

Appendix

Note to Reviewers: Minor Bug in Figure 3

In the original submission, there was a minor bug in the plotting code that incorrectly categorized the objects in Fig. 3. **We emphasize that the quantitative results both in Fig. 4 of the main text and in the full results table of this Appendix are correct and consistent with each other.**

The only changes after fixing the plotting code are as follows:

- the redbox and lunchbox objects are in the medium difficulty and not easy, and
- the minion and dino objects are in the hard difficulty and not medium.

To see the rationale behind the object categorization, see Appendix E.2. The old figure and new figure are shown here for clarity.

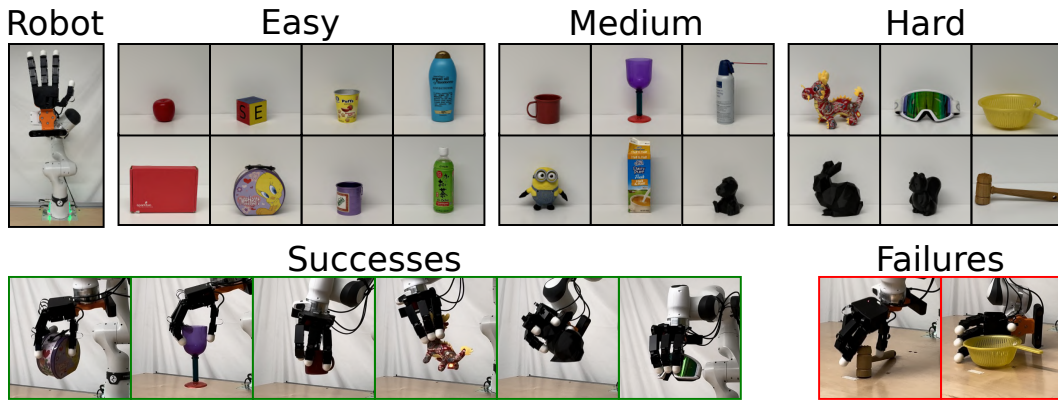


Figure 5: Original figure.

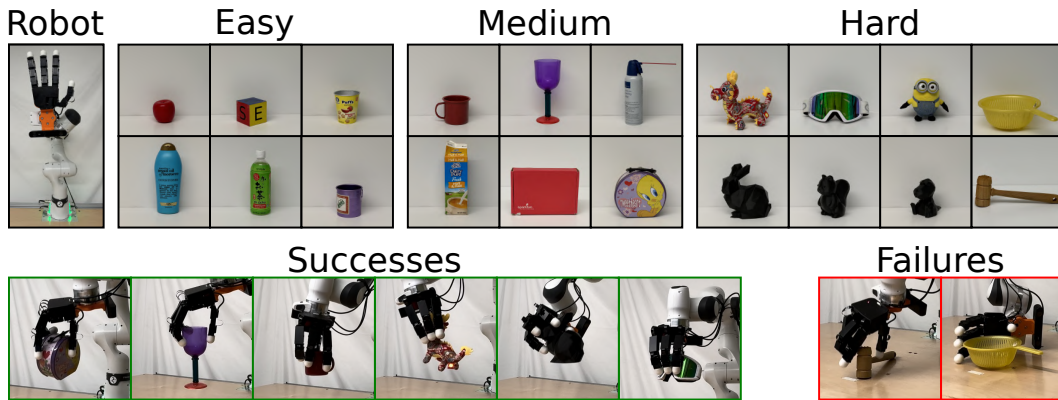


Figure 6: Fixed figure.

11 A Additional Insights and Discussion of Results

12 In this section, we provide further discussion that may aid the interpretation of our results.

13 **How “fair” is the diffusion sampler?** To ensure reasonable performance, we train our generative
 14 model exclusively on high-quality grasps with a probability of grasp success $y_{\text{PGS}} = 1.0$. This is
 15 approximately 300K grasps (8.45% of the training data), roughly 10 times the amount used to train
 16 existing models [1, 2]. Figure 23 shows real-world grasps generated by the diffusion sampler, and
 17 shows that many failed grasps are reasonably “close” to a successful grasp.

18 In general, grasp evaluators have superior data efficiency in the sense that they can train on “bad”
 19 grasps (in our case, there are $\sim 3\text{M}$) which are byproducts of generating “good” ones. We stress
 20 that we do not claim that grasp evaluation is superior to grasp generation as a paradigm - only that
 21 evaluators can achieve robust sim-to-real transfer by leveraging a large volume of negative examples.

22 We emphasize that there are no suitable datasets for dexterous grasping larger than ours on which we
 23 can train our diffusion sampler. To our knowledge, the only large dataset with visual observations
 24 is closed-source [3]. Further, no existing datasets parameterize the grasp execution motion, few
 25 consider a tabletop setting, and many of their grasps intersect with objects in a non-physical way.

26 **Are the simulation results “reasonable”?** Our simulation success rates at first appear much lower
 27 than our hardware success rates and simulation results of similar studies [1, 2]. This discrepancy
 28 arises from subtleties in labeling. In the real world, we only consider collision-free grasps, and
 29 thus report the empirically estimated conditional probability $p(\text{success} \mid \text{no collisions})$. However,
 30 in simulation we evaluate *all* planned grasps, including those in collision (which count as failures),
 31 and instead report $p(\text{success})$. In other words, the real-world experiments answer the question “how
 32 many *collision-free* generated grasps succeed?”, while the simulations answer the question “how
 33 many generated grasps succeed?”

34 Figure 7 shows histograms for all simulated evaluator-based methods, which compares unconditional
 35 success rates and those conditioned on $y_{\text{coll}} \geq 0.8$. The median success rates are 37% and 80%
 36 respectively. Note the similarity between this 80% mark and our evaluator-based hardware results,
 37 which achieve 76-81% across all objects (see Table 4). Lastly, other works differ from ours in three
 38 major ways: they often do not consider tabletop settings (allowing non-physical fully-caging grasps),
 39 they have less object diversity, and they plan power grasps while we plan precision grasps. These
 40 factors make our learning problem more challenging than those in other studies.

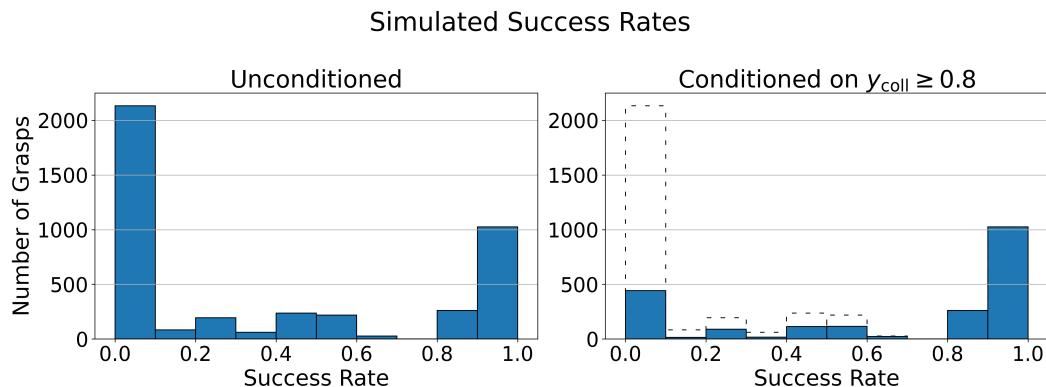


Figure 7: Success rates for all simulated evaluator-based methods. When only considering grasps with $y_{\text{coll}} \geq 0.8$, the median success rate increases from 37% to 80%. 2100/4240 grasps satisfied $y_{\text{coll}} \geq 0.8$. On the right, the unconditioned distribution is plotted with dashed lines for comparison.

