

Shelving, Stacking, Hanging: Relational Pose Diffusion for Multi-modal Rearrangement – Supplementary Material

Section A1 includes additional visualizations of iterative test-time evaluation on simulated shapes and examples of object-scene point clouds that were used as training data. In Section A2, we present details on data generation, model architecture, and training for RPDiff. In Section A3 we elaborate in more detail on the multi-step iterative regression inference procedure which predicts the set of rearrangement transforms. Section A4 describes more details about how the success classifier is trained and used in conjunction with our transform predictor as a simple mechanism for selecting which among multiple candidate transforms to execute. In Section A5, we describe more details about our experimental setup, and Section A6 discusses more details on the evaluation tasks and robot execution pipelines. In Section A7 we present an additional set of ablations to highlight the impact of other hyperparameters and design decisions. Section A8 describes additional implementation details for the real-world executions along with an expanded discussion on limitations and avenues for future work. Finally, Section A9 shows model architecture diagrams a summarized set of relevant hyperparameters that were used in training and evaluation.

A1 Additional Test-time and Training Data Visualizations

Here, we show additional visualizations of the tasks used in our simulation experiments and the noised point clouds used to train our pose regression model. Figure A1 shows snapshots of performing the iterative de-noising at evaluation time with simulated objects, and Figure A2 shows examples of the combined object-scene point clouds and their corresponding noised versions that were used for training to perform iterative de-noising.

A2 Iterative Pose Regression Training and Data Generation

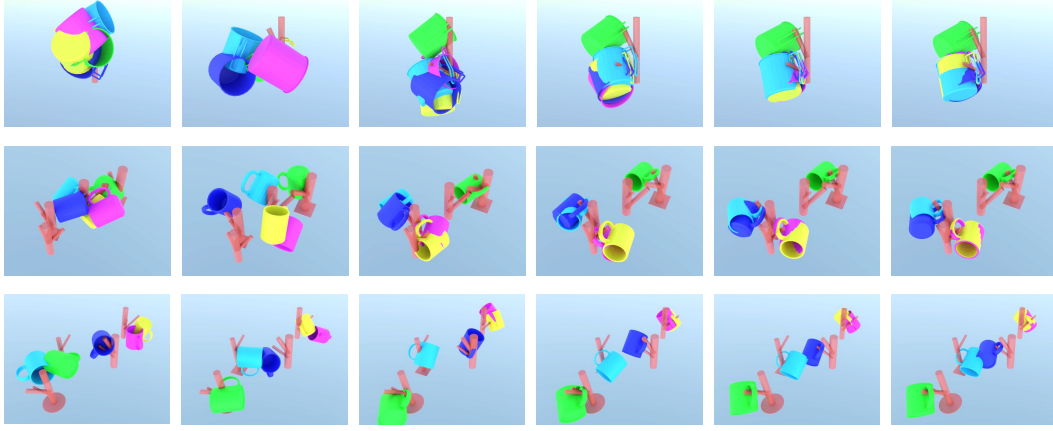
In this section, we present details on the data used for training the pose diffusion model in RPDiff, the neural network architecture we used for processing point clouds and predicting SE(3) transforms, and details on training the model.

A2.1 Training Data Generation

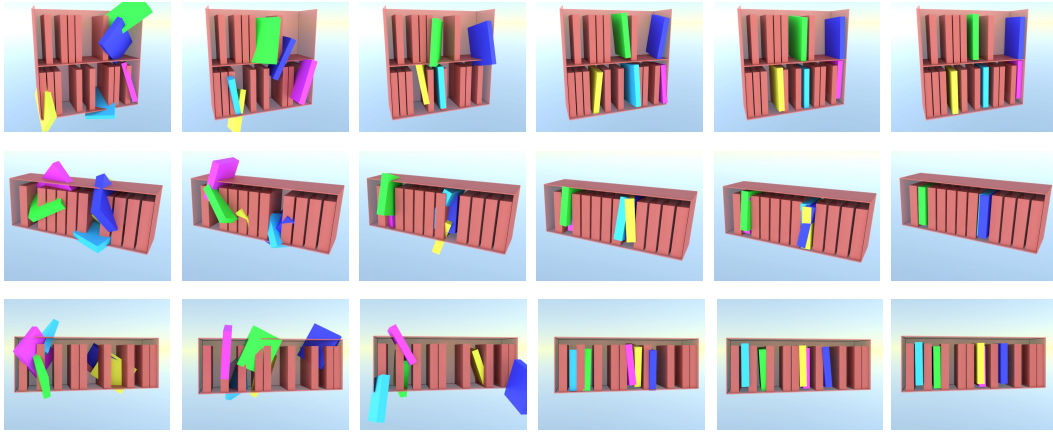
Objects used in simulated rearrangement demonstrations. We create the rearrangement demonstrations in simulation with a set of synthetic 3D objects. The three tasks we consider include objects from five categories: mugs, racks, cans, books, “bookshelves” (shelves partially filled with books), and “cabinets” (shelves partially-filled with stacks of cans). We use ShapeNet [67] for the mugs and procedurally generate our own dataset of .obj files for the racks, books, shelves, and cabinets. See Figure A3 for representative samples of the 3D models from each category.

Procedurally generated rearrangement demonstrations in simulation. The core regression model f_θ in RPDiff is trained to process a combined object-scene point cloud and predict an SE(3) transformation updates the pose of the object point cloud. To train the model to make these relative pose predictions, we use a dataset of demonstrations showing object and scene point clouds in final configurations that satisfy the desired rearrangement tasks. Here we describe how we obtain these “final point cloud” demonstrations

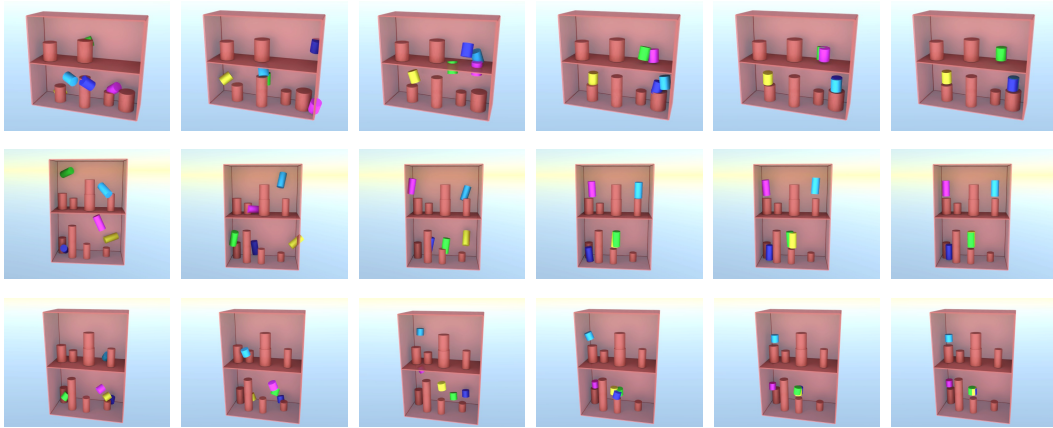
We begin by initializing the objects on a table in PyBullet [19] in random positions and orientations and render depth images with the object segmented from the background using multiple simulated cameras. These depth maps are converted to 3D point clouds and fused into the world coordinate frame using known camera poses. To obtain a diverse set of point clouds, we randomize the number of cameras (1-4), camera viewing angles, distances between the cameras and objects, object scales, and object poses. Rendering point clouds in this way allows the model to see some of the occlusion patterns that occur when the objects are in different orientations and cannot be viewed from below the table. To see enough of the shelf/cabinet region, we use the known state of the shelf/cabinet to position two cameras that roughly point toward the open side of the shelf/cabinet.



(a) Mug/Rack



(b) Book/Shelf



(c) Can/Cabinet

Figure A1: Visualizations of multiple steps of iterative de-noising on simulated objects. Starting from the left side, each object is initialized in a random $SE(3)$ pose in the vicinity of the scene. Over multiple iterations, RPDiff updates the object pose. The right side shows the final set of converged solutions.

