## A Benchmark Details

### A.1 Task Setup

**Robots and Cameras.** The robot is placed at the front of the scene Figure 9, with +x axis facing forward. The base position is determined by the positions of the support surfaces and the scene type category. For example, for DoubleDoorCabineet type, we place the robot [0.2, 0.5]m beneath the lower deck, to mimic cases when the robot needs to fetch items from the cupboard in the kitchen. The camera positions are also sampled based on the supports. To be specific, we use simple heuristic to set position and look-at position [27], to ensure that the scene objects (shelves, tables) and all surfaces are visible in the field of view. This prevents cases where the robot collide into objects that are completely out of the view. However, we should emphasize that occlusion, e.g., shelf boards, baskets, is still a common challenge and primary cause for failure and collision.

**Initialization.** The robot is initialized with a default joint state. All objects are placed at the pose specified by the configuration file, which is stored after the filtering in Section 4.1.

**Evaluation.** After the algorithm terminates, we wait for an extra 10s for the scene to be stabilized. Then, the grasping is considered successful if the following criterion are satisfied:

1. The center of mass of the target object has a z-value $> 0.3$m in the robot-base frame.
2. The center of mass of the target object has a x-value $< 0.0$m in the robot-base frame. (x+ points the front of the robot)
3. The center of mass of all other objects in the scene has a movement $< 0.1$m from its initialization.

Here, these thresholds are carefully tuned to prevent outlier scenarios.

### A.2 Procedural Scenes

We create 13 types of procedural scenes. Figure 8 shows some examples of the scenes. For the EketShelf, we randomize the size of the cells and its placement on the wall. For the LargeShelf, we randomize the height, width, depth of the shelf, the height of each cell, the basket geometry and size, and the placement of the baskets on each layer of the shelf. For the TriangleShelf, we randomize the board, leg thickness and placement, the board contour geometry, the gap between boards and the

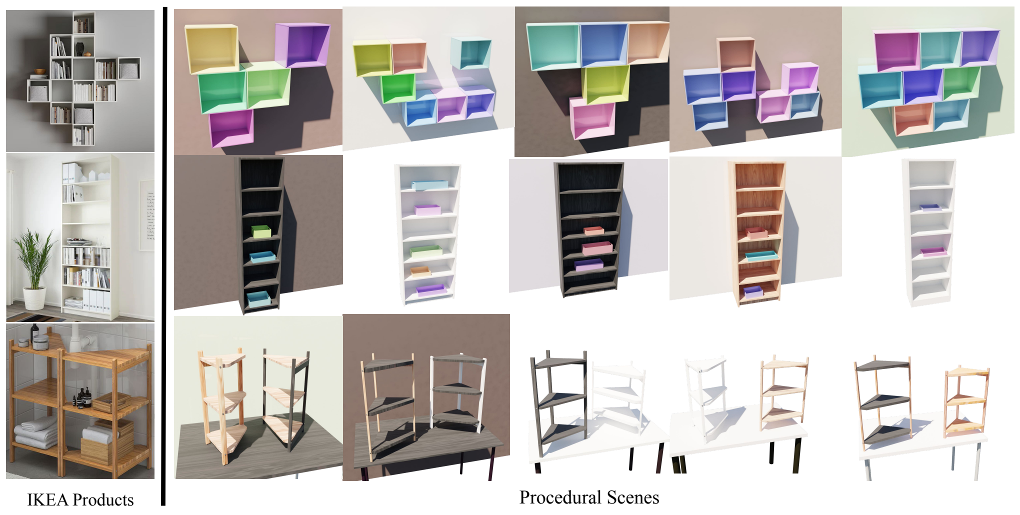

Figure 8: Examples of procedural scenes and their IKEA counterparts. The procedural scenes are rendered with Infinigen [10]. The first row is the EketShelf type, the second row is the LargeShelf type, and the last row is the TriangleShelf type.

