AAAI 2025 Tutorial T04
Time: 2025-02-25 8:30-12:30
Location: Room 118A

Part II: Foundation Models meet Physical Agents

High-Level Decision-Making

AAAI Tutorial: Foundation Models Meet Embodied Agents

Northwestern University
Columbia
MIT
Stanford University

High-Level Policy

- We want to model $\boldsymbol{P(a_t \mid o_t, g)}$ based on:
  - g: natural language goal
  - A: discrete action space with pre-defined skills
  - O: observations from robot sensors

High-Level Policy

- We want to model $P(a_t \mid o_t, g)$ based on:
  - g: natural language goal
  - A: discrete action space with pre-defined skills
  - O: observations from robot sensors

"Place all fruits in the red basket"

<u>Action Space</u>
PickPlace(apple, blue basket)
PickPlace(apple, red basket)
PickPlace(orange, blue basket)
...

or **<u>Natural Language Version</u>**:
"put apple in blue basket"
"put apple in red basket"
...

Example Goal, Action Space, and Observation

High-Level Policy

- What we may not have access to:
  - S: underlying state representation
  - T: transition model that predicts the state changes based on an action

Starting State	**Action**	**Resulting State**
inside(apple, blue basket): True inside(apple, red basket): False	PickPlace(apple, red basket)	inside(apple, blue basket): False inside(apple, red basket): True

Example State Representation and Transition

High-Level Policy

- What we may not have access to:
  - S: underlying state representation
  - T: transition model that predicts the state changes based on an action

Two possible definition of “state”

- The underlying physical state of the world
- The agent’s state representation of the world → what we consider here

Starting State	**Action**	**Resulting State**
inside(apple, blue basket): True inside(apple, red basket): False	PickPlace(apple, red basket)	inside(apple, blue basket): False inside(apple, red basket): True

Example State Representation and Transition

High-Level Policy

- What we may not have access to:
  - S: underlying state representation
  - T: transition model that predicts the state changes based on an action
  - R: reward function in the state-action space based on g

$$r = \begin{cases} 1 & \text{inside(apple, red basket)} \\ 0 & \text{otherwise} \end{cases}$$

Example Reward

High-Level Policy

- What we may not have access to:
  - S: underlying state representation
  - T: transition model that predicts the state changes based on an action
  - R: reward function in the state-action space based on g
  - Expert demonstrations: $(\mathrm{g}, o_1, a_1, o_2, a_2, \ldots)$

High-Level Policy

- How can we come up with a policy based on different assumptions?

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	1

→ Behavior Cloning

High-Level Policy

- How can we come up with a policy based on different assumptions?

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	1

→ Behavior Cloning

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	1	1	1	0

→ Search or planning

High-Level Policy

- How can we come up with a policy based on different assumptions?