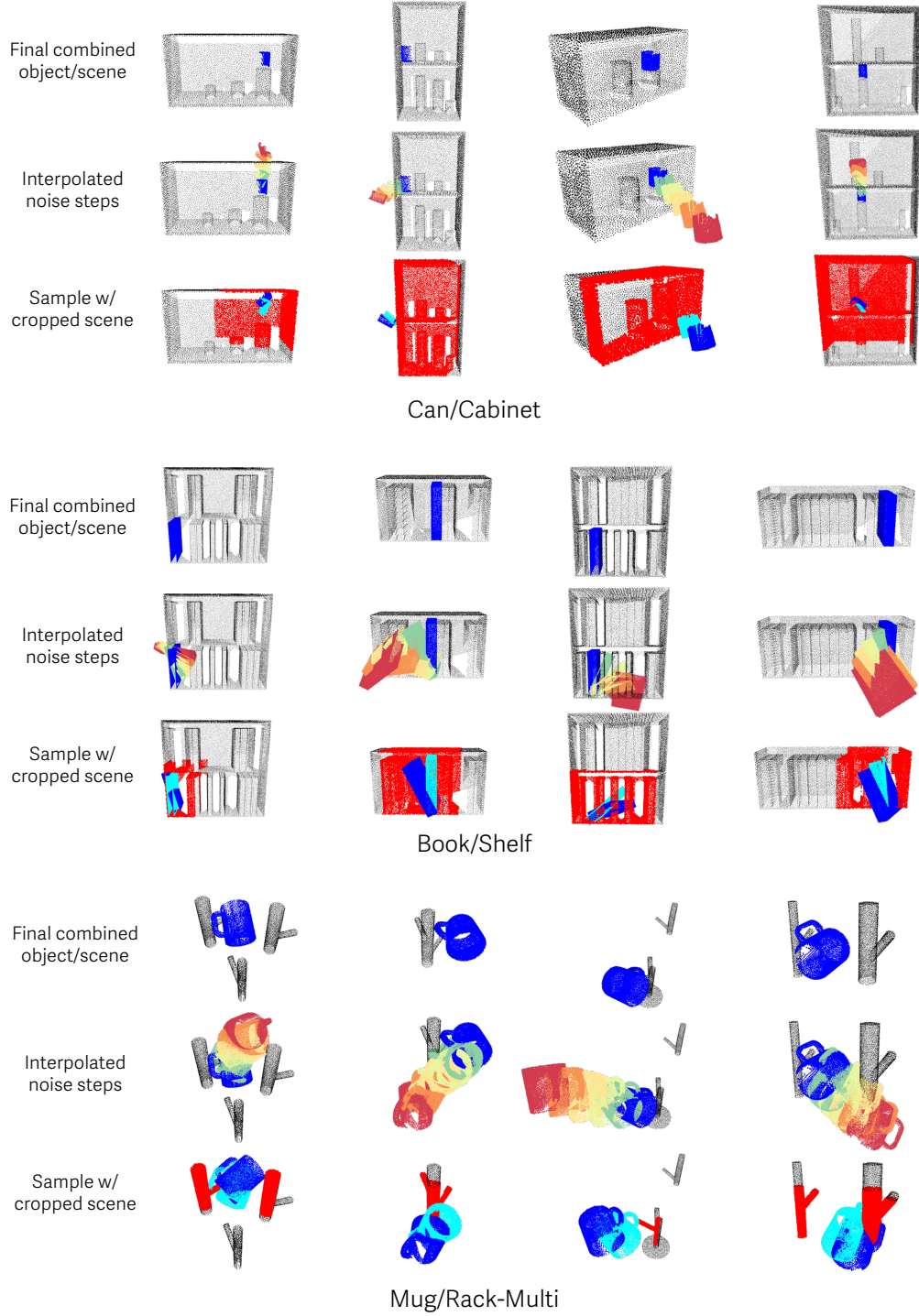


Figure A2: Example point clouds from the demonstrations for each task of **Can/Cabinet** (top), **Book/Shelf** (middle) and **Mug/RackMed-Multi** (bottom). For each task, the top row shows the ground truth combined object-scene point cloud. Scene point clouds are in black and object point clouds are in dark blue. The middle row in each task shows an example of creating multiple steps of noising perturbations by uniformly interpolating a single randomly sampled perturbation transform (with a combination of linear interpolation for the translation and SLERP for the rotation). Different colors show the point clouds at different interpolated poses. The bottom row shows a sampled step among these interpolated poses, with the corresponding “noised” object point cloud (dark blue), ground truth target point cloud (light blue), and cropped scene point cloud (red).

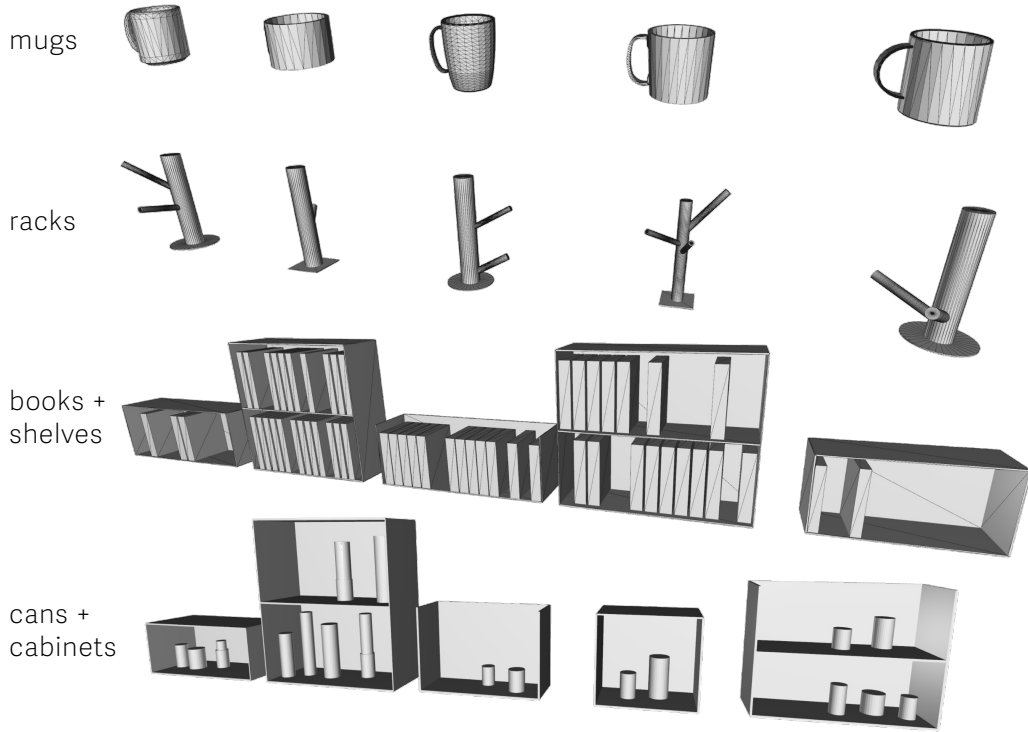


Figure A3: Example 3D models used to train RPDiff and deploy RPDiff on our rearrangement tasks. Mugs are from ShapeNet [67] while we procedurally generated our own synthetic racks, books, cans, shelves, and cabinets.

After obtaining the initial object and scene point clouds, we obtain an $SE(3)$ transform to apply to the object, such that transforming into a “final” object pose using this transform results in the desired placement. This transform is used to translate and rotate the initial object point cloud, such that the combined “final object” and scene point cloud can be used for generating training examples. Figure A2 shows example visualizations of the final point clouds in the demonstrations for each task.

We obtain the final configuration that satisfies these tasks using a combination of privileged knowledge about the objects in the simulator (e.g., ground truth state, approximate locations of task-relevant object parts, 3D mesh models for each object, known placing locations that are available) and human intuition about the task. To create mug configurations that satisfy “hanging” on one of the pegs of a rack, we first approximately locate one of the pegs on one of the racks (we select one uniformly at random) and the handle on the mug (which is straightforward because all the ShapeNet mugs are aligned with the handle pointing in the +y axis of the body frame). We then transform the mug so that the handle is approximately “on” the selected hook. Finally, we sample small perturbations about this nominal pose until we find one that does not lead to any collision/penetration between the two shapes. We perform an analogous process for the other tasks, where the ground truth available slots in the bookshelf and positions that work for placing the mug (e.g., on top of a stack, or on a flat shelf region in between existing stacks) are recorded when the 3D models for the shelves/cabinets are created. The exact methods for generating these shapes and their corresponding rearrangement poses can be found in our code.

A2.2 Pose Prediction Architecture

Transformer point cloud processing and pose regression. We follow the Transformer [15] architecture proposed in Neural Shape Mating [3] for processing point clouds and computing shape features that are fed to the output MLPs for pose prediction.

598 We first downsample the observed point clouds $\mathbf{P}_O \in \mathbb{R}^{N' \times 3}$ and $\mathbf{P}_S \in \mathbb{R}^{M' \times 3}$ using farthest
 599 point sampling into $\bar{\mathbf{P}}_O \in \mathbb{R}^{N \times 3}$ and $\bar{\mathbf{P}}_S \in \mathbb{R}^{M \times 3}$. We then normalize to create $\mathbf{P}_O^{\text{norm}} \in \mathbb{R}^{N \times 3}$
 600 and $\mathbf{P}_S^{\text{norm}} \in \mathbb{R}^{M \times 3}$, based on the centroid of the scene point cloud and a scaling factor that
 601 approximately scales the combined point cloud to have extents similar to a unit bounding box:

$$\begin{aligned} \bar{\mathbf{P}}_S &= \begin{bmatrix} \mathbf{p}_1^S \\ \mathbf{p}_2^S \\ \dots \\ \mathbf{p}_M^S \end{bmatrix} & \bar{\mathbf{P}}_O &= \begin{bmatrix} \mathbf{p}_1^O \\ \mathbf{p}_2^O \\ \dots \\ \mathbf{p}_M^O \end{bmatrix} & \mathbf{p}^{\text{S,cent}} &= \frac{1}{M} \sum_{i=1}^M \mathbf{p}_i^S & \mathbf{a} &= \max\{\mathbf{p}_i^S\} - \min\{\mathbf{p}_i^S\} \\ \mathbf{P}_S^{\text{norm}} &= \begin{bmatrix} \mathbf{p}_1^{\text{S,norm}} \\ \mathbf{p}_2^{\text{S,norm}} \\ \dots \\ \mathbf{p}_M^{\text{S,norm}} \end{bmatrix} & \mathbf{p}_i^{\text{S,norm}} &= \mathbf{a}(\mathbf{p}_i^S - \mathbf{p}^{\text{S,cent}}) \quad \forall i \in 1, \dots, M \\ \mathbf{P}_O^{\text{norm}} &= \begin{bmatrix} \mathbf{p}_1^{\text{O,norm}} \\ \mathbf{p}_2^{\text{O,norm}} \\ \dots \\ \mathbf{p}_M^{\text{O,norm}} \end{bmatrix} & \mathbf{p}_i^{\text{O,norm}} &= \mathbf{a}(\mathbf{p}_i^O - \mathbf{p}^{\text{S,cent}}) \quad \forall j \in 1, \dots, N \end{aligned}$$

