

## 1 Appendix

### 2 A Architecture

3 M $\pi$ Former uses PointNet++ [1] to encode the point cloud and a transformer [2] to fuse the point cloud  
4 features with a representation of the current joint state. The input point cloud has a feature vector of  
5 length 4 for every point. All obstacles are assigned the same feature, all target points are assigned  
6 the same feature, and each robot point, which are sampled deterministically from the robot’s mesh, is  
7 assigned a unique feature to disambiguate points on the arm. Our PointNet++ encoding architecture  
8 consists of three Set Aggregation (SA) layers. SA layers are a sparse 3D analog to convolutional  
9 layers. Each layer receives a point cloud where each point has a feature and outputs a smaller point  
10 cloud by using furthest point sampling to select  $\frac{1}{4}$  of the points. Then, each sampled point is used as  
11 the center of a ball query. The ball query samples up to 64 points inside the ball and concatenates the  
12 ball center’s coordinates to each point’s feature vector. A four-layer MLP is then run on each point  
13 and MaxPool [3] collects the points inside the ball to produce a single feature per ball. The layers’  
14 ball queries have radii of 5, 30, and 50 centimeters respectively. Our input point cloud always has  
15 6,272 points—4,096 obstacle points, 2,048 robot points, 128 target points. The downsampled point  
16 cloud after the third set aggregation layer has 98 points. Finally, we add 3D positional encoding to  
17 each of these 98 points, similar to [4].

18 The transformer takes a sequence of tokens as input, consisting of the 98 output features of the third  
19 SA layer, a token for the current joint configuration, and a learned constant token, similar to the  
20 decoder tokens in [5]. We get the joint angle token by passing the joint angles, which are normalized  
21 to be between -1 and 1, through a single linear layer. Our transformer has 8 layers with an embedding  
22 dimension of 512 and a feed-forward dimension of 2,048. To produce the final output  $\Delta q$ , we take  
23 the last token of the output sequence and map it through a single linear layer.

### 24 B Data

25 Our environments are similar to those demonstrated in M $\pi$ Nets, but they differ in two key ways:  
26 we augmented the cubby design to encourage reasonable expert behavior by adding a floor beneath  
27 the robot, and we increased the complexity of the tabletop environment by adding more objects  
28 and increasing the range of reachable poses. Within these constructed environments, we randomly  
29 sample end effector poses and their corresponding inverse kinematics (IK) solutions, which we  
30 compute using IKFast [6]. For the cubby environments, the poses are all grasping positions inside a  
31 cubby. For the tabletop, the poses are grasps pointing toward the lower hemisphere and placed either  
32 near the table’s surface or on top of the objects. We also add neutral configurations drawn from  
33 uniform distribution around the robot’s default pose to the tabletop data. These poses, for both types  
34 of environments, must be at least 5mm away from obstacles. We then use AIT\* [7] with a path-length  
35 objective combined with a spline-based shortcutting [8] to generate expert demonstrations. In our  
36 planning pipeline, we impose a 20 second time limit in which we sample uniformly from the robot’s  
37 configuration space, marking any sample that is either in self-collision or within 5mm of an obstacle  
38 as invalid. During the smoothing stage, we fit a collision and dynamics-aware spline to the planned  
39 path while shortcutting. We then sample from the spline at a fixed timestep, leading to paths with  
40 similar velocities, but varying lengths.

41 We chose this sampling-based pipeline because it enables us to produce expert demonstrations that  
42 lie precariously close to obstacles. Previously, M $\pi$ Nets [9] demonstrated strong performance when  
43 trained with a so-called *Hybrid Expert*, which uses a reactive controller [10] to follow a planned  
44 end effector path. While this expert is effective for learning, it is highly conservative, preferring to  
45 stay far away from obstacles. In their experiments, the authors demonstrated that the *hybrid expert*  
46 demonstrations are insufficient to learn to solve problems that lie very close to obstacles. With our  
47 sampling expert, we chose a 5mm buffer from obstacles because this is sufficiently close for most  
48 tasks. As we designed our expert, we observed that increasing the collision margin improves learned

collision avoidance, but this limits the expert’s ability (and thus, the policy’s ability) to plan to targets near obstacles.

