Robot Perception and Control

LLM for Robotics

Last updated: Jul / 25 /2024

Kashu Yamazaki

kyamazak@andrew.cmu.edu

From Transformers to Foundation Models

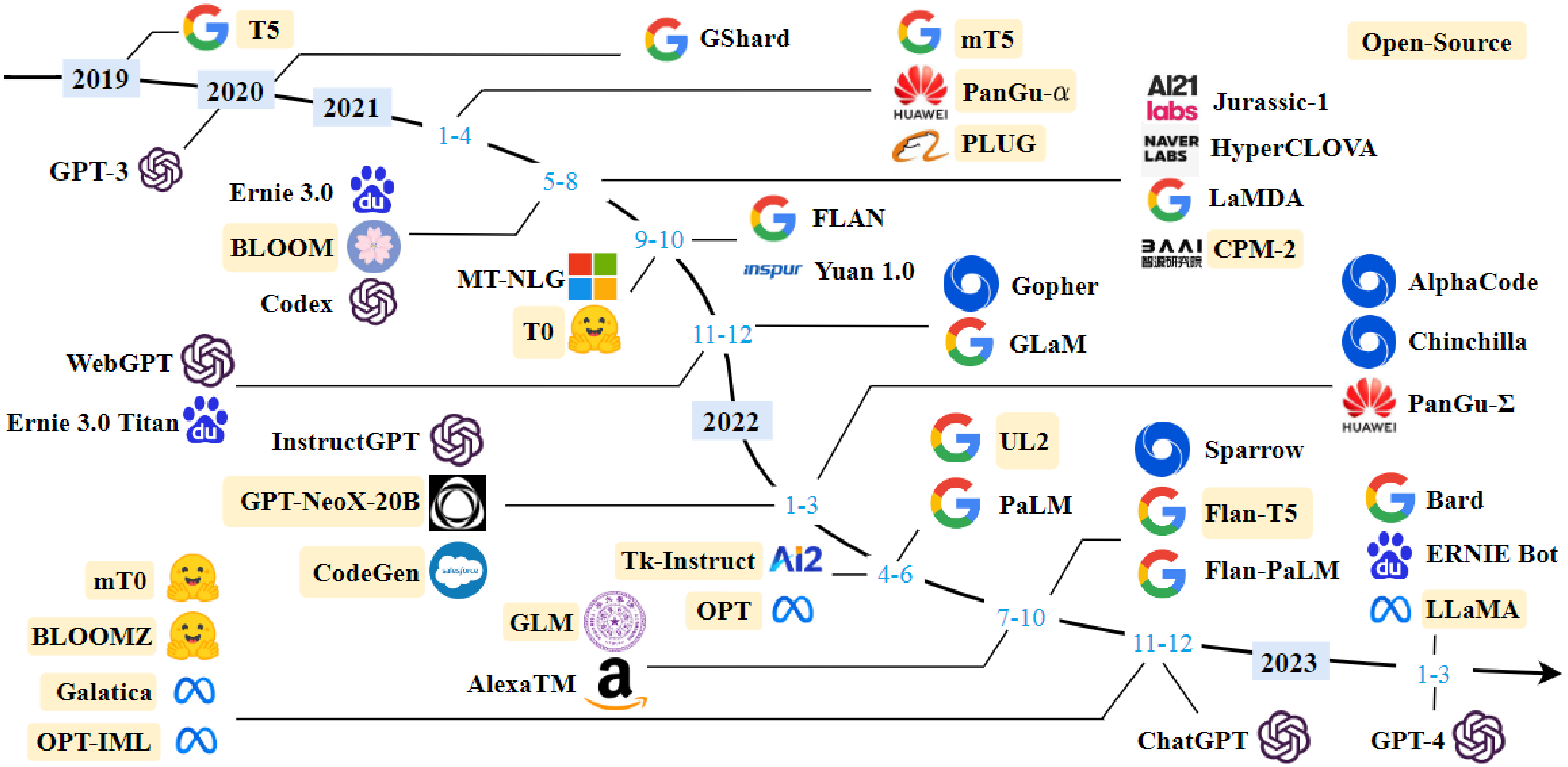


Fig. 1. A timeline of existing large language models (having a size larger than 10B) in recent years. We mark the open-source LLMs in yellow color.

SayCan (1/3)

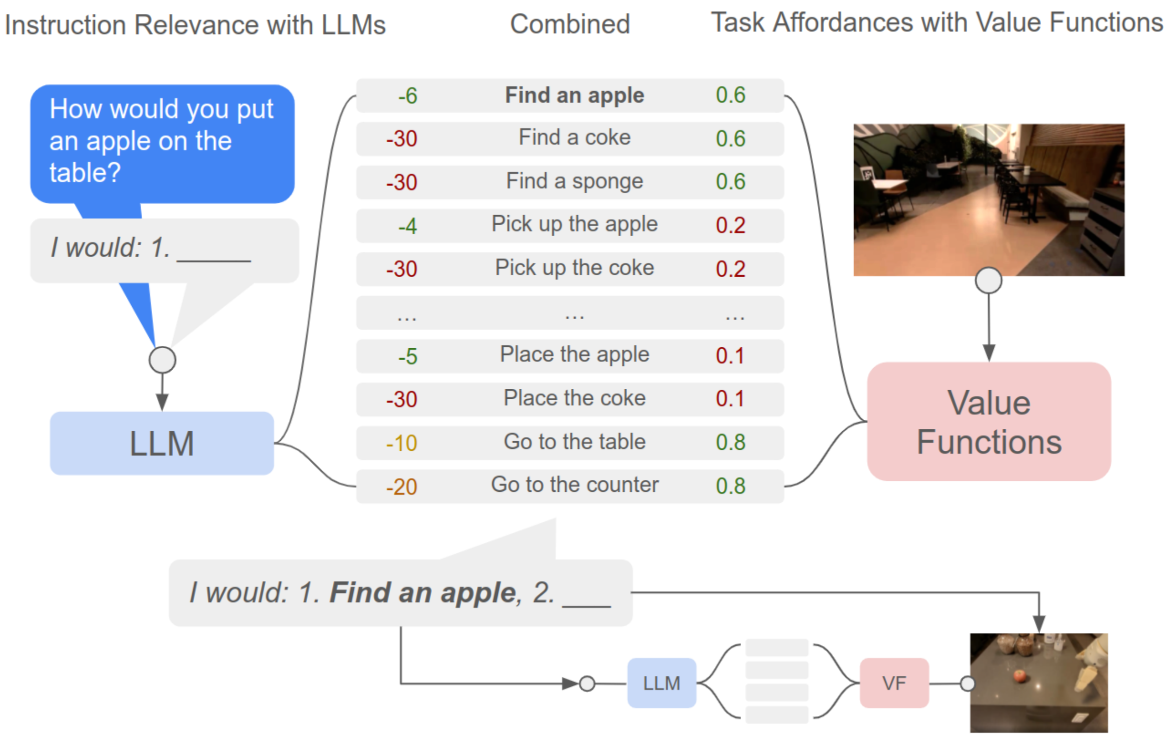
With prompt engineering and scoring we can use LLM to break down an instruction into small, actionable steps. However, the **LLM doesn't know about the scene**, **embodiment** and the **situation it's in**. It needs what is call an affordance function!

