A Real-Robot Rollouts

Dynamic videos of real-robot rollouts can be found in the supplementary video and at our anonymous website: https://retrieval-manipulation-anon.github.io/.

B Data Collection and Affordance Extraction

B.1 Robotic Data

We adopt DROID [2] as our source of robotic data, which includes 76,000 expert trajectories of robots conducting daily tasks, along with corresponding task instructions. To extract affordance information from DROID, we first query instructions for tasks of interest. For example, for the task "open the drawer," we perform a hard search to filter out instructions that do not include the action "open" and the object "drawer."

Next, the filtered instructions are sorted based on the L2 distances between their language embeddings and the query embedding. We generate these embeddings using the "`text-embedding-3-small`" model of OpenAI. Based on the sorted instructions, we then select the Top-*k* episodes for further affordance extraction.

To extract affordance information from these selected episodes, we identify the cartesian position when the gripper is closed as the 3D contact point. We then track the gripper's position for the following 10 time steps or until the gripper stops moving for consecutive steps. This provides us with the 3D post-contact trajectory. Using the provided camera parameters, we project the 3D contact point and the post-contact trajectory onto the first frame of each episode where the object is unobscured. Note that some affordance demonstrations are further refined manually by adding an offset or being removed due to inaccurate camera parameters. Should accurate camera calibration be available, the manual correction is not necessary.

This method ensures a precise and reliable extraction of affordance information from the DROID [2] dataset. Note that this method can also adapt to other robotic datasets (real-world or synthetic), such as [1, 3, 4].

B.2 HOI Data

In addition to the details in the main text, we further note that we only average the hand keypoints within the object mask to determine the contact point and shift the post-contact trajectory accordingly. This ensures that the contact point is within the object for robust affordance transfer.

Note that this affordance extraction procedure can also be applied to other data sources where hand-object interactions are involved, such as more HOI datasets [5, 6, 7], vast amounts of unannotated human egocentric videos on the Internet, and user-provided demonstrations.

B.3 Custom Data

The annotation process for custom data affordance has been discussed in the main text. Additionally, the sources of custom data are highly diverse and configurable. For our experiments, we annotated object images of our interest obtained by simply searching from the Internet (Google, YouTube, etc.).

Notably, custom data can also come from a variety of sources, ranging from user-captured images, cartoon images, AI-generated content, and even sketches, etc., demonstrating the flexibility and diversity of our data sources. This flexibility allows for a wide range of potential applications and extends our affordance memory's scalability to a greater extent.

B.4 Affordance Memory Statistics

The statistics of our affordance memory can be found in Table 5.

Task Name	Icon	Data Source	Size
Open the drawer	🗄	DROID	30
Close the drawer	🗄	DROID	20
Open the cabinet	🚪	DROID	12
Close the cabinet	🚪	DROID	6
Open the microwave	📺	DROID	42
Close the microwave	📺	DROID	10
Open the dishwasher	▤	Custom	10
Open the refrigerator	▯	Custom	20
Open the trashcan	🗑	Custom	20
Pickup the pot	🍲	DROID	11
Pickup the mug	☕	HOI4D	149
Pickup the bowl	🥣	HOI4D	252
Pickup the bottle	🍾	HOI4D	78
Total	/	/	660

Table 5: Affordance memory statistics.

C Implementation Details

C.1 Feature Extraction Using Foundation Models

We use different foundation models as visual feature extractors, including:

- **Stable Diffusion (SD)** [46]. As illustrated in [51], given an original image x_0, we first add noise of time step t to it to move it to distribution x_t, and then feed it to the stable diffusion network f_θ along with t for denoising to extract the intermediate layer activations as the diffusion features (DIFT). We use the same configuration as in [51].
- **DINOv2** [55]. Extracting DINOv2 features is implemented by feeding the original image to the DINOv2 model and extracting the intermediate layer activations of DINOv2 ViT during the feed-forward process.
- **SD-DINOv2** [52]. As in [52], we first extract SD features and DINOv2 features and then do L2 normalization on them to align their scales and distributions. After that, we concatenate these two features together to get the SD-DINOv2 feature.
- **CLIP** [45]. Similar to DINOv2, We extract dense CLIP features by utilizing the intermediate layer activations of CLIP ViT.