When generating partially observed point clouds during inference, we captured depth information from randomized camera positions placed in the scene. In these scenes, we placed the robot at a fixed neutral starting configuration and segmented the robot out of the image. To randomize the camera, it was first placed in the scene at a predefined location facing the robot and obstacles, and was then rotated randomly by up to  $30^\circ$  about the z-axis (rotating side to side), then again by up to  $10^\circ$  about the camera’s local x-axis (tilting up and down). Both of these rotations were applied using a fixed pivot point directly in front of the camera. Finally, the camera was translated randomly along the global z axis and y axes by up to 25cm.

To generate our expert dataset, we used a single desktop with a AMD Ryzen Threadripper 3990X 64-Core Processor. Generating the cubby and tabletop data took four and six days respectively.

## C Loss Functions

**Task Space Loss** The aim of this loss is to compare the physical positions of the policy’s predicted robot joint space configuration and the expert’s joint space configuration. For both configurations, we use forward kinematic functions  $\phi^{\{i\}}(\cdot)$  to map joint angles of the robot  $q$  to 1,024 points  $x^{\{i\}}$  on the robot’s surface, represented in 3D coordinates.

$$L_{\text{BC}}(\hat{\Delta}q) = \sum_{i=0}^{1,024} \|\hat{x}^i - x^i\|_2 + \|\hat{x}^i - x^i\|_1 \quad (1)$$

Like  $M\pi$ Nets, we sum  $L1$  and  $L2$  distances in the loss because the sum penalizes both large and small errors. We use a task space loss following  $M\pi$ Nets, which demonstrated it to be more effective when reasoning about collision avoidance as small perturbations along the kinematic chain can lead to large deviations for the end effector.

**Collision Avoidance Loss** The training data was generated in simulation, giving us access to privileged information unavailable during inference, including a signed-distance representation of the scene. To avoid collisions, we use a hinge-based loss on  $D(x)$ , the signed distance from a point  $x$  on the robot to the nearest surface in the scene. Inspired by motion optimization [11, 12, 13], this loss effectively pushes the robot out of regions of collision. As in Equation 1, we use 1,024 points  $x^{\{i\}}$  on the robot’s surface to measure collision.

$$L_{\text{collision}} = \sum_i h(\hat{x}^i), \text{ where} \quad (2)$$

$$h(\hat{x}^i) = \begin{cases} -D(\hat{x}^i), & \text{if } D(\hat{x}^i) \leq 0 \\ 0, & \text{if } D(\hat{x}^i) > 0 \end{cases}$$

## D ROPE

Our expert-guided fine tuning algorithm *Refining on Optimized Policy Experts (ROPE)* refines a pretrained model to reduce the collision rate using automated labeling of online data generated by the learning agent. This algorithm rolls out short-horizon sequences  $s'$  using the current model, and then if these collide, we generate a corrected sequence  $\hat{s}'$  by optimizing the trajectory out of collision. This optimization uses the collision avoidance loss in Equation 2 to push the sequence out of collision. We use AdamW to perform this optimization for simplicity, although we expect other methods typical to motion optimization such as Gauss-Newton or Levenberg-Marquardt may lead to a faster fine-tuning procedure. Once enough corrected data has been collected, the model is fine tuned using the task space and collision avoidance losses outlined in Appendix C. Algorithm 1 provides pseudocode. During fine-tuning, we continually use the latest policy to perform rollouts, even as it is updated. In our best-performing fine-tuning experiment, we reached peak performance after 21 hours of training.