- A robotic value functions as a way to provide what's feasible in the world given the current scene and embodiment.
- LLM checks what makes sense to do next given the grand plan, and the value function checks what is currently feasible

SayCan (2/3)

SayCan [1] obtains a skill that is both possible and useful with LLMs by:

- asking the LLM to interpret an instruction and **score the likelihood** that an individual skill makes progress towards completing the high-level instruction.
- a **value function** that represents the probability of successfully executing said skill to select the skill to perform

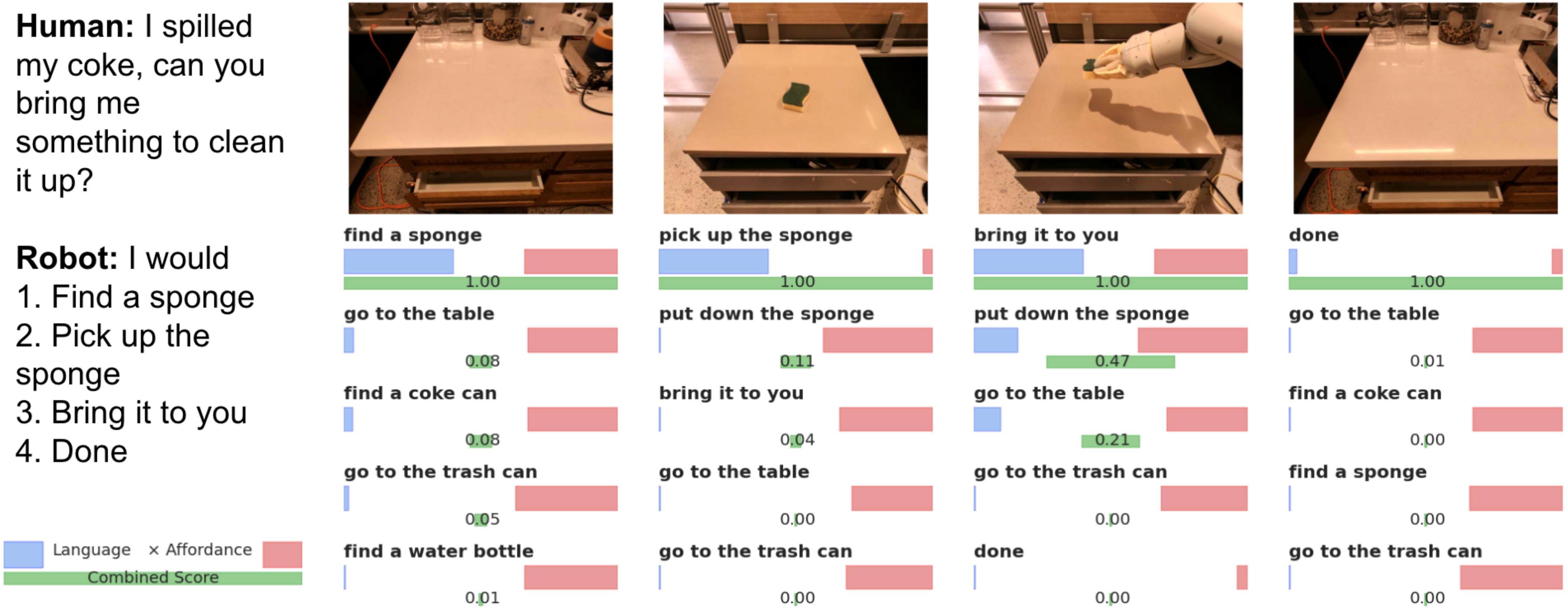

Instruction Relevance with LLMs
Combined
Task Affordances with Value Functions
How would you put an apple on the table?
I would: 1. _____
LLM
Value Functions
I would: 1. Find an apple, 2. ___
LLM
VF

-6	**Find an apple**	0.6
-30	Find a coke	0.6
-30	Find a sponge	0.6
-4	Pick up the apple	0.2
-30	Pick up the coke	0.2
…	…	…
-5	Place the apple	0.1
-30	Place the coke	0.1
-10	Go to the table	0.8
-20	Go to the counter	0.8

SayCan (3/3)

Human: I spilled my coke, can you bring me something to clean it up?

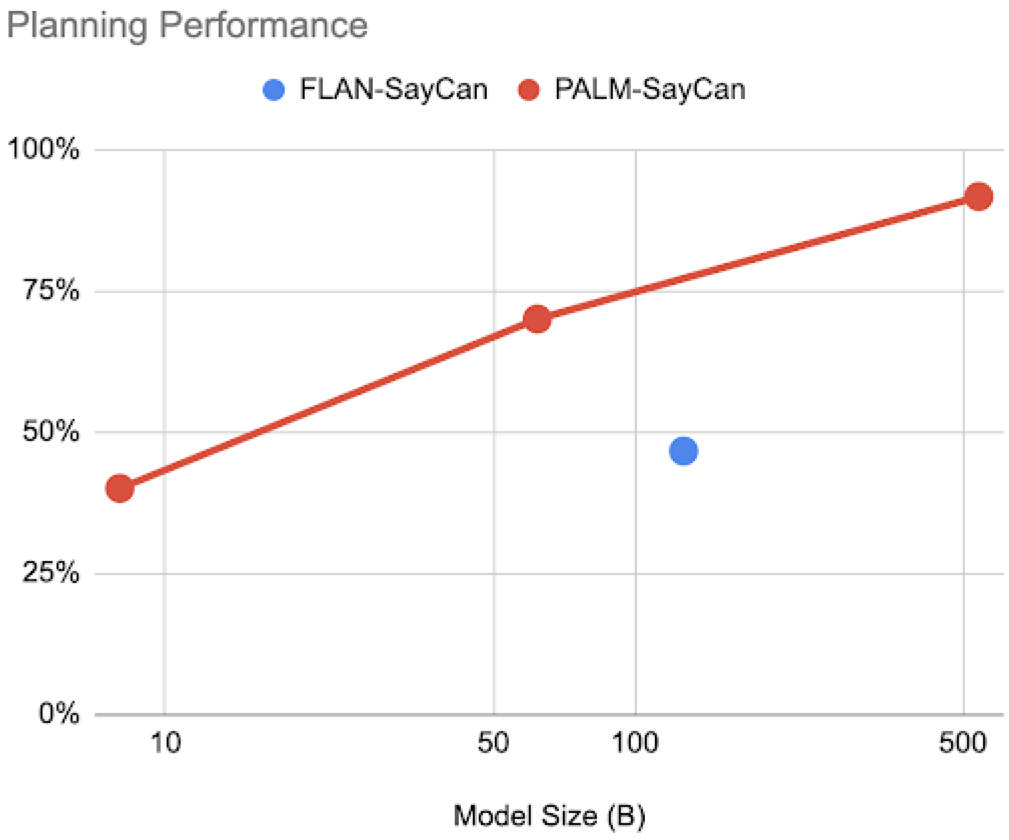
Robot: I would
1. Find a sponge
2. Pick up the sponge
3. Bring it to you
4. Done