602 Next, we “tokenize” the normalized object/scene point clouds into d -dimensional input features
 603 $\phi_O \in \mathbb{R}^{N \times d}$ and $\phi_S \in \mathbb{R}^{M \times d}$ by concatenating a two-dimensional one-hot feature to each point in
 604 $\mathbf{P}_O^{\text{norm}}$ and $\mathbf{P}_S^{\text{norm}}$ (to explicitly inform the Transformer which points correspond to the object and
 605 the scene) and projecting to a d -dimensional vector with a linear layer $\mathbf{W}_{\text{in}} \in \mathbb{R}^{d \times 5}$:

$$\begin{aligned} \phi_S &= \begin{bmatrix} \mathbf{W}_{\text{in}} \bar{\mathbf{p}}_1^{\text{S,norm}} \\ \mathbf{W}_{\text{in}} \bar{\mathbf{p}}_2^{\text{S,norm}} \\ \dots \\ \mathbf{W}_{\text{in}} \bar{\mathbf{p}}_M^{\text{S,norm}} \end{bmatrix} & \bar{\mathbf{p}}_i^{\text{S,norm}} &= \mathbf{p}_i^{\text{S,norm}} \oplus [1, 0] \quad \forall i \in 1, \dots, M \\ \phi_O &= \begin{bmatrix} \mathbf{W}_{\text{in}} \bar{\mathbf{p}}_1^{\text{O,norm}} \\ \mathbf{W}_{\text{in}} \bar{\mathbf{p}}_2^{\text{O,norm}} \\ \dots \\ \mathbf{W}_{\text{in}} \bar{\mathbf{p}}_M^{\text{O,norm}} \end{bmatrix} & \bar{\mathbf{p}}_j^{\text{O,norm}} &= \mathbf{p}_j^{\text{O,norm}} \oplus [0, 1] \quad \forall j \in 1, \dots, N \end{aligned}$$

606 Note we could also pass the point cloud through a point cloud encoder to pool local features together,
 607 as performed in NSM via DGCNN [17]. We did not experiment with this as we obtained satisfactory
 608 results by directly operating on the individual point features, but it would likely perform similarly or
 609 even better if we first passed through a point cloud encoder. We also incorporate the timestep t that
 610 the current prediction corresponds to by including the position-encoded timestep as an additional
 611 input token together with the object point tokens as $\bar{\phi}_O \in \mathbb{R}^{(N+1) \times d}$ where $\bar{\phi}_O = \begin{bmatrix} \phi_O \\ \text{pos_emb}(t) \end{bmatrix}$.

612 We then use a Transformer encoder and decoder to process the combined tokenized point cloud (see
 613 Figure A4 for visual depiction). This consists of performing multiple rounds of self-attention on
 614 the scene features (encoder) and then performing a combination of self-attention on the object point
 615 cloud together with cross-attention between the object point cloud and the output features of the
 616 scene point cloud (decoder):

$$\begin{aligned} q_S &= Q_E(\phi_S) \quad k_S = K_E(\phi_S) \quad v_S = V_E(\phi_S) \\ s_S &= \text{Attention}(q_S, k_S, v_S) = \text{softmax}\left(\frac{q_S k_S^T}{\sqrt{d}}\right) v_S \\ q_O &= Q_D(\bar{\phi}_O) \quad k_O = K_D(\bar{\phi}_O) \quad v_O = V_D(\bar{\phi}_O) \\ s_O &= \text{Attention}(q_O, k_O, v_O) = \text{softmax}\left(\frac{q_O k_O^T}{\sqrt{d}}\right) v_O \\ h_O &= \text{Attention}(q = s_O, k = s_S, v = s_S) = \text{softmax}\left(\frac{s_O s_S^T}{\sqrt{d}}\right) s_S \end{aligned}$$

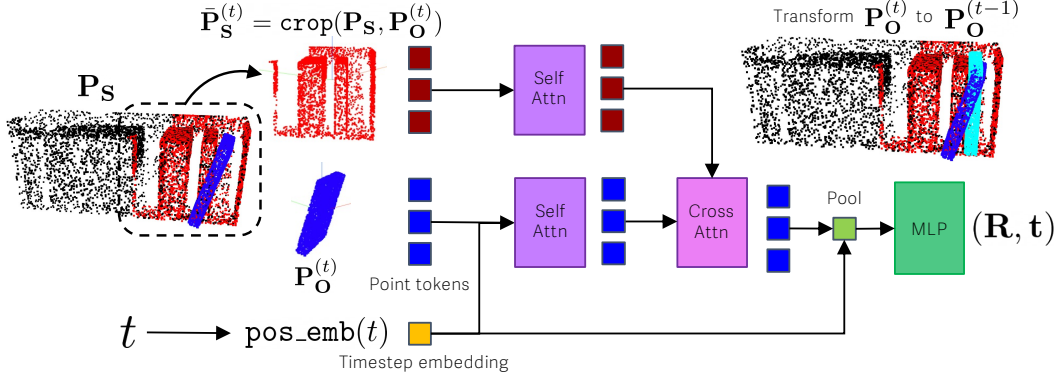


Figure A4: Architecture diagram showing a combination of self-attention and cross-attention among object and scene point cloud for SE(3) transform prediction. The scene point cloud is processed via multiple rounds of self-attention, while the object features are combined via a combination of self-attention and cross-attention with the scene point cloud. The timestep embedding is incorporated as both an input token and via a residual connection with the pooled output feature. The global output feature is used to predict the translation and rotation that are applied to the object point cloud.

This gives a set of output features $h_O \in \mathbb{R}^{(N+1) \times d}$ where d is the dimension of the embedding space. We compute a global feature by mean-pooling the output point features and averaging with the timestep embedding as a residual connection, and then use a set of output MLPs to predict the translation and rotation (the rotation is obtained by converting a pair of 3D vectors into an orthonormal basis and then stacking into a rotation matrix [10, 18]):

$$\begin{aligned} \bar{h}_O &= \frac{1}{2} \left(\frac{1}{N} \sum_{i=1}^{N+1} h_{O,i} + \text{pos_emb}(t) \right) & \bar{h}_O &\in \mathbb{R}^d \\ \mathbf{t} &= \text{MLP}_{\text{trans}}(\bar{h}_O) & \mathbf{t} &\in \mathbb{R}^3 \\ a, b &= \text{MLP}_{\text{rot}}(\bar{h}_O) & a &\in \mathbb{R}^3, b \in \mathbb{R}^3 \\ \hat{a} &= \frac{a}{\|a\|} & \hat{b} &= \frac{b - \langle \hat{a}, b \rangle \hat{a}}{\|b\|} & \hat{c} &= \hat{a} \times \hat{b} \\ \mathbf{R} &= \begin{bmatrix} | & | & | \\ \hat{a} & \hat{b} & \hat{c} \\ | & | & | \end{bmatrix} \end{aligned}$$

Local scene point cloud cropping. As shown in the experimental results, local cropping helps improve performance due to increasing precision while generalizing well to unseen layouts of the scene. Our “Fixed” cropping method uses a box with fixed side length $L_{\text{box}} = L_{\text{min}}$, centered at the current object point cloud iterate across all timesteps, and selects scene point cloud points that lie within this box. Our “Varying” cropping method adjusts the length of the box based on the timestep, with larger timesteps using a larger crop, and smaller timesteps using a smaller crop. We parameterize this as a function of the timestep t via the following linear decay function:

$$L_{\text{box}} = L_{\text{min}} + (L_{\text{max}} - L_{\text{min}}) \frac{T - t}{T}$$

where L_{min} and L_{max} are hyperparameters.

Applying Predicted Transforms to Object Point Cloud. We apply the predicted rotation and translation by first mean-centering the object point cloud, applying the rotation, and then translating back to the original world frame position, and then finally translating by the predicted translation. This helps reduce sensitivity to the rotation prediction, whereas if we rotate about the world frame coordinate axes, a small rotation can cause a large configuration change in the object.

Skill Type	Number of samples
Mug/EasyRack	3190
Mug/MedRack	950
Mug/Multi-MedRack	3240
Book/Shelf	1720
Can/Cabinet	2790

Table 2: Number of demonstrations used in each task. The same set of demonstrations is used to train both our method and each baseline method.

639 A2.3 Training Details

640 Here we elaborate on details regarding training the RPDiff pose diffusion model using the demonstra-
641 tion data and model architecture described in the sections above. A dataset sample consists of a tuple
642 $(\mathbf{P}_O, \mathbf{P}_S)$. From this tuple, we want to construct a perturbed object point cloud $\mathbf{P}_O^{(t)}$ for a particular
643 timestep $t \in 1, \dots, T$, where lower values of t correspond to noised point clouds that are more similar
644 to the ground truth, and larger values of T are more perturbed. At the limit, the distribution of point
645 clouds corresponding to $t = T$ should approximately match the distribution we will sample from
646 when initializing the iterative refinement procedure at test time.

647 Noising schedules and perturbation schemes are an active area of research currently in the diffusion
648 modeling literature [68, 69], and there are many options available for applying noise to the data
649 samples. We apply a simple method that makes use of uniformly interpolated SE(3) transforms.
650 First, we sample one “large” transform from the same distribution we use to initialize the test-time
651 evaluation procedure from – rotations are sampled uniformly from SO(3) and translations are
652 sampled uniformly within a bounding box around the scene point cloud. We then use a combination
653 of linear interpolation on the translations, and spherical-linear interpolation (SLERP) on the rotations,
654 to obtain a sequence of T uniformly-spaced transforms (see Fig. A2 for example visualizations).
655 Based on the sampled timestep t , we select the transform corresponding to timestep t in this sequence
656 as the noising perturbation $\mathbf{T}_{\text{noise}}^{(t)}$, and use the transform corresponding to timestep $t - 1$ to compute
657 the “incremental”/“interval” transform to use as a prediction target. As discussed in Section 3.1, using
658 the incremental transform as a prediction target helps maintain a more uniform output scale among
659 the predictions across samples, which is beneficial for neural network optimization as it minimizes
660 gradient fluctuations [14]. We also provide quantitative evidence that predicting only the increment
661 instead of the full inverse perturbation benefits overall performance. See Section A7 for details.

