

Appendix

A	UniVoxGen	15
A.1	Scene Generation Method	15
A.2	Scene Generation Result	16
B	Oracle Policy Training	16
B.1	Reward Function	16
B.2	Training Randomization	17
B.3	Curriculum Learning Methods	18
B.4	The Architecture of Oracle Policy	18
C	Vision Policy Training	19
C.1	Ablation Study on Scaling the Training Data	19
C.2	Ablation Study on Region of Interest (ROI) Size	20
C.3	Training Details	20
C.4	The Details of 2D-3D Spatial Attention	20
C.5	Processing of Relative Depth	21
C.6	Real-World Reconstruction Results	21
D	Baseline Implementation Details	22
E	Real-Wold Experiment Details	23
E.1	Setup	23
E.2	Real-World System Design	24
E.3	Experiment objects	24
F	Failure Modes Analysis	25

A UniVoxGen

A.1 Scene Generation Method

A diverse and realistic set of scenes is crucial for sim-to-real transfer, requiring an effective scene generation method. Creating a valid scene involves determining the pose for each object and ensuring there is no penetration among any of them. Therefore, during scene generation, we need to check for collisions between different objects. Previous approaches [6, 3, 8, 4, 10, 7] for generating cluttered scenes typically depend on computationally intensive collision detection mechanisms. These methods require importing objects into a simulator and executing a simulation step to detect potential collisions, a process known to be computationally expensive. This inefficiency is particularly pronounced in cluttered scenes, where the dense packing of objects inherently leads to frequent collisions requiring verification. Additionally, some other methods create cluttered layouts by simulating objects dropped from the air, which may result in unstable and unrealistic scene configurations. Our method, UniVoxGen, is specifically designed for fast and realistic scene generation in voxel space. It accelerates the generation process by performing efficient collision checks in voxel space and produces realistic scene layouts using a set of carefully crafted hand-designed rules.

We begin by providing formal definitions for key elements in the voxel space. Let $V^o = \{V_1^o, V_2^o, \dots, V_N^o\}$ represent the voxel representation of a set of objects, and $V^s = \{V_1^s, V_2^s, \dots, V_N^s\}$ represent the voxel representation of the scene. We define a set of operational primitives in voxel space for manipulating voxels. Specifically, $V_i \cup V_j$ denotes the union operation, which combines two voxel sets and is commonly used to add an object’s voxels into the scene. $V_i \cap V_j$ denotes the intersection operation, which retrieves the intersection of two voxel sets and is used to detect potential collisions when adding a new object. $V_i - V_j$ denotes the difference operation, which removes the overlapping portion of V_i with V_j , typically used to remove an object from the scene. Finally, $T(V_i, P), P \in SE(3)$ represents a transformation of a voxel V_i in $SE(3)$ space, commonly used to change the pose of the object. Here, P is a transformation matrix in $SE(3)$ that combines rotation and translation.

Algorithm 1 Scene Generation Algorithm

Require: Number of scenes N

Require: Max objects per scene K

Require: A set of objects \mathcal{O}

```
1: for scene 1 :  $N$  do
2:   Initialize scene voxel  $V^s = \{\}$ 
3:   Sample a target object  $O^{tar}$ 
4:   Sample a pose  $P$  in  $SE(3)$  for  $O^{tar}$ 
5:   Apply  $T(V_i, P)$  to transform the target object
6:   Apply  $V_{O^{tar}} \cup V^s$  to add the target object to the scene
7:   Sample number of obstacle objects  $k \sim [1, \dots, K]$ 
8:   for obstacle  $O_i^{obs}$  1 :  $k$  do
9:     Sample a pose  $P$  in  $SE(3)$  for  $O_i^{obs}$ 
10:    Apply  $T(V_i, P)$  to transform the obstacle object
11:    while  $V_{O_i^{obs}} \cap V^s$  do
12:      Sample a new pose  $P$  in  $SE(3)$  for  $O_i^{obs}$ 
13:      Apply  $T(V_i, P)$  to transform the obstacle object
14:    end while
15:    Apply  $V_{O_i^{obs}} \cup V^s$  to add the obstacle object to the scene
16:  end for
17:  Save the pose  $P$  of each object in the scene
18: end for
```

Based on the previously defined key elements and operation primitives, we further design a set of generation rules $R = \{R_1, R_2, \dots, R_N\}$. UniVoxGen uses these rules to generate diverse cluttered scenes. These scenes may include unsolvable cases, where it is impossible to retrieve the target