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	1

→ Behavior Cloning

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	1	1	1	0

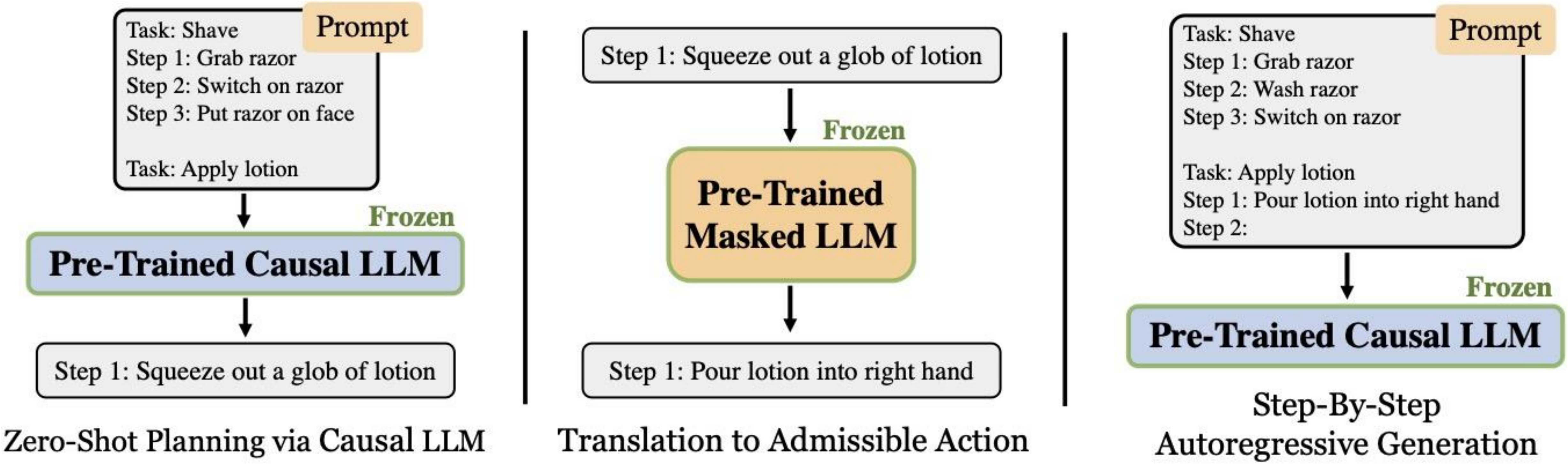
→ Search or planning

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

→ Humans can do it – how can my robot also do this?

LLMs as Planners

- How can we "plan" without state representation, transition function, or a reward function?



LLMs as Zero-Shot Planners. W. Huang, P. Abbeel, D. Pathak*, and I. Mordatch*. ICML 2022.

LLMs as Planners

- What LLMs actually provide: $\boldsymbol{P(a_t \mid g)}$
  - The likelihood of a step conditioned on goal and previous steps
  - Example: P("*Step 1: put apple in red basket*" | "*Place all fruits in the red basket*")

LLMs as Planners

- What LLMs actually provide: $\boldsymbol{P(a_t \mid g)}$
  - The likelihood of a step conditioned on goal and previous steps
  - Example: P("*Step 1: put apple in red basket*" | "*Place all fruits in the red basket*")
  - Key issue: The generation is not conditioned on current state
    - i.e., it is not "embodied" as it can do anything at anytime

LLMs as Planners

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

- How can we model $P(a_t \mid o_t, g)$ based on how we can plan with LLMs?

LLMs as Planners

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

- How can we model $P(a_t \mid o_t, g)$ based on how we can plan with LLMs?
- We can factor into two parts:
  - **LLM Prior**: How likely is a particular skill at current time based on "commonsense"?
  - **Feasibility**: Can the robot perform this skill successfully based on current observation?

SayCan

- Formalizing “feasibility” by a **per-skill value function**
- Intuitively: “If I ask the robot to do [skill] now, will it do it successfully?

Do As I Can, Not As I Say. Ahn et al. CoRL 2022.

