APPENDIX

A. Implementation Details on Fine-Grained Region Proposal

In this section we discuss additional implementation details to obtain candidate regions.

From multi-view RGBD renderings of an asset, we obtain aggregated point cloud $P \in \mathbb{R}^{N\times 3}$ by projecting fore-ground pixels (we obtain this mask from the rendering simulator) of each view to world frame and uniformly down-sample the point cloud. Then for each RGB image, we extract patch-wise features from DINOv2 with registers (ViT-L14) [14] and perform bilinear interpolation to upsample the features to original image size.

To fuse the DINOv2 features from all views to P, we adapt the following procedure from [109]: for each point $p \in P$, we compute it's corresponding pixel on each camera view. We consider it to be visible in a camera view if the projection depth is close to the depth image reading at that pixel by a small threshold. The fused feature for p is the average of DINOv2 features on it's corresponding pixel across all the views p is visible.

With the procedure above we obtain global feature field $F_{\text{global}} \in \mathbb{R}^{N\times d}$, and then we apply PCA to obtain $F_{\text{reduced}} \in \mathbb{R}^{N\times 3}$ to mitigates affect of local texture or appearance on cluster result.

We group object points into candidate regions by running clustering algorithm on F_{reduced}. Since our data processing pipeline handles over-segmentation better than under-segmentation (reasons established in next su-bsection), we first run Mean Shift on the features, and if it found less than 5 clusters over the object, we re-run the k-means to find 5 clusters. Specifically, for articulated objects such as cabinets, we obtain the per-link mask from the rendering process and run the aforementioned clustering pipeline for each link individually. This allows us to find finer object regions such as drawer knobs.

B. Implementation Details on Task Instruction Proposal and Region-Instruction Mapping

After we obtain the region labels for all object points, we visualize the clustering on the view we selected with the following procedure: for each foreground pixel in that view, we compute the corresponding 3D point with using the depth map. Then we find its closest neighbor p in the aggregated pointcloud P, and use the label r_p as the region label for this pixel.

We assign a unique color to all pixels within each cluster and overlay this on the original RGB image. We input the cluster visualization with the original image to the VLM prompt below. The prompt contains only generic instructions and a few text-based examples to illustrate expected output and format. VLM is queried to propose a set of task instructions $\{\mathcal{T}_1, ..., \mathcal{T}_J\}$ closely related to the object, and associate each instruction with a single candidate region. We use GPT-4o [10] for all our experiments.

To convert the instruction-region matching to continuous affordance map $A \in [0,1]^{H\times W}$, we average the features for points in the corresponding region and calculate cosine similarity score of this reference feature and F_{global}. We project this to each camera view following the same procedure as we visualize cluster results, i.e. for each pixel, find its corresponding 3D point's nearest neighbor in P and assign that point's value. All values below 0 are set as 0 to ensure the correct value range of the obtained affordance map.

We found over-segmentation by the clustering step is preferred over under-segmentation. When a object part is over-segmented, empirically GPT-4o is still capable for correctly associating the instruction with one of the regions, and through the cosine similarity calculation, the other not selected regions within the same part is likely still computed to have high cosine similarity to the reference feature. On the other hand, under-segmentation could cause the affordance map to highlight regions that are not most closely related to the instruction, which is not desired for our purpose of finding fine-grained affordance.

We obtain triplets of RGB object image, task instruction, and affordance map (I, $\mathcal{T}$, A), from our data extraction pipeline. We further process the affordance map by setting all values below a threshold 0.5 to 0, with the purpose to create a ground-truth map more focused on the most relevant regions. We then apply a Gaussian blur to A with kernel size = 3, to accommodate for the boundaries created by the previous thresholding process and improve training stability.

Our model contains 3 FiLM-conditioned convolution layers with output channels $[256, 64, 1]$. We use linear layers to predict channel transformations from language embeddings. We initialize the linear layers with weights to be 1 and bias to be 0, adapted from implementation in RT-1 [118]. We use Adam optimizer with a learning rate of 0.001. We train our model for 30 epochs with a batch size 8, which takes approximately 12 hours on a single NVIDIA A6000 GPU.