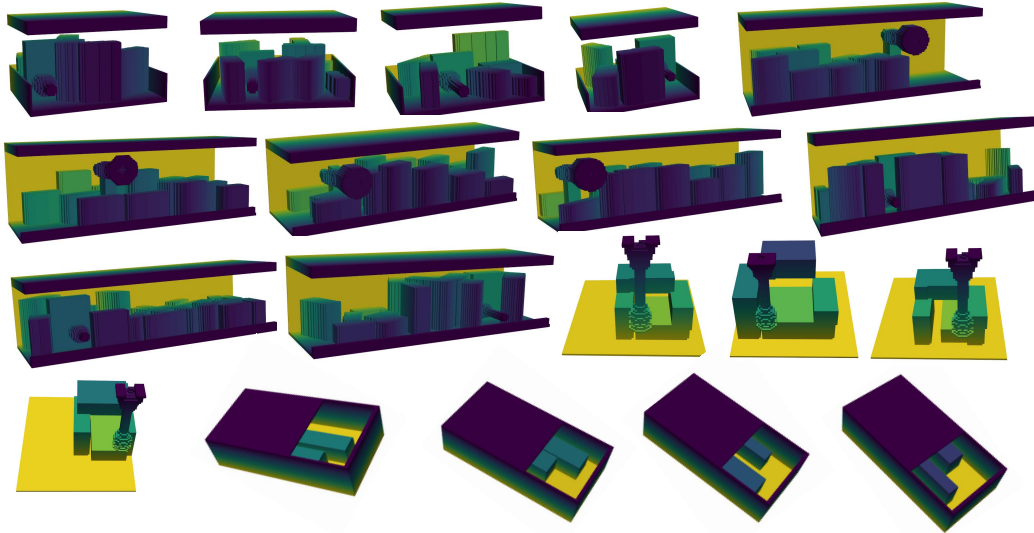


Figure 10: Generated cluttered scenes by UniVoxGen, including shelf, tabletop, drawer, and storage rack environments.

object without colliding with any obstacles. It is worth noting that the inclusion of unsolvable cases is intended to better simulate real-world scenarios, as such situations can occur in practice. The procedure for generating these cluttered scenes is outlined in Algorithm 1. It should be noted that, given the complexity of the various scene generation rules, the steps presented here represent a simplified version of our scene generation rules. The detailed generation rules will be made available in the released source code. Finally, we use UniVoxGen to generate **1 million** cluttered scenes, which are then utilized as training scene data for an oracle policy. It takes 12 hours on the workstation equipped with 8 RTX 4090s to generate these 1 million scenes.

A.2 Scene Generation Result

We utilize UniVoxGen to generate a large number of cluttered environments. Our initial focus is on cluttered shelf environments. Such environments require not only clutter but also a certain degree of structured arrangement to be realistic. On a shelf, objects cannot be arbitrarily piled together, but instead need to exhibit some regularity in their placement to simulate realistic shelf settings. Additionally, we generate cluttered environments for tabletop, drawer, and rack settings to support the needs of extension experiments, as shown in Fig. 10.

B Oracle Policy Training

B.1 Reward Function

Our reward function combines task completion r_{task} , behavioral constraints r_{cons} , and an environment change penalty r_{env} .

Task Reward. The task reward formulation imposes a dual requirement: the target object must be successfully fetched, and the resulting environmental impact, measured over the entire episode, must be limited. This impact is quantified by the total translation (m_{trans}) and total rotation (m_{rot}) accumulated across all obstacles during the episode. Both metrics must remain below their respective predefined thresholds: m_{trans} must be less than the translation threshold σ_{trans} , and m_{rot} must be less than the rotation threshold σ_{rot} . We use a curriculum learning method for the thresholds (σ_{trans} and σ_{rot}), gradually decreasing their values during training. Specifically, σ_{trans} is set according to the sequence [0.03, 0.015, 0.01, 0.005, 0.0] m, and σ_{rot} follows the sequence [0.4, 0.2, 0.16, 0.1,

0.0] radians. The transition to the next value in each sequence occurs once the policy’s performance saturate at the current stage.

Behavioral Constraints Reward. We apply certain behavioral constraints to the flying end-effector to make its behavior more naturalistic.

We use r_{action_range} to constrain the policy’s output from becoming too large. That is:

$$r_{action_range} = \begin{cases} -\lambda_{action_range} \exp((|a| - 3)^2) & \text{if } |a| > 3 \\ 0 & \text{if } |a| \leq 3 \end{cases}$$

To ensure smoother and more continuous actions, we introduce a term, r_{action_rate} , that penalizes large differences between consecutive pose outputs of the policy ($a_{last}, a_{current}$):

$$r_{action_rate} = -\lambda_{action_rate} \exp((a_{last} - a_{current})^2)$$

To prevent fetching the object in singular poses, which could violate robot arm joint limits, we incentivize maintaining the target object close to its initial pose (p_{init}). This objective is achieved using the r_{pose} penalty term, calculated as:

$$r_{pose} = -\lambda_{pose} \exp(p_{curr} - p_{init})$$

To prevent collisions with shelf barriers, we introduce a penalty term, $r_{penetration}$. If a collision, i.e., penetration, occurs, the target object experiences a significant acceleration, denoted as a_{ccel} . This acceleration is then used to compute the penalty as follows:

$$r_{penetration} = -\lambda_{pen} \exp(a_{ccel}^2)$$

To encourage moving the target towards its designated goal position, we employ a reward component, r_{target_move} , to guide the policy’s behavior. This reward incentivizes movement towards the goal and penalizes movement away from it. Let d_{last} denote the distance from the target to the goal position at the previous timestep, and d_{curr} be the current distance to the goal. The term $d_{last} - d_{curr}$ thus represents the progress made towards the goal in the current step, a positive value indicates movement closer to the goal. We use v represents target’s velocity and dt is the duration of the timestep, the reward is defined as:

$$r_{target_move} = \lambda_{target_move} \cdot \frac{d_{last} - d_{curr}}{v \cdot dt}$$

Environment Change Reward. To incentivize the policy to minimize its impact on the environment, we introduce penalties based on the total displacement of all obstacles within each timestep. Specifically, we consider the total translational displacement (m_{trans_step}) and the total rotational displacement (m_{rot_step}) of all obstacles. These penalties are computed as follows:

$$r_{trans_step} = -\lambda_{trans_step} \exp(m_{trans_step}^2)$$

$$r_{rot_step} = -\lambda_{rot_step} \exp(m_{rot_step}^2)$$

All weight coefficients are listed in Table 4. We train our oracle policy with PPO [5], and the training hyper-parameters are shown in Table 3.