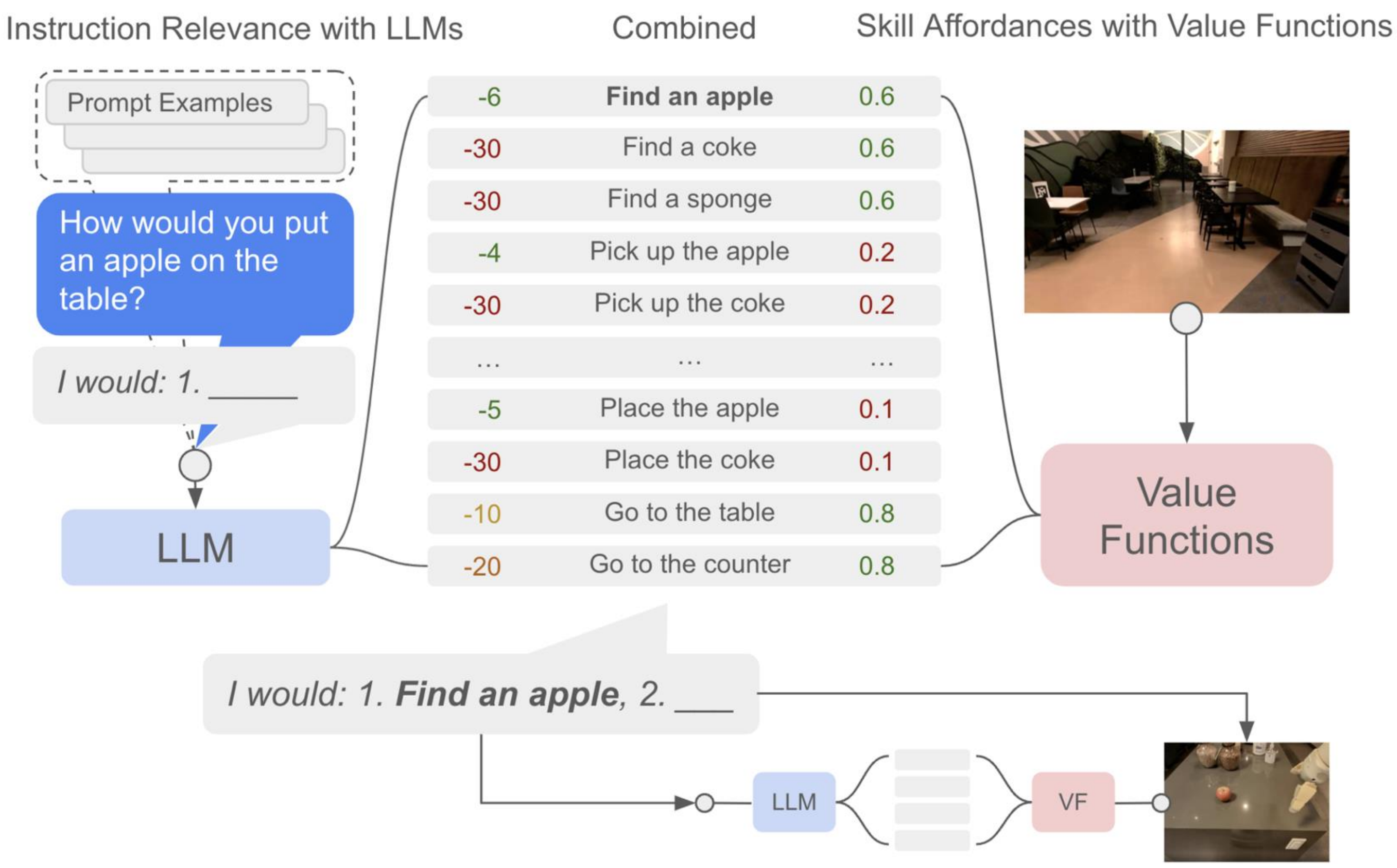
SayCan

- Formalizing “feasibility” by a **per-skill value function**
- Intuitively: “If I ask the robot to do [skill] now, will it do it successfully?
- How to obtain the value function? Based on how to obtain the skills, ...
  - **Reinforcement Learning**: value function is naturally available
  - **Behavior Cloning**: need to train posthoc success detector
  - **Manually-defined primitives**: write rule-based value functions

Do As I Can, Not As I Say. Ahn et al. CoRL 2022.

SayCan

- Full algorithm:
  - Start with high-level goal
  - Compute likelihood of each skill under the LLM
  - Compute the value function of each skill given current observation
  - Multiply their probabilities
  - Choose the most likely one



Do As I Can, Not As I Say. Ahn et al. CoRL 2022.

SayCan

Do As I Can, Not As I Say. Ahn et al. CoRL 2022.

SayCan

- Key Challenges:

SayCan

- Key Challenges:
  - Integrated high-level and low-level decision making:
    - LLMs only plan the next step based on each skill’s text description instead of what it physically does
    - Example: stowing a book on shelf requires first grasping a book in a particular way, but the text description may just be “grasp book”

SayCan

- Key Challenges:
  - Integrated high-level and low-level decision making:
    - LLMs only plan the next step based on each skill's text description instead of what it physically does
    - Example: stowing a book on shelf requires first grasping a book in a particular way, but the text description may just be "grasp book"
  - Robots may fail, but LLMs assume every step is successful

SayCan

- Key Challenges:
  - Integrated high-level and low-level decision making:
    - LLMs only plan the next step based on each skill's text description instead of what it physically does
    - Example: stowing a book on shelf requires first grasping a book in a particular way, but the text description may just be "grasp book"
  - Robots may fail, but LLMs assume every step is successful
  - Priors provided LLMs $P(a_t \mid g)$ do not consider observations
    - Crucial if environments are dynamic or goal is underspecified
    - Example: Given task "place all fruits in the red basket", what if some fruits are already in the basket or they are being taken out by another agent?