41 B Grasp Dataset

42 B.1 Grasp Generation

Parameter	Value	Parameter	Value	Parameter	Value
n_c_per_finger	5	switch_prob	0.5	jitter	0.1
w_fc	0.5	mu	0.98	dist_lower	0.2
w_dis	500	step_size	0.005	dist_upper	0.3
w_pen	300.0	stepsize_period	50	theta_lower	$-\pi / 6$
w_spen	100.0	starting_temp	18	theta_upper	$\pi / 6$
w_joints	1.0	annealing_period	30		
w_ff	3.0	tempdecay	0.95		
w_fp	0.0				
w_tpen	100.0				

Table 3: Grasp generation parameters. Variables names taken from DexGraspNet [4]. Left: Energy function parameters. Middle: Optimization parameters. Right: Initialization parameters.

43 Our grasp generation pipeline is inspired by Wang et al. [4], but required key modifications. Further
44 details about these modifications not discussed in the main text are described here.

- 45 • As mentioned, our data generation pipeline yields *pre-grasp* poses with the fingertips 1.5cm
46 off of the surface of the object, while DexGraspNet places the fingers on or very near the
47 surface. Due to this, when planning a grasp trajectory, we compute a new configuration
48 corresponding to fingertip locations 3.5cm below the object surface, which we call the
49 *post-grasp* pose.
- 50 • We choose to generate *precision grasps* rather than power grasps, since this synergizes with
51 the grasp motion parameterization described in the main text. To implement this, we only
52 specify *contact candidates* on the fingertip of the hand, which function as attractors to the
53 object surface during grasp generation. However, the *surface points* used to penalize hand-
54 object or hand-table collision still cover the whole hand. See Figure 8 for a visualization.

55 We now describe the dataset augmentation procedure in detail. Recall that first, a large set of
56 *nominal* grasps are generated using the modified DexGraspNet pipeline. In this step, the fingertip
57 grasp directions \mathbf{d}_i are generated by computing the direction of each fingertip to the closest point on
58 the mesh. Because there are a large proportion of failures after this step, we augment the dataset
59 with more positive examples by taking all nominal grasps satisfying $y_{PGS} \geq 0.9$, perturbing them
60 slightly, re-evaluating them, and adding them to the dataset.

61 These perturbations are sampled using Halton sequence sampling [5], which generates quasi-random,
62 *low-discrepancy* sequences that sample more uniformly across the input space compared to standard
63 random sampling and reducing clustering. We sample perturbations from the following ranges:

- 64 • wrist translation: each spatial coordinate draws a perturbation from $[-5\text{mm}, 5\text{mm}]$;
- 65 • wrist orientation: each of roll, pitch, and yaw draw a perturbation from $[-2.5^\circ, 2.5^\circ]$;
- 66 • finger joints: each angle draws a perturbation from $[-0.05\text{rad}, 0.05\text{rad}]$;
- 67 • fingertip grasp directions: each of two axes orthogonal to the nominal direction draw
68 perturbations from $[-10^\circ, 10^\circ]$.

69 Each high-success grasp is perturbed five times and re-evaluated on the same object. Additionally,
70 these grasps are perturbed two more times and evaluated on *different, randomly sampled* objects from
71 the dataset. These additional perturbations typically yield unsuccessful grasps, but we believe this
72 protects the model from overfitting and provides useful training signal regarding object geometry.

73 The dataset took about 1 day to generate using 4 Nvidia A100 GPUs. Table 3 summarizes all
74 parameters used for grasp generation.

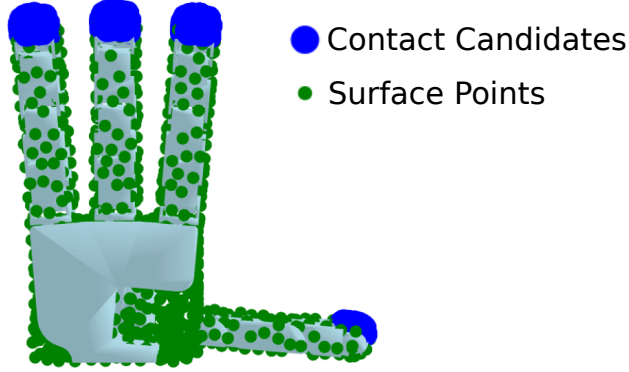


Figure 8: Visualization of the Allegro hand contact candidates and 500 surface points. Contact candidates are used to compute E_{dis} , encouraging hand-object proximity. Surface points are used to compute E_{pen} , which penalizes hand-table penetration to keep the grasps above the table.

75 B.2 Grasp Label Generation in Simulation

76 To generate the labels for the grasps in the dataset as well as corresponding rendered visual data,
 77 we use Isaac Gym [6]. Our simulated evaluation proceeds as follows. We spawn the hand in the
 78 pre-grasp pose, execute the grasp motion parameterized by fingertip directions, then lift the object
 79 20cm off of the table (with gravity enabled).

80 A pick is considered a *simulation success* ($y_{\text{pick}} = 1$) only if the following conditions are met: (1)
 81 the hand contacts the object on at least 3 links; (2) the palm and proximal links of the hand do
 82 not touch the object; (3) the change in the object’s position relative to the hand does not exceed
 83 10cm throughout the lift; (4) the change in the object’s orientation relative to the hand does not
 84 exceed 45 degrees about each Euler angle throughout the lift. A pick is considered *collision-free*
 85 ($y_{\text{coll}} = 1$) only if the hand does not contact the table or object during the pre-grasp pose. Recall that
 86 $y_{\text{PGS}} = y_{\text{pick}} \wedge y_{\text{coll}}$.

87 To generate non-binary labels, each grasp is corrupted with small noise and re-simulated 5 times.
 88 Note that this is a *different perturbation* than the one used for dataset augmentation described in
 89 Appendix B.1. Only the wrist pose is perturbed. The wrist translations are corrupted with uniform
 90 noise drawn from $[-5\text{mm}, 5\text{mm}]$ along each axis, and the wrist orientation is corrupted about the
 91 roll, pitch, and yaw axes with uniform noise drawn from $[-2.5^\circ, 2.5^\circ]$.

92 The simulation timestep is set to 1/60 seconds, but the integrator solves at a rate of 1/120 seconds for
 93 numerical stability using the Truncated Gauss-Seidel (TGS) solver. The TGS solver runs 8 position
 94 and 8 velocity iterations per simulation timestep. “Force at a distance” (i.e., the *contact offset*
 95 parameter) is applied starting from 1mm of separation between collision geometries, substantially
 96 lower than the 1cm distance used by [4] in DexGraspNet. This helps reduce unwanted non-physical
 97 collisions. Geometries are processed by a convex decomposition using the default settings in Isaac
 98 Gym. Each object is spawned 2cm above the table and dropped. The simulation starts when the
 99 object settles. The simulation pipeline is visualized in Figure 9.

100 The entire dataset can be evaluated in about one day using 4 Nvidia A100 GPUs. We note that this
 101 can only occur after the dataset has been generated using the procedure outline in Appendix B.1.