PaLM-SayCan

Just by changing the LLM to a more performant PaLM we got:

- better performance
- chain-of-thought prompting
- handling of queries in other languages

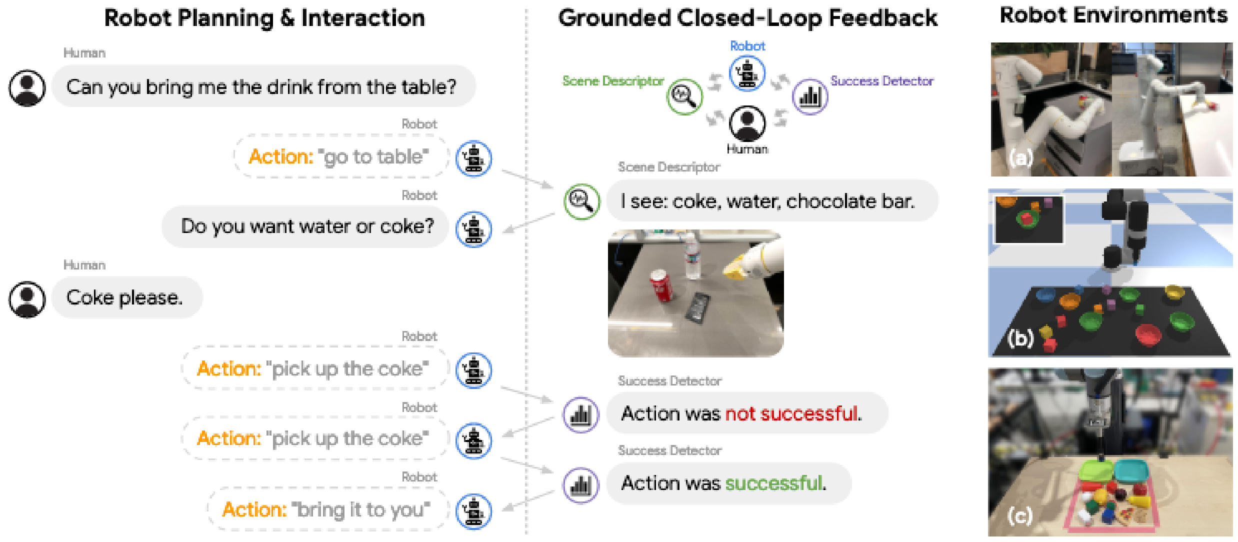


Inner Monologue (1/3)

Inner Monologue [1↑] bring in VLMs to provide feedback about the scene, task success etc.

All these different models talk to each other in natural language so that LLM can understand.

> “ VLMs bring a lot of non-robotic data into our system allowing us to get better planning feedback mechanisms.



Human: Can you bring me the drink from the table?

Robot: Action: "go to table"

Scene Descriptor: I see: coke, water, chocolate bar.

Robot: Do you want water or coke?

Human: Coke please.

Robot: Action: "pick up the coke"

Success Detector: Action was not successful.

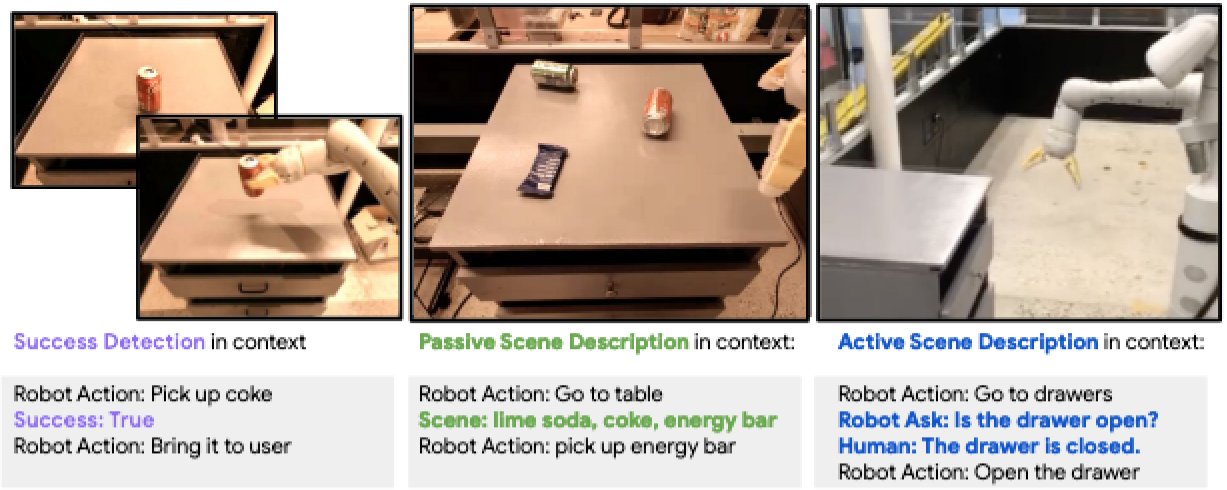
Robot: Action: "pick up the coke"

Success Detector: Action was successful.

Robot: Action: "bring it to you"

Inner Monologue (2/3)

- **Success Detection** gives task-specific task completion information.
- **Passive Scene Description** gives structured semantic scene information at every planning step.
- **Active Scene Description** gives unstructured semantic information only when queried by the LLM planner.