C. Mturk Interface

Fig 6 is the Amazon MTurk interface we use to collect human affordance annotations on our valuation sets and DROID images.

D. MTurk Task Assignments and Label Post-processing

To obtain the ground-truth for evaluations, we prompt the image and corresponding text to Amazon MTurk workers and ask them to draw a fine-grained mask on the image for the region they believe corresponds to the text. On average, worker complete labelling assignment for each image in 40 seconds, for which they are compensated for 0.6 dollars.

For each (text, image) query pair, we collect annotations from 7 MTurk workers and apply a pixel-wise voting scheme. A pixel is marked as part of the ground truth mask (value 1) if more than three workers label it accordingly.

E. Details for Evaluation on AGD20K Dataset

We evaluate our model on the easy split of AGD20K and compare with the baseline performance reported in [117]. To avoid numerical instability in KLD computation as we

```
## System Prompt

Given (a) the category of an object, (b) a reference image of the original image, (c) an image
    visualizing clustering of the object into a few regions, each indicated by a distinct color, and (d)
    a related list of used colors, please:
1. Identify specific regions of the object that serve different purposes in various manipulation tasks.
- Focus on crucial parts and offer detailed and fine-grained descriptions of the regions of interest.
- For each identified region, provide both a Region Description and a Region for action xxx. For example,
     "handle of plastic bag -- region for agent to hold and lift the bag." Use double quotes only to
    represent a string element.
- Avoid using multiple single quotes for an element. For example, instead of "adjusting the lamp's
    position", use "adjusting the lamp's position." Avoid trivial regions like: power cord, power plug,
    seal, small edges, small corners, different sides of the walls or body, interior base, or exterior
    edge.
2. Match the colored region in (c) with the proposed task in step 1, considering the functionality and
    the granularity of the task. The requirements are as follows:
- Compare the original image to the proposals to find the colored region that matches the description,
    considering the context provided by the explanation. For example, if a given proposal image indicates
     that the red region covers the handle, and the description mentions a task related to the handle,
    you should identify the answer as "Red."
- When the described region is clustered into more than one cluster on the image, pick the cluster that
    is most appropriate according to the explanation provided in the description.
- If the described region is within a cluster but still contains other parts of the object, you should
    still select the cluster.
- Consider the reason/explanation to make the final decision, and provide your best guess.
If you cannot identify the counterpart on the given image, give your best guess. Do not say you cannot
    identify something.

## User Prompt

The first image is the original image of the object, and the second image shows the clustering of regions
     in colors.
This is the color list: {CLUSTER_COLORS}, and this is the object category:{OBJ_CATEGORY}.
I need you to propose the task-guided fine-grained region description and match region description with
    the one most appropriate color in the images.
Output format: Start with the word "ANSWER: ", followed by a dict, where each key-value pair is in the
    format of "region description -- region for xxx" : "Color", separated by commas. Specifically, the
    content after "ANSWER: " should be parseable with Python's ast.literal_eval() and nothing else. All
    elements in the dict keys or values should be enclosed by double quotes only. The color could only be
     one color, with the first letter of the color name capitalized and the rest in lowercase.
```