102 B.3 Simulated Label Distribution

103 Figure 10 shows the distribution of grasp labels across the dataset of 3.5M grasps. The median and
 104 IQR of each label is as follows:

- 105 • y_{PGS} : 0.0 (0.0, 0.48)
- 106 • y_{pick} : 0.2 (0.0, 0.8)

Simulated Grasp Evaluation

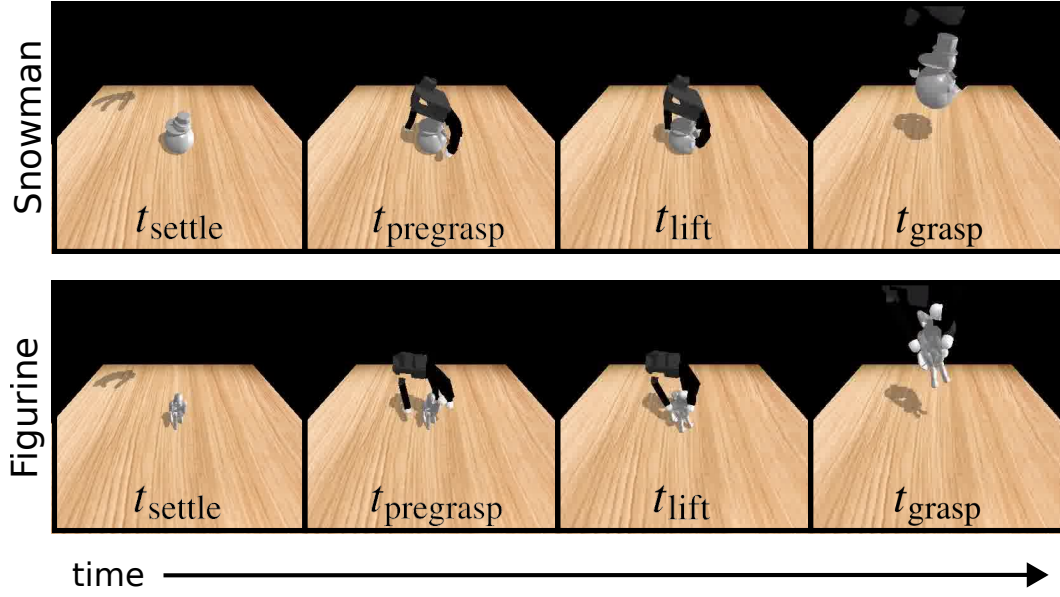


Figure 9: Visualization of grasp evaluation in simulation on a snowman object (top) and a figurine object (bottom). After the object settles on the table (t_{settle}), the Allegro hand is moved to the pre-grasp position (t_{pregrasp}), executes the grasp (t_{grasp}), and lifts the object (t_{lift}). At t_{pregrasp} , we compute y_{coll} by checking for hand collisions with the object and table. At t_{lift} , we compute y_{pick} by checking if the object pose relative to the hand significantly changed from t_{pregrasp} .

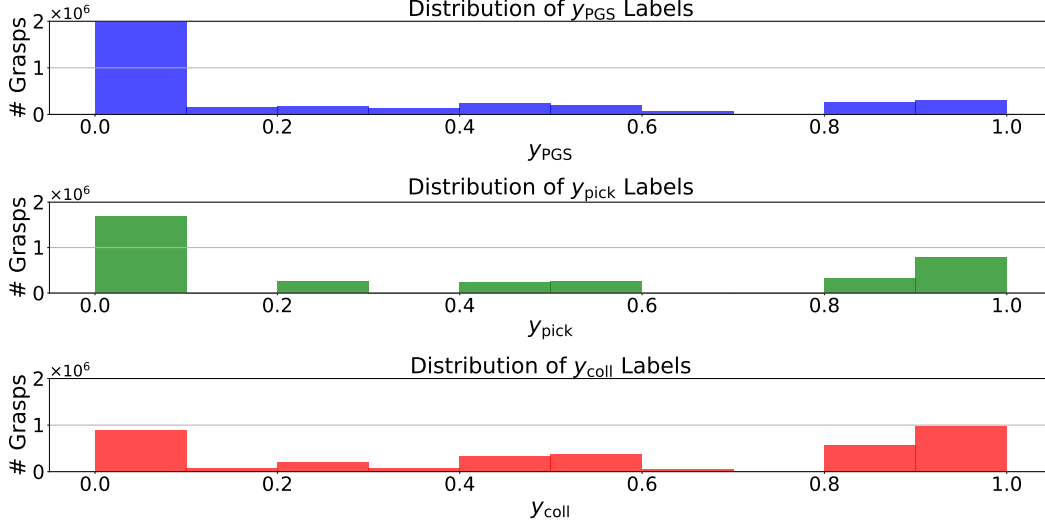


Figure 10: Dataset label distribution across 3.5M grasps.

107 • y_{coll} : 0.08 (0.6, 1.0)

108 B.4 Object Dataset and Generation

109 The objects considered in this work are a strict subset of those in DexGraspNet [4]. DexGraspNet
 110 contains 5.3K unique objects, while we only consider 4.3K of them. The main reason for this is
 111 that we assume a tabletop setting while DexGraspNet does not, which necessitates a canonical “up”

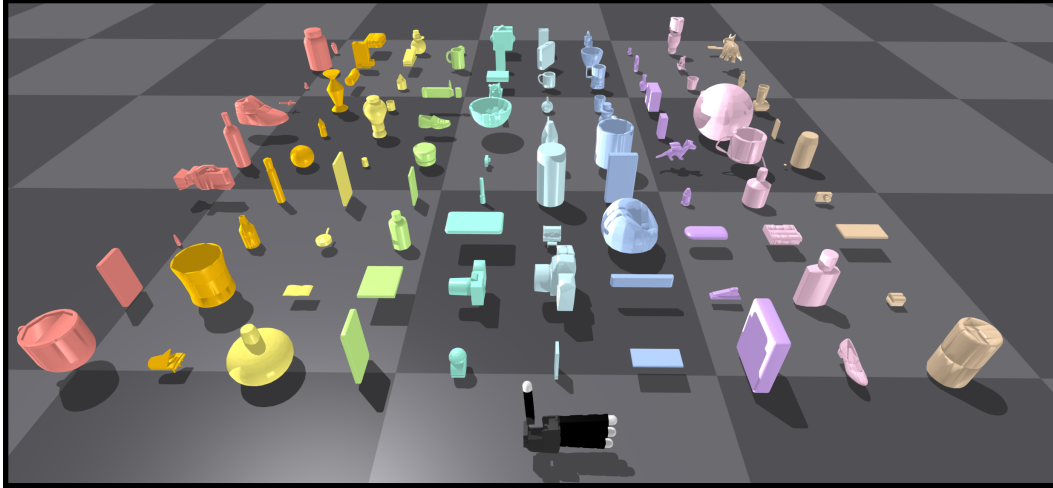


Figure 11: Visualization of a random subset of 100 objects out of the 212 test objects, with the Allegro hand shown at the bottom for scale. The object dataset is a subset of the DexGraspNet dataset’s meshes [4]. See their work for more details.

direction. Thus, many objects were manually processed to be oriented in a reasonable manner. We only use an object if after this reorientation, the object successfully settles when dropped from a height of 2cm. If the object tips over or does not settle after 200 timesteps in the Isaac Gym simulation, then it is excluded. We consider an object to have remained upright if the real part of its relative quaternion remains greater than 0.95 throughout the simulation. We consider it to have settled if the translational components stay within 1mm along each axis and roll, pitch, and yaw stay within $1e-2$ radians for 15 consecutive timesteps. Figure 11 visualizes a subset of our object dataset. To introduce more scale diversity while retaining a dataset size under 2.5TB, a subset of approximately 1.7K of the objects is duplicated at three distinct scales, while the remaining objects are used at one scale each, resulting in approximately 7.7K objects. The large dataset storage requirement comes from the storage of multiple 3D NeRF density grids used to represent each grasp for the NeRF evaluator (more details about the NeRF density grid representation in Appendix D.3.2).