Table 4: Statistics of different types of procedural scenes of the testing tasks.

| | # Scenes | # Tasks | # On-Table | # On-Shelf | # In-Drawer | # In-Basket |
|---|---|---|---|---|---|---|
| CellShelfDesk | 10 | 600 | 265 | 235 | 0 | 100 |
| Desk | 6 | 360 | 360 | 0 | 0 | 0 |
| DeskWall | 7 | 420 | 255 | 165 | 0 | 0 |
| DoubleDoorCabinet | 6 | 360 | 0 | 353 | 0 | 7 |
| Drawer | 6 | 0 | 0 | 0 | 360 | 0 |
| DrawerShelf | 10 | 600 | 0 | 48 | 499 | 53 |
| EketShelf | 5 | 300 | 0 | 300 | 0 | 0 |
| LargeShelf | 10 | 600 | 0 | 481 | 0 | 119 |
| LargeShelfDesk | 9 | 540 | 208 | 266 | 0 | 66 |
| LayerShelf | 10 | 600 | 0 | 362 | 0 | 238 |
| RoundTable | 9 | 540 | 237 | 0 | 0 | 303 |
| SingleDoorCabinetDesk | 7 | 420 | 201 | 214 | 0 | 5 |
| TriangleShelfDesk | 5 | 300 | 0 | 300 | 0 | 0 |
| Total | 100 | 6000 | 1526 | 2724 | 891 | 859 |

size and height of the table-top and the position and rotation of 2 shelves. In Table 4, we show all types of the scenes and statistics in the 6k testing tasks.

### A.3 Task Examples

In Figure 9, we show the examples of our evaluation task from the view of one of the camera, with the Franka robot in place.

## B Baseline Implementations

### B.1 Sense-Plan-Act

For all sense-plan-act methods, i.e, CGN-CuRobo, CGN-RRTConnect, CGN-Cabinet, the CGN takes the point-cloud and the target-object segmentation mask as input, and outputs the candidate grasp poses. The pre-grasp pose is 4cm retracted along the approach direction [32] and the post-grasp pose is 2cm lifted from the grasp pose. These offset values are carefully tuned on held-out validation tasks. For CuRobo [13] and RRTConnect [42] algorithms, the scene point-cloud is converted to mesh with marching cube algorithm with a voxel size of 5mm. For Cabinet [9], the input points to the collision checker and waypoint predictor are first moved to the origin based on the position of the target object. For CGN-CuRobo, we use sample-surface approximation for the partially observed object in the retrieval phase (when it is attached to the end-effector). For CGN-RRTConnect, we use PyBullet and Trimesh for mesh-based collision checking.

### B.2 Imitation Learning

For our imitation learning models, we use PointNet++ [44] as the encoder on point cloud of the target object, the scene and the robot separately. Furthermore, we encode the proprioceptive states with a MLP encoder. We concatenate all embeddings, i.e., target object point cloud embedding, scene point cloud embedding, robot point cloud embedding and the proprioceptive state embedding. Then, we project it to a fixed size as the final embedding to feed into the policy network.

For the MLP variant, we use the embedding of current step and use a 3-layer MLP to output the delta joint movement. The output is parameterized as a Tanh Gaussian distribution. For the Transformer variant, we use the consecutive last 4 steps' embeddings to feed into an Optimus policy head [45] and output a Tanh Gaussian mixture distribution for the joint movement.

Additionally, we downsample the input point cloud with farthest point sampling and the delta joint commands are re-scaled to fit the Tanh distribution. For E2EImit methods, we train a approach