Conditioning LLMs on Observations

- How can we model $P(a_t \mid o_t, g)$ directly with LLMs?

Conditioning LLMs on Observations

- How can we model $P(a_t \mid o_t, g)$ directly with LLMs?
- Possible ideas:

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	**0**

  - Textualize o_t and put that in the prompt at each timestep -- no training needed

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	**1**

  - Train o_t into the LLMs and make them multimodal

Conditioning LLMs on Observations

- How can we model $P(a_t \mid o_t, g)$ directly with LLMs?
- Possible ideas:

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

- Textualize o_t and put that in the prompt at each timestep -- no training needed

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	1

- Train o_t into the LLMs and make them multimodal

Conditioning LLMs on Observations

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

- Condition LLMs by textualizing observations
- Key Questions:
  - What to textualize from $\boldsymbol{o_t}$?
  - How to textualize from $\boldsymbol{o_t}$?

Inner Monologue

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

- Textualize $\boldsymbol{o_t}$ using a combination of the following:
  - **Success detector**: per-skill detector that says whether previous skill was successful
  - **Structured scene description**: structured text provided by specialized perception modules, such as object detectors
  - **Unstructured scene description**: unstructured text provided by another multi-modal LLMs or a human
- After every skill, provide the textualized $\boldsymbol{o_t}$, and repeat the SayCan-style next skill selection

Inner Monologue. Huang*, Xiao*, Xia* et al. CoRL 2022.

Inner Monologue

Code as Policies

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

- Alternatively, formulating it as a **code-writing problem**
  - Provide basic perception APIs and control primitives:
    - `detect_objects`: return a list of present objects
    - `pick_place`: motion primitive for picking up and placing an object

Code as Policies. Liang et al. ICRA 2023.

Code as Policies

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

- Alternatively, formulating it as a **code-writing problem**
  - Provide basic perception APIs and control primitives:
    - \`detect_objects\`: return a list of present objects
    - \`pick_place\`: motion primitive for picking up and placing an object
- Why might this be a good idea?
  - LLMs can actively decide how to textualize $\boldsymbol{o_t}$ to benefit its decision-making instead of passively being provided the same information
  - Programmatic structure enables use of precise numerical values, function composition, and logical structure, similar to how humans write policy,

Code as Policies. Liang et al. ICRA 2023.

Code as Policies

High-Level Session:

```
# draw a square around the sweeter fruit.
say('ok - drawing a square around the sweeter fruit')
sweeter_fruit_name = parse_obj_name('the sweeter fruit', f'objects = {get_obj_names()}')
sweeter_fruit_square_shape_pts = parse_shape_pts(f'a 10cm square around the {sweeter_fruit_name}')
draw(sweeter_fruit_square_shape_pts)
```

"parse_obj_name" Session:

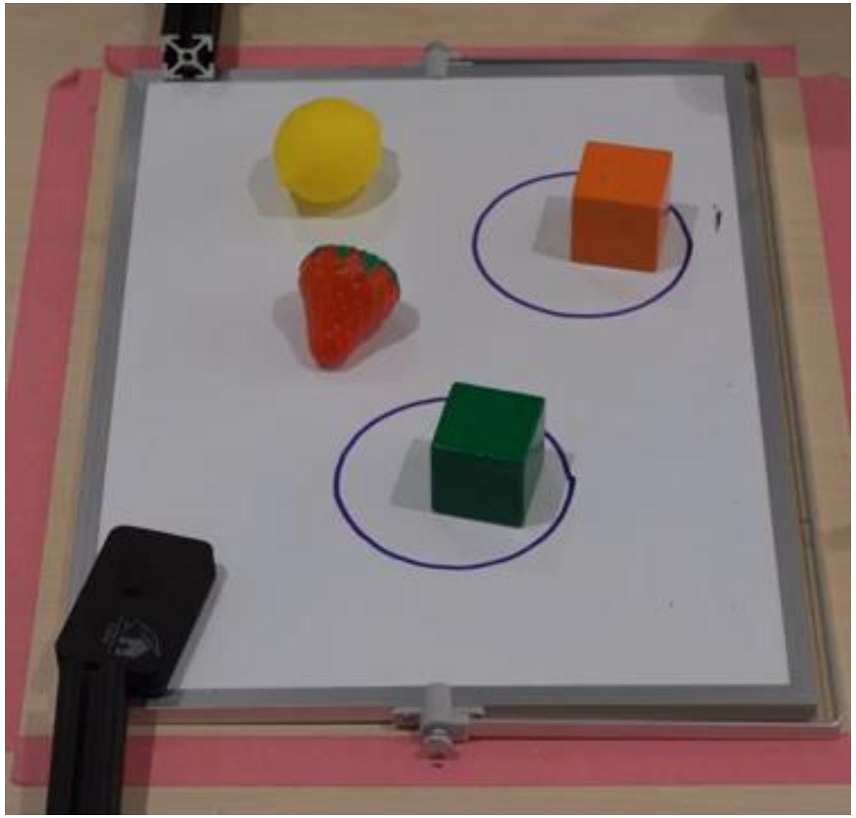
```
objects = ['green block', 'orange block', 'strawberry', 'lemon']
# the sweeter fruit.
ret_val = 'strawberry'
```

"parse_shape_pts" Session:

```
# a 10cm square around the strawberry.
strawberry_name = parse_obj_name('the strawberry', f'objects = {get_obj_names()}')
strawberry_pos = get_obj_pos(strawberry_name)
square_shape = make_square(size=0.1, center=strawberry_pos)
square_shape_pts = get_points_from_polygon(square_shape)
ret_val = square_shape_pts
```

Function Generator Session:

```
# define function: square_shape = make_square(size=0.1, center=strawberry_pos).
def make_square(size, center):
    square = Polygon([
        center + [-size/2, -size/2],
        center + [size/2, -size/2],
        center + [size/2, size/2],
        center + [-size/2, size/2]
    ])
    return square

# define function: square_shape_pts = get_points_from_polygon(square_shape).
def get_points_from_polygon(square_shape):
    return np.array(square_shape.exterior.coords)
```



Code as Policies. Liang et al. ICRA 2023.

Conditioning LLMs on Observations

- How can we model $P(a_t \mid o_t, g)$ with LLMs?
- Possible ideas:

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0

  - Textualize o_t and put that in the prompt at each timestep -- no training needed

