Towards Open-World Grasping with Large Vision-Language Models

Supplementary Material

A LVLM prompts

A.1 Prompts

Prompts for the three use cases considered in this work, namely: **open-ended referring segmentation**, **grounded grasp planning**, and **grasp ranking** can be found below.

- **Open-ended referring segmentation:** referring_segmentation.txt
  Takes an observation image, a marked copy with highlighted instance masks and label IDs and an open-ended language query referring to a particular object instance, and outputs the label ID of the referred object. The LVLM is encouraged to provide chains-of-thought reasoning in cases where the input query contains complex expressions that involve multiple object and spatial relations.
- **Grounded grasp planning:** grasp_planning.txt
  Takes the marked image and the label ID of the target object to be grasped, and outputs a plan to ensure the target object will be graspable. The plan consists of `remove` actions for blocking objects and a final `pick` action for the target.
- **Grasp ranking:** grasp_ranking.txt
  Takes a cropped bounding box image around the next object to be picked, marked with grasp proposals from a 4-DoF grasp synthesis model and a set of label IDs, and outputs a sorted list of grasp IDs, from most to least confidence for a successful grasp. The LVLMs is encouraged to reason about the object shape and their neighbouring objects before producing a final ranking.

A.2 Visual Prompt Design

In the following, we summarize the key visual prompting elements that were used for prompting the LVLM in the context of OWG:

Clarity of visual markers The most common failure mode of visual marker prompting with GPT-4v is that it sometimes struggles to discriminate which ID corresponds to what segment. Especially in cluttered scenes, label IDs might severely overlap within small frame regions. Several techniques can assist in making the markers more clear to the LVLM: a) we adopt the algorithm of [1] for overlaying numeric IDs within the frame with minimal overlap, b) we paint both the internal of each segment's mask and its ID with the same unique color (colors are chosen to be visually distinguishable), and c) increase the resolution of the marked image and the size layout of the markers.

Reference Image If not highlighting the internal of each segment, GPT-4v sometimes refers to regions with wrong IDs, especially in highly cluttered scenes. But if the masks are highlighted with high opacity, then the appearance of the object becomes less visible and GPT-4v struggles to recognize it. We propose a technique to ameliorate this is by passing both the original (reference) and the marked image and constructing a text prompt that explains that the latter corresponds to annotated segments of the first.

8th Conference on Robot Learning (CoRL 2024), Munich, Germany.

Chain-of-thoughts Chain-of-Thought (CoT) prompting is a well-established methodology for guiding LLMs to perform multi-step reasoning and reduce hallucinations [2]. We find that LVLMs share similar properties and prompting them to reason about their final answer before producing it can robustify the response quality. For grounding, we ask GPT-4v to decompose the input instruction in steps and refer to all intermediate referenced objects. For grasp planning, we ask it to explicitly mention all object IDs that are blocking the target object, before producing a plan. For grasp ranking, we decompose the prompt in three steps: (i) identify the category of the target object and provide a general description of what constitutes a good grasp for it given its shape, (ii) list the grasp IDs that will most likely lead to contact with neighboring objects, and (iii) rank the grasp IDs based on the previous two steps.

Self-consistency Even with zero temperature, we observe that the outputs of GPT-4v are not always reproducible. We find that sometimes GPT-4v might produce different responses at different runs, even with exactly the same prompt. In an attempt to reduce the effect of this phenomenon and robustify LVLM outputs, we use the self-consistency method developed for LLMs [3]. In particular, we ask GPT-4v to provide multiple responses, parse each one separately and then perform majority voting to determine the most consistent output.

B Robot experiments

B.1 Setups

Our object catalog for seen/unseen trials is shown in Fig. 1. In Gazebo, isolated scenarios are generated by ensuring all spawned objects have a fixed 3D distance, while in cluttered scenarios we ensure contact between the target object and neighbouring objects, by first spawning the target and then sampling different poses for other object models around it. In real-robot experiments, we manually setup the scenes while making sure to replicate the setup as close as possible for fair comparisons between baselines. In all trial scenes that contain distractor objects, the user instruction refers to some property that disambiguates the target instance from other objects of the same category, using names, attributes and spatial relations. We also conduct experiments without distractors for affordance-based queries, which require semantic reasoning to be correctly grounded.

Figure 1: Seen *(left column)* and unseen *(right column)* object used in our robot experiments in Gazebo *(top)* and the real world *(bottom)*.

For real robot experiments, we use the default `torchvision` implementation of Mask-RCNN, with the model weights provided by PyTorch Hub, fine-tuned in a few annotated scenes captures from our robot setup. For grasp synthesis, we generate a top-down orthographic projection of the scene, both for color and for depth (i.e. reverse depth - heightmap). This is the input we pass to the pretrained GR-ConvNet. In order to align regions from the 2D frame where Mask-RCNN provides segmentations and the orthograpic projection where our grasp synthesis model provides grasp poses, we use the Hungarian matching algorithm to match the centers of outputs from both models, after projected to 3D and transformed to a world reference frame (robot base), using 3D euclidean distance as the cost function.