662 The main hyperparameter for this procedure is the number of steps T . In our experiments, we
663 observed it is important to find an appropriate value for T . When T is too large, the magnitude of the
664 transforms between consecutive timesteps is very small, and the iterative predictions at evaluation
665 time make tiny updates to the point cloud pose, oftentimes failing to converge. When T is too small,
666 most of the noised point clouds will be very far from the ground truth and might look similar across
667 training samples but require conflicting prediction targets, which causes the model to fit the data
668 poorly. We found that values in the vicinity of $T = 5$ work well across our tasks ($T = 2$ and $T = 50$
669 both did not work well). This corresponds to an average perturbation magnitude of 2.5cm for the
670 translation and 18 degrees for the rotation.

671 After obtaining the ground truth prediction target, we compute the gradient with respect to the loss
672 between the prediction and the ground truth, which is composed of the mean-squared translation error,
673 a geodesic rotation distance error [12, 13], and the chamfer distance between the point cloud obtained
674 by applying the predicted transform and the ground-truth next point cloud. We also found the model
675 to work well using either just the chamfer distance loss or the combined translation/rotation losses.

676 We trained a separate model for each task, with each model training for 500 thousand iterations on
677 a single NVIDIA V100 GPU with a batch size of 16. We used a learning rate schedule of linear
678 warmup and cosine decay, with a maximum learning rate of $1e-4$. Training takes about three days.
679 We train the models using the AdamW [70] optimizer. Table 2 includes the number of demonstrations
680 we used for each task.

A3 Test time evaluation

Here, we elaborate in more detail on the iterative de-noising procedure performed at test time. Starting with \mathbf{P}_O and \mathbf{P}_S , we sample K initial transforms $\{\hat{\mathbf{T}}_k^{(I)}\}_{k=1}^K$, where initial rotations are drawn from a uniform grid over $\text{SO}(3)$, and we uniformly sample translations that position the object within the bounding box of the scene point cloud. We create K copies of \mathbf{P}_O and apply each corresponding transform to create initial object point clouds $\{\hat{\mathbf{P}}_{O,k}^{(I)}\}_{k=1}^K$ where $\hat{\mathbf{P}}_{O,k}^{(I)} = \hat{\mathbf{T}}_k^{(I)} \mathbf{P}_O$. We then perform the following update for I steps for each of the K initial transforms:

$$\hat{\mathbf{T}}^{(i-1)} = (\mathbf{T}_{\Delta}^{\text{Rand}} \hat{\mathbf{T}}_{\Delta}) \hat{\mathbf{T}}^{(n)} \quad \hat{\mathbf{P}}_O^{(n-1)} = (\mathbf{T}_{\Delta}^{\text{Rand}} \hat{\mathbf{T}}_{\Delta}) \hat{\mathbf{P}}_O^{(i)}$$

where transform $\hat{\mathbf{T}}_{\Delta}$ is obtained as $\hat{\mathbf{T}}_{\Delta} = f_{\theta}(\hat{\mathbf{P}}_O^{(i)}, \mathbf{P}_S, \text{pos_emb}(\text{i_to_t}(i)))$. Transform $\mathbf{T}_{\Delta}^{\text{Rand}}$ is sampled from a timestep-conditioned uniform distribution that converges toward deterministically producing an identity transform as i tends toward 0. We obtain the random noise by sampling from a Gaussian distribution for both translation and rotation. For the translation, we directly output a 3D vector with random elements. For the rotation, we represent the random noise via axis angle 3D rotation $\mathbf{R}_{\text{aa}}^0 \in \mathbb{R}^3$ and convert it to a rotation matrix using the $\text{SO}(3)$ exponential map [71] (and a 3D translation $\mathbf{t}^0 \in \mathbb{R}^3$). We exponentially decay the variance of these noise distributions so that they produce nearly zero effect as the iterations tend toward 0. We perform the updates in a batch. The full iterative inference procedure can be found in Alg. 1.

Evaluation timestep scheduling and prediction behavior for different timestep values.. The function `i_to_t` is used to map the iteration number i to a timestep value t that the model has been trained on. This allows the number of steps during evaluation (I) to differ from the number of steps during training (T). For example, we found values of $T = 5$ to work well during training but used a default value of $I = 50$ for evaluation. We observed this benefits performance since running the iterative evaluation procedure for many steps helps convergence and enables “bouncing out” of “locally optimal” solutions. However, we found that if we provide values for i that go beyond the support of what the model is trained on (i.e., for $i > T$), the predictions perform poorly. Thus, the function `i_to_t` ensures all values $i \in 1, \dots, I$ are mapped to an appropriate value $t \in 1, \dots, T$ that the model has seen previously.

There are many ways to obtain this mapping, and different implementations produce different kinds of behavior. This is because different `i_to_t` schedules emphasize using the model in different ways since the model learns qualitatively different behavior for different values of t . Specifically, for smaller values of t , the model has only been trained on “small basins of attraction” and thus the predictions are more precise and local, which allows the model to “snap on” to any solution in the immediate vicinity of the current object iterate. Figure A5 shows this in a set of artificially constrained evaluation runs where the model is constrained to use the *same* timestep for every step $i = 1, \dots, I$.

However, this can also lead the model to get stuck near regions that are far from any solution. On the other hand, for larger perturbations, the data starts to look more multi-modal and the model averages out toward either a biased solution in the direction of a biased region, or just an identity transform that doesn’t move the object at all.

We find the pipeline performs best when primarily using predictions corresponding to smaller timesteps, but still incorporating predictions from higher timesteps. We thus parameterize the timestep schedule `i_to_t` such that it exponentially increases the number of predictions used for smaller values of t . While there are many ways this can be implemented, we use the following procedure: we construct an array D of length I where each element lies between 1 and T , and define the mapping `i_to_t` as

$$t = \text{i_to_t}(i) = D_i \quad \text{subscript } i \text{ denotes the } i\text{-th element of } D$$

The array D is parameterized by a constant value A (where higher value of A corresponds to using more predictions with smaller timesteps, while $A = 1$ corresponds to using each timestep an equal number of times) and ensures that predictions for each timestep are made at least once:

Algorithm 1 Rearrangement Transform Inference via Iterative Point Cloud De-noising

```
1: Input: Scene point cloud  $\mathbf{P}_S$ , object point cloud  $\mathbf{P}_O$ , number of parallel runs  $K$ , number of
   iterations to use in evaluation  $I$ , number of iterations used in training  $T$ , pose regression model  $f_\theta$ ,
   success classifier  $h_\phi$ , function to map from evaluation iteration values to training iteration values
   i_to_t, parameters for controlling what fraction of evaluation iterations correspond to smaller
   training timestep values  $A$ , local cropping function crop, distribution for sampling external pose
   noise  $p_{\text{AnnealedRandSE}(3)}$ 

   # Init transforms, transformed object, and cropped scene
2: for  $k$  in  $1, \dots, K$  do
3:    $\mathbf{R}_k^{(H)} \sim p_{\text{UnifSO}(3)}(\cdot)$ 
4:    $\mathbf{t}_k^{(H)} \sim p_{\text{UnifBoundingBox}}(\cdot \mid \mathbf{P}_O, \mathbf{P}_S)$ 
5:    $\hat{\mathbf{T}}_k^{(H)} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$ 
6:    $\hat{\mathbf{P}}_{O,k}^{(H)} = \hat{\mathbf{T}}_k^{(H)} \mathbf{P}_O$ 
7:    $\hat{\mathbf{P}}_{S,k}^{(H)} = \text{crop}(\hat{\mathbf{P}}_{O,k}^{(H)}, \mathbf{P}_S)$ 
8: end for
   # Init set of transform and final point cloud solutions and classifier scores
9: init  $\mathcal{S} = \emptyset$ 
10: init  $\mathcal{T} = \emptyset$ 
11: init  $\mathcal{P} = \emptyset$ 
   # Iterative pose regression
12: for  $i$  in  $I, \dots, 1$  do
   # Map evaluation timestep to in-distribution training timestep
13:    $t = \text{i\_to\_t}(i, A)$ 
14:   for  $k$  in  $1, \dots, K$  do
15:      $\hat{\mathbf{T}}_{\Delta,k} = f_\theta(\mathbf{P}_{O,k}, \hat{\mathbf{P}}_{S,k}, \text{pos\_emb}(t))$ 
16:     if  $i > (0.2 * I)$  then
       # Apply random external noise, with noise magnitude annealed as  $i$  approaches 0
17:        $\mathbf{T}_{\Delta,k}^{\text{Rand}} \sim p_{\text{AnnealedRandSE}(3)}(\cdot \mid i)$ 
18:     else
       # Remove all external noise for the last 20% of the iterations
19:        $\mathbf{T}_{\Delta,k}^{\text{Rand}} = \mathbf{I}_4$ 
20:     end if
21:      $\hat{\mathbf{T}}_k^{(i-1)} = \mathbf{T}_{\Delta,k}^{\text{Rand}} \hat{\mathbf{T}}_{\Delta,k} \hat{\mathbf{T}}_k^{(i)}$ 
22:      $\hat{\mathbf{P}}_{O,k}^{(i-1)} = \mathbf{T}_{\Delta,k}^{\text{Rand}} \hat{\mathbf{T}}_{\Delta,k} \hat{\mathbf{P}}_{O,k}^{(i)}$ 
23:      $\hat{\mathbf{P}}_{S,k}^{(i-1)} = \text{crop}(\hat{\mathbf{P}}_{O,k}^{(i-1)}, \mathbf{P}_S, t, T)$ 
24:     if  $i == 1$  then
       # Predict success probabilities from final objects
25:        $s_k = h_\phi(\hat{\mathbf{P}}_{O,k}^{(0)}, \mathbf{P}_S)$ 
       # Save final rearrangement solutions and predicted scores
26:        $\mathcal{S} = \mathcal{S} \cup \{s_k\}$ 
27:        $\mathcal{T} = \mathcal{T} \cup \{\hat{\mathbf{T}}_k^{(0)}\}$ 
28:        $\mathcal{P} = \mathcal{P} \cup \{\hat{\mathbf{P}}_{O,k}^{(0)}\}$ 
29:     end if
30:   end for
31: end for
   # Decision rule (e.g., argmax) for output
32:  $k^{\text{out}} = \text{argmax}(\mathcal{S})$ 
33:  $\mathbf{T}^{\text{out}} = \mathcal{T}[k^{\text{out}}]$ 
   # Return top-scoring transform and full set of solutions for potential downstream planning/search
34: return  $\mathbf{T}^{\text{out}}, \mathcal{T}, \mathcal{P}, \mathcal{S}$ 
```