---

**Algorithm 1:** Refining on Optimized Policy Experts

---

**Result:**  $\pi$ 

```
1  $\pi \leftarrow \pi_{\text{pretrained}}$ 
2  $b \leftarrow \text{Batch Size}$ 
3  $r \leftarrow \text{Correction Ratio}$ 
4  $D_{\text{expert}} \triangleright \text{Dataset containing expert demos}$ 
5  $B_{\text{coll}} \leftarrow \{\}$   $\triangleright \text{Collision correction demos}$ 
6  $B_{\text{free}} \leftarrow \{\}$   $\triangleright \text{Collision-free expert demos}$ 
7 for {state, next_state, tgt, scene} in  $D_{\text{expert}}$  do
8    $s \leftarrow \text{state}$ 
9   for  $j \leftarrow 1$  to  $N$  do
10     $s' \leftarrow \pi(s, \text{tgt})$ 
11     $\triangleright \text{If } s' \text{ collides, correct \& add to buffer}$ 
12    if  $\text{COLLIDES}(s', \text{scene})$  then
13       $\bar{s}' \leftarrow \text{CORRECT}(s', \text{scene})$   $\triangleright \text{Apx Eqn 2}$ 
14       $\text{ADD}(B_{\text{coll}}, \{s, \bar{s}', \text{tgt}, \text{scene}\})$ 
15      break
16     $\triangleright \text{If rollout finishes without collision, add original example to buffer}$ 
17    if  $\text{REACHED}(s', \text{tgt})$  or  $j = N$  then
18       $\text{ADD}(B_{\text{free}}, \{\text{state}, \text{next\_state}, \text{tgt}, \text{scene}\})$ 
19      break
20     $s \leftarrow s'$ 
21  end
22  if  $|B_{\text{coll}}| > rb$  and  $|B_{\text{free}}| > (1 - r)b$  then
23     $\triangleright \text{Make batch \& clear buffers}$ 
24     $B \leftarrow \{\text{POP}(B_{\text{coll}}, rb), \text{POP}(B_{\text{free}}, (1 - r)b)\}$   $\triangleright \text{Compute loss, gradient update}$ 
25     $\pi \leftarrow \text{UPDATE}(\pi, B)$ 
26  end
27   $\triangleright \text{Reached validation accuracy or timeout}$ 
28  if  $\text{TERMINATION\_CONDITION}(\pi)$  then
29    terminate
30 end
```

---

## 89 E Training Implementation

90 *Avoid Everything* was trained on an NVIDIA 4090 in batches of 50 using AdamW [14] with a  
91 learning rate of  $5e-5$  and a linear warmup of 5000 steps from  $1e-5$ . On the cubby environment,  
92 the model was trained for 1.2 million steps, which took approximately four days.

93 During training, we add small amounts of random noise to the input configurations, which [15]  
94 showed leads to improved robustness. Like  $M\pi$ Nets, the training scenes are constructed from  
95 primitives, so point clouds can be generated on the fly during training by sampling points from the  
96 surfaces of these primitives. Robot points are sampled deterministically from the mesh of the robot.  
97 When *Avoid Everything* runs on the real robot, we mask out the robot points in the depth cloud and  
98 re-insert them using the same deterministically sampled points from training.

## 99 F Partial Observability for Analytic Planners

100 Figure 1 show examples of the perceptual pipelines we used for both RRTConnect [18] and  
101 cuRobo [19]. We evaluated RRTConnect with the commonly used motion planning library  
102 MoveIt! [20] paired with Octomap [16] for perception. We used an Octomap with a resolution  
103 of 5mm and RRTConnect [18] (with a 5s timeout) as the planner. In the cubby settings, we found