B.2 Baseline Implementation

CROG CROG receives an single 448×448 RGB view and a natural language query, and provides both an instance segmentation mask for the target object, as well as a set of 4-DoF grasp proposals, assuming that the gripper approaches the object aligned with the perspective of the camera. We use the checkpoint provided by the original paper, trained in the multiple split of OCID-VLG dataset, which contains 90k scene-query-grasp data from around 1,000 unique scenes from 31

Name	Attribute	Spatial Rel.	Visual Rel.	Sem. Rel.	Multi-hop	Affordance	Total
42	26	33	19	13	24	16	173

Table 1: Number of samples in grounding evaluation dataset.

object categories. The model uses CLIP's pretrained ResNet-50 visual and BERT text encoders, but fine-tunes them end-to-end in OCID scenes for joint grounding and grasp synthesis tasks.

SayCan-IM Our SayCan-IM baseline follows the implementation publicly released by the SayCan work [4], which can be found in this HTTP URL. In particular, the pipeline uses the ViLD [5] open-vocab object detector to turn the input observation image into a list of object names and then lets the LLM generate a sequence of pick-and-place actions to perform in order to solve the task given by the user. We made the following modifications to the above baseline:

1. In the original implementation, the robot only has access to a `pick_and_place` skill, and the output plan is confined to only selecting what objects to pick and where to place them (based on the detected object list from ViLD). In our implementation, we also provide a `visual_grounding` tool, which lets the LLM invoke CLIP [6] to rank a list of candidate objects with a given text description and select the most similar one. This is to allow the LLM reason about attribute concepts besides object category (e.g. *"get the blue mug"*).
2. Besides the names of the appearing objects, we also provide their bounding box coordinates, as detected by ViLD, in `x1y1x2y2` format in the prompt. This was introduced in order to enable the LLM to also reason about the locations of objects and resolve spatial relation queries, as well as reason about the feasibility of grasping objects by checking whether their bounding boxes overlap.
3. We replace the `pick_and_place` primitive skill with two distinct skills: `remove` and `pick`. The first skill corresponds to removing a blocking object in order to clear the path for grasping the target. The second skill corresponds to picking the target object that the user requested. Both skills use GR-ConvNet [7] under-the-hood to sample grasp proposals, select the one with higher predicted grasp quality, and use an IK solver to control the robot arm.
4. We used the observe-reason-act prompting style first introduced by Inner Monologue [8] and later improved by ReAct [9]. Unlike the vanilla implementation, which simply produces a plan of steps without feedback, with this technique we let the LLM plan one step at a time, and integrate feedback from the environment (e.g. CLIP outputs, grasp failures etc.) before planning again.

The system prompt and in-context examples used in our SayCan-IM baseline are shown in Fig. 2. As we mention in our main paper, for the real robot experiments, we replace ViLD-RPN with a Mask-RCNN [10] for instance segmentation, and use CLIP with prompts for all object used in experiments to recognize the categories and provide the object list state to the LLM.

C Offline grounding experiments

C.1 OCID Dataset Details

We manually annotate 173 images from OCID dataset with the following query types: a) **name** (open-vocabulary object descriptions), b) **attribute**, c) **spatial relations**, d) **visual relations**, e) **semantic relations**, f), **multi-hop reasoning**, and g) **user-affordances**. The number of annotations per query type given in Table 1. We make sure to include unique test scenes from the dataset and include images with heavy clutter. The target of each scene within a query type is unique, and we make sure to include images with distractor objects (of the same category as the target) for all query types that require relational reasoning (all except name and affordance).

Regarding our custom FGVP-CLIP baseline (FGVP*), we present analytical comparisons and ablation in the following subsection.

C.2 Baselines Implementation and Ablations

We utilize the provided demo applications for the end-to-end methods (SEEM, PolyFormer) to conduct grounding experiments manually. For CLIP-based baselines, we re-implement all methods from the corresponding papers (ReCLIP, RedCircle, FGVP) . We use the ViT-B visual encoder to extract features from image segments and the default BERT text encoder to represent the input query. CLIP-based baselines compute the cosine similarity between segment and text features to rank them and select the most similar segment as the final result via the argmax operator. Ground-truth masks are used for all CLIP-based baselines, similar to GPT-4v ones. We would like to highlight that in the original papers, the aforementioned baselines use potential post-processing steps to enhance the grounding capabilities of CLIP. In particular, ReCLIP uses syntactic parsing to extract entity and relation words/phrases from the input query, as well as spatial relation resolution heuristics (e.g. 'left', 'on' etc. - designed specifically for the RefCOCO dataset) to process the relations analytically and combine CLIP predictions only for the entities. RedCircle and FGVP additionally utilize a *"subtraction"* post-processing step, where they further subtract from the similarity values the average in a set of mined hard-negative queries (again selected for a specific dataset). We believe that such steps constitute domain-aware hand-crafted efforts, which even though helpful, do not represent the challenges of open-ended generalization, which is the primary focus of this work. As a result, we do not consider such post-processing steps in our baseline implementation.

Comparisons with end-to-end approaches The need for manual annotations to exhaust all possible language query inputs, as well as the need for manual testing via online demo applications for the considered specialist end-to-end methods (SEEM, PolyFormer) restrained us from conducting experiments in large-scale. Instead, we originally conducted experiments in a smaller subset of 52 images. Results are given in Table 3. Results follow similar patterns to the larger test set of the main paper. Specialist models (SEEM, PolyFormer) struggle with even simple name queries, scoring below 15% on average. This is potentially due to the high discrepancy between the training distribution of RefCOCO and Visual Genome and our test data, as well as the lack of relational and affordance-based language in these datasets. GPT-4v-based methods still compare favourably to CLIP-based baselines, even in the SoM setting where single marked image is used. Overall, our OWG-grounder achieves an averaged mIoU score of 70.4%, which is almost $\times 2$ from the previous approach.