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	1

  - Train o_t into the LLMs and make them multimodal

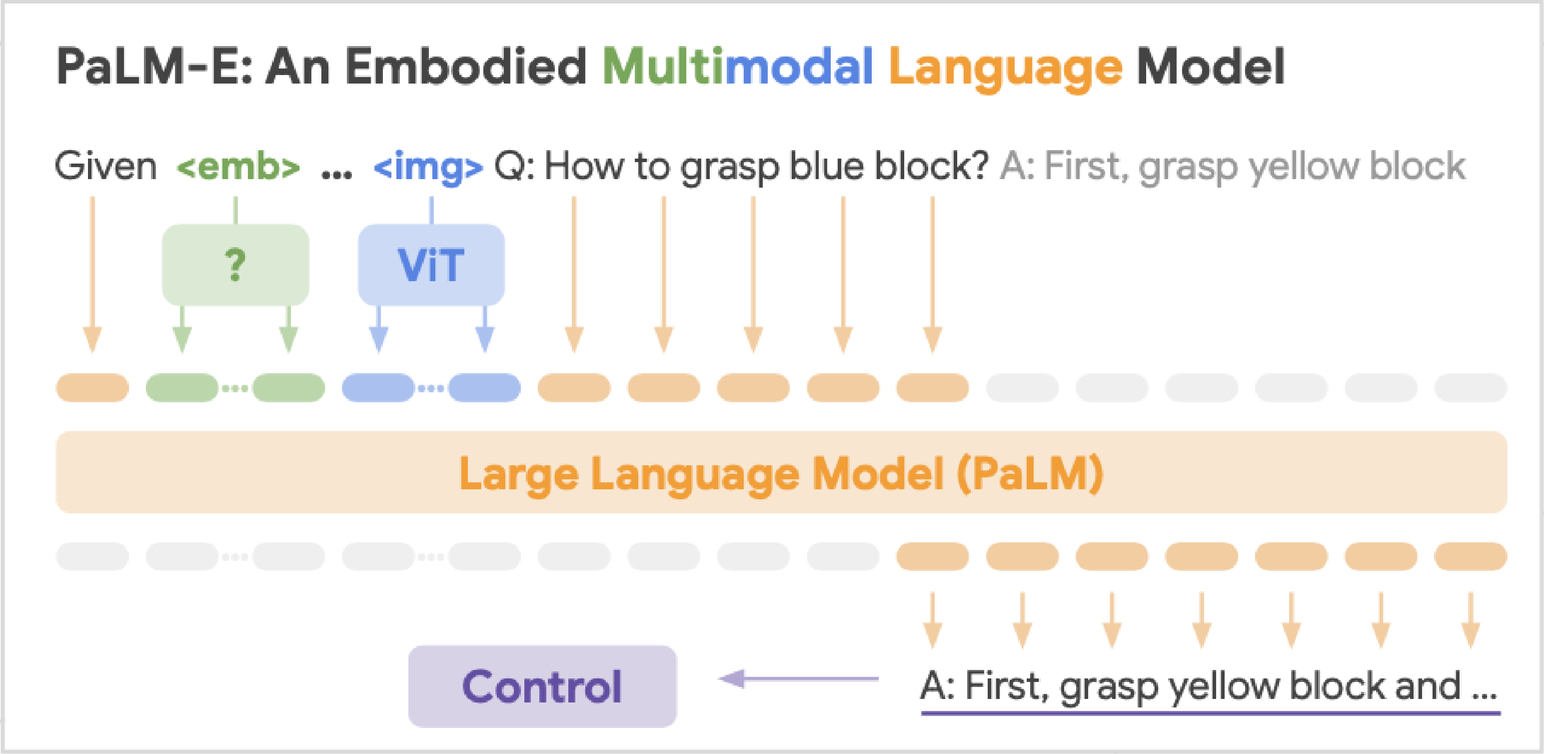
PaLM-E

- Collect multi-modal expert demonstrations: $(\, g, o_1, a_1, o_2, a_2, \ldots \,)$
- Start with pre-trained LLM and vision encoder
- Finetune them on the collected demonstrations and other vision-language data

PaLM-E. Driess et al. ICML 2023.

PaLM-E

- Collect multi-modal expert demonstrations: $(\, g, o_1, a_1, o_2, a_2, \ldots \,)$
- Start with pre-trained LLM and vision encoder
- Finetune them on the collected demonstrations and other vision-language data



Modeling Reward instead of Policy

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0/1

- Recall we have been discussing how to model $P(a_t \mid o_t, g)$ directly

Modeling Reward instead of Policy

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	0	0	0	0/1

- Recall we have been discussing how to model $P(a_t \mid o_t, g)$ directly

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	1	**1**	**1**	*	0

- Also recall that if we have a state representation and a transition model, we can alternatively model reward $P(r_t \mid s_{t-1}, a_{t-1}, s_t, g)$. Then we can use planning or reinforcement learning to obtain actions/policies.