B.5 Train, Validation, and Test Split

We create our train, validation, and test splits of the datasets very carefully. We make sure that each unique object mesh only exists in one of these three sets.

We split the 4305 unique object meshes into 3981 (92.5%) train objects, 216 (5%) validation objects, and 108 (2.5%) test objects. Each object may appear at more than one scale, so this results in 7695 different objects after all scaling operations, split into 7108 (92.5%) train objects, 375 (5%) validation objects, and 212 (2.5%) test objects. This results in a total of 3,531,098 grasps, with 3,261,228 (92.5%) for training, 170,128 (5%) for validation, and 99,742 (2.5%) for testing.

C Object Representations

C.1 Neural Radiance Fields

We train NeRFs using both simulated data and real-world data and compare them here.

For NeRFs trained on simulated data, the object is first spawned and allowed to settle on the table. Then, we uniformly sample 100 images over a spherical cap above the object with a radius of 0.45m and a polar angle sampled from $U(0, \frac{\pi}{4})$, and then train for 400 iterations with `nerfstudio` [7, 8].



Figure 12: Comparison between NeRFs trained on simulated snowman data and real-world dragon data, which demonstrates the qualitative similarity between the two NeRFs. First row: two of the RGB images used for NeRF training. Second row: Camera poses used for NeRF training. Third row: NeRF-rendered RGB image and accumulation image.

For NeRFs trained on real-world data, we first collect 100 images while moving the wrist-mounted camera along a hard-coded trajectory encircling the object and then train for 400 iterations with `nerfstudio` [7, 8]. This trajectory consists of 3 spirals around the object, with the object placed roughly in the center of the spirals.

Figure 12 shows a qualitative comparison between the RGB images and NeRFs trained on simulated data versus real-world data. Both types of NeRFs exhibit floater artifacts [9], which must either be processed away or ignored by downstream models.

We train without scene contraction, auto-scaling poses, centering method, or orientation method, and use a scale factor of 1.0, which ensures that the coordinate space in the NeRF is not modified from the given data. The remainder of the parameters follow the default `nerfacto` settings in `nerfstudio`.

C.2 Basis Point Sets

Methods that represent the object as basis point sets (BPSs) use the following procedure.

1. Train a NeRF following the procedure described in Appendix C.1.
2. Sample a point cloud with 5000 points using `nerfstudio` [8] by rendering out depth images with opacity exceeding 0.5 in the axis-aligned bounding box parameterized by the lower and upper bounds $[-0.2m, -0.2m, 0.0m] \times [0.2m, 0.2m, 0.3m]$ using a z-up convention.

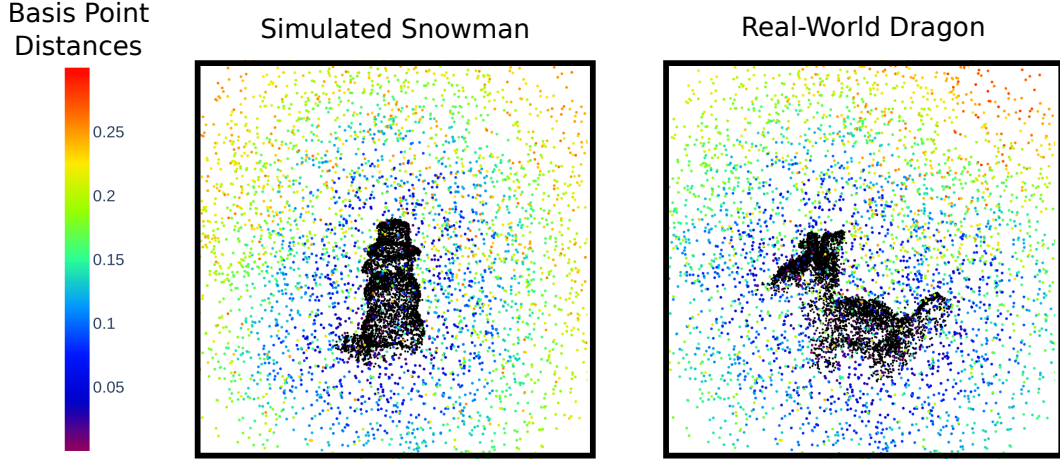


Figure 13: Visualization of a point cloud (5000 black points) and a basis point set (4096 points colored by distance to the point cloud) generated by a NeRF for both a simulated snowman object (left) and a real-world dragon object (right).

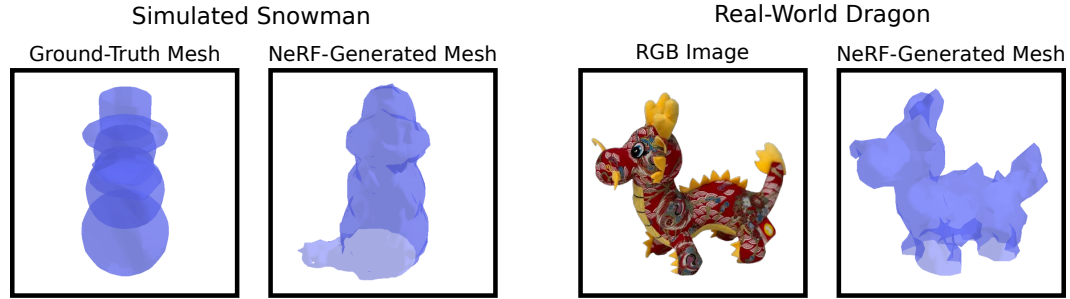


Figure 14: We use NeRFs to generate meshes for two purposes: analytic grasp planning methods (FRoGGeR) and collision-free motion planning from the start pose to the pre-grasp pose (all methods). Left: Comparison between the ground-truth mesh and NeRF-generated mesh of a simulated snowman object, where the extra vertices at the bottom are due to artifacts from shadows/lighting effects. Right: Comparison between an RGB image and the NeRF-generated mesh of a real-world dragon object. We do not have a ground-truth mesh for the real-world dragon object.

- 156 3. Preprocess the point cloud to remove most outliers and floaters. First, we use open3d [10]
 157 to remove statistical outliers (`nb_neighbors=20`, `std_ratio=2.0`) and radius out-
 158 liers (`nb_points=16`, `radius=0.05`). Next, we construct an undirected graph of the
 159 remaining points, where points are nodes and an edge exists between two nodes if the points
 160 are within 1mm. Floaters are removed by keeping the largest connected component.
- 161 4. Generate a random set of 4096 basis points within a sphere of radius 0.3m, centered 0.15m
 162 above the table. This set of basis points remains fixed for all experiments. We utilize
 163 bps [11] for basis point set operations.
- 164 5. Compute basis point set values by calculating the distance from each basis point to its
 165 closest point in the point cloud.

166 Figure 13 shows an example of a point cloud and BPS for both a simulated and real-world object.

167 C.3 Meshes

168 We use triangle meshes for two purposes: the analytic grasp planning method (FRoGGeR) that we
 169 use as a baseline and collision-free motion planning from the start pose to the pre-grasp pose (all
 170 methods).