CLIP Visual Prompt Ablations To further analyze the performance of CLIP-based baselines, we conduct ablation studies where we use specific elements of each method. In particular, we study: a) effect of using **multi-templates** for the text prompt, where we average text embeddings from multiple versions of the query, using templates from the original paper, b) averaging similarity scores from the visual prompt and **crops** of each segment, as originally proposed in ReCLIP, c) different visual prompt schemes, like drawing a boundary (**rectangle** or **ellipse** - as in RedCircle), converting

w/ Crop	w/ White-Back.	w/ Blur-Rev	w/ Gray-Rev	w/ Multi Temp.	Rect.	Ellipse	Mask	mIoU
							✗	18.3
						✗		31.1
					✗			34.8
				✗	✗			33.7
			✗		✗			24.6
		✗			✗			26.3
		✗	✗		✗			34.9
		✗	✗		✗		✗	41.5
		✗	✗	✗	✗		✗	43.0
	✗	✗	✗	✗	✗		✗	**51.8**
✗	✗	✗	✗	✗	✗		✗	51.2

Table 2: Component ablation studies for CLIP-based visual prompting. Results in %.

Method	Found. Model	Name	Attribute	Spatial Relation	Visual Relation	Semantic Relation	Affordance	Multi-hop	Avg.
PolyFormer	-	20.9	13.3	2.6	0.8	3.1	6.7	8.3	8.0
SEEM	-	23.3	10.1	4.6	10.5	10.2	7.9	17.5	12.1
ReCLIP	CLIP	36.9	40.0	12.7	14.2	20.1	23.0	34.0	25.9
RedCircle	CLIP	33.3	21.1	19.7	15.4	18.8	24.0	47.4	25.7
FDVP	CLIP	25.1	19.0	23.7	25.2	12.3	22.5	22.8	21.6
SoM	GPT-4v	40.1	25.0	23.3	40.3	42.5	60.0	21.2	36.1
OWG (Ours)	GPT-4v	83.3	80.1	45.7	55.4	78.8	90.3	59.4	70.4

Table 3: Segmentation - mIoU(%) results in different language input types for cluttered indoor scenes from OCID.

to **grayscale** or **blurring** the rest of the frame (as proposed in FGVP), as well as a prompt that we discover ourselves works good, using a **white background** for the rest of the frame. We note that in our paper's results the element combinations we used are the following:
ReCLIP: rectangle prompt, multi-templates, blur-reverse + crop,
RedCircle: ellipse prompt, multi-templates, gray-reverse + blur-reverse,
FGVP: mask prompt, multi-templates, gray-reverse + blur-reverse

Ablation results are shown in Table 2. Our findings are the following: 1) drawing a rectangle prompt outperforms ellipse and mask (object contours) in itself, but ensembling rectangles and masks gives the best result, 2) using multiple text templates outperforms single-template only when ensembling multiple visual inputs, c) the most effective component is our method of replacing the rest of the frame with white background, compared to grayscale and reverse operators of FGVP, while ensembling all together gives the best performance. We call our custom FGVP baseline FGVP*. We present analytical results per query type for CLIP-based baselines versus GPT-4v methods, as in the original paper, for our extended evaluation dataset in Table **??**. FGVP*represents the best configuration of CLIP-based visual prompting as found by our ablation experiments. Results follow similar patterns to the smaller subset of the main paper, with a significant performance boost for CLIP-based baselines. However, GPT-4v-based methods still compare favourably to CLIP-based baselines, even in the SoM setting where single marked image is used. Our OWG visual prompt scheme dramatically outperforms all baselines, with a margin of 27.7% from SoM and 29.0% from the best found CLIP visual prompt methodology, showcasing its superiority in cluttered indoor scenes context as in OCID.

C.3 Instance Segmentation Ablations

We use the checkpoints provided by the authors for UOIS [11] unseen object instance segmentation, as well as the ViRL-RPN checkpoint and hyper-params from the implementation in this HTTP URL. For SAM, we use the ViT-L variant of the released SAM [12] checkpoints, and search for optimal hyper-parameters for automatic mask generator, resulting in the following configuration: `points_per_side=24, pred_iou_thresh=0.92, stability_score_thresh=0.95`. We apply non-maximum suppression with an `iou_threshold=0.5` and remove nested masks, i.e. masks that are completely inside other masks of higher score threshold. This step aids in keeping only object-level SAM predictions and decreasing the over-segmentation behavior that default SAM provided in our first implementation. In turn, this leads to less cluttered visual markers for our OWG grounding module. Example instance segmentation masks for the different methods are illustrated in Fig. 4.

D GPT-4v Example Responses

In Figs. 5, 6, 7, we provide example responses for grounding different types of language queries in OCID scenes. We observed that GPT-4v, augmented with marked image prompting, can ground not just object-related queries but also complex referring expressions that require reasoning about space, visual attributes, semantics and user-affordances. Interestingly, we find that GPT-4v responds

to queries that require symbolic reasoning concepts such as counting and negation, which are notoriously hard to emerge in specialist grounding models. In Fig. 8, we provide some example responses corresponding to failure cases. Main failure modes include: a) grounding a distractor instead of the desired object, b) not finding the object of interest at all, c) providing a correct reasoning and identifying the target in the raw image, but providing a wrong ID of an irrelevant object.

References

[1] J. Yang, H. Zhang, F. Li, X. Zou, C. yue Li, and J. Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *ArXiv*, abs/2310.11441, 2023. URL https://api.semanticscholar.org/CorpusID:266149987.