B.2 Training Randomization

To improve the robustness of the oracle policy and diversify the demonstration set, we apply extensive domain randomization during training. We randomize the objects’ mass and center of mass, friction coefficients, the shelf’s upper-barrier height, camera pose, and the end-effector’s initial pose; in addition, we inject noise into observations to emulate real-world imperfections such as calibration bias and readout noise.

Hyper-parameters	Values
<i>Num. envs</i>	1024
<i>Num. steps for per update</i>	24
<i>Num. minibatches</i>	4
<i>Num. learning epochs</i>	1500
<i>learning rate</i>	0.0003
<i>clip range</i>	0.2
<i>entropy coefficient</i>	0.0
<i>kl threshold</i>	0.02
<i>max gradient norm</i>	1.0
λ	0.95
γ	0.99

Table 3: Hyper-parameters for the oracle policy learning.

Hyper-parameters	Values
λ_{task}	5.0
λ_{action_range}	1.5
λ_{action_rate}	0.0001
λ_{pose}	0.6
$\lambda_{penetration}$	9.0
λ_{target_move}	0.06
λ_{trans_step}	10.0
λ_{rot_step}	10.0

Table 4: Hyper-parameters for the reward function.

B.3 Curriculum Learning Methods

We explore various curriculum learning strategies. The first step is to assign difficulty levels to the task scenarios. Specifically, we generate a diverse set of scenarios based on predefined rules, and then categorize them into five difficulty levels according to the occlusion rate—defined as the percentage of the target object’s surface area occluded by obstacles when viewed from the front.

Approach 1. *Per-Environment Difficulty Curriculum.* ✗ We adopt a locomotion-style curriculum [13] as our first attempt: in Isaac Gym, each environment is assigned a difficulty level; upon task success its difficulty increases, otherwise it decreases. However, this scheme does not yield any improvement over direct training on all scenarios.

Approach 2. *Unified Difficulty Curriculum.* ✗ Our second attempt departs from the locomotion-style curriculum. In Isaac Gym, all parallel environments are kept at the same difficulty level, rather than assigning each environment its own difficulty. Once the policy converges at the current difficulty, we advance all environments to the next level. However, this method also fails to produce satisfactory results.

Approach 3. *Policy-Driven Difficulty Curriculum.* ✗ Our third attempt abandons the use of manually defined metrics (i.e., occlusion rate) for defining difficulty levels, rather than relying on the policy’s own performance. Specifically, we first train the policy on all scenarios until convergence, then evaluate it and label the failed scenarios as higher-difficulty cases. Training then continues exclusively on these failed scenarios, and this process is repeated for five iterations. However, this method also fails to deliver satisfactory results.

Approach 4. *Task-Conditioned Curriculum.* ✓ Our final method is a σ -based curriculum (§ 3.3): σ is large early for exploration and is gradually reduced to enhance precision and stability.

Our key insight is that, whether difficulty is defined manually or in a policy-driven way, the chosen scene difficulty axis is highly homogeneous and lacks qualitative transitions across levels; knowledge learned in easy scenarios transfers poorly to hard ones, failing to provide effective learning gradients or a skill ladder. Unlike scene-based curricula, the σ -based curriculum decouples training from scene parameters and schedules solely based on task difficulty via σ , thereby avoiding scene-difficulty specification and directly improving task performance and training stability.

B.4 The Architecture of Oracle Policy

Encoder. The oracle policy receives an observation composed of three parts: the scene representation z_t , proprioception p_t , and the previous action a_{t-1} . The term $p_t \in \mathbb{R}^{12}$ encodes the end-effector pose relative to the target placement and decomposes as $p_t = [\text{rot}_t, \text{trans}_t]$, where

$\text{rot}_t \in \mathbb{R}^9$ denotes the rotation matrix (flattened to 9 entries) and $\text{trans}_t \in \mathbb{R}^3$ is the translation vector. The previous action satisfies $\mathbf{a}_{t-1} \in \mathbb{R}^6$. The scene encoder network (§ 3.3) maps the raw scene input to $\mathbf{z}_t \in \mathbb{R}^{64}$. The final observation is the concatenation $\mathbf{o}_t = [\mathbf{z}_t, \mathbf{p}_t, \mathbf{a}_{t-1}] \in \mathbb{R}^{82}$.

Decision. The decision module is a three-layer MLP that maps the observation to an action. Formally, the policy π implements $\pi : \mathbb{R}^{82} \rightarrow \mathbb{R}^6$ and $\mathbf{a}_t = \pi(\mathbf{o}_t)$, where $\mathbf{a}_t \in \mathbb{R}^6$.