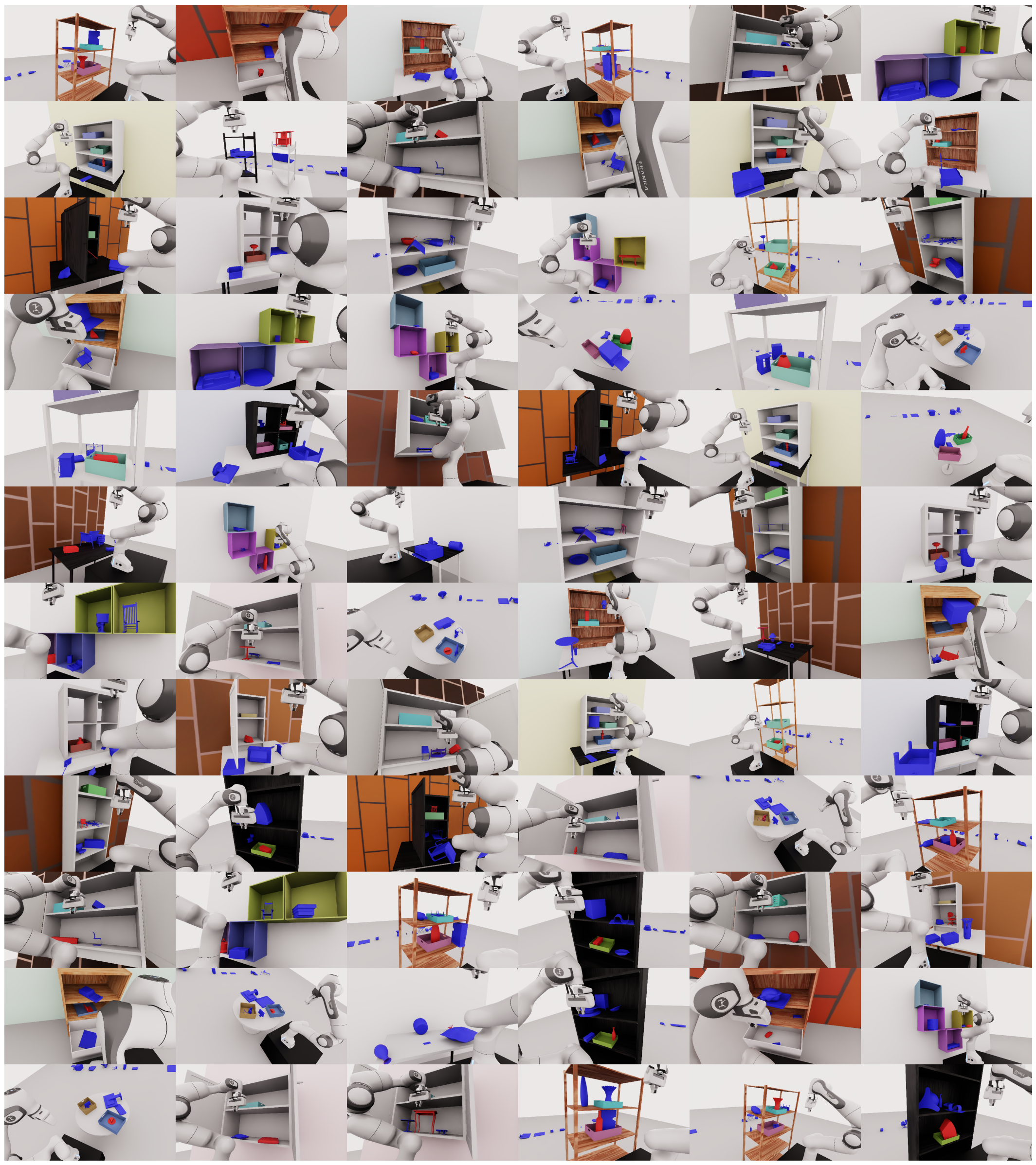
Figure 9: Examples of grasping tasks from one of the camera view when the robot is in place. The scenes are rendered with Isaac-Sim [8].

phase model and a retrieval phase model separately. The retrieval phase model is also used for the CGN-CuRobo-Imit methods. All models are trained on our demonstration dataset with Adam [46] optimizer. We select the best hyper-parameters by validating on held-out validation tasks.

## C Real Robot Experiment

### C.1 Experiment Setup

We demonstrate that robot fetching from complex environment is a challenging problem in practice. Here, we implement the CGN-RRTConnect baseline and test on a diverse set of objects and environments in order to compare performance between different scenes.

**Hardware and Evaluation Setup.** For our hardware, we used the 7-DOF Franka Emika Research 3 as our robot arm, and a Intel Realsense L515 LiDAR camera to capture the RGB-D images of

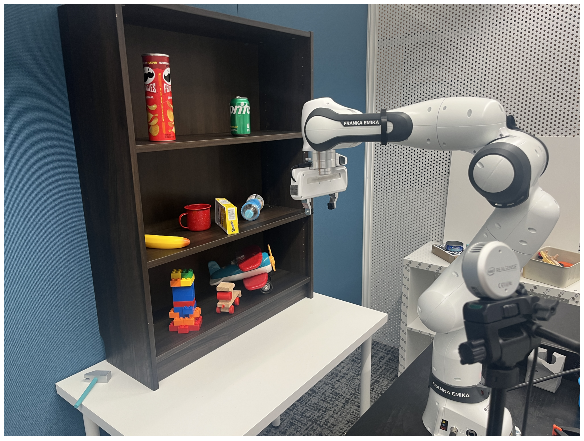

Figure 10: Hardware setup of our real world experiment. The Franka Emika Research 3 is placed in front of the scene and we use Realsense L515 pointing towards the scene to capture RGB-D information.

the scene [1]. In order to evaluate on a diverse set of environments, we performed experiments on a tabletop, two distinct shelves and various baskets. Figure 10 shows our hardware setup and the example scenes are shown in Figure 7.