[2] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. *ArXiv*, abs/2205.11916, 2022. URL https://api.semanticscholar.org/CorpusID:249017743.

[3] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. H. hsin Chi, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171, 2022. URL https://api.semanticscholar.org/CorpusID:247595263.

[4] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

[5] X. Gu, T.-Y. Lin, W. Kuo, and Y. Cui. Open-vocabulary object detection via vision and language knowledge distillation. In *International Conference on Learning Representations*, 2021. URL https://api.semanticscholar.org/CorpusID:238744187.

[6] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. URL https://arxiv.org/abs/2103.00020.

[7] S. Kumra, S. Joshi, and F. Sahin. Gr-convnet v2: A real-time multi-grasp detection network for robotic grasping. *Sensors (Basel, Switzerland)*, 22, 2022. URL https://api.semanticscholar.org/CorpusID:251706781.

[8] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. R. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:250451569.

[9] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022. URL https://api.semanticscholar.org/CorpusID:252762395.

[10] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask r-cnn. 2017. URL https://api.semanticscholar.org/CorpusID:54465873.

[11] C. Xie, Y. Xiang, A. Mousavian, and D. Fox. Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics*, 37:1343–1359, 2020. URL https://api.semanticscholar.org/CorpusID:220546289.

[12] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. B. Girshick. Segment anything. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3992–4003, 2023. URL https://api.semanticscholar.org/CorpusID:257952310.

You are highly skilled in robotic task planning, identifying what object to grasp from a given user's instruction and planning how to grasp it successfully. If the object is in sight, you need to directly manipulate it. If the object is not in sight, you need to use tools to find the object first. If the target object is overlapping with other objects, you need to remove all the blocking objects before picking up the target object. The overlap condition requires that: For two bounding boxes [x1_a, y1_a, x2_a, y2_a] and [x1_b, y1_b, x2_b, y2_b], they overlap if:
x1_a <= x2_b and x2_a >= x1_b
y1_a <= y2_b and y2_a >= y1_b
Remember your last step plan needs to be "done".

Consider the following tools the robot can use:
1. visual_grounding <list of object IDs> <text description to be grounded> (e.g. visual_grounding [3,4,7,11] 'blue and green'). The visual grounding tool should be used to determine which of the objects in the input ID list best matches the text description.
2. remove <object ID> (e.g remove [6]). The remove tool should be used to pick blocking objects and move them to free space, in order to make the target object more graspable.
3. pick <object ID> (e.g pick [1]). The pick tool should be used to pick the target object and give it to the user.
You are only allowed to use the provided skills. It's essential to stick to the format of these basic skills. When creating a plan, replace the object ID placeholders with specific object instances using their unique ID in square brackets as in the examples above.

Use the following two examples as a demonstration of how you should interact with the tools to control the robot and understand what the response format should be like.

Start of episode
User Input: "It's dark around here, I need some light".

Observation:
'ID': 1, 'name': 'binder', 'x1y1x2y2': [0.49, 0.11, 0.68, 0.49]
'ID': 2, 'name': 'cereal box', 'x1y1x2y2': [0.2, 0.27, 0.42, 0.48]
'ID': 3, 'name': 'food bag', 'x1y1x2y2': [0.66, 0.6, 0.78, 0.81]
'ID': 4, 'name': 'food can', 'x1y1x2y2': [0.3, 0.66, 0.38, 0.78]
'ID': 5, 'name': 'cereal box', 'x1y1x2y2': [0.23, 0.09, 0.47, 0.31]
'ID': 6, 'name': 'kleenex', 'x1y1x2y2': [0.37, 0.47, 0.55, 0.61]
'ID': 7, 'name': 'flashlight', 'x1y1x2y2': [0.45, 0.79, 0.48, 0.95]
'ID': 8, 'name': 'marker', 'x1y1x2y2': [0.47, 0.73, 0.59, 0.85]
'ID': 9, 'name': 'glue stick', 'x1y1x2y2': [0.45, 0.63, 0.48, 0.7]
#Robot thought: The user mentions "It's dark around here, I need some light", indicating that they would like to use a tool that can act as a light source, such as a light bulb, a lamp or a flashlight. A flashlight exists in the scene with ID [7], so I should pick it to provide it to the user. After inspecting the location coordinates of flashlight [7] compared to other objects, it appears that there is an overlap with the marker [8]. x1=0.47 for marker [8] is between x1=0.45 and x2=0.48 of flashlight [7], and y2=0.85 for marker [8] is between y1=0.79 and y2=0.95 of flashlight [7]. This indicates that directly picking flashlight [7] could lead to a failure, so I should first remove the neighboring marker [8].
#Robot action: remove [8]
Observation: You failed to grasp object [8]
#Robot thought: I failed to grasp the marker [8], so I should try again.
#Robot action: remove [8]
Observation: You successfully removed object [8]
#Robot thought: Since the target (flashlight [7]) is now available, I should pick it to give it to the user.
#Robot action: pick [7]
Observation: You successfully picked object [7]
#Robot thought: I have completed the task.
#Robot action: done

End of episode

Start of episode
User Input: "get the corn flakes next to the gray keyboard".