C Vision Policy Training

C.1 Ablation Study on Scaling the Training Data

In our ablation study on scaling the training data, we examine its impact on both the occupancy pre-training (Fig. 11) and the policy learning (Fig. 12). We use suction cup as the end-effector tool to complete this experiment.

Scaling the Training Data for Occupancy Pre-Training. This study examines how the amount of data utilized for pre-training the 3D vision encoder via an occupancy prediction task affects the ultimate performance of the model. All experiments employ varying dataset sizes for pre-training the 3D vision encoder but maintain a fixed dataset size of 500K for policy training. We find a clear correlation between larger datasets and improved policy performance. Starting with 500 scenes, the success rate is 62.33%. As the dataset size increases to 5k and 50k, the success rate improves to 70.72% and 72.50%, respectively. As the dataset size increases to 100k and 250k, the performance continues to improve. The largest dataset, with 500k scenes, achieves the best performance, reaching a success rate of 81.46%. This demonstrates that pre-training on larger datasets significantly enhances the policy performance, providing a more comprehensive understanding of the 3D scene.

Scaling the Training Data for Policy Learning. We seek to determine how the quantity of data available for policy learning influences the model’s ultimate performance. For this investigation, all experiments maintain a fixed data volume for 3D vision encoder pre-training but use different amounts of data for training the policy. With a dataset of 500 state-action pairs, the success rate is 48.23%. As the dataset increases to 5k and 50k, the success rate rises to 54.66% and 65.55%, respectively. As the dataset size increases to 100k and 250k, performance continues to improve. The largest dataset of 500k state-action pairs yields the best performance, achieving a success rate of 81.46%. These results highlight that increasing the training data size for the decision module significantly improves task success, emphasizing the importance of a sufficiently large dataset for accurate and reliable policy performance.

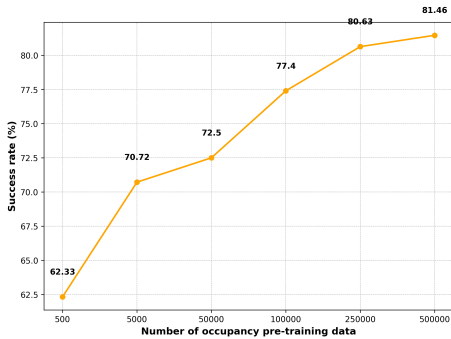


Figure 11: Occ Pre-Training Scaling Law

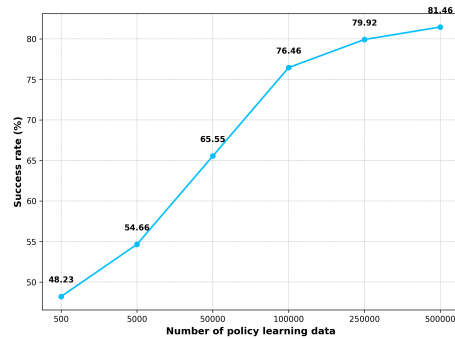


Figure 12: Policy Learning Scaling Law

Overall, our findings highlight the critical role of large-scale data in both occupancy pre-training and policy learning, underscoring the importance of scaling the dataset for improving the overall performance and accuracy of the 3D vision policy.

C.2 Ablation Study on Region of Interest (ROI) Size

Focusing solely on the region of interest is an effective approach that enhances the policy’s generalization ability and aids policy learning. We define our Region of Interest (ROI) by cropping an $H \times W \times Z$ cm cubic space around the end-effector.

In our ablation study on the ROI size (Table. 5), we investigated the impact of varying sizes on policy performance. The results show that a compact ROI, specifically $20 \times 20 \times 30$ cm, leads to the best performance, achieving the highest success rate (81.46%) and the lowest translation (2.78 cm) and rotation (0.36 rad) errors. It is crucial to note that *the optimal ROI size is not arbitrarily small but is closely tied to the scale of the target objects*. In our experiments, the target objects average approximately $5 \times 10 \times 12$ cm. Therefore, an ROI of $20 \times 20 \times 30$ cm is large enough to fully encompass the target while providing essential local context, yet small enough to filter out most distracting information. As the ROI size increases beyond this optimal range, both the success rate and accuracy metrics decrease. This is because larger ROIs introduce more irrelevant information from distant areas, which reduces the model’s ability to focus on the target region and leads to less precise predictions.

These findings highlight the advantage of using a *task-appropriate, localized ROI*. This approach not only improves policy performance but also enhances generalization, making the network more robust to scene variations. It also accelerates policy inference while maintaining high performance, especially in cluttered environments.

ROI Size (cm)	Success Rate (%) \uparrow	Translation (cm) \downarrow	Rotation (rad) \downarrow
$20 \times 20 \times 30$	81.46%	2.78	0.36
$40 \times 80 \times 30$	76.93%	3.24	0.43
$60 \times 150 \times 30$	74.63%	3.67	0.47

Table 5: **Ablation study on Region of Interest (ROI) size.** A compact ROI of $20 \times 20 \times 30$ cm achieves the best results, with the highest success rate and lowest errors. This highlights the benefit of a localized ROI that is appropriately scaled to the task, as larger ROIs degrade performance by including irrelevant information from distant areas.