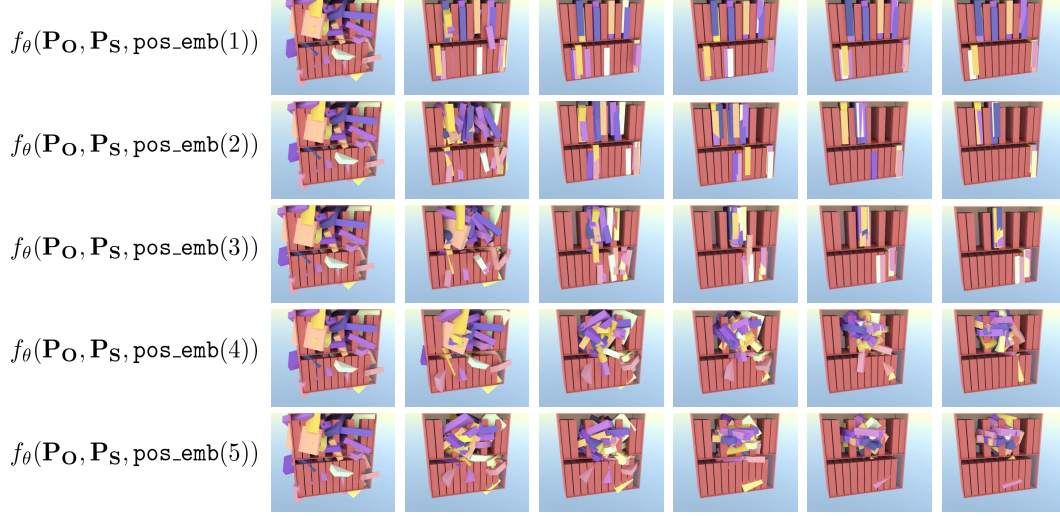


Figure A5: Examples of running our full iterative evaluation procedure for I steps with the model constrained to use a fixed value for t on each iteration. This highlights the different behavior the model has learned for different timesteps in the de-noising process. For timesteps near 1, the model has learned to make very local updates that “snap on” to whatever features are in the immediate vicinity of the object. As the timesteps get larger, the model considers a more global context and makes predictions that reach solutions that are farther away from the initial object pose. However, these end up more biased to a single solution in a region where there may be many nearby solutions (see the top row of shelves where there are four slots that the model finds when using timestep 1, but the model only reaches two of them with timestep $t = 2$ and one of them with $t = 3$). For even larger values of t , the model has learned a much more biased and “averaged out” solution that fails to rotate the object and only approximately reaches the scene regions corresponding to valid placements.

$$B = [A^T, A^{T-1}, \dots, A^2, A^1]$$

Exponentially decreasing values

$$C = \left\lceil \frac{A * I}{\sum_{i=1}^T A_i} \right\rceil$$

Normalize, scale up by I , and round up (minimum value per element is 1)

$$\bar{C} = \left\lceil \frac{C * I}{\sum_{i=1}^T C_i} \right\rceil$$

Normalize again so $\sum_{i=1}^T \bar{C}_i \approx I$ with $\bar{C}_i \in \mathbb{N} \forall i = 1, \dots, T$

$$\bar{C}_1 = \bar{C}_0 - \left(\sum_{i=1}^T \bar{C}_i - I \right)$$

Ensure $\sum_{i=1}^T \bar{C}_i = I$ exactly

727 Then, from \bar{C} , we construct multiple arrays with values ranging from 1 to T , each with lengths
728 corresponding to values in \bar{C} ,

$$\bar{D}_1 = [\bar{D}_{1,1} \ \bar{D}_{1,2} \ \dots] \text{ with } \bar{D}_{1,k} = 1 \ \forall k \in 1, \dots, \bar{C}_1$$

$$\bar{D}_2 = [\bar{D}_{2,1} \ \bar{D}_{2,2} \ \dots] \text{ with } \bar{D}_{2,k} = 2 \ \forall k \in 1, \dots, \bar{C}_2$$

...

$$\bar{D}_T = [\bar{D}_{T,1} \ \bar{D}_{T,2} \ \dots] \text{ with } \bar{D}_{T,k} = T \ \forall k \in 1, \dots, \bar{C}_T$$

729 and then stack these arrays together to obtain D as a complete array of length I :

730

$$D = [\bar{D}_1 \ \bar{D}_2 \ \dots \ \bar{D}_T]$$

731 A4 Success Classifier Details

732 In this section, we present details on training and applying the success classifier h_ϕ that we use for
733 ranking and filtering the set of multiple predicted SE(3) transforms produced by RPDiff.

734 **Training Data.** To train the success classifier, we use the demonstrations to generate positive and
735 negative examples, where the positives are labeled with success likelihood 1.0 and the negatives have
736 success likelihood 0.0. The positives are simply the unperturbed final point clouds and the negatives
737 are perturbations of the final object point clouds. We use the same sampling scheme of sampling a
738 rotation from a uniform distribution over $SO(3)$ and sampling a translation uniformly from within a
739 bounding box around the scene point cloud.

740 **Model Architecture.** We use an identical Transformer architecture as described in Section A2,
741 except that we use a single output MLP followed by a sigmoid to output the predicted success
742 likelihood, we do not condition on the timestep, and we provide the uncropped scene point cloud.

743 **Training Details.** We supervise the success classifier predictions with a binary cross entropy loss
744 between the predicted and ground truth success likelihood. We train for 500k iterations with batch size
745 64 on a V100 GPU which takes 5 days. We augment the data by rotating the combined object-scene
746 point cloud by random 3D rotations to increase dataset diversity.

747 A5 Experimental Setup

748 This section describes the details of our experimental setup in simulation and the real world.

749 A5.1 Simulated Experimental Setup

750 We use PyBullet [19] and the AIRobot [72] library to set up the tasks in the simulation and quantita-
751 tively evaluate our method along with the baselines. The environment consists of a table with the
752 shapes that make up the object and the scene, and the multiple simulated cameras that are used to
753 obtain the fused 3D point cloud. We obtain segmentation masks of the object and the scene using
754 PyBullet’s built-in segmentation abilities.

755 A5.2 Real World Experimental Setup

756 In the real world, we use a Franka Robot arm with a Robotiq 2F140 parallel jaw gripper attached
757 for executing the predicted rearrangements. We also use four Realsense D415 RGB-D cameras
758 with known extrinsic parameters. Two of these cameras are mounted to provide a clear, close-up
759 view of the object, and the other two are positioned to provide a view of the scene objects. We
760 use a combination of Mask-RCNN, density-based Euclidean clustering [73], and manual keypoint
761 annotation to segment the object, and use simple cropping heuristics to segment the overall scene
762 from the rest of the background/observation (e.g., remove the table and the robot from the observation
763 so we just see the bookshelf with the books on it).

764 A6 Evaluation Details

765 This section presents further details on the tasks we used in our experiments, the baseline methods
766 we compared RPDiff against, and the mechanisms we used to apply the predicted rearrangement to
767 the object in simulation and the real world.

768 A6.1 Tasks and Evaluation Criteria

769 **Task Descriptions.** We consider three relational rearrangement tasks for evaluation: (1) hanging a
770 mug on the hook of a rack, where there might be multiple racks on the table, and each rack might
771 have multiple hooks, (2) inserting a book into one of the multiple open slots on a randomly posed
772 bookshelf that is partially filled with existing books, and (3) placing a cylindrical can upright either
773 on an existing stack of cans or on a flat open region of a shelf where there are no cans there. Each
774 of these tasks features many placing solutions that achieve the desired relationship between the
775 object and the scene (e.g., multiple slots and multiple orientations can be used for placing, multiple
776 racks/hooks and multiple orientations about the hook can be used for hanging, multiple stacks and/or
777 multiple regions in the cabinet can be used for placing the can, which itself can be placed with either
778 flat side down and with any orientation about its cylindrical axis).

Evaluation Metrics and Success Criteria. To quantify performance, we report the average success rate over 100 trials, where we use the ground truth simulator state to compute success. For a trial to be successful, the object **O** and **S** must be in contact and the object must have the correct orientation relative to the scene (for instance, the books must be *on* the shelf, and must not be oriented with the long side facing into the shelf). For the can/cabinet task, we also ensure that the object **O** did not run into any existing stacks in the cabinet, to simulate the requirement of avoiding hitting the stacks and knocking them over.

We also quantify coverage via recall between the full set of predicted solutions and the precomputed set of solutions that are available for a given task instance. This is computed by finding the closest prediction to each of the precomputed solutions and checking whether the translation and rotation error between the prediction and the solution is within a threshold (we use 3.5cm for the translation and 5 degrees for the rotation). If the error is within this threshold, we count the solution as “detected”. We compute recall for a trial as the total number of “detected solutions” divided by the total number of solutions available and report overall recall as the average over the 100 trials. Precision is computed in an analogous fashion but instead checks whether each prediction is within the threshold for at least one of the ground truth available solutions.