Observation:
'ID': 1, 'name': 'cereal box', 'x1y1x2y2': [0.57, 0.28, 0.82, 0.47]
'ID': 2, 'name': 'marker', 'x1y1x2y2': [0.39, 0.44, 0.7, 0.64]
'ID': 3, 'name': 'flashlight', 'x1y1x2y2': [0.38, 0.75, 0.49, 0.85]
'ID': 4, 'name': 'cereal box', 'x1y1x2y2': [0.24, 0.46, 0.45, 0.69]
'ID': 5, 'name': 'keyboard', 'x1y1x2y2': [0.6, 0.8, 0.95, 0.97]
'ID': 6, 'name': 'keyboard', 'x1y1x2y2': [0.19, 0.56, 0.32, 0.67]
'ID': 7, 'name': 'shampoo', 'x1y1x2y2': [0.74, 0.48, 0.81, 0.6]
'ID': 8, 'name': 'unknown', 'x1y1x2y2': [0.62, 0.64, 0.71, 0.74]
'ID': 9, 'name': 'kleenex', 'x1y1x2y2': [0.46, 0.2, 0.56, 0.41]
'ID': 10, 'name': 'soda can', 'x1y1x2y2': [0.32, 0.69, 0.38, 0.71]
'ID': 11, 'name': 'ball', 'x1y1x2y2': [0.55, 0.57, 0.63, 0.69]
'ID': 12, 'name': 'hand towel', 'x1y1x2y2': [0.17, 0.71, 0.35, 0.95]
'ID': 13, 'name': 'stapler', 'x1y1x2y2': [0.53, 0.7, 0.57, 0.84]
'ID': 14, 'name': 'ball', 'x1y1x2y2': [0.6, 0.7, 0.67, 0.79]
'ID': 15, 'name': 'food bag', 'x1y1x2y2': [0.17, 0.41, 0.28, 0.58]
'ID': 16, 'name': 'food can', 'x1y1x2y2': [0.65, 0.32, 0.71, 0.4]
'ID': 17, 'name': 'sponge', 'x1y1x2y2': [0.35, 0.59, 0.42, 0.7]
#Robot thought: The user mentions "get the corn flakes next to the gray keyboard", so I should: 1) identify the gray keyboard, 2) identify the corn flake objects, 3) filter the one whose next to the gray keyboard, 4) pick the filtered corn flakes. There are two keyboard items in the scene ([5], [6]), I should find which one is the gray one. I will invoke the visual grounding tool to determine which one of the two keyboards is gray.
#Robot action: visual_grounding [5,6] `gray'
Observation: Keyboard [6] is gray.
#Robot thought: I identified the gray keyboard [6]. Now I have to identify the corn flake objects. There are two cereal box items in the scene ([1], [4]). I have to look at their location coordinates and identify which one is next to the keyboard.
Keyboard [6] coordinates are: (x1=0.19, y1=0.56, x2=0.55,y2=0.76), so the center would be: (x0=0.255, y0=0.66)
Cereal box [1] coordinates are: (x1=0.57, y1=0.28, x2=0.82, y2=0.47), so the center would be: (x0=0.695, y0=0.375)
Cereal box [4] coordinates are: (x1=0.24, y1=0.46, x2=0.45, y2=0.69), so the center would be: (x0=0.345, y0=0.575)
Therefore, the cereal box that is closest to the keyboard [6] is cereal box [4]. After inspecting the location coordinates of cereal box [4] compared to other objects, it appears that there is an overlap with four other objects:
Marker [2] coordinates are: (x1=0.39, y1=0.44, x2=0.7,y2=0.64) which are overlapping with cereal box [4] coordinates: (x1=0.24, y1=0.46, x2=0.45, y2=0.69).
Keyboard [6] coordinates are: (x1=0.19, y1=0.56, x2=0.55,y2=0.76) which are overlapping with cereal box [4] coordinates: (x1=0.24, y1=0.46, x2=0.45, y2=0.69).
Soda can [10] coordinates are: (x1=0.32, y1=0.69, x2=0.38,y2=0.71) which are overlapping with cereal box [4] coordinates: (x1=0.24, y1=0.46, x2=0.45, y2=0.69).
Sponge [17] coordinates are: (x1=0.35, y1=0.59, x2=0.42,y2=0.7) which are overlapping with cereal box [4] coordinates: (x1=0.24, y1=0.46, x2=0.45, y2=0.69).
This indicates that directly picking cereal box [4] could lead to a failure, so I should first remove the neighbouring objects ([2], [6], [10], [17]).
#Robot action: remove [2]
Observation: You successfully removed object [2]
#Robot thought: The marker [2] is out of the way. I will proceed with removing the keyboard [6].
#Robot action: remove [6]
Observation: You failed to grasp object [6]
#Robot thought: I failed to grasp the keyboard [6], so I should try again.
#Robot action: remove [6]
Observation: You failed to grasp object [6]
#Robot thought: I failed to grasp the keyboard [6], so I should try again.
#Robot action: remove [6]
Observation: You successfully removed object [6]
#Robot thought: The marker [2] and the keyboard [6] are out of the way. I will proceed with removing the soda can [10].
#Robot action: remove [10]
Observation: You successfully picked object [10]
#Robot thought: The marker [2], the keyboard [6] and the soda can [10] are out of the way. I will proceed with removing the sponge [17].
#Robot action: remove [17]
Observation: You failed to grasp object [17]
#Robot thought: I failed to grasp the sponge [17], so I should try again.
#Robot action: remove [17]
Observation: You successfully picked object [17]
#Robot thought: The marker [2], the keyboard [6], the soda can [10] and the sponge [17] are out of the way. Since the target (cereal box [4]) is now available, I should pick it to give it to the user.
#Robot action: pick [4]
Observation: You successfully picked object [4]
#Robot thought: I have completed the task.
#Robot action: done

End of episode

Now begin:
Start of episode
User Input: "{user_input}".