C.2 IMD Metric Calculation

As in [56], Instance Matching Distance (IMD) is originally proposed to examine pose prediction accuracy. Given a source image $I^{\mathcal{S}}$ and a target image $I^{\mathcal{T}}$, their normalized and masked feature maps $F^{\mathcal{S}}$ and $F^{\mathcal{T}}$, and a source instance mask $M^{\mathcal{S}}$, the IMD metric is defined as:

$$\mathrm{IMD}(I^{\mathcal{S}}, I^{\mathcal{T}}, M^{\mathcal{S}}) = \sum_{p \in M^{\mathcal{S}}} \left\| F^{\mathcal{S}}(p) - \mathrm{NN}(F^{\mathcal{S}}(p), F^{\mathcal{T}}) \right\|_2, \tag{3}$$

where p denotes a pixel within the source instance mask, $F^{\mathcal{S}}(p)$ is the source feature vector at pixel p, and $\mathrm{NN}(F^{\mathcal{S}}(p), F^{\mathcal{T}})$ denotes the nearest neighbor vector in the target feature map $F^{\mathcal{T}}$ with respect to the source feature vector. IMD measures the similarity of two images via the average feature distance of corresponding pixels [56]. Using IMD in the geometrical retrieval stage, we can accurately retrieve the demonstration where the object is oriented in the most similar way as in the observation.

C.3 Baseline Methods

- **Where2Act** [14] is designed for articulated object manipulation only, which takes an object point cloud as input and predicts point-wise actionability scores, action proposals, and action scores with three separate models. Another drawback of this method is that it processes the point cloud in a task-agnostic way, leading to ambiguity of the generated affordance. We adopt it to the evaluation tasks by 1) randomly sampling the contact point from the predicted top-5 actionable points, 2) proposing 100 actions using the action proposal model, and 3) selecting the action with the highest action score.
- **VRB** [12] predicts the contact point and direction only on 2D images. To make it applicable in real manipulation tasks, we lift the estimated 2D affordance to 3D using our proposed sampling-based affordance lifting module.
- **Robo-ABC** [44] is initially designed for object grasping only, where only the contact point of a source demonstration retrieved by CLIP [45] feature similarity is transferred on the 2D image, followed by AnyGrasp [17] for grasp pose selection. For a fair comparison, we feed it with our collected affordance memory. To extend it for articulated objects, we use the proposed 2D affordance transfer module to transfer both the contact point and post direction. Subsequently, we follow the same procedure as in our method to lift 2D affordance to 3D.

D Experiment Details

In the simulation, we utilize a flying Franka Panda gripper for simplicity. We utilize cuRobo [19] motion planner for position control of the gripper.

In the real world, we adopt two different robotic systems. In the Franka Emika robotic arm setting, we leverage an on-hand RealSense D415 camera for RGBD perception and utilize *MoveIt!* [18] motion planner for the transformation from the target end-effector pose to joint position trajectories. In the Unitree robot dog setting, we leverage a Unitree B1 dog with a Z1 arm, along with a Robotiq 2F-85 parallel gripper. The RealSense D415 camera is also on-hand mounted, and we control the arm using the Z1 SDK for delta cartesian-space control.

For grasp generation, we utilize AnyGrasp [17] to produce grasp proposals, along with GSNet [16] with a relatively low graspness score threshold and collision threshold for more dense grasp proposals in case there is no grasp pose close enough.

E Downstream Application Details

E.1 Training ACT Policy

For policy distillation, we utilize an ACT policy [30] to perform imitation learning from our self-collected demonstrations. ACT is based on CVAE Transformer architecture and adopts the idea of action chunking to mitigate compounding errors that are common in behavior cloning (BC). More details can be found in their original paper [30].

We use 5 RGB views ($5 \times 640 \times 480 \times 3$) and the robot's proprioception as observation. We set the chunk size to 60, and the latent space dimension to 512. We use L1 loss plus KL divergence regularization for supervision. The number of training iterations is set to 200K, and we set the learning rate to 1×10^{-5} and batch size to 8.

E.2 One-Shot Visual Imitation Details

For one-shot visual imitation conditioned on human preference, we pick out demonstrations either from our own in-domain demonstrations or from out-of-domain cartoon images (*Tom and Jerry* in this case). We ground and choose the first frame of interaction for I^S and use the custom data annotation method for affordance extraction. We then skip the hierarchical retrieval step and directly use these chosen demonstrations for affordance transfer and lifting, followed by 3D affordance execution.