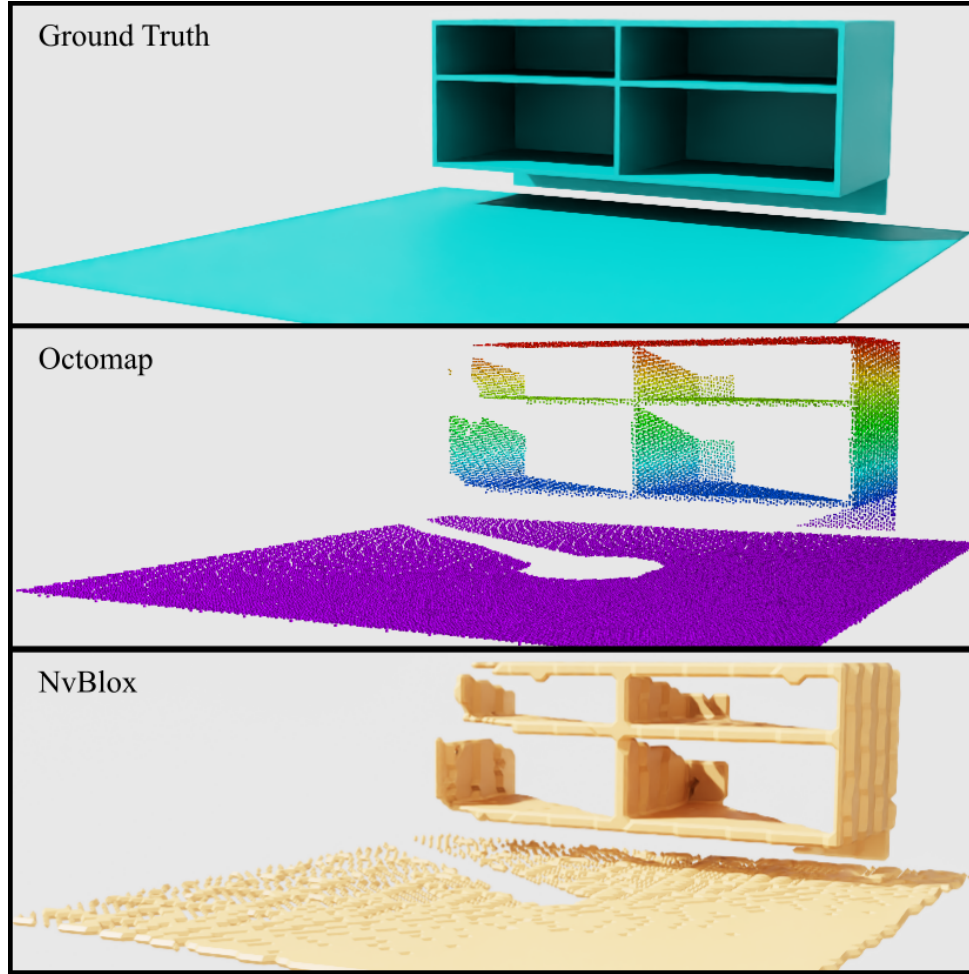


Figure 1: A typical failure case for classical planners is that they do not account for collisions in unobserved regions. In this example, the reconstructions from both Octomap [16] and NvBlox [17] leave large holes due to occlusion. *Avoid Everything* is able to leverage learned priors to produce safe movement without an explicit reconstruction.

104 that the planner found a solution in 99.52% of the problems and we attribute the remaining to noise  
 105 that could be addressed with a longer timeout. However, of these successful plans, over 67% of them  
 106 had collisions. RRTConnect produced fewer collisions (53%) in the tabletop setting, likely due to  
 107 fewer or smaller holes in the point cloud.

108 We ran a similar test using a trajectory optimization method designed to produce smooth trajectories,  
 109 cuRobo [19] and NvBlox [17]. This technique finds a path in 94.74% of of cubby problems, but  
 110 22.88% of these trajectories have collisions. We set the nvBlox resolution to 1cm for this test after  
 111 consulting with the authors of cuRobo [19]. While cuRobo also performed better in the tabletop  
 112 setting, the difference was not as large as RRTConnect (see Table 2). An advantage of these classical  
 113 methods is that they did not require special tuning or training for either environment. Despite *Avoid*  
 114 *Everything* having stronger performance in both environments, we do not expect it to generalize to  
 115 wholly new settings as classical methods can.