{episode_history}
Observation: {obs}
#Robot thought:"""

Figure 2: System prompt and in-context examples for the LLM in the SayCan-IM baseline. The input field `user_input` corresponds to the task given by the user, `obs` contains the scene state or feedback from actions that is updated in each episode step, and `episode_history` the previous observation-thought-action cycles in the current episode.

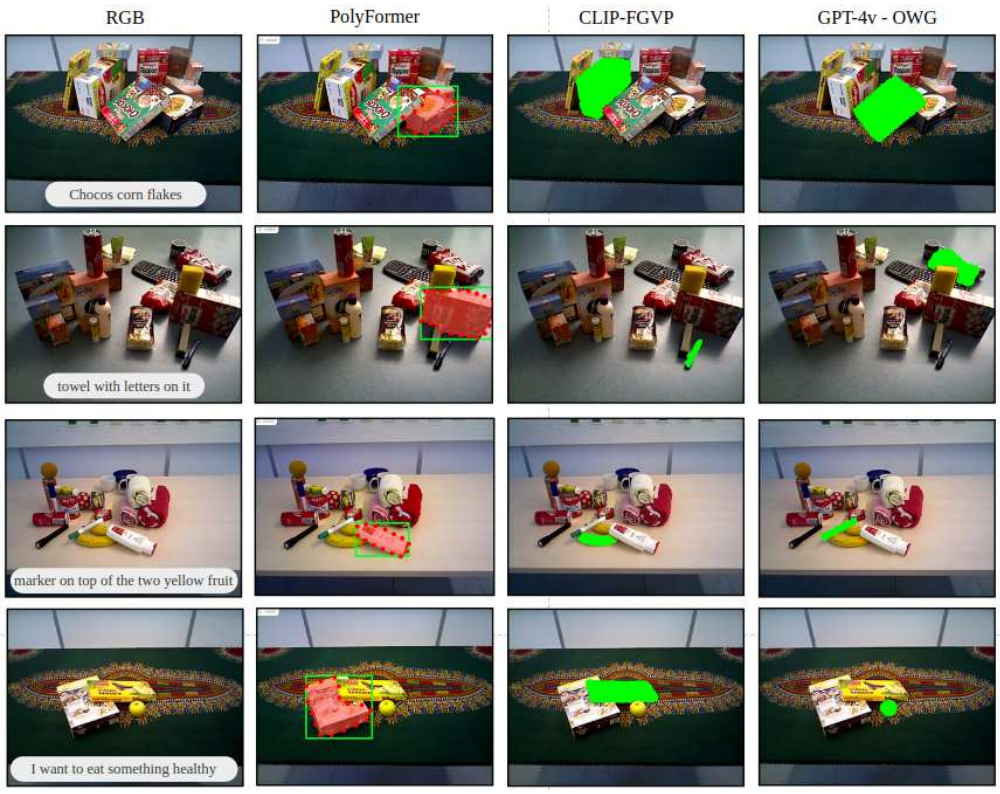


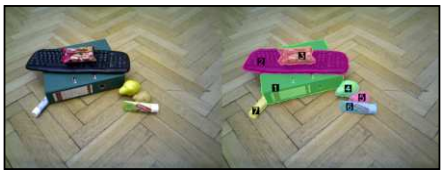
Figure 3: Example output segmentation masks of different grounding methods for OCID scenes.

Figure 4: Comparison of different segmentation models for seven example cluttered indoor scenes from OCID dataset

Object Brand Name

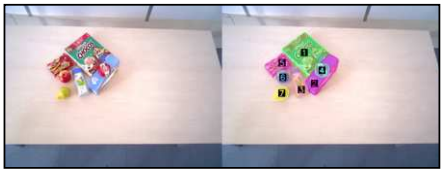
User: Feh package
GPT-4v: To find the object that matches the description "Feh package," I will first look at the raw image to identify any packages that could potentially match the description. Upon examining the raw image, I notice a package with the text "Feh" on it. This package is located towards the left side of the image, standing upright, and appears to be a box of some sort. Next, I will look at the marked image to find the same package and determine its unique ID. The package with the text "Feh" on it in the raw image corresponds to the package in the marked image with the ID [12] on it. The package is in the same position in both images, confirming that this is the correct object.
My final answer is: [12].

Object Attribute: Maturity



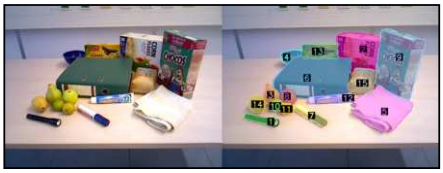
User: non ripe pear
GPT-4v: To find the 'non ripe pear,' I will first identify the pears in the raw image. There are two pears visible on the floor. To determine which one is not ripe, I will look for visual cues such as color. Typically, a non-ripe pear may have a greener and firmer appearance compared to a ripe one, which would be more yellow and possibly softer looking. In the raw image, the pear on the left appears greener and is likely the non-ripe pear, while the pear on the right seems more yellow and could be considered ripe. Now, I will look at the marked image to find the ID corresponding to the greener pear. In the marked image, the greener pear on the left has the ID [4]. Therefore, this is the ID that corresponds to the 'non ripe pear.'
My final answer is: [4]

Object Content Type



User: noodles
GPT-4v: To find the object that matches the description "noodles," I will first examine the raw image to identify any items that could be noodles. Upon inspection, I see a package with an image that appears to represent noodles on the left side of the table. Next, I will look at the marked image to find the same package and determine its unique ID. The package with the noodles is highlighted with a unique color and is associated with the ID [5].
My final answer is: [5].