A6.2 Baseline Implementation and Discussion

In this section, we elaborate on the implementation of each baseline approach in more detail and include further discussion on the observed behavior and failure modes of each approach.

A6.2.1 Coarse-to-Fine Q-attention (C2F-QA).

C2F-QA adapts the classification-based approach proposed in [8], originally designed for pick-and-place with a fixed robotic gripper, to the problem of relational object rearrangement. We voxelize the scene and use a local PointNet [74] that operates on the points in each voxel to compute per-voxel input features. We then pass this voxel feature grid through a set of 3D convolution layers to compute an output voxel feature grid. Finally, the per-voxel output features are each passed through a shared MLP which predicts per-voxel scores. These scores are normalized with a softmax across the grid to represent a distribution of “action values” representing the “quality” of moving the centroid of the object to the center of each respective voxel. This architecture is based on the convolutional point cloud encoder used in Convolutional Occupancy Networks [7].

To run in a coarse-to-fine fashion, we take the top-scoring voxel position (or the top- k voxels if making multiple predictions), translate the object point cloud to this position, and crop the scene point cloud to a box around the object centroid position. From this cropped scene and the translated object, we form a combined object-scene input point cloud and re-voxelize just this local portion of the point cloud at a higher resolution. We then compute a new set of voxel features with a separate high-resolution convolutional point cloud encoder. Finally, we pool the output voxel features from this step and predict a distribution over a discrete set of rotations to apply to the object. We found difficulty in using the discretized Euler angle method that was applied in [8], and instead directly classify in a binned version of $SO(3)$ by using an approximate uniform rotation discretization method that was used in [75].

We train the model to minimize the cross entropy loss for both the translation and the rotation (i.e., between the ground truth voxel coordinate containing the object centroid in the demonstrations and the ground truth discrete rotation bin). We use the same object point cloud perturbation scheme to create initial “noised” point clouds for the model to de-noise but have the model directly predict how to invert the perturbation transform in one step.

Output coverage evaluation. Since C2F-QA performs the best in terms of task success among all the baselines and is naturally suited for handling multi-modality by selecting more than just the argmax among the binned output solutions, we evaluate the ability of our method and C2F-QA to achieve high coverage among the available placing solutions while still achieving good precision (see Section 5.2). To obtain multiple output predictions from C2F-QA, we first select multiple voxel positions using the top- k voxel scores output by the PointNet \rightarrow 3D CNN \rightarrow MLP pipeline. We then copy the object point cloud and translate it to each of the selected voxel positions. For each selected position, we pool the local combined object-scene point cloud features and use the pooled features to predict a distribution of scores over the discrete space of rotations. Similar to selecting multiple

voxel positions, we select the top- k scoring rotations and use this full set of multiple translations + multiple rotations-per-translation as the set of output transforms to use for computing coverage.

Relationship to other “discretize-then-classify” methods. C2F-QA computes per-voxel features from the scene and uses these to output a normalized distribution of scores representing the quality of a “translation” action executed at each voxel coordinate. This idea of discretizing the scene and using each discrete location as a representation of a translational action has been successfully applied by a number of works in both 2D and 3D [44, 47, 76]. In most of these pipelines, the translations typically represent gripper positions, i.e., for grasping. In our case, the voxel coordinates represent a location to move the object for rearrangement.

However, techniques used by “discretize-then-classify” methods for rotation prediction somewhat diverge. C2F-QA and the recently proposed PerceiverActor [47] directly classify the best discrete rotation based on pooled network features. On the other hand, TransporterNets [44] and O2O-Afford [46] exhaustively evaluate the quality of different rotation actions by “convolving” some representation of the object being rearranged (e.g., a local image patch or a segmented object point cloud) in *all* possible object orientations, with respect to *each* position in the entire discretized scene (e.g., each pixel in the overall image or each point in the full scene point cloud). The benefit is the ability to help the model more explicitly consider the “interaction affordance” between the object and the scene at various locations and object orientations and potentially make a more accurate prediction of the quality of each candidate rearrangement action. However, the downside of this “exhaustive search” approach is the computational and memory requirements are much greater, hence these methods have remained limited to lower dimensions.

A6.2.2 Relational Neural Descriptor Fields (R-NDF).

R-NDF [20] uses a neural field shape representation trained on category-level 3D models of the objects used in the task. This consists of a PointNet encoder with $SO(3)$ -equivariant Vector Neuron [77] layers and an MLP decoder. The decoder takes as input a 3D query point and the output of the point cloud encoder, and predicts either the occupancy or signed distance of the 3D query point relative to the shape. After training, a point or a rigid set of points in the vicinity of the shape can be encoded by recording their feature activations of the MLP decoder. The corresponding point/point set relative to a new shape can then be found by locating the point/point set with the most similar decoder activations. These point sets can be used to parameterize the pose of local oriented coordinate frames, which can represent the pose of a secondary object or a gripper that must interact with the encoded object.

R-NDFs have been used to perform relational rearrangement tasks via the process of encoding task-relevant coordinate frames near the object parts that must align to achieve the desired rearrangement, and then localizing the corresponding parts on test-time objects so a relative transform that aligns them can be computed. We use the point clouds from the demonstrations to record a set of task-relevant coordinate frames that must be localized at test time to perform each of the tasks in our experiments. The main downside of R-NDF is if the neural field representation fails to faithfully represent the shape category, the downstream corresponding matching also tends to fail. Indeed, owing to the global point cloud encoding used by R-NDF, the reconstruction quality on our multi-rack/bookshelf/cabinet scenes is quite poor, so the subsequent correspondence matching does not perform well on any of the tasks we consider.

A6.2.3 Neural Shape Mating (NSM) + CVAE.

Neural Shape Mating (NSM) [3] uses a Transformer to process a pair of point clouds and predict how to align them. The method was originally deployed on the task of “mating” two parts of an object that has been broken but can be easily repurposed for the analogous task of relational rearrangement given a point cloud of a manipulated object and a point cloud of a scene/“parent object”. Architecturally, NSM is the same as our relative pose regression model, with the key differences of (i) being trained on arbitrarily large perturbations of the demonstration point clouds, (ii) not using local cropping, and (iii) only making a single prediction. We call this baseline “NSM-base” because we do not consider the auxiliary signed-distance prediction and learned discriminator proposed in the original approach [3]. As shown in Table 1, the standard version of NSM fails to perform well on any of the tasks that feature multi-modality in the solution space (nor can the model successfully fit the demonstration data). Therefore, we adapted it into a conditional variational autoencoder (CVAE) that at least has the capacity to learn from multi-modal data and output a distribution of transformations.

We use the same Transformer architecture for both the CVAE encoder and decoder with some small modifications to the inputs and outputs to accommodate (i) the encoder also encoding the ground truth de-noising transforms and predicting a latent variable z , and (ii) the decoder conditioning on z in addition to the combined object-scene point cloud to reconstruct the transform. We implement this with the same method that was used to incorporate the timestep information in our architecture – for the encoder, we include the ground truth transform as both an additional input token and via a residual connection with the global output feature, and for the decoder, we include the latent variable in the same fashion. We also experimented with concatenating the residually connected features and did not find any benefit. We experimented with different latent variable dimensions and weighting coefficients for the reconstruction and the KL divergence loss terms, since the CVAE models still struggled to fit the data well when the KL loss weight was too high relative to the reconstruction. However, despite this tuning to enable the CVAE to fit the training data well, we found it struggled to perform well at test time on unseen objects and scenes.

A6.3 Common failure modes

This section discusses some of the common failure modes for each method on our three tasks.

For **Book/Shelf**, our method occasionally outputs a solution that ignores an existing book already placed in the shelf. We also sometimes face slight imprecision in either the translation or rotation prevents the book from being able to be inserted. Similarly, the main failure modes on this task from the baselines are more severe imprecision. C2F-QA is very good at predicting voxel positions accurately (i.e., detecting voxels near open slots of the shelf) and the rotation predictions are regularly close to something that would work for book placement, but the predicted book orientations are regularly too misaligned with the shelf to allow the insertion to be completed.

For **Mug/Rack**, a scenario where our predictions sometimes fail is when there is a tight fit between the nearby peg and the handle of the mug. For C2F-QA, the predictions appear to regularly ignore the location of the handle when orienting the mug – the positions are typically reasonable (e.g., right next to one of the pegs on a rack) but the orientation oftentimes appears arbitrary. We also find C2F-QA achieves the highest training loss on this task (and hypothesize this occurs for the same reason).

Finally, for **Can/Cabinet**, a common failure mode across the board is predicting a can position that causes a collision between the can being placed and an existing stack of cans, which we don’t allow to simulate the requirement of avoiding knocking over an existing stack.

A6.4 Task Execution

This section describes additional details about the pipelines used for executing the inferred relations in simulation and the real world.

A6.4.1 Simulated Execution Pipeline

The evaluation pipeline mirrors the demonstration setup. Objects from the 3D model dataset for the respective categories are loaded into the scene with randomly sampled position and orientation. We sample a rotation matrix uniformly from $SO(3)$, load the object with this orientation, and constrain the object in the world frame to be fixed in this orientation. We do not allow it to fall on the table under gravity, as this would bias the distribution of orientations covered to be those that are stable on a horizontal surface, whereas we want to evaluate the ability of each method to generalize over all of $SO(3)$. In both cases, we randomly sample a position on/above the table that are in view for the simulated cameras.