## 116 G Point Cloud Completion with Classical Pipeline

117 When capturing point clouds with a depth camera, obstructions in the scene create holes in the  
 118 point cloud. As discussed in section 5.1.3, classical methods often produce a valid path through

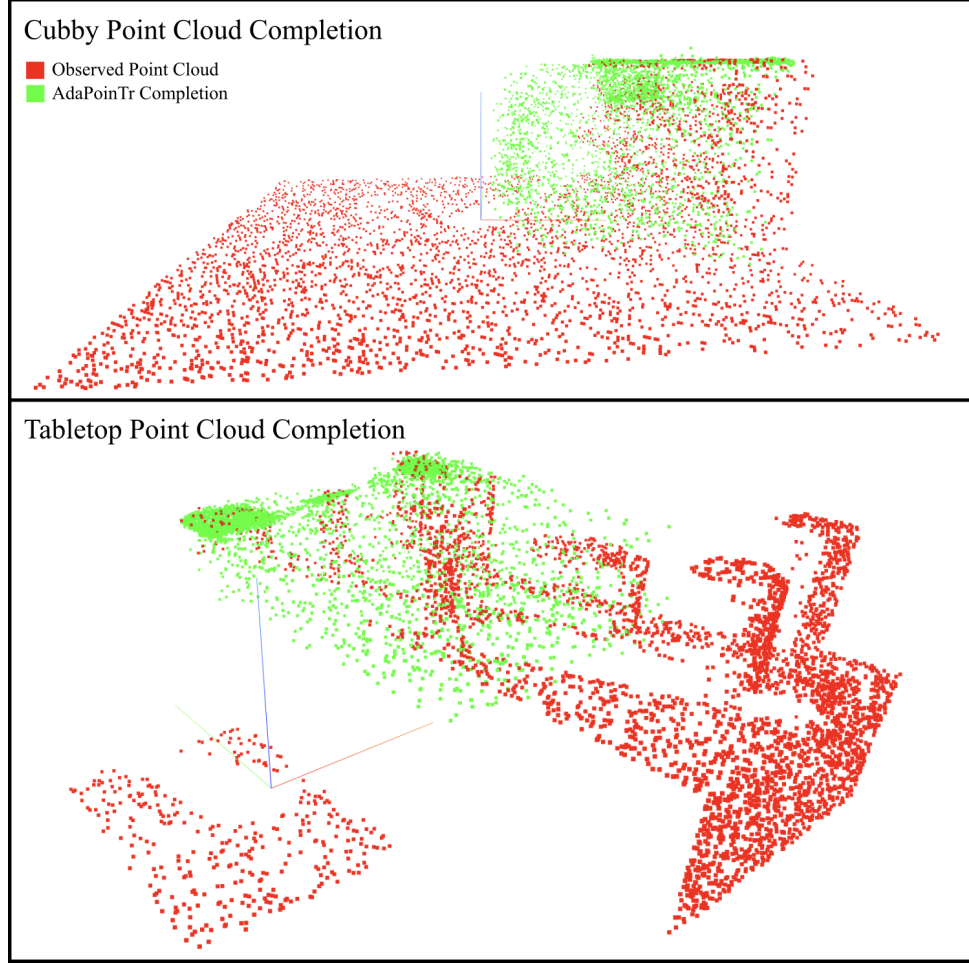


Figure 2: Learned Point cloud completion is a common technique to address unobserved regions of a point cloud. However, when we used the pretrained state-of-the-art point cloud completion network AdaPoinTr [21], we found that it produced highly inaccurate results for our scenes, likely due to distribution shift. In the cubby scene (top), the point cloud completion adds volume to the front of the cubby, making it hard to plan. In the tabletop scene (bottom), the completion misses a large portion of the scene and fails to capture the geometry of the objects.

the observed point cloud but collide with the scene in the unobserved regions. This problem is particularly pronounced in our RRTConnect [18] baseline because the planner searches for any valid feasible path by sampling in free space. Since the unobserved regions are registered as free space, the planner is just as likely to plan through these regions as any other free space in the scene. Instead of using OctoMap to directly represent the points captured from the camera, we could instead use a point cloud completion network, such as the state-of-the-art method AdaPoinTr [21], to estimate the completed shape of the point cloud before constructing the OctoMap and using it for planning. However these techniques are subject to their training distribution and are typically trained on specialized datasets such as ShapeNet [22] and do not generalize. We attempted to use this strategy as a baseline, but found that when pretrained with the Projected ShapeNet-55 dataset, the AdaPoinTr model cannot accurately complete our scenes (see Figure 2), leading to low success rates for the planner. This was particularly pronounced in the cubby setting, where the RRTConnect planner’s reaching success rate (RSR) was 8.84% and among these solutions, the scene collision rate (SCR) was 80.09%. This is a significant degradation from using OctoMap without completion where RSR is 99.52% and SCR is 67.16%. The low planning success rate after completion is largely due to the fact that the completed point clouds obscured either the starting configuration or