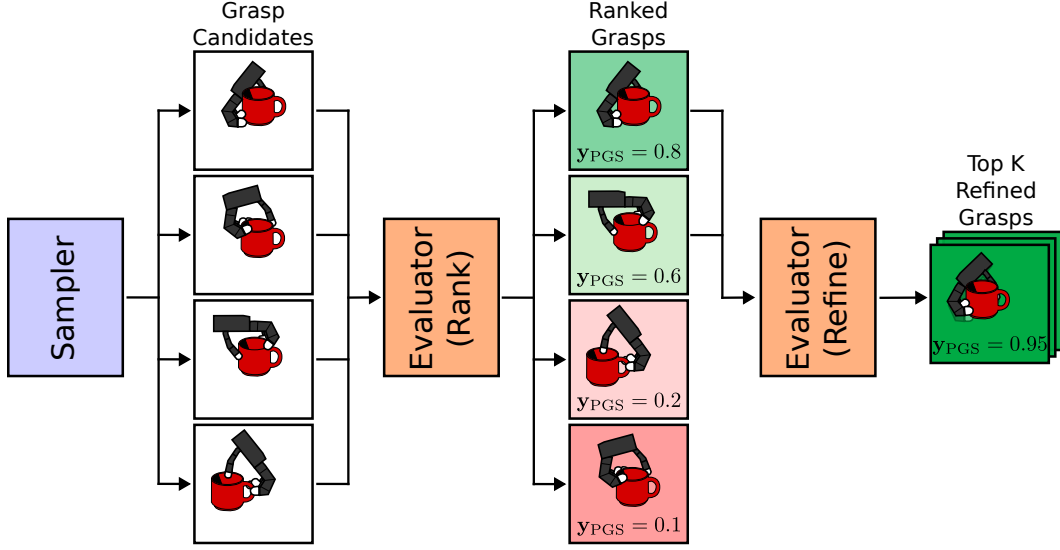


Figure 15: Our grasp planners consist of a *sampler* and an *evaluator*. First, the sampler generates a batch of grasp candidates. The evaluator ranks them, and the top K grasps are refined using sampling-based optimization, where the objective is given by the evaluator.

For both applications, the mesh is generated by the following procedure.

1. Training a NeRF following the procedure described in Appendix C.1.
2. Using `scikit-image` [12] to perform marching cubes, which extracts a 2D surface mesh from a 3D volume with a NeRF density level set of 15 within a bounding box of $[-0.2m, -0.2m, 0.0m] \times [0.2m, 0.2m, 0.3m]$ (with z being the up-direction).
3. Using `trimesh` [13] to remove floaters. This is done by only keeping connected components with at least 31 edges.

Figure 14 shows examples of NeRF-generated meshes. We found the parameters above to be reasonable for all test objects.

D Grasp Planning

As explained in the main text, during grasp planning, a batch of candidate grasps is sampled, and only the top K of these are retained for the refinement phase. In our simulation experiments, we let $K = 5$, which are all refined and executed in simulation. In our real-world experiments, we let $K = 40$, which are all refined and the best is executed on hardware. See Figure 15 for a schematic of the sample/refine process.

Recall that our refinement is sampling-based, where the previous iterate is perturbed by some number of samples and the best one is chosen to be the next iterate. For all perturbations, we use zero-mean Gaussian noise as follows.

- Wrist translation: standard deviation of 5mm per spatial coordinate.
- Wrist and grasp orientations: the noises are sampled using a standard deviation of 0.05m from $\mathfrak{so}(3)$, which are then converted to elements of $SO(3)$ via the exponential map.
- Joint angles: standard deviation of 0.01 radians.

The refinement is run for 50 iterations. Other refinement strategies could work as well, such as gradient-based methods or other sampling-based methods.

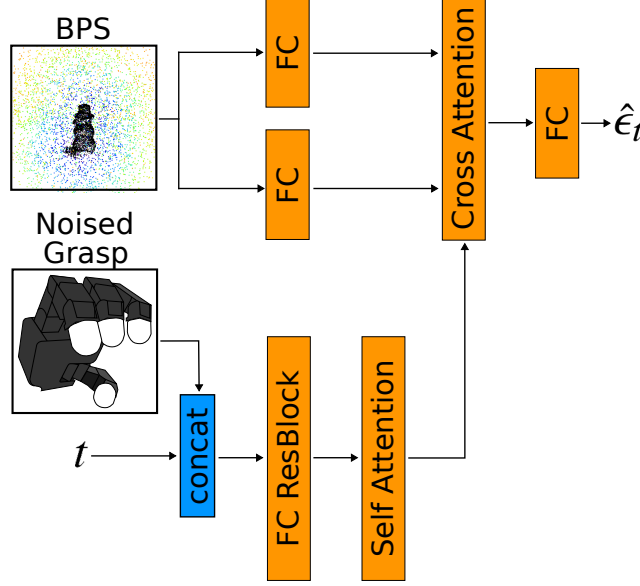


Figure 16: **Diffusion Sampler architecture.** Our model takes in a basis point set, a noised grasp, and a diffusion timestep. The noised grasp and the diffusion timestep are processed into a query using a self-attention block. The basis point set is processed into a key-value pair. These are embedded together using a cross-attention block and used to compute the predicted noise. This is a similar architecture to the one used by Weng et al. [1].

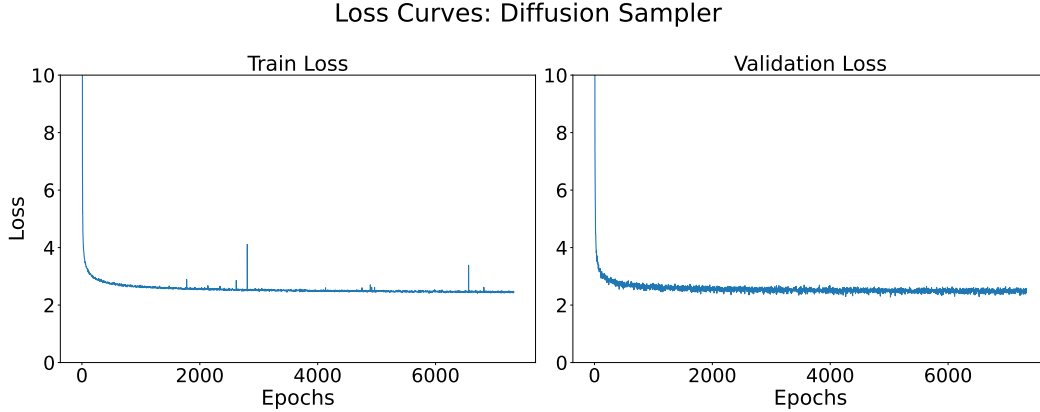


Figure 17: Train and validation loss curves for the diffusion sampler.

To allow grasp parameters to be represented as vector inputs or outputs, we parameterize them as $\mathbf{g} = (\mathbf{x}, \mathbf{r}, \theta, \mathbf{d}_1, \dots, \mathbf{d}_{n_f}) \in \mathbb{R}^{9+n_j+3n_f}$, where $\mathbf{x} \in \mathbb{R}^3$ is the wrist position, $\mathbf{r} \in \mathbb{R}^6$ is the wrist orientation represented by a continuous 6-D rotation vector [14], $\theta \in \mathbb{R}^{n_j}$ is the pre-grasp joint configuration, and $\mathbf{d}_i \in \mathbb{R}^3$ is the direction in which the i^{th} fingertip moves during the grasp.

