

Appendices

A Additional Results

A.1 Occlusion Qualitative Results

Fig. 6 presents the qualitative examples of our occlusion experiment in the simulated environment. We show that our method, leveraging the generalizable semantic features of DINOv2, demonstrates robustness under varying levels of occlusion. This includes extreme case with up to 50% point cloud occluded which is considered the most difficult scenario in real-world applications.

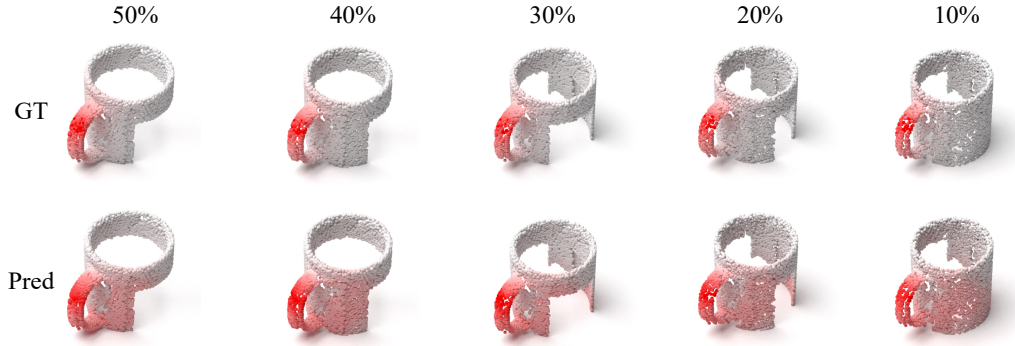


Figure 6: Qualitative examples of different occlusion level.

A.2 RoboPoint Qualitative Results

Fig. 7 shows the affordance prediction from RoboPoint [51]. While RoboPoint, as a vision-language model, demonstrates strong object localization capabilities, it lacks the precision required to infer fine-grained affordance regions on objects, as illustrated in Fig. 7. Most of its predictions successfully fall onto the area of the object, but don't onto the correct part of the object which hinders its application for more complex manipulation tasks.

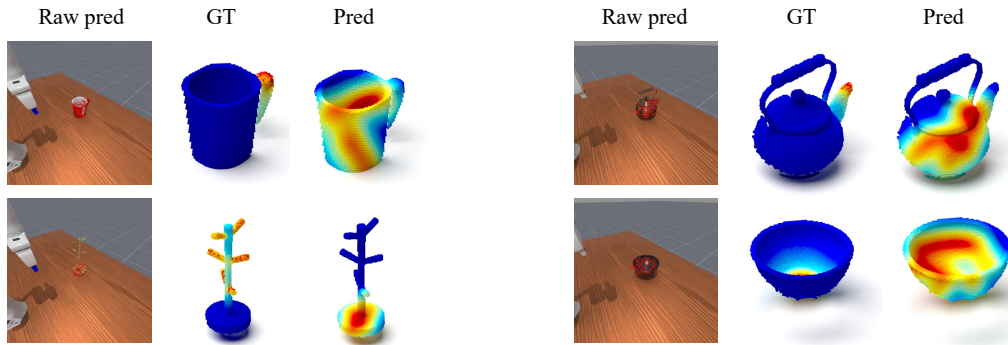


Figure 7: Qualitative examples of RoboPoint predictions.

A.3 OOAL Qualitative Results

We highlight the limitations of recent work OOAL [4], which employs vision foundation models (VFMs) for 2D affordance grounding. OOAL exhibits two main drawbacks: (1) sensitivity to view-point changes, and (2) a noticeable sim-to-real performance gap. We show the qualitative examples in Fig. 8. We evaluate OOAL on data collected in the simulation environment using multiple viewpoints. The results reveal inconsistent predictions across viewpoints and a significant performance

506 drop, despite the data is collected in the simulation. This highlights the benefit of affordance ground-
 507 ing in the 3D point cloud space, which reduces the sim-to-real gap and provides geometric awareness
 508 and robustness to viewpoint variations.