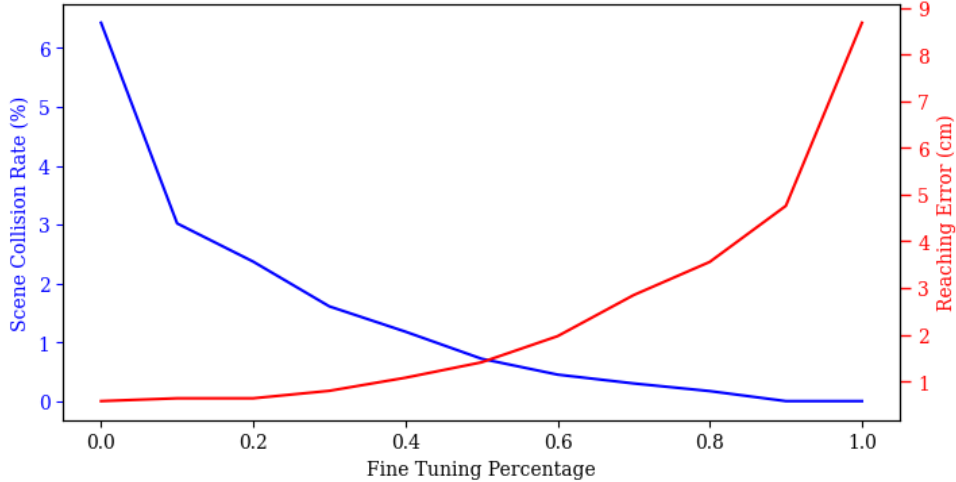


Figure 3: Fine-tuning can be run with different proportions  $r$  of hard negative examples. As  $r$  increases, the collision rate goes down and target error increases. We attribute this phenomenon to the model overfitting to the hard negatives and forgetting the original behavior cloning objective.

target pose, making it impossible to find a valid plan. Point cloud completion performed better in the tabletop settings, where the RSR is 74.14% and SCR is 41.03%. However, these metrics are still significantly lower than OctoMap without completion, where RSR was 99.62% and SCR was 53.30%. Given the performance demonstrated in the original AdaPoinTr publication [21], we suspect that this performance could be significantly improved by retraining the model on a selection of our scenes, but due to resource constraints, we leave this investigation to future work.

## H Maintaining Reaching Performance After Fine Tuning

**ROPE** We aimed to determine the efficacy of *ROPE* by varying the ratio of hard negative examples in each fine-tuning batch. We set this parameter  $r$  as a constant value for the entire fine-tuning procedure and studied how different values change the performance (see Figure 3). For these experiments, we looked only at the cubby setting and used fully observed point clouds, similar to those used during training. We observed a monotonic decrease in collision rate as  $r$  increased. However, we also observed a monotonic increase in the reaching error, *i.e.* the minimum distance from the target after rolling out for 70 time steps. With no fine-tuning, we measured an average reaching error of 0.58cm and a collision rate of 6.43%. At  $r = 20\%$ , we observe an average reaching error of 0.64cm with a collision rate of 2.37%. At  $r = 50\%$ , collision rate is below 1%, but reaching error averages 1.41cm. We chose  $r = 20\%$  for our other experiments, but the choice of this parameter should be determined by the downstream application and the criticality of collision avoidance. We did not experiment with varying  $r$  during fine-tuning as a function of performance, but we hypothesize that setting it as a function of performance would improve results.