C.3 Training Details

Pre-training for occupancy prediction used 40 GeForce RTX 4090 GPUs on 500K scenes and finished in 16 hours. Policy learning then trained on 500K demonstration frames for 8 hours. In real-world tests, the system runs at over 10 fps on a single NVIDIA GeForce RTX 4090.

C.4 The Details of 2D-3D Spatial Attention

Rather than averaging multi-view features (which assumes equal contribution), we fuse them using Deformable Cross-Attention (DCA) [12], guided by a grid of 3D queries $Q \in \mathbb{R}^{C \times H \times W \times Z}$.

As shown in Fig. 7, for each 3D query, defined by its position p and feature q_p , we project it onto each camera view t via $\mathcal{P}(p, t) : \mathbb{R}^3 \mapsto \mathbb{R}^2$. We retain only the set of views where the projection is valid, \mathcal{V}_{hit} . The aggregated 3D feature \mathcal{F}^p at voxel p is then computed by averaging the attention outputs from all visible views:

$$\mathcal{F}^p = \text{DCA}(q_p, \{\mathcal{F}_t^{2D}\}) = \frac{1}{|\mathcal{V}_{\text{hit}}|} \sum_{t \in \mathcal{V}_{\text{hit}}} \text{DA}(q_p, \mathcal{P}(p, t), \mathcal{F}_t^{2D}),$$

where \mathcal{F}_t^{2D} is the 2D feature map of view t . The Deformable Attention (DA) module is defined as:

$$\text{DA}(q, p_{\text{ref}}, X) = \sum_{m=1}^{N_{\text{head}}} W_m \left(\sum_{k=1}^{N_{\text{key}}} A_{mk} \cdot W'_m X(p_{\text{ref}} + \Delta p_{mk}) \right).$$

Here, for each attention head m and key k , the module uses the query q to predict a 2D sampling offset Δp_{mk} relative to the reference point $p_{\text{ref}} = \mathcal{P}(p, t)$ and an attention weight A_{mk} . It then

samples the 2D feature map X at these sparse, dynamically determined locations ($p_{\text{ref}} + \Delta p_{mk}$) and computes a weighted sum. W_m and W'_m are standard learnable projection matrices.

This approach allows the final 3D feature volume, $\mathcal{F} = \{\mathcal{F}^p\}$, to unequally weight views based on visibility and content, enhancing robustness to occlusion and blur.

C.5 Processing of Relative Depth

Depth estimates from DepthAnything [9] are inherently scale-ambiguous. To create a consistent input for our feature extractor, we first normalize each predicted depth map using a per-image **min-max scaling** to bring its values into the range $[0, 1]$.

Concretely, given an RGB image $I \in \mathbb{R}^{H \times W \times 3}$, DepthAnything produces a single-channel depth map $D \in \mathbb{R}^{H \times W}$. This map is normalized as:

$$\hat{D} = \frac{D - \min(D)}{\max(D) - \min(D) + \varepsilon}$$

where $\min(D)$ and $\max(D)$ are the minimum and maximum values of the specific depth map D , and ε is a small constant for numerical stability.

To prepare this normalized map for a standard ResNet backbone, we treat \hat{D} as a grayscale image. It is first replicated across three channels to match the expected input format ($H \times W \times 3$). Then, this 3-channel tensor undergoes a second normalization using the standard ImageNet mean and standard deviation values. The resulting tensor is finally fed into the ResNet backbone (with the final max-pooling layer removed) to extract the feature map F .

C.5.1 Mapping Relative Depth to Metric Voxels

Our method does not perform an explicit conversion from relative depth to metric depth, nor does it use depth values to unproject 2D features into a 3D point cloud. Instead, the core of our approach is to project the centers of a predefined 3D metric voxel grid (with a cell size of 5 mm) onto the input camera views. This forward projection is based on the standard pinhole camera model and utilizes the provided camera matrices.

This multi-view projection strategy is key to resolving spatial ambiguity. By using multiple views, points that might lie along a single viewing ray from one camera are disambiguated through parallax, as they project to distinct 2D locations in the second view.

Consequently, the network is tasked with implicitly learning the transformation from relative depth features to a metrically-scaled occupancy grid, guided by these strong multi-view geometric constraints. The entire process is optimized end-to-end with direct supervision from the ground-truth metric voxel data. In summary, our methodology leverages the geometric accuracy of the pinhole camera model for projection and the power of end-to-end learning to fuse features from relative depth into a usable metric representation.

C.6 Real-World Reconstruction Results

We first pre-train the encoder on an occupancy-prediction auxiliary task in simulation, which yields high-quality reconstruction not only in simulation but also on real hardware (strong sim-to-real, as shown in Fig. 13). Under heavy occlusions, the model infers the occluded volume to recover a complete scene representation, and it remains robust to transparent, reflective, and irregular objects. Importantly, we do not feed the reconstructed voxels (final occupancy map) to the downstream policy. Instead, the policy receives intermediate latent features from the pre-trained encoder; although it never observes the final occupancy prediction, these auxiliary-task-enriched features allow it to implicitly capture the scene’s complete 3D geometry.