D.1 Diffusion Sampler

Figure 16 shows the diffusion sampler architecture inspired by Weng et al. [1]. As their implementation is not publicly available, we re-implemented their architecture to the best of our ability via the textual description. We use an embedding dimension of 128 and a sequence length of 4 for all attention modules. The sequence length is created by reshaping the outputs of the fully-connected layers (to (4, 128)). We use Denoising Diffusion Implicit Models (DDIM) [15] with a linear sched-

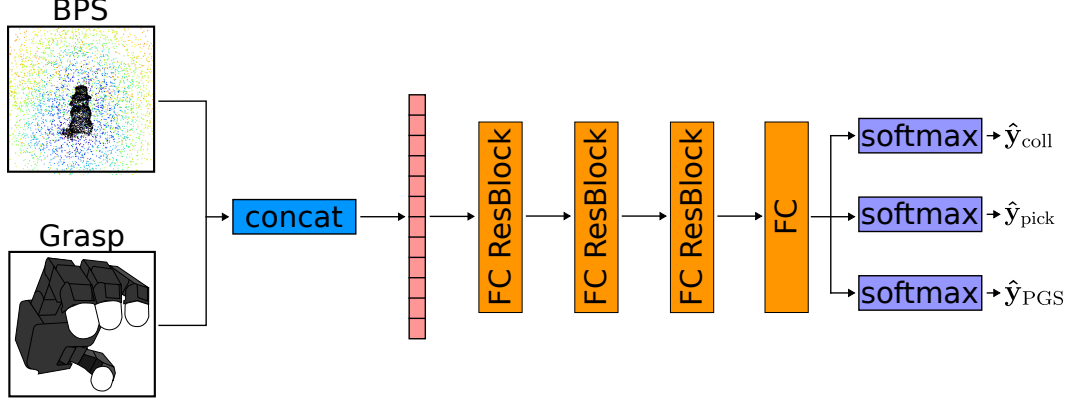


Figure 18: **BPS Evaluator architecture.** Our model takes in a basis point set and a grasp, which are concatenated together and then passed through three fully-connected resblocks and a final fully-connected layer, which returns logits for 3 labels: whether (i) there are unwanted collisions in the scene, (ii) the pick succeeded in the simulator, and (iii) both (i) and (ii) are true. See the official implementation from Mayer et al. [2] [here]. This is the same architecture used by Weng et al. [1] and our work.

205 ule on the noise variance β that moves from 0.0001 to 0.02 over a total of 1000 diffusion timesteps.
 206 Figure 17 shows the train and validation loss curves for this model.

207 Although the diffusion sampler is trained to only generate successful grasps, it can still produce
 208 unsuccessful grasps as seen in previous works [1, 2]. While it could typically generate feasible-
 209 looking grasps on a diverse range of objects, its most common failure modes were (1) generating
 210 grasps that were “close” to successful but with fingers slightly too close or far to successfully execute,
 211 and (2) generating a grasp that was substantially too far from the object.

212 D.2 Sampling from a Fixed Dataset

213 To disentangle the importance of the learned evaluator from the impact of the diffusion sampler, we
 214 perform additional experiments in which we replace the diffusion sampler with a simple baseline
 215 in which we store a fixed set of 4349 grasps from the training set. More specifically, this was
 216 generated by storing one grasp per training set object that exhibited a high probability of success
 217 ($y_{\text{PGS}} \geq 0.9$). For objects that did not have any such grasps, we did not store any. We emphasize
 218 that these grasps were from training set objects, so they were not designed for the unseen test objects
 219 used for simulation evaluation or the real-world objects used for hardware evaluation.

220 D.3 Grasp Evaluators

221 Recall that the evaluators are trained by regressing on three distinct soft labels: y_{PGS} , y_{coll} , and y_{pick} .
 222 Although we only use the \hat{y}_{PGS} prediction at inference time, we choose to regress the other two labels
 223 \hat{y}_{coll} and \hat{y}_{pick} at train time, as we found that these labels provide an additional signal about *why* a
 224 given grasp may be failing.

225 D.3.1 BPS Evaluator Details

226 Figure 18 shows the architecture used for the BPS evaluator. We used the official implementation
 227 from Mayer et al. [2] [here], which is the same architecture used by Weng et al. [1].

228 D.3.2 NeRF Evaluator Details

229 Figure 19 shows the architecture used for the NeRF evaluator. We use a novel grasp representation
 230 that leverages NeRF features directly.

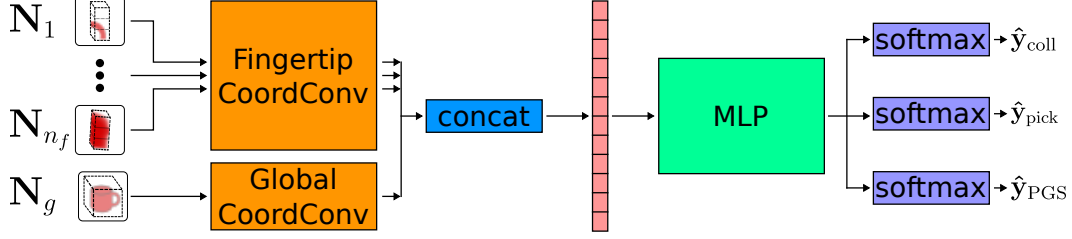


Figure 19: **NeRF Evaluator architecture.** Our model takes in local and global NeRF feature grids, passing them through 3D CoordConv networks and an MLP, which returns logits for 3 labels: whether (i) there are unwanted collisions in the scene, (ii) the pick succeeded in the simulator, and (iii) both (i) and (ii) are true. The Fingertip CoordConv weights are shared for all fingertips.

We will motivate this representation and then describe it in detail. A key factor for grasp success is the surface geometry at the contact points. Modeling this from images is difficult, and few (if any) fast, reliable surface reconstruction methods are adequate for grasp planning. Here, we use NeRF features that we hypothesize capture accurate estimates of the object surface. Centered at finger i 's position \mathbf{x}_i , a square of side length 0.06m is swept along \mathbf{d}_i by 0.08m to recover a rectangular prism which is discretized into a 4D tensor $\mathbf{N}_i \in \mathbb{R}^{4 \times 31 \times 31 \times 41}$, where the first channel dimension is the NeRF density and the last 3 are spatial coordinates. These grid dimensions overapproximate both the fingertip size and the grasp depth to ensure the geometric information captured is not too local. Further, to capture the global object geometry, we generate a grid centered on the estimated object centroid with side lengths 0.4m $\mathbf{N}_g \in \mathbb{R}^{4 \times 41 \times 41 \times 41}$. The centroid is estimated by assuming uniform mass density and integrating over spatial regions with NeRF density exceeding 15.0.

In summary, our architecture uses local NeRF densities and global NeRF densities as inputs. The local NeRF densities are sampled grids approximating the fingertip swept volumes when moving along the grasp directions. The global NeRF densities are a sampled grid of fixed size to capture the object's global geometric features.

D.4 FROGGER Details

For all experiments that use FROGGER [16, 17], we use the open-source implementation provided at <https://github.com/alberthli/frogger>. The procedure for acquiring the meshes used for planning is described in Appendix C.3.

Out of the 100 real-world pick attempts using FROGGER, there were a total of 49 failures. 16 failures were caused by planned grasps that failed to lift the object, and 33 failures were caused by planning issues in which no grasp could be found within the timeout period.