Success Detection in context

Robot Action: Pick up coke
Success: True
Robot Action: Bring it to user

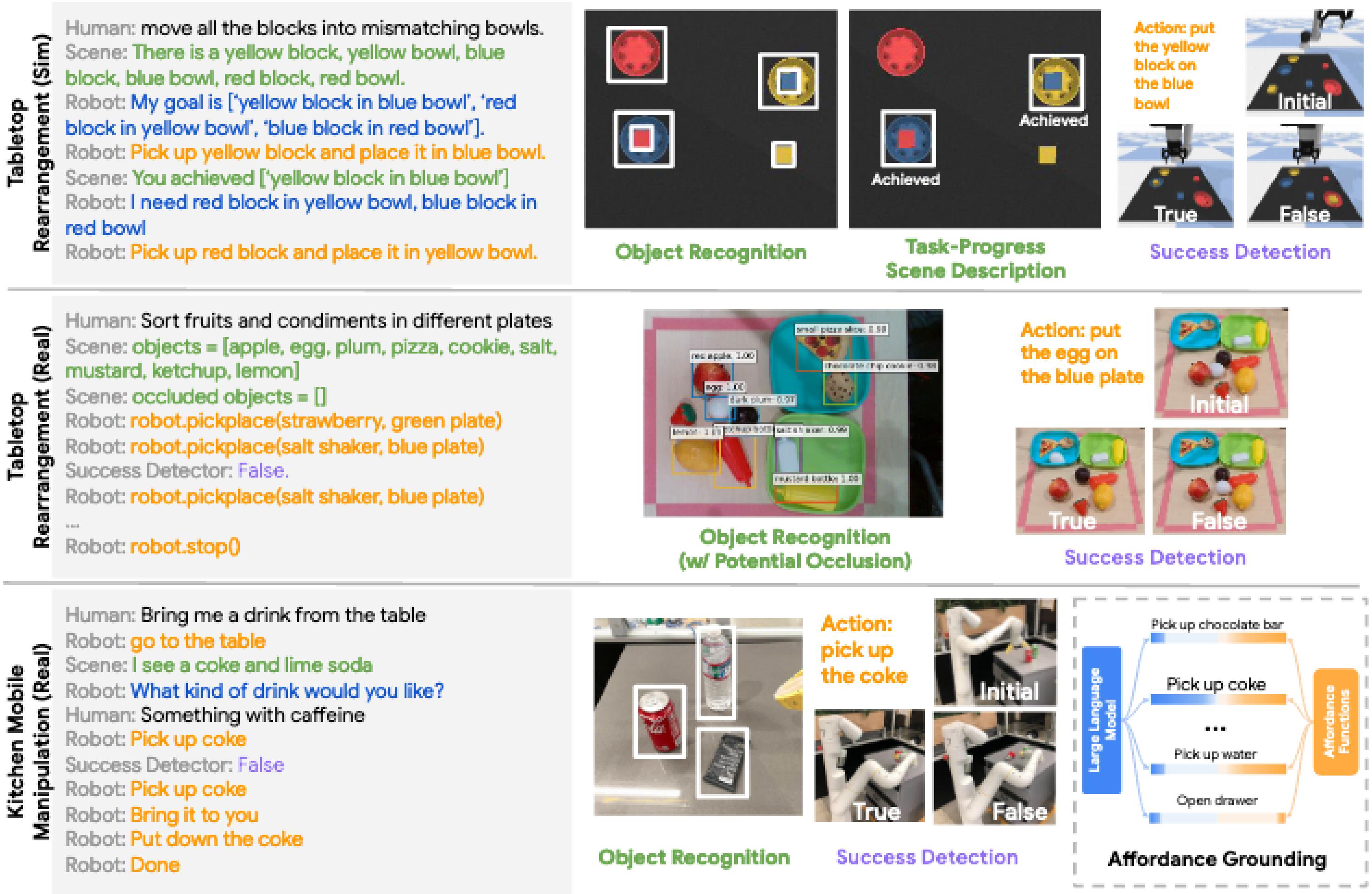
Passive Scene Description in context:

Robot Action: Go to table
Scene: lime soda, coke, energy bar
Robot Action: pick up energy bar

Active Scene Description in context:

Robot Action: Go to drawers
Robot Ask: Is the drawer open?
Human: The drawer is closed.
Robot Action: Open the drawer

Inner Monologue (3/3)



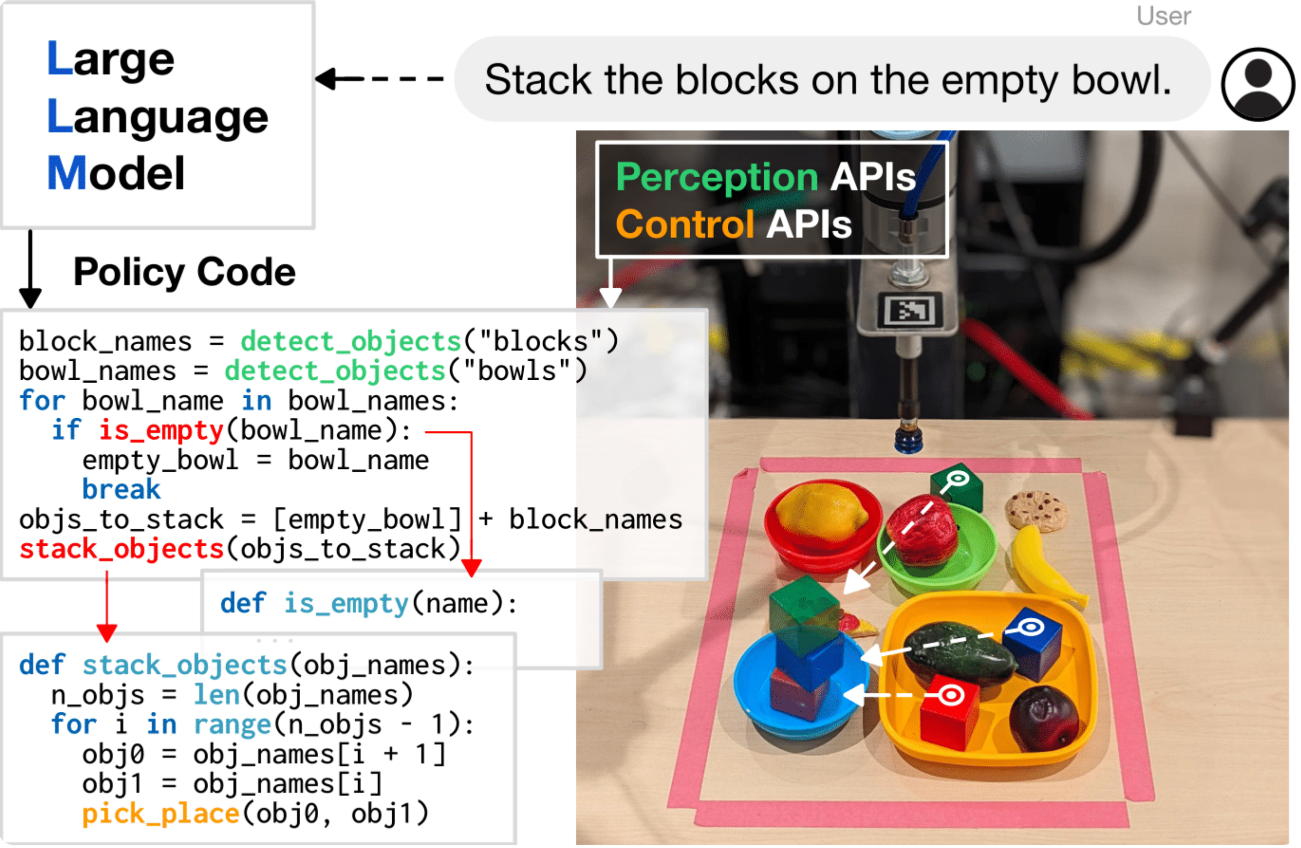
Human: move all the blocks into mismatching bowls.
Scene: There is a yellow block, yellow bowl, blue block, blue bowl, red block, red bowl.
Robot: My goal is ['yellow block in blue bowl', 'red block in yellow bowl', 'blue block in red bowl'].
Robot: Pick up yellow block and place it in blue bowl.
Scene: You achieved ['yellow block in blue bowl']
Robot: I need red block in yellow bowl, blue block in red bowl
Robot: Pick up red block and place it in yellow bowl.

