} is the travelled distance between the start and end pose. Intuitively, an optimal trajectory, ignoring the kinematics constraint of the robot, should have $r_\xi = 1$. We provide additional visualisations to analyse the effect of trajectory ranking in Fig. 6, where the red curves denote the ground-truth trajectory, and blue ones are the trajectories sampled from RK-Diffuser. We observe that the RK-Diffuser is capable to generate shorter trajectories while trained with sub-optimal planner demonstrations.