FROGGER allows users to either plan with a floating hand or the full hand-arm system. We opt to use the full hand-arm system so that the generated grasps account for kinematic reachability of grasp poses when using a given arm, resulting in fewer downstream motion planning failures.

E Experiment Details

E.1 Simulation: Ablation Study on Dataset Size Details

We hypothesize that learned grasp evaluators could help achieve robust sim-to-real transfer for multi-finger grasp planners, but only if the associated data are large-scale and account for realistic perceptual inputs.

Looking at prior works, we see a clear trend toward larger multi-fingered grasp datasets used for training grasp evaluators (see Figure 20). Recognizing this trend, our goal with this ablation is to study how much the increased scale of our dataset (3.5M grasps across 4.3K objects) impacts

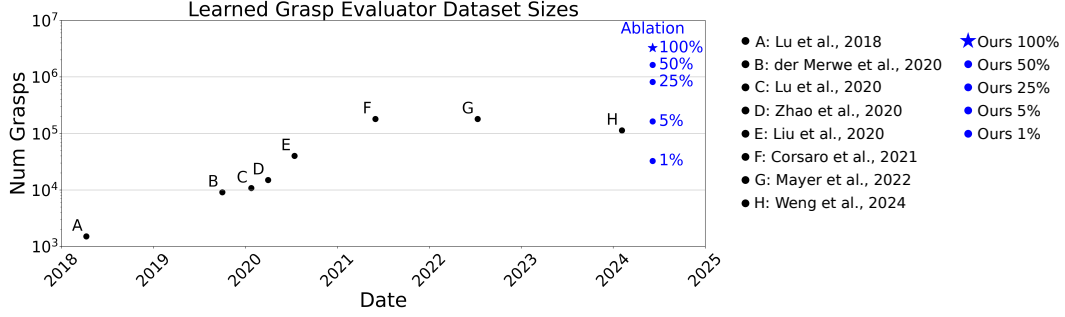


Figure 20: Comparison between learned grasp evaluator dataset sizes in prior works over time, with our dataset and our ablations shown on the right. The full training set contains 3.26M grasps.

Loss Curves: Ablation Study on Evaluator Training Set Size

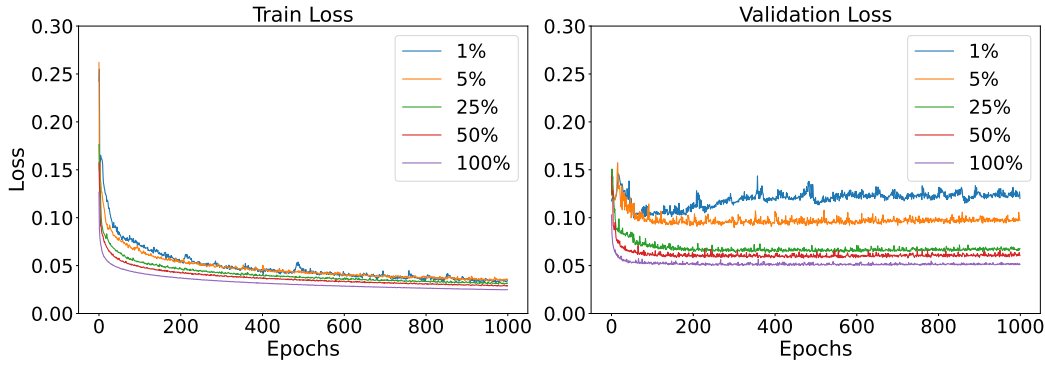


Figure 21: Comparison of loss curves across each model trained on a different fraction of the training dataset and validated on the same validation dataset. These show a clear correlation between dataset scale and validation performance. The full training set contains 3.26M grasps.

performance, which we measure by examining the improvement in the simulated probability of grasp success gained by evaluator refinement.

Figure 21 shows the train and validation loss curves of each model trained on a different fraction of the training dataset and validated on the same validation dataset. We see that all models are trained to convergence and achieve similar train losses, but their corresponding validation losses show a consistent correlation between dataset scale and validation performance.

E.2 Hardware: Object Selection

Figure 22 shows the real-world objects labeled with their corresponding names. Easy objects are those that have high-quality grasps when approached from any or most directions. For example, tall, cylindrical objects can be grasped overhead or from the side. Medium objects are those with more unusual geometry, including “distractors,” or objects which are harder to grasp when the hand is arbitrarily oriented. For example, the mug has a handle and the lunchbox is more easily pinched in the thin direction. Lastly, hard objects contain very unusual, non-convex geometry/visual features, or are difficult to grasp without affordances. For example, the goggles are both reflective and transparent, which poses a challenge for object reconstruction algorithms, and the bunny’s ears are a large distractor for grasp planners. The bunny, squirrel, and dino were custom printed as hard test objects. The strainer and mallet were chosen to be out-of-distribution objects, providing a ceiling on the performance of our grasping algorithms. No method obtained any successful grasps on the strainer.

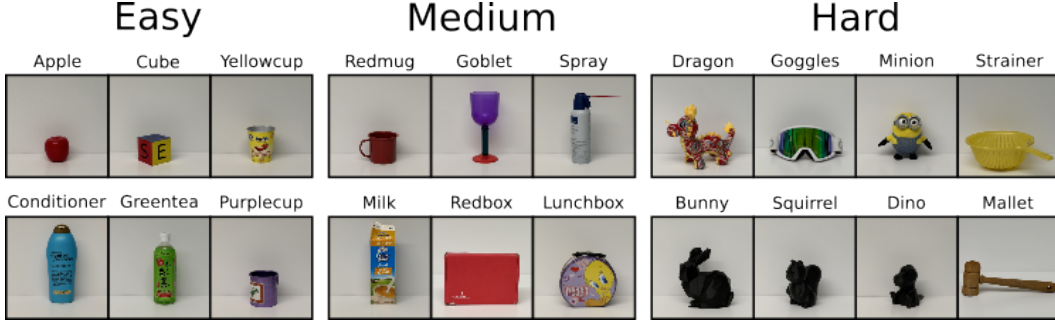


Figure 22: Real-world objects labeled with their names corresponding to Table 4.

283 E.3 Hardware: Detailed Results

284 Table 4 presents the detailed quantitative results of all grasp planning methods on various objects in
 285 the real world.

Difficulty	Objects	Methods				
		No Evaluator		Evaluator		
		FRoGGeR	Diffusion No Evaluator	Diffusion BPS	Diffusion NeRF	Fixed BPS
Easy	Apple	4/5	3/5	5/5	5/5	5/5
	Cube	3/5	1/5	5/5	4/5	5/5
	Yellowcup	5/5	4/5	5/5	5/5	5/5
	Conditioner	4/5	5/5	5/5	4/5	5/5
	Greentea	3/5	4/5	4/5	5/5	5/5
	Purplecup	5/5	5/5	5/5	5/5	5/5
	TOTAL	24/30 (80%)	22/30 (73%)	29/30 (97%)	28/30 (93%)	30/30 (100%)
Medium	Redmug	2/5	4/5	5/5	5/5	5/5
	Goblet	5/5	3/5	5/5	5/5	5/5
	Spray	4/5	0/5	2/5	4/5	3/5
	Milk	1/5	5/5	5/5	5/5	5/5
	Redbox	5/5	1/5	4/5	5/5	3/5
	Lunchbox	2/5	0/5	5/5	2/5	4/5
	TOTAL	19/30 (63%)	13/30 (43%)	26/30 (87%)	26/30 (87%)	25/30 (83%)
Hard	Dragon	0/5	0/5	2/5	2/5	4/5
	Goggles	0/5	0/5	4/5	2/5	4/5
	Minion	0/5	3/5	5/5	5/5	5/5
	Strainer	0/5	0/5	0/5	0/5	0/5
	Bunny	0/5	1/5	5/5	4/5	3/5
	Squirrel	1/5	2/5	4/5	3/5	4/5
	Dino	2/5	2/5	5/5	5/5	5/5
	Mallet	5/5	0/5	1/5	1/5	0/5
	TOTAL	8/40 (20%)	8/40 (20%)	26/40 (65%)	22/40 (55%)	25/40 (62%)
All	TOTAL	51/100 (51%)	43/100 (43%)	81/100 (81%)	76/100 (76%)	80/100 (80%)

Table 4: Results of different grasp planning methods on various objects in the real world. Images of these objects can be found in Figure 22.