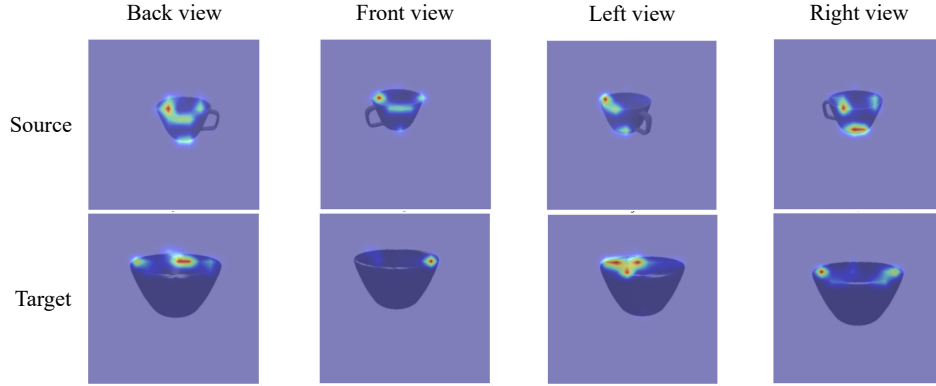


Figure 8: **Qualitative examples of OOAL predictions for four different views.** Each row renders the affordance heatmaps of the same source/target object across views.

509 A.4 Manipulation Qualitative Results

510 Fig. 9 shows the example executions of the different types of actions in the simulator using our
 511 proposed method.