Human: Sort fruits and condiments in different plates
Scene: objects = [apple, egg, plum, pizza, cookie, salt, mustard, ketchup, lemon]
Scene: occluded objects = []
Robot: robot.pickplace(strawberry, green plate)
Robot: robot.pickplace(salt shaker, blue plate)
Success Detector: False.
Robot: robot.pickplace(salt shaker, blue plate)
...
Robot: robot.stop()

Human: Bring me a drink from the table
Robot: go to the table
Scene: I see a coke and lime soda
Robot: What kind of drink would you like?
Human: Something with caffeine
Robot: Pick up coke
Success Detector: False
Robot: Pick up coke
Robot: Bring it to you
Robot: Put down the coke
Robot: Done

Code as Policies

Code as Policies [1 ⮥] used LLMs to generate code to **directly control the robot**.



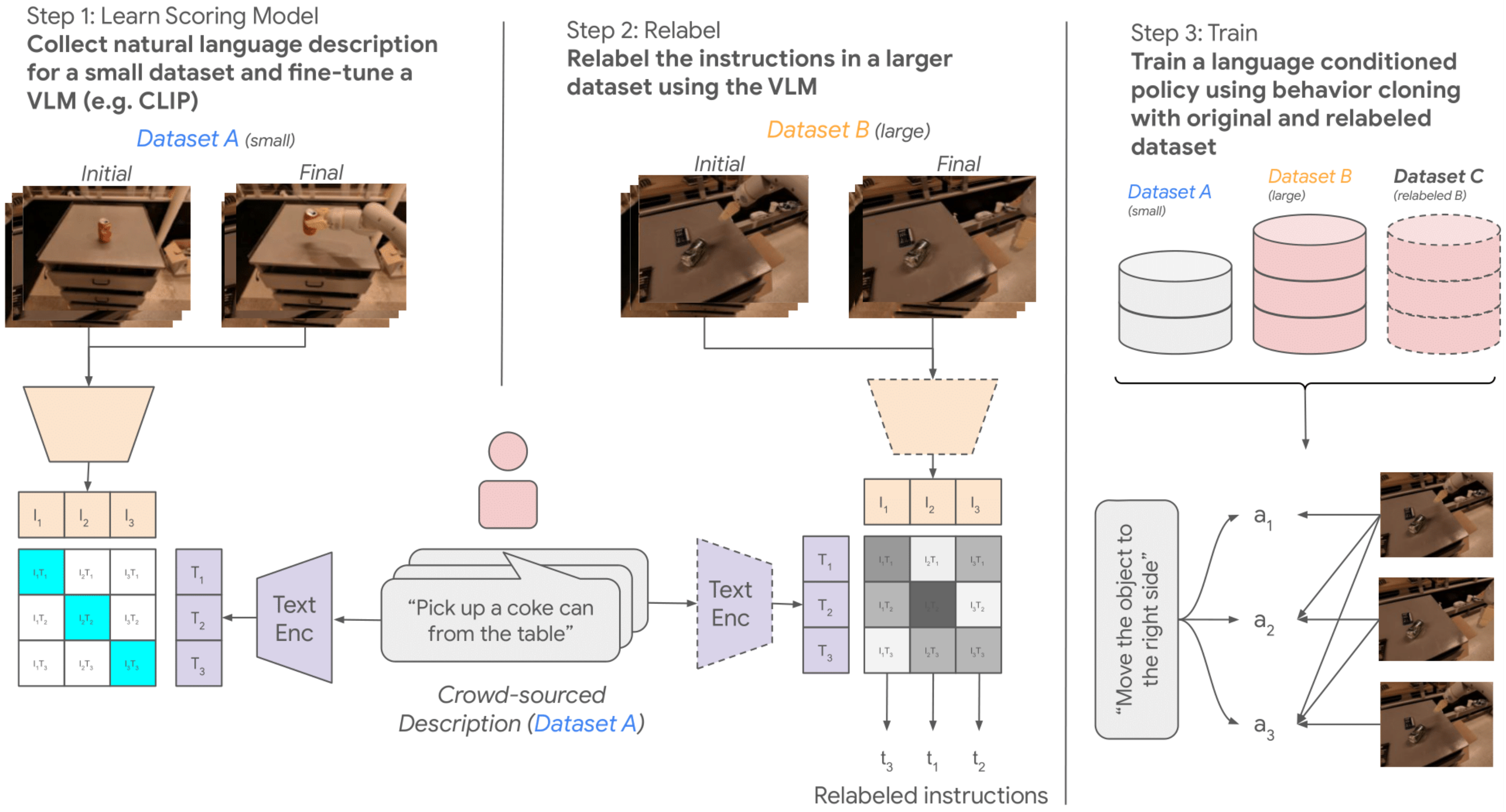
```
block_names = detect_objects("blocks")
bowl_names = detect_objects("bowls")
for bowl_name in bowl_names:
  if is_empty(bowl_name):
    empty_bowl = bowl_name
    break
objs_to_stack = [empty_bowl] + block_names
stack_objects(objs_to_stack)
```

```
def is_empty(name):
  ...
```

```
def stack_objects(obj_names):
  n_objs = len(obj_names)
  for i in range(n_objs - 1):
    obj0 = obj_names[i + 1]
    obj1 = obj_names[i]
    pick_place(obj0, obj1)
```

DIAL

DAIL [1 ⤴] show that VLMs can significantly expand language labels without collecting any additional robot data.