We tested on a diverse set of objects in our experiment. Figure 11 shows examples of the objects used in our experiments.

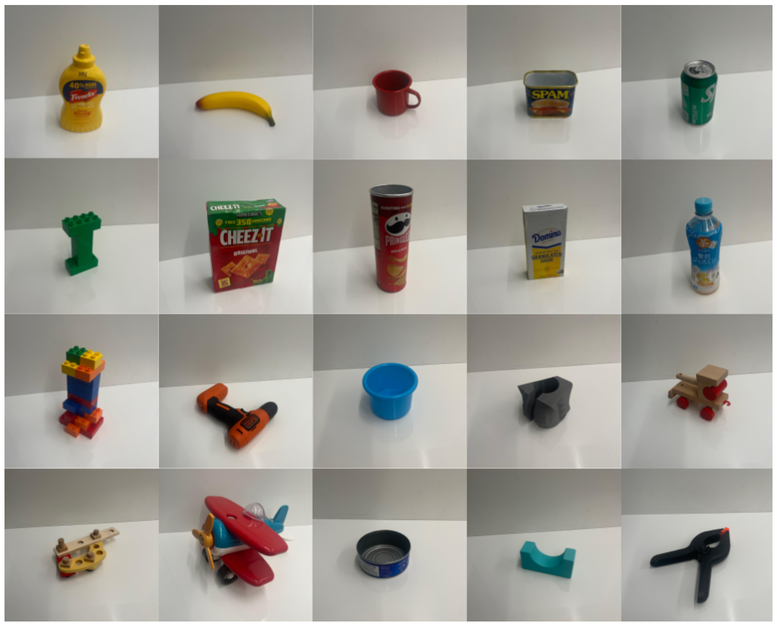

Figure 11: Examples of objects used in our real world experiments.

**Algorithm.** With our real-world robot, the CGN-RRTConnect baseline is implemented as follows.

1. Capture the RGB-D image of the scene and get the target object segmentation mask by prompting Seg-Any.
2. Use the point-cloud and the mask to query CGN for candidate grasp poses on the target object.
3. Use MoveIt! [12] to search for motion to the pre-grasp poses based on the confidence scores.
4. Move from pre-grasp pose to grasp pose with linear motion from pilz's LIN planner [12].

5. Crop out the object from the point-cloud, add a placeholder to the robot, and plan the motion to the initial pose.

### C.2 Experiment Results

Table 5 shows the detailed results of our real-world experiment. Comparing between table-top and shelf cases, we see the % success significantly decreases in shelf scenes due to the increased difficulty of the environment. In addition, we also show that grabbing objects from baskets within each respective scene also results in lower % success due to physical constraints.

| | # Success | # Attempts | % Success |
|---|---|---|---|
| TableTop | 20 | 52 | 38.5 % |
| TableTop-Basket | 3 | 16 | 18.8 % |
| Shelf | 11 | 83 | 13.3 % |
| Shelf-Basket | 1 | 16 | 6.25% |
| Total | 35 | 167 | 21.0 % |

Table 5: Results of real-world fetching experiment by environment types.

**Failure Analysis.** To further understand the behavior of the CGN-RRTConnect baseline, we broke down each unsuccessful attempt into four categories of failure:

1. No Grasp Poses (NGP), where CGN failed to find any grasps.
2. Motion Planning Failure (MPF), where CGN returned grasps, but RRT-Connect found no valid trajectories.
3. Invalid Grasp Pose (INV), where the robot attempts to grab the object, but the grasp is unstable.
4. Collision Failure (CF), where the robot or target object collides/disrupts the environment.

| | % Success | % NGP | % MPF | % INV | % CF |
|---|---|---|---|---|---|
| TableTop | 38.5 | 13.5 | 0 | 36.5 | 11.5 |
| TableTop-Basket | 18.8 | 31.3 | 18.8 | 25.0 | 6.25 |
| Shelf | 13.3 | 15.7 | 31.3 | 12.1 | 27.7 |
| Shelf-Basket | 6.25 | 37.5 | 43.8 | 6.3 | 6.3 |
| Total | 21.0 | 18.6 | 21.6 | 20.4 | 18.6 |

Table 6: Results of different failure types in each type of environment.