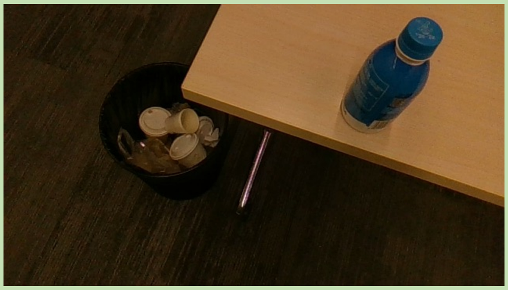
E.3 LLM/VLM Integration Details

For LLM/VLM integration, we utilize GPT-4V (`gpt-4-vision-preview`) [64] for task decomposition and scene understanding. We also use Grounded-SAM [65] for object detection and segmentation to produce 3D bounding boxes of objects in the scene.

Specifically, we define 3 basic primitives: `grasp()`, `move_to()`, and `release()` for VLM output. Note that these three primitives do not involve heuristics on specific object manipulation. Other than these primitives, when the VLM finds out there are relevant demonstrations in the affordance memory, it will schedule the proposed RAM system as a retrieval-augmented module to perform the action as a whole, followed by other defined primitives.

An example of our prompt and the VLM output is shown in Fig. 7.

Prompt



```
===========

You are an intelligent robot dog that has an arm with a parallel gripper
for object manipulation.

You are given a human instruction and a scene observation. Your task is
to correctly manipulate the objects safely conditioned on the
instruction.

===========
```

```
You have a series of primitives and demonstrations you can leverage to perform the task. Based on the instruction, you can freely decompose it into
several sub-tasks that are easier to finish and then chain them together.

First, you are endowed with 3 primitives, which are:
1. grasp(), which takes in object name, detects the object, moves to a graspable pose, and closes the gripper.
2. move_to(), which takes in a 6D pose and does motion planning to it.
3. release(), open the gripper to release the holding object.

The calling format should be like release(), etc.

Apart from the primitives, you also have an affordance memory, which contains lots of tasks that already have plenty of demonstrations. You are also
equipped with a powerful robotic system called RAM, which can utilize those demonstrations in your affordance memory to directly perform
corresponding sub-tasks, without calling the primitives. The tasks in your affordance memory are:

- Open: drawer, cabinet, microwave, dishwasher, refrigerator, trashcan
- Close: drawer, cabinet, microwave
- Pickup: pot, mug, bowl, bottle

The calling format should be like Open_drawer(), etc.

For the instruction and observation, please output a sequence of actions or primitives in the format of function calling as above. And finally, you
should convert the sequence into a python list for convenience.

===========

A basic example:

Human instruction: Please open the drawer and pick up the utensil.
Observation: A closed drawer.

Your output:
<agent>
+++
First, I will decompose the instruction into several sub-tasks. The instruction is 'Please open the drawer and pick up the utensil.', so I need to
first open the drawer, and then pick up the utensil.
Sub-tasks:
1. open the drawer
2. pick up the untensil
+++
Then, for each subtask, I will see whether it can directly be performed using my affordance memory.
For 1, there are demonstrations regarding opening drawers, so I will directly use it.
For 2, there are no demonstrations for utensils, but picking up bottle is similar. Although usable, I will use primitives to ensure safety.
Action sequence:
1. Open_drawer()
2. grasp(utensil)
3. move_to(utensil_above_position)
4. move_to(release_position)
5. release()
+++
Finally, I will convert the sequence into a python list:
<output>
['Open_drawer()', 'grasp(utensil)', 'move_to(utensil_above_position)', 'move_to(release_position)', 'release()']
</output>
+++
</agent>

===========

Now you need to clear this table, please find a feasible way of doing that. Follow the example above:
```

VLM Output

```
Human instruction: Please clear the table.
Observation: There is a bottle on the table, and a trashcan with cups and bottles next to it.

<agent>
+++
First, I will decompose the instruction into several sub-tasks. The instruction is 'Please clear the table', so I need to first pick up the bottle
from the table, move it to the trashcan, and release it.
Sub-tasks:
1.Pick up the bottle from the table.
2.Move the bottle to the trashcan.
3.Release the bottle into the trashcan.
+++
Then, for each subtask, I will see whether it can directly be performed using my affordance memory.
For 1, there are demonstrations regarding picking up a bottle, so I will directly use it.
For 2 and 3, I will use the primitives to ensure the correct and safe movement to the trashcan and releasing the bottle.
Action sequence:
1.Pickup_bottle()
2.move_to(trashcan_position)
3.release()
+++
Finally, I will convert the sequence into a python list:
<output>
['Pickup_bottle()', 'move_to(trashcan_position)', 'release()']
</output>
+++
</agent>
```

Figure 7: Example prompt and VLM outputs of our LLM/VLM integration system.