Step 1: Learn Scoring Model
Collect natural language description for a small dataset and fine-tune a VLM (e.g. CLIP)

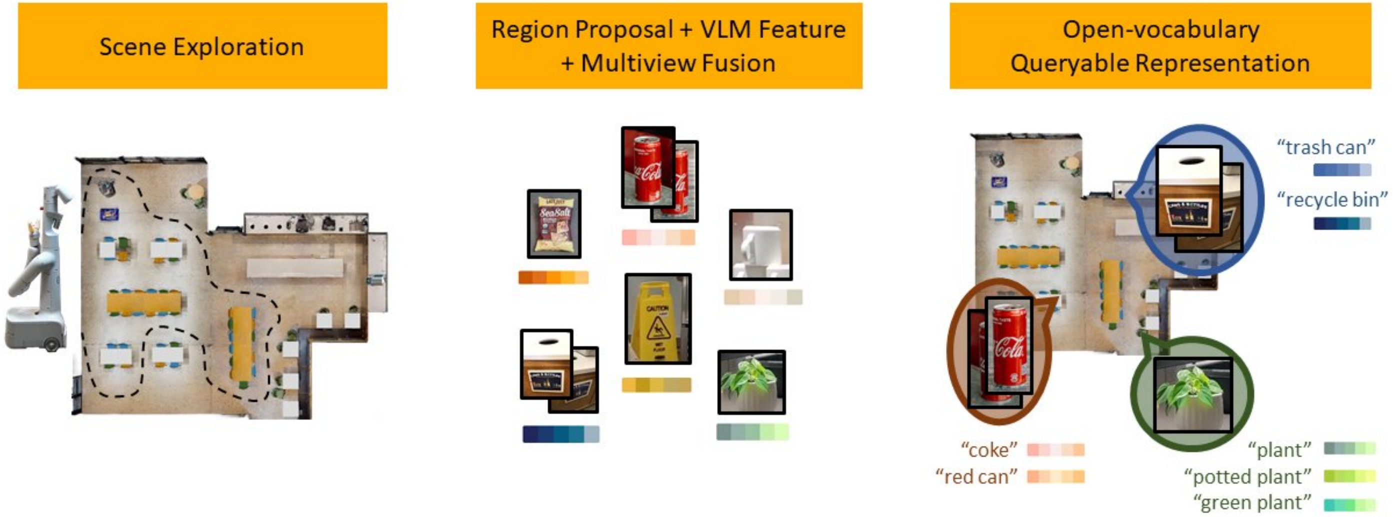
Step 2: Relabel
Relabel the instructions in a larger dataset using the VLM

Step 3: Train
Train a language conditioned policy using behavior cloning with original and relabeled dataset

NLMap (1/4)

NLMap [1] showed VLMs can be used to query objects in the scene and allow for open-vocabulary queries in SayCan. NLMap addresses two core problems:

1. How to maintain open-vocabulary scene representations that are capable of locating arbitrary objects?
2. How to merge such representations within long-horizon LLM planners to imbue them with scene understanding?



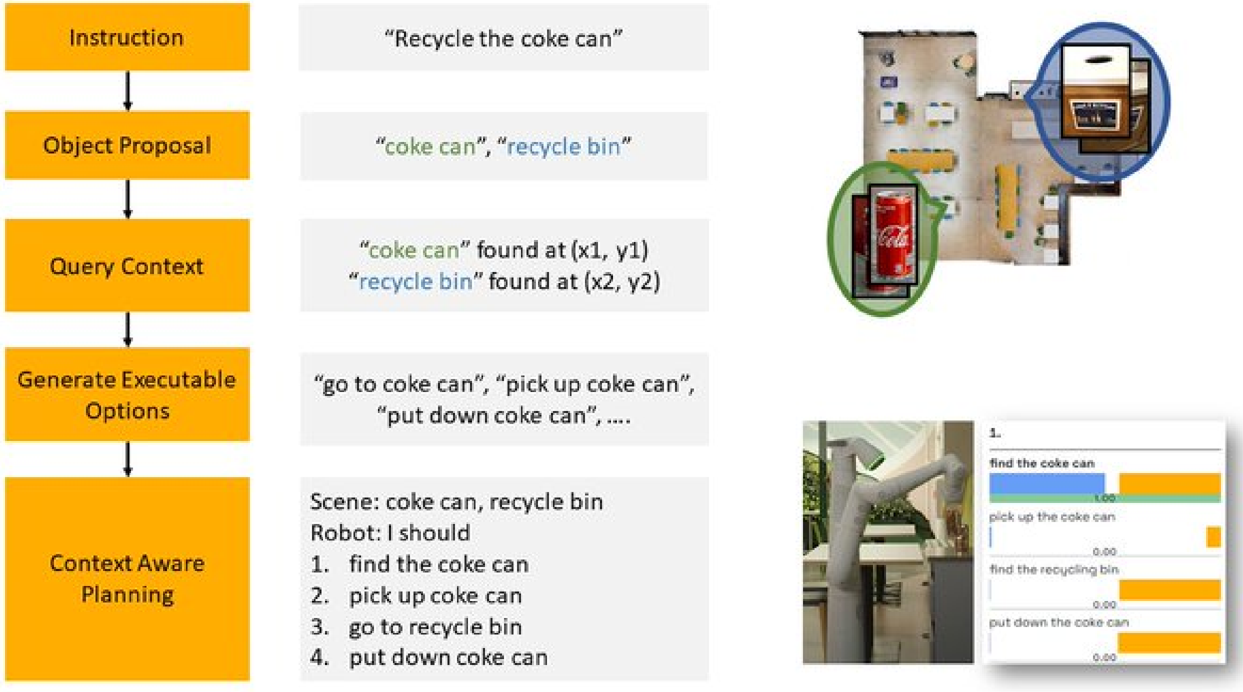
Scene Exploration

Region Proposal + VLM Feature + Multiview Fusion

Open-vocabulary Queryable Representation

NLMap (2/4)

NLMap builds a natural language queryable scene representation with VLMs. An LLM-based object proposal module infers involved objects to query the representation for object availability and location. LLM planner (SayCan) then plans conditioned on such information.



Instruction

"Recycle the coke can"

Object Proposal

"coke can", "recycle bin"

Query Context

"coke can" found at (x1, y1)
"recycle bin" found at (x2, y2)

Generate Executable Options

"go to coke can", "pick up coke can", "put down coke can", ….