Object Attribute: Colors

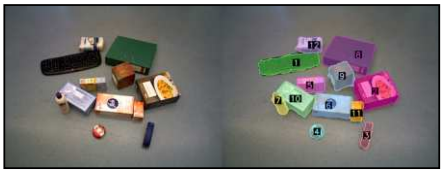


User: green and red corn flakes
GPT-4v: To find the "green and red corn flakes," I will first look at the raw image to identify the box of corn flakes. In the image, there are two cereal boxes visible. One of them has a predominantly green color scheme with some red elements, which matches the description of "green and red corn flakes." This box is on the right side of the image, next to another cereal box with a blue and yellow color scheme. Now, I will switch to the marked image to find the same box of corn flakes with the corresponding ID. The box with the green and red color scheme is highlighted with the same color as the ID number 9.
My final answer is: [9].

Object Functionality

User: red soft thing for the hair
GPT-4v: To find the object that matches the description "red soft thing for the hair," I will follow these steps: 1. Identify any red objects in the raw image, 2. Determine which of these red objects could be described as "soft" and used for the hair. In the raw image, there is a red object with white text that appears to be a towel, which is typically soft and can be used for drying hair. This object is located in the center of the image, draped over other items. Now, I will look at the marked image to find the same object and its corresponding ID. The red towel in the marked image has been highlighted with a unique color and is labeled with the ID number 4.
My final answer is: [4].

Object Attribute: Shape



User: cube
GPT-4v: To find the object that matches the description "cube," I will first look at the raw image to identify any objects that have a cubic shape. In the raw image, there is an object that appears to be a cube towards the right side of the image, slightly above the center. It has a rusted appearance and seems to be a cube due to its equal-length edges and square faces. Now, I will look at the marked image to find the same object with its corresponding ID. The object with the cubic shape in the same location is highlighted with a color and has the ID number 9 next to it.
My final answer is: [9]

Figure 5: Example GPT-4v responses (1/3): object-related queries

Relations: Relative Location w.r.t Object

User: food box under the choco-banana

GPT-4v: To find the "food box under the choco-banana," I will first identify the choco-banana in the raw image. The choco-banana appears to be a box with an image of a banana and chocolate on it, located towards the bottom right of the image. Now, I will look for any food box that is positioned under it. In the raw image, there is a box directly underneath the choco-banana box. This box is likely the "food box" mentioned in the description. To find the corresponding ID, I will now switch to the marked image and look for the same box under the choco-banana box, which is labeled with ID 8. Directly underneath the choco-banana box with ID 8, there is another box highlighted with a different color and labeled with ID 1. This is the food box that is under the choco-banana.
My final answer is: [1].

Relations: Absolute Location

User: leftmost toothpaste product

GPT-4v: To find the "leftmost toothpaste product," I will first identify any toothpaste products in the raw image. Looking at the raw image, I can see two items that resemble toothpaste tubes. They are located towards the center of the image, one standing upright and the other lying down to its right. Now, I will switch to the marked image to find the corresponding IDs for these toothpaste products. The leftmost toothpaste product, which is standing upright, has a highlighted color and an ID next to it. The ID next to this toothpaste product is [10].
My final answer is: [10].

Relations: Same Attribute

User: mug of same color as the keyboard

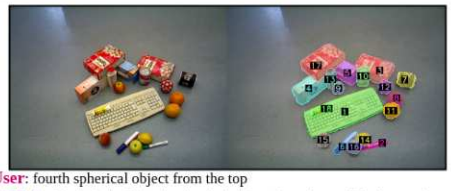
GPT-4v: To find the mug of the same color as the keyboard, I will first identify the color of the keyboard in the raw image. The keyboard appears to be white. Next, I will look for a mug that is also white. In the raw image, there are two mugs on the table to the left of the keyboard. One of these mugs is white, which matches the color of the keyboard. Now, I will locate this white mug in the marked image to find its corresponding ID. In the marked image, the white mug is highlighted with a unique color and has the ID number next to it. The white mug has the ID [11] next to it.
My final answer is: [11].

Relations: Superlative Attribute

User: thickest marker

GPT-4v: To find the "thickest marker," I will first identify all the markers in the raw image. There are two markers visible: one with a blue cap and one with a green cap. Between these two, the marker with the blue cap appears to be thicker than the one with the green cap. Now, I will look at the marked image to find the ID corresponding to the thicker marker with the blue cap. In the marked image, the thicker marker with the blue cap is highlighted and has the ID number next to it. The thicker marker with the blue cap is associated with the ID [6].
My final answer is: [6].

Multi-Hop Reasoning: Counting



User: fourth spherical object from the top

GPT-4v: To find the "fourth spherical object from the top" in the raw image, I will first identify all the spherical objects and then count from the top of the image down to the fourth one. In the raw image, starting from the top, I see several spherical objects which appear to be fruits and a ball. Counting these objects from the top, the first spherical object is an orange, the second is another orange, the third is the red ball, and the fourth is yet another orange. Now, I will look at the marked image to find the ID corresponding to the fourth spherical object from the top, which is the orange. In the marked image, this orange is highlighted with a color and has the ID number next to it. Upon examining the marked image, I can see that the fourth spherical object from the top, the orange, has the ID number 11 next to it.
My final answer is: [11].