Table 6 shows the failure types of all attempts in each environment category. Comparing table top scenes to shelf scenes, shelves have a much higher failure rate due to a higher number of motion planning and collisions. This is attributed to the difficulty of motion planning within a shelf scene. In addition, basket scenes had a higher rate of NGP failure compared to their normal counterparts, as the baskets produce occlusions that limit the pointcloud input into CGN. We also found that many of the perspectives in shelf and basket scenes caused CGN to output very few + low confidence grasp propositions, leading us to believe that CGN is sensitive to perspective and produces better results on table top scenes.

**Limitations.** Due to limited resources, we can only evaluate on a smaller number of fetching cases, comparing to the simulation benchmark.

## D Video Demos

In the appendix, we show the examples of success and failure cases of the baselines, both in simulation benchmark and real-world experiments. Additionally, we show the examples of the issues mentioned in the ablation analysis (Section 6.2).

## E Additional Related Works

**Grasp Pose Prediction.** Predicting grasp poses for various novel objects has been an important long-standing research challenge in robotics [47]. It becomes more challenging to predict accurate and diverse grasp poses from noisy and partially observed sensory readings like depth maps and partial point clouds [1, 5]. Recently, learning-based grasp pose synthesis has become a crucial solution paradigm to this problem [32], owing to the power of neural networks to handle high-dimensional inputs and flexibility to various objects with different geometries and physical properties. To train the neural network, methods create and utilize large-scale object grasp pose datasets [40, 2, 48]. The valid grasp poses of each object are labeled with analytic metrics [3, 5, 2], or with physics simulation [34, 1]. Notably, researchers have applied the grasp pose prediction models to build robust grasping systems that can clear various unknown objects from cluttered bins [5] and table-tops [1].

Grasp pose prediction models play a crucial role in the baselines we tested. We use the Contact-GraspNet [1] to predict 6D grasp poses from partial point clouds to command the motion generation module. However, as we will show in Section 6.2, the model is not powerful enough to solve the benchmark. Furthermore, having an accurate grasp pose is not the only bottleneck challenge to grasping objects from more challenging cluttered environments, e.g., shelves, cabinets, and drawers, that are crucial for applications like service robots [7].

**Motion Generation for Robot Arm.** To generate collision-free trajectories is one of the fundamental problems for robot arm control. Given the obstacles of the environment, sampling-based motion planning algorithms like RRT [49] and its variants [42] are the common choice to command the arm to the target pose [12]. Meanwhile, optimization-based motion generation methods [13] have been proposed as an alternative to the classical approach. CuRobo [13] has much higher efficiency than the sampling-based motion planning algorithms, as it can be computed in parallel on GPUs. However, these methods assume a known environment and obstacles. They do not account for the partial observation problem, which is a common challenge for in-the-wild applications like home robots. To overcome this issue, [9, 50] propose to learn a neural collision checker for partially observed scenes and movable objects. After training on large-scale synthetic data, these models have shown promising results in tackling the partial observation problem in robot object rearrangement tasks.

However, in Section 6.2, we find that the partial observation problem still remains a challenge to existing motion generation methods in grasping. This suggests huge space for improvement in the motion generation methods to tackle real-world challenges.

**Imitation Learning.** Imitation learning [51, 52, 45] has become a promising approach to learning large-scale behavior models [35, 36] for robots. However, despite great effort from the community, researchers still lack enough data to train powerful large behavior models. Moreover, a majority of the expert data [35, 53, 54, 55] are collected on table-top environments and lack diversity in scene variation. With our procedural scenes and tasks, our simulation benchmark can generate a large quantity of diverse grasping demonstrations in various environments. Thus, our benchmark also serves as a platform for imitation learning research of large behavior models on grasping. We provide a procedural demonstration synthetic data generator for diverse and abundant expert demos. Furthermore, in Section 6.1, we find that combining imitation learning and the common grasping pipeline achieves SOTA performance on our benchmark, which suggests a promising direction for new grasping systems.