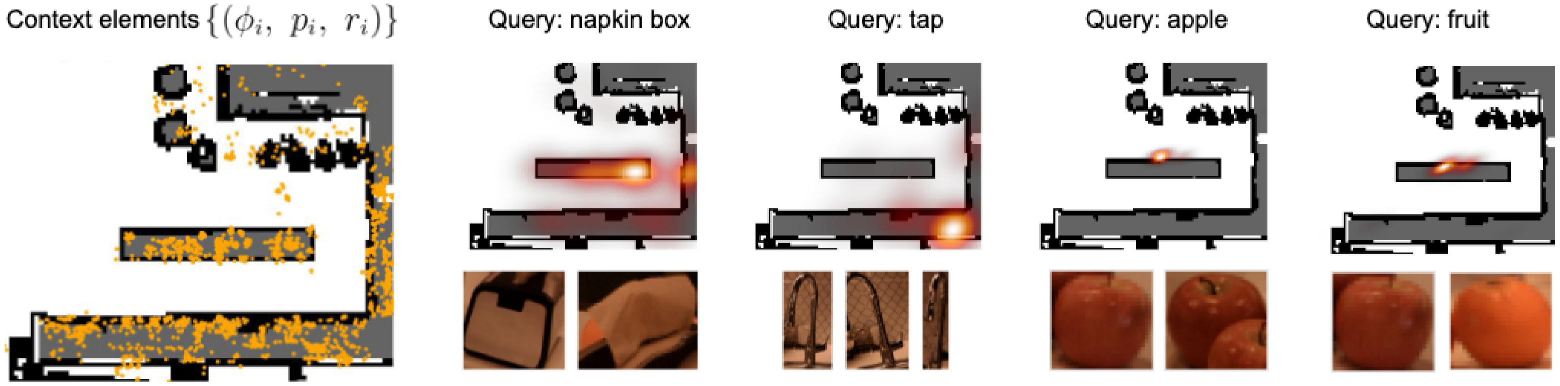
Context Aware Planning

Scene: coke can, recycle bin
Robot: I should
1. find the coke can
2. pick up coke can
3. go to recycle bin
4. put down coke can

NLMap (3/4)

Natural Language Queryable Scene Representation:

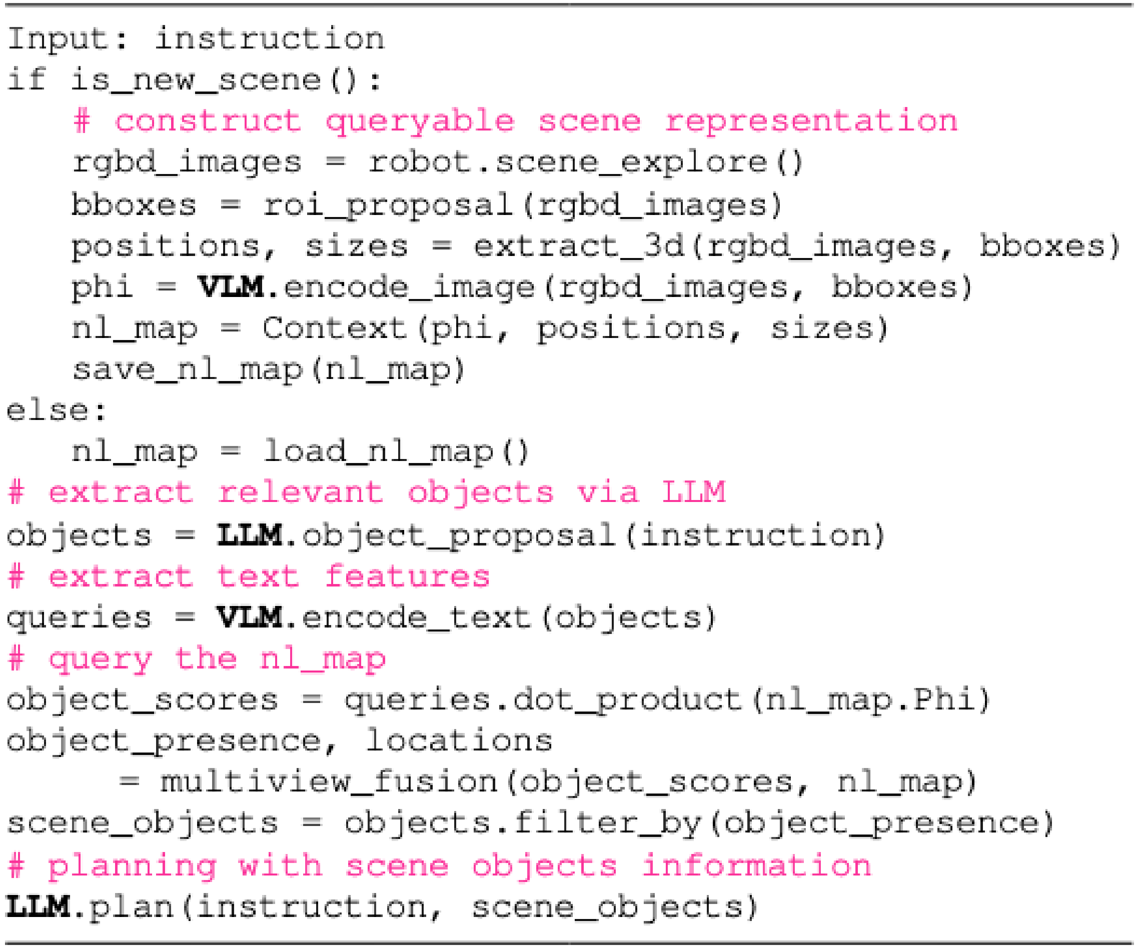
1. The agent explores the scene and provides a class-agnostic bounding box proposal based on objectness.
2. Extract 512d CLIP features and 512d ViLD features of each bounding box and represent them as a feature point cloud $C = (\varphi_i, p_i, r_i)_{i=1\ldots N}$.
3. When queried with a piece of text, visualize the heatmap of matches based on the alignment of text and visual features.



NLMap (4/4)

To complete a task specified by human instruction, the robot will query the scene representation for relevant information.

1. parsing natural language instruction into a list of relevant object names
2. using the names as keys to query object locations and availability.
3. generating executable options based on what’s found in the scene, then plan and execute as instructed.