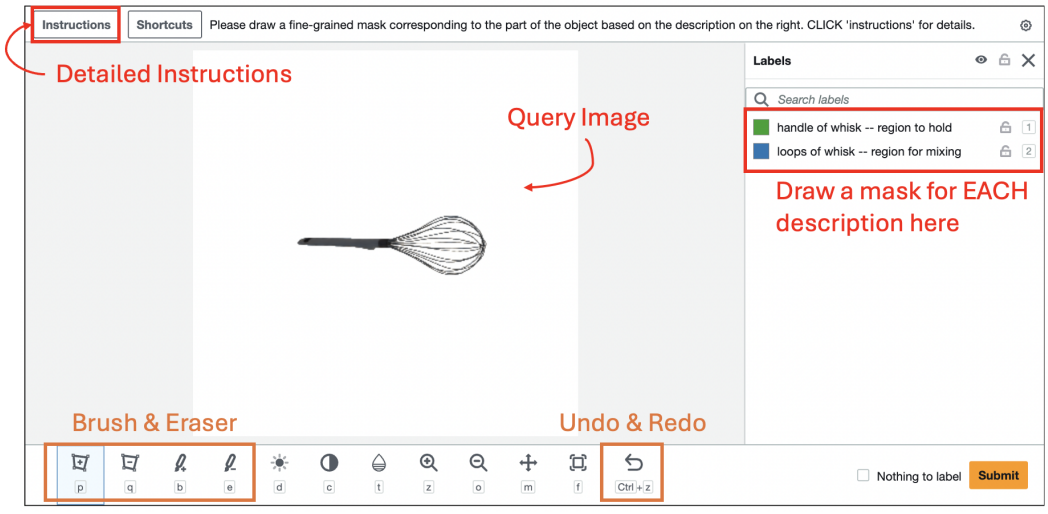

Fig. 6: Amazon MTurk Annotation Interface.

consider per-pixel affordance instead of per-image affordance as in other works, we post-process our model's prediction by adding a small ϵ to each pixel before normalization.

F. Details for Evaluation on DROID Images

We implement the baselines as follows:

- CLIP: we obtain the per-patch visual feature of query image and use bilinear interpolation to original image

height and width. We compute the per-pixel cosine similarity with the text feature of query instruction and clip the minimum value to be 0.
- OpenSeeD: Since OpenSeeD is an open-vocabulary segmentation model, we use the same query instructions as for other methods to query the predicted mask. We set all pixels in the predicted object mask to 1 and 0 otherwise. The prediction image is all 0 if no mask is found.

G. Policy Learning in Simulation – Environment Setup

Our simulation environment in OmniGibson contains one Fetch robot, for which we use operational space controller for the end-effector pose, multi-finger gripper controler for the gripper, and kept location of the robot base fixed. Grasping is physically simulated for all tasks.

We use a key-frame based policy for both demonstrations and learnt policies. Key-frames are commonly used by prior works [112, 113, 119–121] as "important or bottleneck steps of gripper during task execution". To execute an action <end-effector pose, gripper action>, we first command the end-effector controller of an interpolated trajectory from current to target pose, then execute the gripper action afterwards.

We use 3 cameras at the front, left, and right of the workspace for *Pouring* and *Inserting*. We use 2 cameras on both sides of the robot for *Opening* as the articulated objects are typically large in size and would occlude the other cameras.

H. Policy Learning in Simulation – Tasks

Below we discuss the details of environment setup, scripted policy steps, success criteria, and evaluation generalization setting for each task.

Pouring The environment includes a beer bottle, a bowl, and a pot plant. The scripted policy involves four key-frames: reaching a pre-grasp pose next to the bottle, grasping the bottle, lifting and moving it next to the bowl, and tilting it to pour into the bowl. Success is defined by the alignment and tilting of the bottle's opening directly over the bowl.

At training time, object poses are randomized within a [±5cm, ±3cm, 0] range, with the bowl and the pot plant positions randomly swapped. This randomization is maintained during evaluations to test the system with varied object poses.

Different object models for the beer bottle and bowl are used for the novel object instance evaluation. The beer bottle is replaced with a Coke can in the novel object category evaluation. For the novel instruction, the task is changed to watering the pot plant.

Opening The task environment features a cabinet with a revolute door. The task sequence includes two steps: reaching and grasping the cabinet door handle, followed by pulling it open. The task is considered successful if the door opens to at least 45 degrees.