**Dagger** One of the most common techniques for fine-tuning a learned policy is DAgger[23]. DAgger aids in accounting for distribution shift by asking the expert to provide demonstrations at every state the pretrained policy would visit. Likewise, *ROPE* can be seen as a way to account for distribution shift by correcting the policy when it fails. While DAgger is a generally useful tool for imitation learning, it requires making many costly calls to the expert. In our case, each expert demonstration requires 20 seconds of computation time, which adds up quickly if a demonstration is needed at every state visited by the policy. We implemented two versions of DAgger as comparisons and show the performance in Table 3. In the first version, we ran the pretrained *Avoid Everything* through its entire training data, collected the trajectories with collisions, and requested an expert demonstration at every step leading up the collision. We found that this technique can improve



performance, reducing the pretrained collision rate of 6.43% in cubby setting to 4.08%, but it is not better than *ROPE*, which reduces the collision rate to 2.37%. We attribute this to the fact that the DAGger corrections use the same expert, which often veers very close (5mm) to obstacles. To verify this, we tested a second version of DAGger that uses a more conservative expert for corrections—one with a 2cm collision buffer. We label this more conservative technique *Cons. DAGger* in Table 3. As discussed in Section 4, this expert is more limited in the problems it can solve, *e.g.* not those that either start or end within 2cm of obstacles. However, we found that this version of DAGger significantly improves collision avoidance without negatively impacting reaching performance, dropping collision rate in the cubby setting to 1.28%. We observe a similar drop in the tabletop setting, bringing pretrained collision rate from 11.26% to 2.31%. Running DAGger, however, is very computationally intensive—collecting DAGger demonstrations for the policy’s failures on our training dataset required nearly five days on a desktop with an NVIDIA 3090 GPU and an AMD Ryzen Threadripper 3990X 64-Core Processor.

When used alone, *ROPE* outperformed DAGger with the original 5mm expert in both the cubby and tabletop settings. Meanwhile, fine-tuning with *Cons. DAGger* outperforms both. However, we did not find *ROPE* to be mutually exclusive of DAGger. With both versions of DAGger, we were able to further improve performance by using *ROPE* as a second fine-tuning step. The best performance came from stacking the conservative DAGger technique with *ROPE*, with success rates of 95.71% and 91.97% in the cubby and tabletop settings respectively.

## I Real Robot Experiments

We used a dual-computer setup running ROS to control our Franka Emika Panda robot. The control computer, which runs a real-time linux kernel, has Intel(R) Core(TM) i7-4770 CPU with 16 Gigabytes of RAM. The second computer, which runs *Avoid Everything*, has an Intel(R) Core(TM) i9-9900K CPU, 32 Gigabytes of RAM, and an NVIDIA Titan RTX GPU. We use a Kinect V2 for perception, which captures point clouds at approximately 10Hz. We use [24] for eye-on-hand calibration and [25] to remove the robot from the depth cloud; we then re-insert these robot points into the cloud using the deterministic sampling method described in Section E. We are able to run the model at approximately 25Hz on our hardware, which allows for reactive motion. We send each predicted action directly to a lower level joint controller [26].

The model is able to react to moving obstacles in the scene, but due to speed of our camera, it can take up to 140ms—100ms for the camera update, 40ms for the model update—for the robot to react to an obstacle. We expect that this reactivity could be improved with a faster camera, a faster GPU, or both. We used our best performing checkpoint, which was first fine-tuned with the conservative DAGger pipeline and then fine-tuned with *ROPE* (see Section 5.1.4).

One challenge in our setup is that the gripper of the Franka is nearly symmetric about the axis that points from the wrist to the midpoint of the fingers. Our training data consisted of randomly generated poses, but these poses typically sampled from only half of the rotations about this axis. When we provided an out-of-distribution pose where the 180° rotation about this axis would be in distribution, we observed the robot typically tries to exploit the symmetry of the gripper and reach the symmetric in-distribution pose. Depending on the application, these 180° rotations may or may not be acceptable. We believe this could be fixed by increasing the variation of target poses in the training set, adding a unique per-point embedding to the gripper points to distinguish orientations, or both.