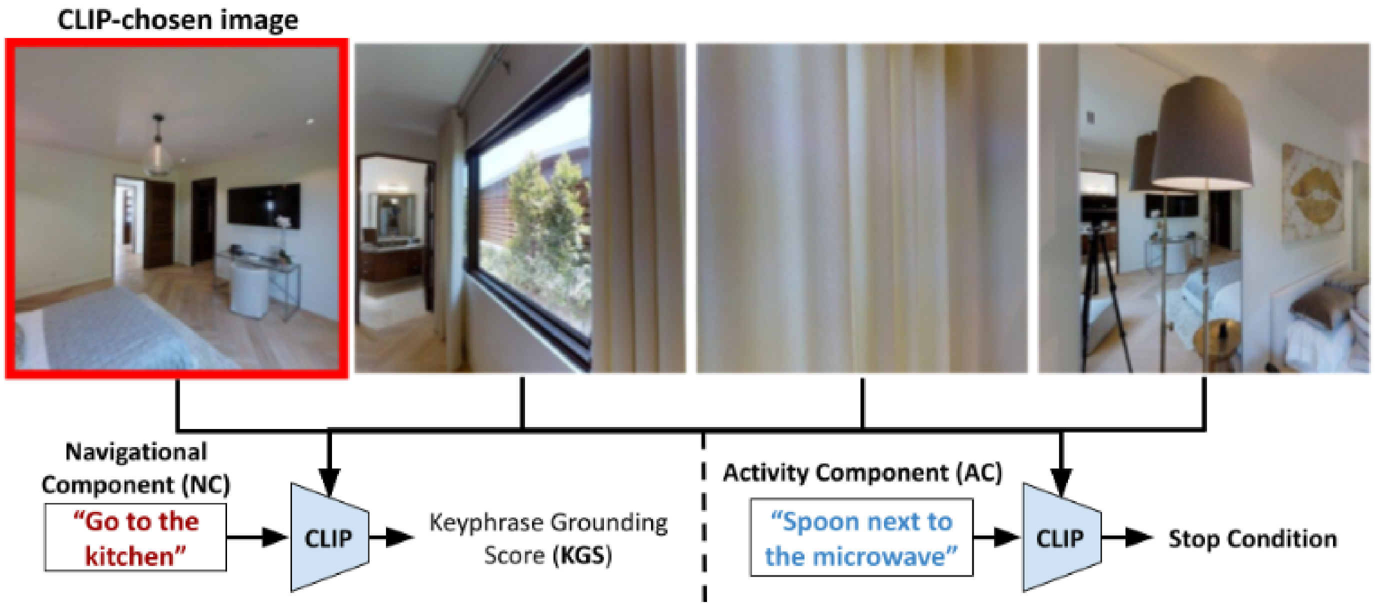
```
Input: instruction
if is_new_scene():
    # construct queryable scene representation
    rgbd_images = robot.scene_explore()
    bboxes = roi_proposal(rgbd_images)
    positions, sizes = extract_3d(rgbd_images, bboxes)
    phi = VLM.encode_image(rgbd_images, bboxes)
    nl_map = Context(phi, positions, sizes)
    save_nl_map(nl_map)
else:
    nl_map = load_nl_map()
# extract relevant objects via LLM
objects = LLM.object_proposal(instruction)
# extract text features
queries = VLM.encode_text(objects)
# query the nl_map
object_scores = queries.dot_product(nl_map.Phi)
object_presence, locations
       = multiview_fusion(object_scores, nl_map)
scene_objects = objects.filter_by(object_presence)
# planning with scene objects information
LLM.plan(instruction, scene_objects)
```

CLIP-Nav (1/3)

CLIP-Nav [1↑] examines CLIP’s capability in making sequential navigational decisions, and study how it influences the path that an agent takes.

1. **Instruction Breakdown**: Decompose coarse-grained instructions into keyphrases using LLMs.
2. **Vision-Language Grounding**: Ground keyphrases in the environment using CLIP.
3. **Zero-Shot Navigation**: Utilize the CLIP scores to make navigational decisions.

CLIP-Nav (2/3)


CLIP-chosen image
Navigational Component (NC)
"Go to the kitchen"
CLIP
Keyphrase Grounding Score (KGS)
Activity Component (AC)
"Spoon next to the microwave"
CLIP
Stop Condition

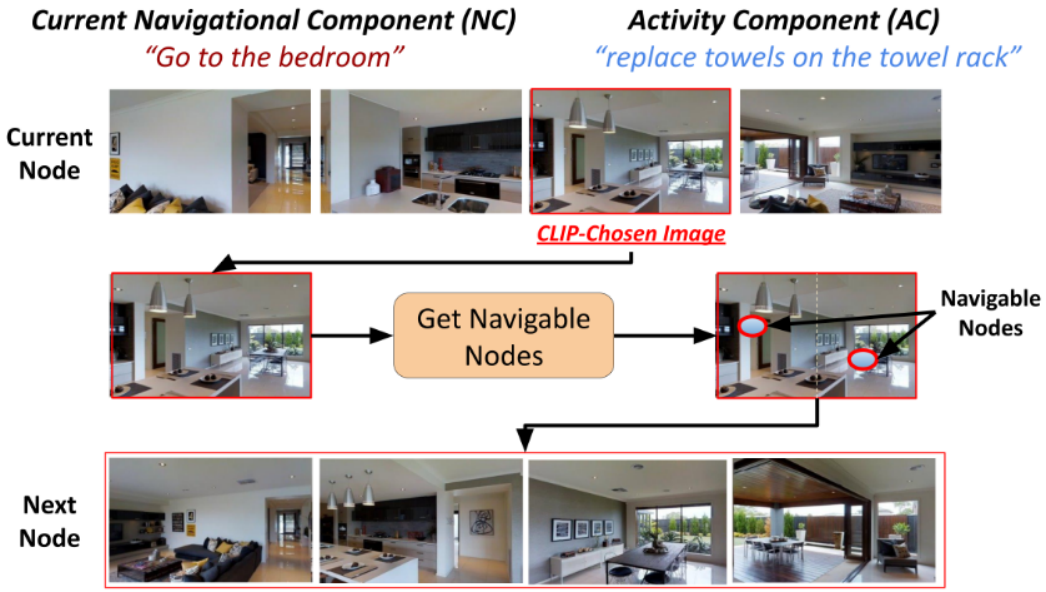
- Ground the NC on all the split images to obtain Keyphrase Grounding Scores (KGS). The *CLIP-chosen image* represents the one with the highest KGS, which drives the navigation algorithms.
- Ground the AC and use the grounding score to determine if the agent has reached the target location (*stop condition*).

CLIP-Nav (3/3)

At each time step:

1. split the panorama into 4 images, and obtain the CLIP-chosen image
2. obtain adjacent navigable nodes visible from this image using the Matterport Simulator, and choose the closest node.

This is done iteratively till the *Stop Condition* is reached.


Current Navigational Component (NC)
"Go to the bedroom"
Activity Component (AC)
"replace towels on the towel rack"
Current Node
CLIP-Chosen Image
Get Navigable Nodes
Navigable Nodes
Next Node