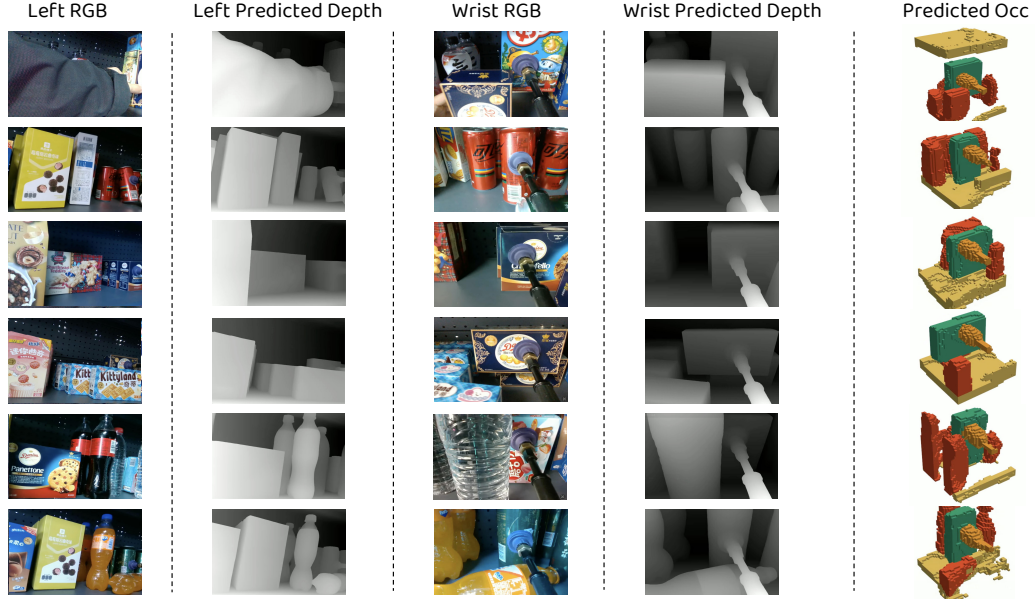


Figure 13: Real-world occupancy reconstruction results, capable of handling diverse scenarios: varying object shapes, different materials, and complex layouts.

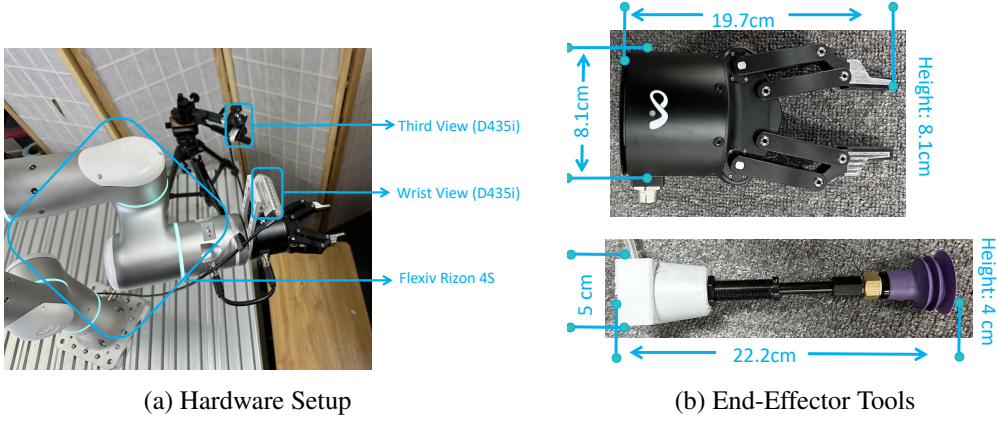


Figure 14: We illustrate our real-robot experimental hardware setup and the two types of end-effector tools employed: one suction cup and one parallel gripper.

D Baseline Implementation Details

Baselines utilizing RGB (Diffusion Policy), point cloud (3D Diffusion Policy), RGB-D voxels, raw depth, and predicted depth all share the same action generation method, differing only in their input representations. Actions are generated via a Denoising Diffusion Probabilistic Model (DDPM) [2], which utilizes 1000 denoising steps during training and 100 steps for inference, its network architecture is a three layer MLP.

RGB (Diffusion Policy) [1]. We use dual-view RGB images as input and extract features following the same processing pipeline as Diffusion Policy (i.e., through ResNet and Spatial Softmax). Subsequently, actions are generated using DDPM.

Point Cloud (3D Diffusion Policy) [11]. Our input is a point cloud generated by fusing two camera depth maps. Following the approach in DP3, we employ a Simple PointNet to extract features from this point cloud.