## References

- [1] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. URL <https://arxiv.org/abs/1706.02413>.
- [2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. URL <https://arxiv.org/abs/1706.03762>.
- [3] D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, 2012. URL <https://arxiv.org/abs/1202.2745>.
- [4] W. Yuan, A. Murali, A. Mousavian, and D. Fox. M2t2: Multi-task masked transformer for object-centric pick and place. *arXiv preprint arXiv:2311.00926*, 2023. URL <https://arxiv.org/abs/2311.00926>.
- [5] T. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *ArXiv*, abs/2304.13705, 2023. URL <https://arxiv.org/abs/2304.13705>.
- [6] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. URL [http://www.programmingvision.com/rosen\\_diankov\\_thesis.pdf](http://www.programmingvision.com/rosen_diankov_thesis.pdf).
- [7] M. P. Strub and J. D. Gammell. Advanced bit\* (abit\*): Sampling-based planning with advanced graph-search techniques. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 130–136, 2020. URL <https://arxiv.org/abs/2002.06589>.
- [8] K. K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. *2010 IEEE International Conference on Robotics and Automation*, pages 2493–2498, 2010. URL <https://ieeexplore.ieee.org/document/5509683>.
- [9] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox. Motion policy networks. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022. URL <https://arxiv.org/abs/2210.12209>.
- [10] K. V. Wyk, M. Xie, A. Li, M. A. Rana, B. Babich, B. N. Peele, et al. Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior. *IEEE Robotics and Automation Letters*, 7:3202–3209, 2022. URL <https://arxiv.org/abs/2109.10443>.
- [11] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE international conference on robotics and automation*, pages 489–494. IEEE, 2009. URL <https://ieeexplore.ieee.org/document/5152817>.
- [12] J. Dong, M. Mukadam, F. Dellaert, and B. Boots. Motion planning as probabilistic inference using gaussian processes and factor graphs. In *Robotics: Science and Systems*, volume 12, 2016.
- [13] A. Fishman, C. Paxton, W. Yang, D. Fox, B. Boots, and N. D. Ratliff. Collaborative interaction models for optimized human-robot teamwork. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11221–11228, 2020. URL <https://ieeexplore.ieee.org/document/9341369>.
- [14] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. URL <https://arxiv.org/abs/1711.05101>.



- [15] L. Ke, J. Wang, T. Bhattacharjee, B. Boots, and S. S. Srinivasa. Grasping with chopsticks: Combating covariate shift in model-free imitation learning for fine manipulation. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6185–6191, 2021. URL <https://arxiv.org/abs/2011.06719>.
- [16] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34:189 – 206, 2013. URL <https://link.springer.com/article/10.1007/s10514-012-9321-0>.
- [17] A. Millane, H. Oleynikova, E. Wirbel, R. Steiner, V. Ramasamy, D. Tingdahl, and R. Siegwart. nvblox: Gpu-accelerated incremental signed distance field mapping, 2023. URL <https://arxiv.org/abs/2311.00626>.
- [18] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 2:995–1001 vol.2, 2000. URL <https://ieeexplore.ieee.org/document/844730>.
- [19] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, et al. Curobo: Parallelized collision-free minimum-jerk robot motion generation. *arXiv preprint arXiv:2310.17274*, 2023. URL <https://arxiv.org/abs/2310.17274>.
- [20] S. Chitta, I. Sucan, and S. Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012. URL <https://ieeexplore.ieee.org/document/6174325>.
- [21] X. Yu, Y. Rao, Z. Wang, J. Lu, and J. Zhou. Adapointr: Diverse point cloud completion with adaptive geometry-aware transformers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.
- [22] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.
- [23] S. Ross, G. J. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2010. URL <https://arxiv.org/abs/1011.0686>.
- [24] M. Esposito. realtime\_urdf\_filter. [https://github.com/IFL-CAMP/easy\\_handeye](https://github.com/IFL-CAMP/easy_handeye), 2024.
- [25] N. Blodow. realtime\_urdf\_filter. [https://github.com/blodow/realtime\\_urdf\\_filter](https://github.com/blodow/realtime_urdf_filter), 2024.
- [26] M. Bhardwaj. franka\_motion\_control. [https://github.com/mohakbhardwaj/franka\\_motion\\_control](https://github.com/mohakbhardwaj/franka_motion_control), 2024.