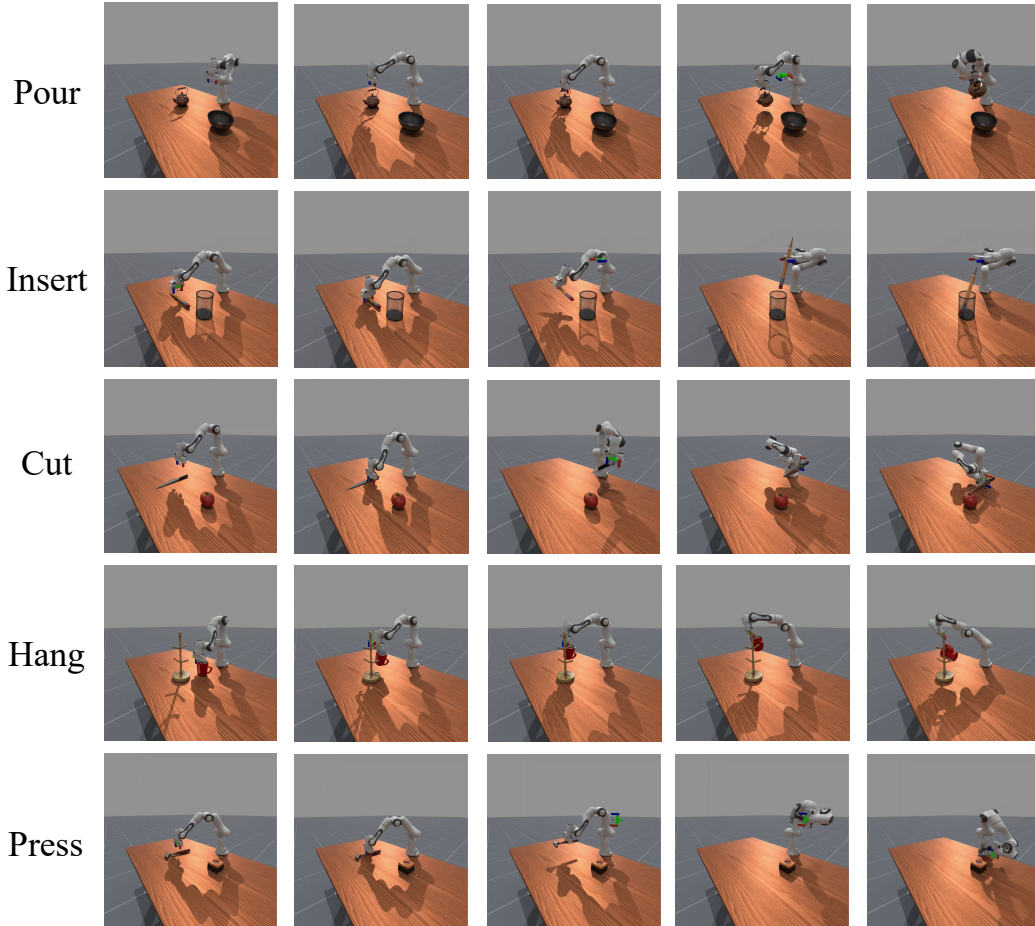


Figure 9: **Qualitative results in simulation environment.**

B Implementation Details

B.1 Baselines

RoboPoint RoboPoint [51] is a vision-language model finetuned on an image-language affordance dataset. To apply it to a point cloud, we first prompt RoboPoint to predict affordance pixels in the image, which are then projected onto the point cloud using the corresponding depth information. Across all views, we aggregate these predictions by averaging the affordance scores assigned from different views weighted by depth consistency across views.

IAGNet IAGNet [27] is a training-based method which takes an image, with an annotated object bounding box, and an object point cloud as input. To adapt the model for the object-to-object setting, we use the same interaction image, with bounding boxes annotated for both the source and target objects during training and testing, for each interacting object pair. This modification preserves the core idea of extracting semantic interaction information from 2D images without altering the model architecture.

UAD UAD (Tang et al., ICRA2025) is a concurrent work that also projects features from DINOv2 onto the point cloud to segment out the part and uses vision-language models for affordance auto-annotation. Our work diverges from UAD in task settings, and we plan to add it as a future baseline after UAD is open-sourced.

B.2 Model Details

Point Cloud Encoder We employ a patch-based architecture for point cloud encoding that processes the input through local grouping and feature extraction. The encoder first groups points using kNN, then processes each local patch through a specialized patch encoder, and finally incorporates positional information through learnable embeddings. This design enables effective capture of both local geometric structures and global spatial relationships.

Hyperparameter	Value
Output dimension	512
Number of groups	256
Group size	64
Group radius	0.15
Position embedding dimension	512
Patch encoder hidden dims	[784, 512]

Table 4: Hyperparameters of the point cloud encoder.

Decoder We employ a transformer-based decoder which additionally adopts joint-attention to enable feature communication between the source and target objects. We provide detailed hyperparameters in Table 5.

Hyperparameter	Value
Number of attention heads	8
Attention head dimension	512
Dropout	0.1
Cross attention dimension	512
Point cloud embedding dimension	512
Activation function	GELU
Output MLP dimensions	[512, 256]
Normalization type	LayerNorm
Normalization epsilon	1e-5

Table 5: Hyperparameters of the decoder.

538 B.3 Prompt Template

539 We use GPT-4o for constraint function generation. Note that our pipeline can be integrated with any
 540 advanced LLMs for constraint function generation. We provide an example prompt template below.

Prompt Template Certain Constraint

You are given two 3D objects represented as point clouds. Each point is associated with an affordance score predicted by a perception model. Your task is to propose a constraint function for the specified affordance type that evaluates how well the source object can interact with the target object.

Inputs:

- Source Object Name: [SRC_OBJECT_NAME]
- Target Object Name: [TGT_OBJECT_NAME]
- Interaction Type: [AFFORDANCE] (e.g., pour, hang, press, cut, put, plug in)
- Source Point Cloud: $\{(x_i, y_i, z_i)\}_{i=1}^N$, with affordance scores $\{a_i\}_{i=1}^N$
- Target Point Cloud: $\{(x_j, y_j, z_j)\}_{j=1}^M$, with affordance scores $\{b_j\}_{j=1}^M$

Task: Generate a constraint function that evaluates the quality of an affordance-specific interaction between source and target objects. The function should consider high-affordance regions, interaction-specific spatial constraints, and physical plausibility.

Code Skeleton:

```
def compute_alignment_score(src_aff, tgt_aff, src_pcd, tgt_pcd):
    score = 0
    """
    # TODO: Implement affordance alignment constraint

    return score
```

Constraints:

- The function should use the information from high-affordance regions of both objects
- The evaluation must reflect the semantic meaning of the specified affordance type
- Consideration should be given to the physical feasibility of the interaction
- All constraints should be combined into a single cost value (lower is better)

541

542 B.4 Example Constraint Functions

543 We provide an example of a generated constraint function in Fig. 10.

```
1  def compute_alignment_score(src_aff, tgt_aff, src_pcd, tgt_pcd):
2
3      # Get the center of source object affordance regions
4      src_aff_center = src_aff[0]['centroid']
5
6      # Get the center of target object affordance regions
7      tgt_aff_center = tgt_aff[0]['centroid']
8
9      # Affordance alignment score
10     # The part to be inserted should be aligned with the container
11     horizontal_alignment = np.linalg.norm(
12         src_insert_center[:2] - tgt_container_center[:2]
13     )
14     aff_alignment_score = horizontal_alignment * 10
15
16     return aff_alignment_score
```

Figure 10: Example of the generated function for object alignment.