Multi-Hop Reasoning: Negation

User: shampoo that is not standing upright

GPT-4v: To find the shampoo that is not standing upright, I will first look at the raw image to identify any objects that resemble shampoo bottles. In the raw image, there are two bottles on the table. One is standing upright, and the other is lying on its side. Now, I will look at the marked image to find the ID of the shampoo bottle that is not standing upright. The bottle lying on its side has a unique color and a numeric ID next to it. This ID is [3].
My final answer is: [3].

Figure 6: Example GPT-4v responses (2/3): relational reasoning queries

Semantic Reasoning: Utility

User: something to put my cereal in

GPT-4v: To find "something to put my cereal in," I'm looking for a bowl or a similar container that would be appropriate for holding cereal. In the raw image, I can see a bowl on the left side of the table, near the center of the image. It's placed in front of a cereal box and to the left of a keyboard. Now, I'll look at the marked image to find the same bowl and its corresponding ID. The bowl is in the same location as in the raw image, and it has a unique color and a numeric ID next to it. The ID next to the bowl is [10].
My final answer is: [10].

Semantic Reasoning: Material

User: something that grows organically

GPT-4v: In the raw image, there is an orange, which is an object that grows organically. Now, I will look at the marked image to find the same orange and its corresponding ID. The orange in the marked image is highlighted with a unique color and has the numeric ID 3 next to it.
My final answer is: [3].

User Affordance: State

User: my hands are wet

GPT-4v: To find the object that best matches the description "My hands are wet," I need to identify an object that could be associated with wet hands. In the context of the items present in the image, the most likely candidate would be a towel or a cloth that one would use to dry their hands. Looking at the raw image, I can see a pink towel-like object on the left side of the image. This object seems to be the one that would be used to dry wet hands. Now, I will look at the marked image to find the same towel-like object and its corresponding ID. The pink towel-like object in the marked image has the ID number highlighted in the same color as the object's mask. The ID number for the towel-like object is [18].
My final answer is: [18].

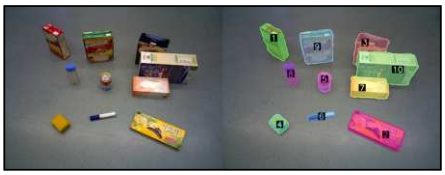
User Affordance: Function

User: I need a toy to release some stress

GPT-4v: To find a toy to release some stress, I will look for an object that is commonly used for stress relief, such as a stress ball or a fidget toy. In the raw image, I can see an object that resembles a ball with a pattern on it, which could potentially be a toy used for stress relief. It is located towards the right side of the image, near the center, and is red with white spots. Now, I will look at the marked image to find the same object and its corresponding ID. The object that matches the description and location in the raw image is highlighted with a unique color and has the ID number next to it. In the marked image, the ball with the red and white pattern has been highlighted and is associated with the ID number [7].
My final answer is: [7].

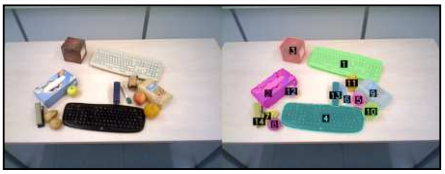
Figure 7: Example GPT-4v responses (3/3): semantics-related queries

User: canned food
GPT-4v: To find the "canned food" in the raw image, I will look for an object that resembles a can or a container typically used for preserving food. In the raw image, I can see a box of what appears to be cereal on the right side, but that is not canned food. However, there is no object that clearly resembles a traditional metal can used for canned food. Since there is no object that matches the description of "canned food" in the raw image, I cannot provide an ID from the marked image. Therefore, there is no label ID to match the description of "canned food" in these images.
My final answer is: there is no label ID for "canned food" in the provided images.



User: corn flakes that is laying on its side

GPT-4v: To find the "corn flakes box that is laying on its side," I will first look at the raw image to identify the box that resembles a corn flakes box and is positioned on its side. In the raw image, there is a box with what appears to be a corn flakes design on the bottom right, and it is indeed laying on its side. Next, I will look at the marked image to find the same box and determine its unique ID. The box that matches the description in the marked image is highlighted with a color and has the number 2 next to it.
My final answer is: [2].



User: stapler next to the marker
GPT-4v: To find the stapler standing next to the marker, I will first identify these objects in the raw image and then locate their corresponding IDs in the marked image. In the raw image, I can see a stapler to the left of a black keyboard, and immediately to the right of the stapler is a marker. Now, I will switch to the marked image to find the IDs that correspond to these objects. In the marked image, the stapler has a unique color and ID next to it, and the marker also has a different color and ID. The stapler is marked with the ID [7], and the marker is right next to it with the ID [6].
My final answer is: [7].

User: blue and black marker
GPT-4v: ITo find the "blue and black marker," I will first look at the raw image to identify any markers that match this description. In the raw image, there are several markers on the table. I can see one marker that appears to have a blue cap and a black body, which is located towards the bottom right of the table. Now, I will switch to the marked image to find the same marker with the corresponding ID. The marker with the blue cap and black body in the marked image has a unique color and ID number next to it. It is the same marker that I identified in the raw image, and it has the ID number [12] next to it.
My final answer is: [12].

Figure 8: Example of GPT-4v response failures.