During training, the position and orientation of the cabinet are randomized within a range of [±5cm, ±5cm, 0] for position, and ±15 degrees around the z-axis for rotation. This randomization is also applied during evaluations to assess performance with varied object poses.

For the novel object instance evaluation, a different cabinet model is used. A small refrigerator substitutes the cabinet in the novel object category setting, testing adaptability to different objects. Novel instruction scenarios are not evaluated for this task.

Insertion The environment contains a marker, a carrot, and a pencil holder. The task involves two key steps: picking up the marker and positioning it directly above the pencil holder's opening in an upright orientation. The task is considered successful if the marker is in the holder.

During training, the positions of the pen and carrot are randomized within ±1.5 cm in the x-direction, and the pencil holder is adjusted within ±3 cm in both x and y directions. The pen and carrot positions are also randomly swapped. The same randomization parameters are used during evaluation.

A different marker model, varying in color and size, is used for evaluating a new object instance. The pencil holder is replaced with a coffee cup for the novel object category evaluation. The task of inserting the pen is changed to inserting the carrot for the novel instruction evaluation.

I. Policy Learning in Simulation – Details on Baseline Visual Representations

Herein we introduce our implementation for each baseline visual representations.

- Vanilla policy: original rgb observation from each camera.
- w/ DINOv2: we first obtain per-pixel DINOv2 features for the rgb image of each camera. Then, we have a trainable 1D convolution layer with kernel size of 1 to reduce the number of channels to 3.
- w/ CLIP: we obtain CLIP text embedding for each detailed instruction for the task. Then we calculate the cosine similarity against per-pixel CLIP visual embedding of each camera observation.
- w/ Voltron [71]: we load a frozen Voltron (V-cond) model and obtain the visual embedding conditioned on task description. We interpolate the per-patch embedding to pixel space, and use a trainable 1D convolution layer with kernel size 1 to reduce the number of channels from 384 to 3. We have also experimented with using a trainable multi-head attention pooling layer for feature extraction, as suggested by the original paper, yet haven't observed improved performance.

J. Training Details

We trained each policy for 4000 epochs. Training with batch size of 3 on a single NVIDIA A40 GPU takes approximately 16 hours for *Pouring*, and 8 hours for *Opening* and *Inserting*.

During training, we normalized the channels for visual observation by: normalize visual representation to $(-1, 1)$, clip (x, y, z) to the min and max workspace bounds, and set depth for all out-of-bound points to 0. Additionally, we append channels according to pixel location, following the

original implementations in RVT. We apply random cropping augmentation to visual input during each training step.

K. Policy Learning in Real-World – Environment Setup

Our real-world evaluation platform uses a Franka arm mounted in a tabletop setup built with Vention frames. Since the learned policy outputs 6-DoF end-effector poses and gripper actions, we use position control in all experiments, which is running at a fixed frequency of 20 Hz. Specifically, given a target end-effector pose in the world frame, we first clip the pose to the pre-defined workspace. Then we perform linear interpolation from the current pose of the robot to the target pose with a step size of 5mm for position and 1 degree for rotation. To move to each interpolated pose, we first calculate inverse kinematics to obtain the target joint positions based on current joint positions using the IK solver implemented in PyBullet [122]. Then we use the joint impedance controller from Deoxys [123] to reach to the target joint positions. Two RGB-D cameras, Orbbec Femto Bolt, are mounted on the left side and the right side of the robot facing the workspace center. The cameras capture RGB images and point clouds at a fixed frequency of 20 Hz.

L. Policy Learning in Real-World –Tasks

We mirror the setup in simulation to evaluate on three similar tasks in the real-world: watering plant, opening drawer, and inserting pen into pen holder. We collect a total of 10 demonstrations for each task and train a policy using the same training procedure described above. The demonstrations are collected using kinesthetic teaching, consisting of varying numbers of keyframes (as described above) that are required to complete the task. Success rates are visually examined by the operator. 10 trials with varying object configurations are performed, and the average success rate for each task is reported.