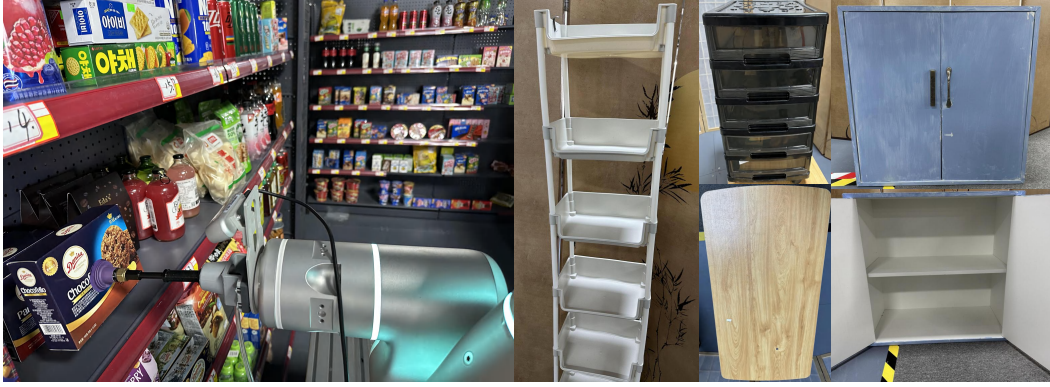


Figure 15: Illustration of experimental scenarios. The main experiments are conducted in a replicated retail store, supplemented by evaluations on a storage rack, in a cabinet, in a drawer, and on tabletops.

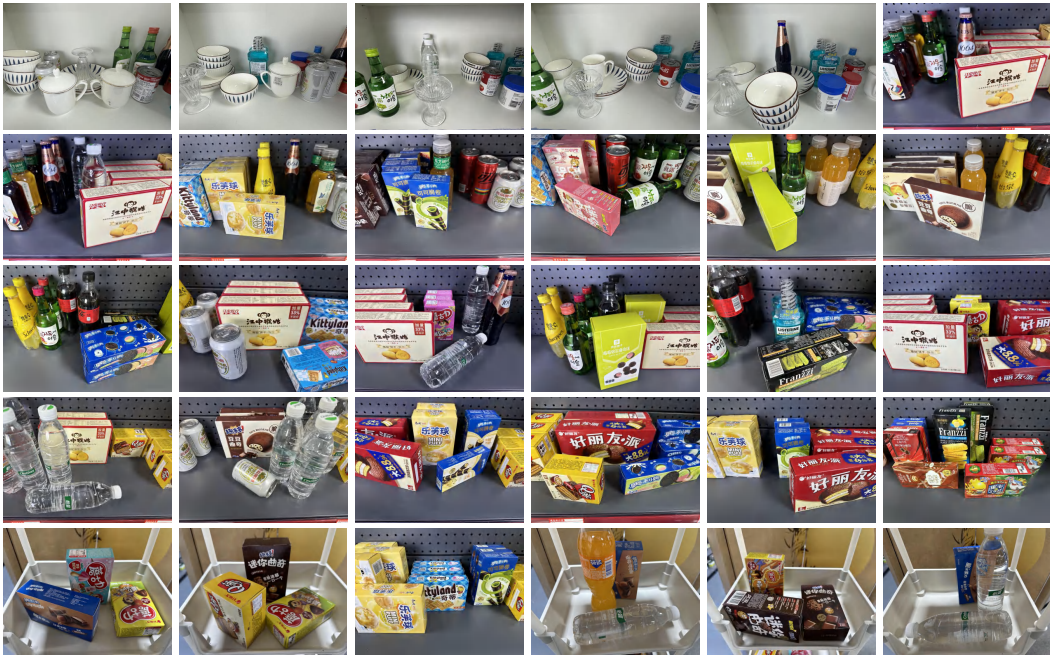


Figure 16: Layouts for real-robot experiments.

RGB-D Voxel. We project RGB images from two viewpoints into 3D space using camera extrinsics and depth maps, thereby forming a voxel representation. A 3D convolutional network (3D ConvNet) is then employed to extract features from these voxels. Then use DDPM to generate actions.

Raw Depth. We use ResNet to extract features from two viewpoint depth maps, after which actions are generated via DDPM.

Predicted Depth. We utilize Depth Anything to obtain predicted depth maps. Features are then extracted from these maps using ResNet. Subsequently, actions are generated via DDPM.

E Real-World Experiment Details

E.1 Setup

Hardware. In our real-world experiments, we use the Flexiv Rizon 4S robotic arm. Two Intel D435i cameras provide input for our 3D vision policy from different perspectives, exclusively utilizing their

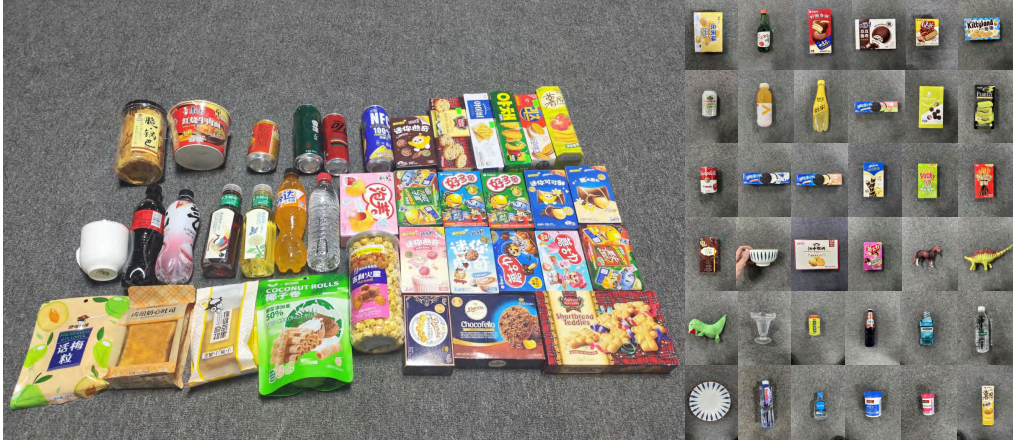


Figure 17: Objects used in real-robot experiments, including a diverse range of items with various shapes (such as boxed, bottled, and irregular forms) and materials (including transparent and reflective types).