After loading object and the scene, we obtain point clouds \mathbf{P}_O and \mathbf{P}_S and use RPDiff to obtain a rearrangement transform to execute. The predicted transformation is applied by resetting the object state to a “pre-placement” pose and directly actuating the object with a position controller to follow a straight-line path. Task success is then checked based on the criteria described in the section above.

Pre-placement Offset and Insertion Controller. Complications with automatic success evaluation can arise when directly resetting the object state based on the predicted transform. To avoid such complications, we simulate a process that mimics a closed-loop controller executing the last few inches of the predicted rearrangement from a “pre-placement” pose that is a pure translational offset from the final predicted placement. For our quantitative evaluations, we use the ground truth state of

the objects in the simulator together with prior knowledge about the task to determine the direction of this translational offset. For the mug/rack task, we determine the axis that goes through the handle and offset by a fixed distance in the direction of this axis (taking care to ensure it does not go in the opposite direction that would cause an approach from the wrong side of the rack). For the can/cabinet task and the book/bookshelf task, we use the known top-down yaw component of the shelf/cabinet world frame orientation to obtain a direction that offsets along the opening of the shelf/cabinet.

To execute the final insertion, we reset to the computed pre-placement pose and directly actuate the object with a position controller to follow a straight line path from the pre-placement pose to the final predicted placement. To simulate some amount of reactivity that such an insertion controller would likely possess in a full-stack rearrangement system, we use the simulator to query contact forces that are detected between the object and the scene. If the object pose is not close to the final predicted pose when contacts are detected, we back off and sample a small “delta” translation and body-frame rotation to apply to the object before attempting another straight line insertion. These small adjustments are attempted up to a maximum of 10 times before the execution is counted as a failure. If, upon detecting contact between the object and the scene, the object is within a threshold of its predicted place pose, the controller is stopped and the object is dropped and allowed to fall under gravity (which either allows it to settle stably in its final placement among the scene object, or causes it to fall away from the scene). We use this same procedure across all methods that we evaluated in our experiments.

We justify the use of this combination of a heuristically-computed pre-placement pose and “trial-and-error” insertion controller because (i) it removes the need for a full object-path planning component that searches for a feasible path the object should follow to the predicted placement pose (as this planning problem would be very challenging to solve to due all the nearby collisions between the object and the scene), (ii) it helps avoid other artificial execution failures that can arise when we perform the insertion from the pre-placement pose in a purely open-loop fashion, and (iii) it enables us to avoid complications that can arise from directly resetting the object state based on the predicted rearrangement transform.

A6.4.2 Real World Execution Pipeline

Here, we repeat the description of how we execute the inferred transformation using a robot arm with additional details. At test time, we are given point clouds \mathbf{P}_O and \mathbf{P}_S of object and scene, and we obtain \mathbf{T} , the SE(3) transform to apply to the object from RPDiff. \mathbf{T} is applied to \mathbf{O} by transforming an initial grasp pose $\mathbf{T}_{\text{grasp}}$, which is obtained using a separate grasp predictor [10], by \mathbf{T} to obtain a placing pose $\mathbf{T}_{\text{place}} = \mathbf{T}\mathbf{T}_{\text{grasp}}$. As in the simulation setup, we use a set of task-dependent heuristics to compute an additional “pre-placement” pose $\mathbf{T}_{\text{pre-place}}$, from which we follow a straight-line end-effector path to reach $\mathbf{T}_{\text{place}}$. We then use off-the-shelf inverse kinematics and motion planning to move the end-effector to $\mathbf{T}_{\text{grasp}}$ and $\mathbf{T}_{\text{place}}$.

To ease the burden of collision-free planning with a grasped object whose 3D geometry is unknown, we also compute an additional set of pre-grasp and post-grasp waypoints which are likely to avoid causing collisions between the gripper and the object during the execution to the grasp pose, and collisions between the object and the table or the rest of the scene when moving the object to the pre-placement pose. Each phase of the overall path is executed by following the joint trajectory in position control mode and opening/closing the fingers at the correct respective steps. The whole pipeline can be run multiple times in case the planner returns infeasibility, as the inference methods for both grasp and placement generation have the capacity to produce multiple solutions.

A7 Extra Ablations

In this section, we perform additional experiments wherein different system components are modified and/or ablated.

With vs. Without Success Classifier. We use neural network h_ϕ to act as a success classifier and support selecting a “best” output among the K predictions made by our iterative de-noising procedure. Another simple mechanism for selecting an output index k_{exec} for execution would be to uniformly sample among the K outputs. However, due to the local nature of the predictions at small values of t and the random guess initializations used to begin the inference procedure, some final solutions end

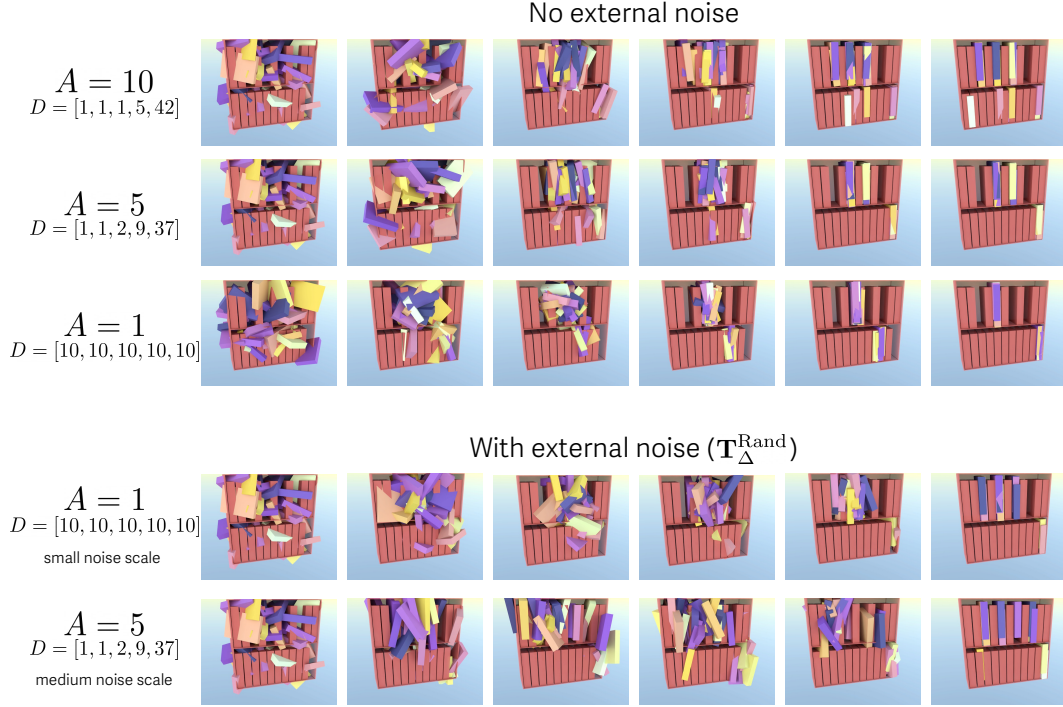


Figure A6: Examples of running our full iterative evaluation procedure for I steps with different values of A (and subsequently, D) in our `i_tot` function (which maps from test-time iteration values $n = 1, \dots, I$ to the timestep values $t = 1, \dots, T$ that were used in training), and with different amounts of external noise $\mathbf{T}_{\Delta}^{\text{Rand}}$ added from the annealed external noise distribution $p_{\text{AnnealedRandSE}(3)}(\cdot)$. We observe that with large values of A , the model makes more predictions with smaller values of t . These predictions are more local and the overall solutions converge to a more broad set of rearrangement transforms. This sometimes leads to “locally optimal” solutions that fail at the desired task (see top right corner with $A = 10$). With small A , the early iterations are more biased toward the average of a general region, so the set of transforms tends to collapse on a single solution within a region. By incorporating external noise, a better balance of coverage for smaller values of A and “local optima” avoidance for larger values of A can be obtained.

in configurations that don’t satisfy the task (see the book poses that converge to a region where there is no available slot for placement in Figure A5 for $A = 10$).

Therefore, a secondary benefit of incorporating h_{ϕ} is to filter out predictions that may have converged to these “locally optimal” solutions, as these resemble some of the negatives that the classifier has seen during training. Indeed, we find the average success rate across tasks with RPDiff when using the success classifier is 0.88, while the average success when uniformly sampling the output predictions is 0.83. This difference is relatively marginal, indicating that the majority of the predictions made by the pose de-noising procedure in RPDiff are precise enough to achieve the task. However, the performance gap indicates that there is an additional benefit of using a final success classifier to rank and filter the outputs based on predicted success.

Noise vs. No Noise. In each update of the iterative evaluation procedure, we update the overall predicted pose and the object point cloud by a combination of a transform predicted by f_{θ} and a randomly sampled “external noise” transform $\mathbf{T}_{\Delta}^{\text{Rand}}$. The distribution that $\mathbf{T}_{\Delta}^{\text{Rand}}$ is sampled from is parameterized by the iteration number i to converge toward producing an identity transform so the final pose updates are purely a function of the network f_{θ} .

The benefit of incorporating the external noise is to better balance between precision and coverage. First, external noise helps the pose/point cloud at each iteration “bounce out” of any locally optimal regions and end up near regions where a high quality solution exists. Furthermore, if there are many high-quality solutions close together, the external noise on later iterations helps maintain some variation in the pose so that more overall diversity is obtained in the final set of transform solutions.

For instance, see the qualitative comparisons in Figure A6 that include iterative predictions both with and without external noise. For a value of $A = 1$ in `i_to_t`, only two of the available shelf slots are found when no noise is included. With noise, however, the method finds placements that cover four of the available slots. Quantitatively, we also find that incorporating external noise helps in terms of overall success rate and coverage achieved across tasks. The average (Success Rate, Recall) across our three tasks with and without noise was found to be (0.88, 0.44) and (0.83, 0.36), respectively.