C Failure Cases Analysis

We present an empirical analysis of failure cases in this section. Our manipulation pipeline involves three key stages: constructing a semantic point cloud from multiview RGB-D observations, grounding affordance on the partially reconstructed point cloud, and executing actions via a low-level motion planner. As a result, failure cases primarily fall into three categories: (1) inaccurate point cloud reconstruction due to sensor noise, (2) incorrect affordance predictions caused by object self-occlusion and unobservable regions, and (3) inverse kinematics (IK) failure during motion planning.

Sensor noise We use the Orbbec Femto Bolt camera, a Time-of-Flight-based sensor, for capturing RGB-D observations. It provides significantly higher-quality depth compared to stereo-based sensors. However, it may still produce erroneous depth measurements when object materials are highly reflective or when objects are too dark in color, preventing the sensor from receiving sufficient reflected light to compute accurate depth. We provide an example of an incorrectly reconstructed point cloud in Fig. 11.

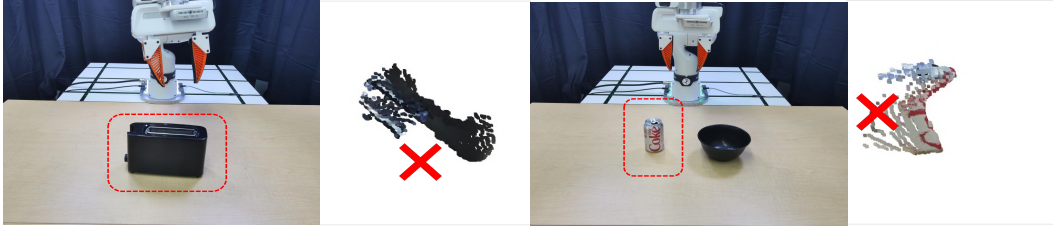


Figure 11: Example of incorrect point cloud.

Object Self-occlusion Our pipeline assumes that affordance regions, such as the mouth of a teapot, are not occluded. However, in practice, these regions can be self-occluded by the object itself. For example, the bottom of a bottle is difficult to observe even with a top-down camera view. This limitation could potentially be addressed by recent advances in single-view or sparse-view reconstruction methods (e.g., TRELLIS, Xiang et. al., CVPR2025), which are orthogonal to our approach. We leave the integration of such techniques for future work.

IK Failure Since we directly optimize the final interaction pose of the object, the resulting pose may be infeasible for execution due to IK failure in the motion planner. We use a screw-based planner, and most IK failures occur when the final pose requires drastic rotational movements. We provide some visualization of IK failure cases in Fig. 13.

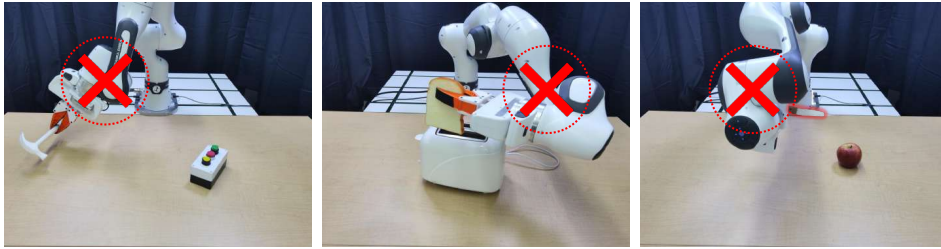


Figure 12: Example of IK failure.

D Environment Setup

Our task uses a Franka Research 3 robot, a 7-DOF manipulator. To capture comprehensive visual information and mitigate occlusions, we employ two Femto Bolt RGB-D cameras. These Time-of-Flight (ToF) based cameras are positioned on the left and right sides in front of the robot’s workspace, providing robust depth perception across the scene.