RGB streams. One camera is mounted on the robotic arm, while the other offers a third-person view, as shown in Fig. 14 (a).

End-Effector Tools. We utilize two end-effector tools: a suction cup and a parallel gripper. The suction cup, which was custom-made by us via 3D printing, is used to handle boxed objects. The parallel gripper, Flexiv GRAV model, is employed for grasping bottled objects (illustrated in Fig. 14 (b)).

Experiment Scenes. Our main real-world experiments are primarily conducted in a replicated retail environment. Additionally, we perform experiments in settings involving a cabinet and a rack, all scenes all shown in Fig. 15. And all 30 experimental layouts are shown in Fig. 16.

E.2 Real-World System Design

Our real-world system employs a *hierarchical, asynchronous control architecture* to ensure smooth and stable robot execution. The system is decoupled into two main layers: a high-level *Policy Layer* and a low-level *Control Layer*.

The *Policy Layer* operates at a low frequency of **10 Hz**. In this layer, the policy network receives the latest sensor observations, performs inference, and generates a target end-effector (EE) pose (position and orientation). This target pose is then sent as a command to the lower-level controller.

The *Control Layer* is an *IK-based controller* that runs asynchronously at a high frequency of **100 Hz**. This layer is responsible for translating the target EE pose into a smooth trajectory of joint positions (\mathbf{q}_{pos}). It receives commands from the policy layer asynchronously and interpolates the robot’s current pose towards the target. In each control cycle, it calculates a small, kinematically valid motion step using an inverse kinematics solver, ensuring that the robot’s velocity limits are respected. This decoupling of low-frequency policy decisions from high-frequency motion control is crucial for stable and continuous real-world performance.

E.3 Experiment objects

The objects used in our experiments encompass a diverse range of types, including those that are boxed, bottled, transparent, reflective, or irregularly shaped. The experimental objects are displayed in Fig. 17.



Figure 18: Illustration of Limitations. (Top row) Actions exceeding joint limits due to an attempted approach from above. (Bottom row, left) A large-volume object demonstrating the need for dual-arm manipulation. (Bottom row, right) A completely occluded and initially unreachable target, necessitating a multi-step reasoning process involving moving the obstacle, fetching the target, and subsequently returning the obstacle to its original position.

F Failure Modes Analysis

When scenes become excessively complex, there is a possibility that the policy’s output will lead to joint limit violations for the robotic arm, causing the task to fail. The top of Fig. 18 depicts a scenario where the robot’s intended rotational approach from above to fetch an object results in exceeding its joint limits and a collision with the upper barrier. Furthermore, when an object is too large and too heavy, we need to employ dual-arm coordination, as shown in the bottom of Fig. 18. Finally, when the target is completely occluded to the point of being unreachable, reasoning capabilities are required to first move the obstacles aside, then retrieve the target, and subsequently restore the obstacles to their original positions (Fig. 18). All of these can be considered as future work.

References

- [1] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.
- [2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.
- [3] Yitong Li, Ruihai Wu, Haoran Lu, Chuanruo Ning, Yan Shen, Guanqi Zhan, and Hao Dong. Broadcasting support relations recursively from local dynamics for object retrieval in clutters. In *Robotics: Science and Systems*, 2024.
- [4] Yaoyao Qian, Xupeng Zhu, Ondrej Biza, Shuo Jiang, Linfeng Zhao, Haojie Huang, Yu Qi, and Robert Platt. Thinkgrasp: A vision-language system for strategic part grasping in clutter. *arXiv preprint arXiv:2407.11298*, 2024.
- [5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [6] Kentaro Wada, Stephen James, and Andrew J Davison. Safepicking: Learning safe object extraction via object-level mapping. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10202–10208. IEEE, 2022.
- [7] Yongliang Wang and Hamidreza Kasaei. Learning dual-arm push and grasp synergy in dense clutter. *arXiv preprint arXiv:2412.04052*, 2024.
- [8] Kechun Xu, Shuqi Zhao, Zhongxiang Zhou, Zizhang Li, Huaijin Pi, Yifeng Zhu, Yue Wang, and Rong Xiong. A joint modeling of vision-language-action for target-oriented grasping in clutter. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11597–11604. IEEE, 2023.
- [9] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *arXiv preprint arXiv:2406.09414*, 2024.
- [10] Yuxiang Yang, Jiangtao Guo, Zilong Li, Zhiwei He, and Jing Zhang. Ground4act: Leveraging visual-language model for collaborative pushing and grasping in clutter. *Image and Vision Computing*, 151:105280, 2024.
- [11] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *ICRA 2024 Workshop on 3D Visual Representations for Robot Manipulation*, 2024.
- [12] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.
- [13] Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher Atkeson, Sören Schwertfeger, Chelsea Finn, and Hang Zhao. Robot parkour learning. In *Conference on Robot Learning (CoRL)*, 2023.