Number of diffusion steps T during training. The total number of steps T and the noise distribution for obtaining perturbation a transform $\mathbf{T}_{\text{noise}}^{(t)}$ affects the magnitude of the translation and rotation predictions that must be made by the model f_θ . While we did not exhaustively search over these hyperparameters, early in our experiments we found that very small values of T (e.g., $T = 2$) cause the predictions to be much more imprecise. This is due to the averaging that occurs between training samples when they are too far away from the ground truth. In this regime, the examples almost always “look multi-modal” to the model. On the other hand, for large values of T (e.g., $T = 50$), the incremental transforms that are used to de-noise become very small and close to the identity transform. When deployed, models trained on this data end up failing to move the object from its initial configuration because the network has only learned to make extremely small pose updates.

We found a moderate value of $T = 5$ works well across each of our tasks, though other similar values in this range can likely also provide good performance. This approximately leads the average output scale of the model to be near 2.5cm translation and 18-degree rotation. We also observe a benefit in biasing sampling for the timesteps $t = 1, \dots, T$ to focus on smaller values. This causes the model to see more examples close to the ground truth and make more precise predictions on later iterations during deployment. We achieve this biased sampling by sampling t from an exponentially decaying categorical probability distribution over discrete values $1, 2, \dots, T$.

Incremental targets vs. full targets. As discussed in Section 3.1, encouraging the network f_θ to predict values with roughly equal magnitude is beneficial. To confirm this observation from the literature, we quantitatively evaluate a version of the de-noising model f_θ trained to predict the full de-noising transform $[\mathbf{T}_{\text{noise}}^{(t)}]^{-1}$. The quantitative (Success Rate, Recall) results averaged across our three tasks with the incremental de-noising targets are (0.88, 0.44), while the model trained on full de-noising targets are (0.76, 0.34). These results indicate a net benefit in using the incremental transforms as de-noising prediction targets during training.

Value of A in `i_to_t`. In this section, we discuss the effect of the value A in the `i_to_t` function used during the iterative evaluation procedure. The function `i_to_t` maps evaluation iteration values i to timestep values t that were seen during training. For instance, we may run the evaluation procedure for 50 iterations, while the model may have only been trained to take values up to $t = 5$ as input. Our `i_to_t` function is parameterized by A such that larger values of A lead to more evaluation iterations with small values of t . As A approaches 1, the number of iterations for each value of t becomes equal (i.e., for $A = 1$, the number of predictions made for each value of t is equal to I/T).

Figure A6 shows qualitative visualizations of de-noising the pose of a book relative to a shelf with multiple available slots with different values of A in the `i_to_t` function. This example shows that the solutions are more biased to converge toward a single solution for smaller values of A . This is because more of the predictions use larger values of t , which correspond to perturbed point clouds that are farther from the ground truth in training. For these perturbed point clouds, their association with the correct target pose compared to other nearby placement regions is more ambiguous. Thus, for large t , the model learns an averaged-out solution that is biased toward a region near the average of multiple placement regions that may be close together. On the other hand, for large A , more predictions correspond to small values of t like $t = 1$ and $t = 0$. For these timesteps, the model has learned to precisely snap onto whatever solutions may exist nearby. Hence, the pose updates are more local and the overall coverage across the K parallel runs is higher. The tradeoff is that these predictions are more likely to remain stuck near a “locally optimal” region where a valid placement pose may not exist. Table 3 shows the quantitative performance variation on the **Book/Shelf** task for different values of A in the `i_to_t` function. These results reflect the trend toward higher coverage and marginally lower success rate for larger values of A .

Metric	Value of A in i_t_o_t				
	1	2	5	10	20
Success Rate	1.00	0.95	0.96	0.94	0.90
Recall (coverage)	0.37	0.41	0.48	0.48	0.52

Table 3: Performance for different values of A in i_t_o_t. Larger values of A obtain marginally better precision at the expense of worse coverage (lower recall).

A8 Further Discussion on Real-world System Engineering and Limitations

This section provides more details on executing rearrangement via pick-and-place on the real robot (to obtain the results shown in Figures 1 and 4) and discusses additional limitations of our approach.

A8.1 Executing multiple predicted transforms in sequence in real-world experiments

The output of the pose diffusion process in RPDiff is a set of K $SE(3)$ transforms $\{\mathbf{T}_k^{(0)}\}_{k=1}^K$. To select one for execution, we typically score the outputs with success classifier h_ϕ and search through the solutions while considering other feasibility constraints such as collision avoidance and robot workspace limits. However, to showcase executing a diverse set of solutions in our real-world experiments, a human operator performs a final step of visually inspecting the set of feasible solutions and deciding which one to execute. This was mainly performed to ease the burden of recording robot executions that span the space of different solutions (i.e., to avoid the robot executing multiple similar solutions, which would fail to showcase the diversity of the solutions produced by our method).

A8.2 Expanded set of limitations and limiting assumptions

Section 7 mentions some of the key limitations of our approach. Here, we elaborate on these and discuss additional limitations, as well as potential avenues for resolving them in future work.

- We train from scratch on demonstrations and do not leverage any pre-training or feature-sharing across multiple tasks. This means we require many demonstrations for training. A consequence of this is that we cannot easily provide enough demonstrations to train the diffusion model in the real world (while still enabling it to generalize to unseen shapes, poses, and scene layouts). Furthermore, because we train only in simulation and directly transfer to the real world, the domain gap causes some challenges in sim2real transfer, so we do observe worse overall prediction performance in the real world. This could be mitigated if the number of demonstrations required was lower and we could train the model directly on point clouds that appear similar to those seen during deployment.
- In both simulation and the real world, we manually completed offset poses for moving the object before executing the final placement. A more ideal prediction pipeline would involve generating “waypoint poses” along the path to the desired placement (or even the full collision-free path, e.g., as in [78]) to support the full insertion trajectory rather than just specifying the final pose.
- Our method operates using a purely geometric representation of the object and scene. As such, there is no notion of physical/contact interaction between the object and the scene. If physical interactions were considered in addition to purely geometric/kinematic interactions/alignment, the method may be even more capable of accurate final placement prediction and avoid some of the small errors that sometimes occur. For instance, a common error in hanging a mug on a rack is to have the handle *just* miss the hook on the rack. While these failed solutions are geometrically very close to being correct, physically, they are completely different (i.e., in one, contact occurs between the two shapes, while in the other, there is no contact that can support the mug hanging).
- Our method operates using 3D point clouds which are currently obtained from depth cameras. While this permits us to perform rearrangements with a wide variety of real-world objects/scenes that can be sensed by depth cameras, there are many objects which cannot be observed by depth cameras (e.g., thin, shiny, transparent objects). Investigating a way to perform similar relational object-scene reasoning in 6D using signals extracted from RGB sensors would be an exciting avenue to investigate.

1102 A9 Model Architecture Diagrams

	Parameter	Value
	Number of \mathbf{P}_O and \mathbf{P}_S points	1024
	Batch size	16
	Transformer encoder blocks	4
	Transformer decoder blocks	4
	Attention heads	1
	Timestep position embedding	Sinusoidal
	Transformer embedding dimension	256
	Training iterations	500k
1103	Optimizer	AdamW
	Learning rate	1e-4
	Minimum learning rate	1e-6
	Learning rate schedule	linear warmup, cosine decay
	Warmup epochs	50
	Optimizer momentum	$\beta_1 = 0.9, \beta_2 = 0.95$
	Weight decay	0.1
	Maximum training timestep T	5
	Maximum \mathbf{P}_S crop size L_{\max}	\mathbf{P}_S bounding box maximum extent
	Minimum \mathbf{P}_S crop size L_{\min}	18cm

1104 Table 4: Training hyperparameters

	Parameter	Value
	Number of evaluation iterations I	50
	Number of parallel runs K	32
	Default value of A in <code>i_to_t</code>	10
1105	Expression for $p_{\text{AnnealedRandSE}(3)}(\cdot \mid i)$	$\mathcal{N}(\cdot \mid 0, \sigma(i))$
	$\sigma(i)$ in $p_{\text{AnnealedRandSE}(3)}$ (for trans and rot)	$a * \exp(-bi/I)$
	Value of a (axis-angle rotation, in degrees)	20
	Value of b (axis-angle rotation)	6
	Value of a (translation, in cm)	3
	Value of b (translation)	6

1106 Table 5: Evaluation hyperparameters

Downsample point clouds	$(N + M) \times 3$
One-hot concat	$(N + M) \times 5$
Linear	$(N + M) \times d$
Concat pos_emb(t)	$(N + M + 1) \times d$
$\left[\begin{array}{c} \text{Self-attention (scene)} \end{array} \right]_{\times 4}$	$M \times d$
$\left[\begin{array}{c} \text{Self-attention (object)} \\ \text{Cross-attention (object, scene)} \end{array} \right]_{\times 4}$	$(N + 1) \times d$
Global Pooling	d
Residual pos_emb(t)	d
MLP (translation)	$d \rightarrow 3$
MLP \rightarrow orthonormalize (rotation)	$d \rightarrow 6 \rightarrow 3 \times 3$

Table 6: Transformer architecture for predicting SE(3) transforms

Downsample point clouds	$(N + M) \times 3$
One-hot concat	$(N + M) \times 5$
Linear	$(N + M) \times d$
$\left[\begin{array}{c} \text{Self-attention (scene)} \end{array} \right]_{\times 4}$	$M \times d$
$\left[\begin{array}{c} \text{Self-attention (object)} \\ \text{Cross-attention (object, scene)} \end{array} \right]_{\times 4}$	$N \times d$
Global Pooling	d
MLP \rightarrow sigmoid (success)	$d \rightarrow 1$

Table 7: Transformer architecture for predicting success likelihood