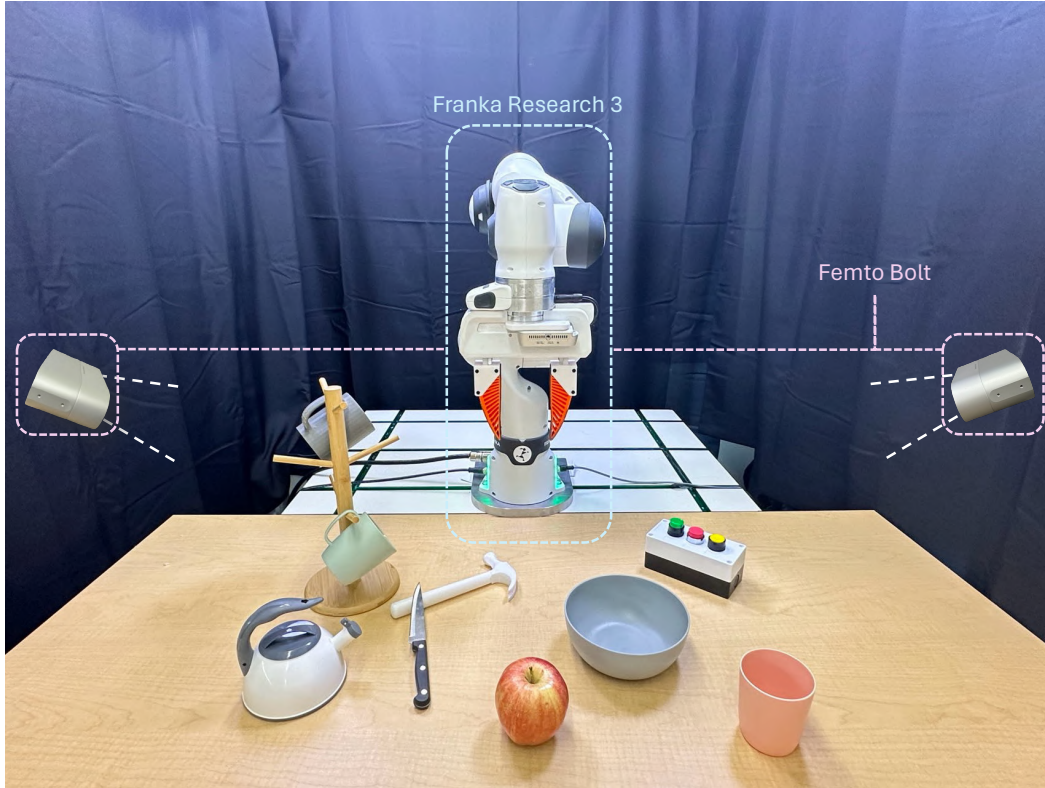


Figure 13: Physical setup.

We utilize a diverse set of objects for our challenging object-to-object interaction tasks. These tasks require the model to not only identify the correct manipulation point on the source object but also to understand the intended interaction point on the target object. This necessitates a deep understanding of object to object affordances. Our tasks include:

- **Pouring water from a teapot into a bowl or:** In this task, the functional affordance is the teapot’s spout, which must be correctly aligned with the receiving area of the bowl (e.g., its bottom center) or the plate. The model must comprehend this specific object-to-object affordance for pouring. For instance, while the edge of the bowl might offer a grasping affordance, it is not the correct target for the pouring task. This task critically evaluates the model’s ability to understand task-specific affordances.
- **Cutting fruit with a knife:** The model must identify the sharp edge of the knife (source object affordance) and the appropriate surface of the fruit (target object affordance) to perform a cutting action. This involves understanding the functional relationship between the knife’s blade and the fruit’s body.
- **Pressing a button with a hammer:** This task requires the model to recognize the head of the hammer as the striking surface and the button as the target. The interaction involves a precise contact between the hammer’s affordance point and the button’s surface.
- **Hanging a cup onto a mugtree:** The model needs to identify the handle of the cup (source object affordance) and a free peg on the mug tree (target object affordance). Successful execution depends on aligning the cup’s handle to engage with the mugtree’s peg.

These tasks are designed to test the model’s understanding of object affordance and how they relate to each other in the context of specific goals.