286 Qualitatively, we observed that our evaluator-based methods very rarely exhibited *edge-seeking*
 287 behavior, wherein the fingers are placed on corners or edges of objects, and has been noted as
 288 a common failure mode by other works [16, 17]. We suspect this may be a result of our label

smoothing, though we do not rigorously test this. However, “distractor” geometries like the bunny ears or spray nozzle frequently caused grasp planning failures, attracting the fingers towards them. Methods like FROGGER rely on online mesh reconstruction, so objects like the goggles caused issues due to transparency/reflectivity. Lastly, many of the diffusion sampler’s unrefined grasps appeared “close” to success even when failing, which suggests that generative modeling may be a viable strategy with more data. Some examples are shown in Figure 23.

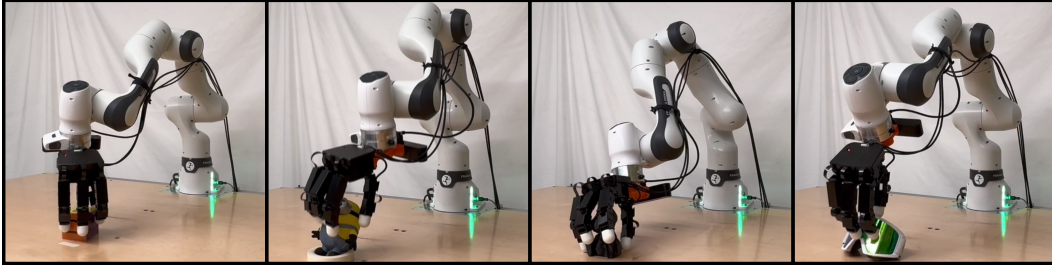


Figure 23: Some examples of failed grasps generated by the diffusion sampler. Though they failed, the grasps appear qualitatively reasonable.

E.4 Hardware: Motion Planning

To evaluate a given grasp in the real world, we need to use a motion planner to move the hand to the corresponding pre-grasp pose. We use cuRobo [18], which performs parallelized collision-free motion generation. In particular, we use cuRobo’s graph planner, which uses Probabilistic Road Map (PRM) to find a collision-free path. Object collision avoidance requires an object mesh, which is acquired through a process explained in Appendix C.3. We use a collision buffer of 1mm.

When executing a grasp on hardware, we decompose the motion into three stages: (1) start pose to pre-grasp pose, (2) pre-grasp pose to grasp pose, (3) grasp pose to lift pose.

The first stage is the most challenging motion planning problem because the hand needs to find a collision-free path to get very close to the object. To simplify this problem, we utilize inverse kinematics to adjust the pre-grasp finger joints, moving them an additional 3cm backward along the fingertip grasp directions.

In addition to ensuring kinematic feasibility, we also need to avoid damaging the cabling of the wrist-mounted camera during grasp execution. To achieve this, we filter out grasp samples if either (1) the normal direction of the palm is within 60 degrees from the upward direction of the world frame or (2) the direction from the palm to the middle finger deviates by more than 60 degrees from the forward direction of the world frame.

To account for these checks, we request 40 grasps from the grasp planner, which are then sorted by the grasp planner’s metric. If no metric is available, such as in the case of a diffusion sampler without an evaluator, the grasps remain unsorted. Next, we utilize cuRobo to solve all 40 motion planning problems in parallel and then execute the first grasp for which a motion plan is successfully found.

References

- [1] Z. Weng, H. Lu, D. Kragic, and J. Lundell. Dexdiffuser: Generating dexterous grasps with diffusion models, 2024.
- [2] V. Mayer, Q. Feng, J. Deng, Y. Shi, Z. Chen, and A. Knoll. Ffhnet: Generating multi-fingered robotic grasps for unknown objects in real-time. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 762–769, 2022. doi:10.1109/ICRA46639.2022.9811666.
- [3] D. Turpin, T. Zhong, S. Zhang, G. Zhu, J. Liu, R. Singh, E. Heiden, M. Macklin, S. Tsogkas, S. Dickinson, and A. Garg. Fast-grasp’d: Dexterous multi-finger grasp generation through differentiable simulation, 2023.
- [4] R. Wang, J. Zhang, J. Chen, Y. Xu, P. Li, T. Liu, and H. Wang. DexGraspNet: A large-scale robotic dexterous grasp dataset for general objects based on simulation. *arXiv preprint arXiv:2210.02697*, 2022.
- [5] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960. doi:10.1007/BF01386213. URL <https://doi.org/10.1007/BF01386213>.
- [6] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance GPU-Based physics simulation for robot learning, 2021.
- [7] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multi-resolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, July 2022. ISSN 1557-7368. doi:10.1145/3528223.3530127. URL <http://dx.doi.org/10.1145/3528223.3530127>.
- [8] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. Mcallister, J. Kerr, and A. Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings, SIGGRAPH ’23*. ACM, July 2023. doi:10.1145/3588432.3591516. URL <http://dx.doi.org/10.1145/3588432.3591516>.
- [9] F. Warburg*, E. Weber*, M. Tancik, A. Hołyński, and A. Kanazawa. Nerfbusters: Removing ghostly artifacts from casually captured nerfs. In *International Conference on Computer Vision (ICCV)*, 2023.
- [10] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.
- [11] S. Prokudin, C. Lassner, and J. Romero. Efficient learning on point clouds with basis point sets. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4332–4341, 2019.
- [12] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi:10.7717/peerj.453. URL <https://doi.org/10.7717/peerj.453>.
- [13] Dawson-Haggerty et al. trimesh. URL <https://trimesh.org/>.
- [14] Y. Zhou, C. Barnes, L. Jingwan, Y. Jimei, and L. Hao. On the continuity of rotation representations in neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [15] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv:2010.02502*, October 2020. URL <https://arxiv.org/abs/2010.02502>.

- 361 [16] A. H. Li, P. Culbertson, J. W. Burdick, and A. D. Ames. Frogger: Fast robust grasp generation
362 via the min-weight metric. *arxiv:2302.13687*, February 2023.
- 363 [17] A. H. Li, P. Culbertson, and A. D. Ames. Toward an analytic theory of intrinsic robustness for
364 dexterous grasping, 2024.
- 365 [18] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane,
366 H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox. Curobo: Parallelized collision-free
367 robot motion generation. In *2023 IEEE International Conference on Robotics and Automation*
368 *(ICRA)*, pages 8112–8119, 2023. doi:[10.1109/ICRA48891.2023.10160765](https://doi.org/10.1109/ICRA48891.2023.10160765).