PDDL

- A standardized framework to specify (mostly symbolic) planning problems
- What it typically requires:
  - discrete action space (e.g., “PickPlace”)
  - state representation (e.g., “inside”)
  - goal state (e.g., “inside(apple, red basket)”)
    - can be considered as a sparse reward function
  - transition function for each action
- Well-suited for the **task planning problem** with high-level action space

LLM + P

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	N/A	1	1	*	0

LLM + P

Goal	Skills	Obs	State	Transition	Reward	Demos
1	1	N/A	1	1	*	0

- Example:
  - Prompt contains:
    - language description of the current state
    - language description of goal
  - LLM generates:
    - starting state in defined state representation
    - goal state (i.e., a sparse reward function)

Tidy-Up Problem PDDL Generated by LLM+P

Problem (P): You are a home robot with one gripper. The distance between coffee table and side table is 10. The distance between coffee table and pantry is 20... You are at the coffee table. There is a mustard bottle... Your goal is to move objects to their destinations...

Problem PDDL generated by LLM+P:

```
(:objects coffee-table side-table
recycle-bin pantry - location
mustard-bottle soup-can - object)
(:init (= (total-cost) 0) (=
(distance coffee-table side-table)
10) (= (distance coffee-table
pantry) 20) ... (robot-at
coffee-table) (at mustard-bottle
coffee-table) (at soup-can
side-table) (hand-empty) )
(:goal (and (at mustard-bottle
pantry) (at soup-can recycle-bin)))
(:metric minimize (total-cost)) )
```

LLM + P. Liu*, Jiang* et al. 2023.

LLM + P

Domain	Success Rate %				
	LLM^-	LLM	LLM^{ToT}	$LLM{+}P^-$	LLM+P
BARMAN	0	0	0	0	**20 (100)**
BLOCKSWORLD	20	15 (30)	0 (5)	0	**90**
FLOORTILE	0	0	0	0	**0**
GRIPPERS	25 (60)	35 (50)	10 (20)	0	**95 (100)**
STORAGE	0	0 (25)	0	0	**85**
TERMES	0	0	0	0	**20**
TYREWORLD	5	15	0	0	**10 (90)**

- **Take-away**: If state and transition function are accessible, with a high-level action space, specifying goal state (i.e., the reward function) with LLMs and use classical planning algorithms is often more effective.

LLM + P. Liu*, Jiang* et al. 2023.

AAAI 2025 Tutorial T04
Time: 2025-02-25 8:30-12:30
Location: Room 118A

Part II: Foundation Models meet Physical Agents

Low-Level Decision-Making

AAAI Tutorial: Foundation Models Meet Embodied Agents

Northwestern University COLUMBIA MIT Stanford University

Low-Level Action Space

- The same key problem $\boldsymbol{P(a_t \mid o_t, g)}$ but now with low-level action space
  - a_t: joint space commands or end-effector commands
  - g: natural language goal
  - o_t: observations from robot sensors

Low-Level Action Space

- The same key problem $\boldsymbol{P(a_t \mid o_t, g)}$ but now with low-level action space
  - a_t: joint space commands or end-effector commands
  - g: natural language goal
  - o_t: observations from robot sensors
- Why it's much more challenging?
  - Higher-frequency: typically 10-20 Hz
  - Longer-horizon: large compounded errors over easily 1000s of steps
  - Continuous and high-dimensional: brings challenging optimization landscape with lots of local minima
  - In the context of this tutorial, low-level actions do not live in the same semantic abstraction with language compared to high-level actions.

Low-Level Action Space

Goal	Skills	Obs	State	Transition	Reward	Demos
1	0	1	0	0	0	1

- If expert demonstrations $(g, o_1, a_1, o_2, a_2, \ldots)$ are assumed:
  - We can directly model $\boldsymbol{P(a_t \mid o_t, g)}$
  - VLMs can be finetuned to become Vision-Language-Action models (VLAs), discussed in the next session of the tutorial.

Low-Level Action Space

Goal	Skills	Obs	State	Transition	Reward	Demos
1	0	1	0	0	0	1

- If expert demonstrations $(g, o_1, a_1, o_2, a_2, \ldots)$ are assumed:
  - We can directly model $P(a_t \mid o_t, g)$
  - VLMs can be finetuned to become Vision-Language-Action models (VLAs), discussed in the next session of the tutorial.

Goal	Skills	Obs	State	Transition	Reward	Demos
1	0	1	1	1	*	0

- Alternatively, we can model these and then “solve for” actions:
  - A state representation $\boldsymbol{S}$
  - Transition function: $\boldsymbol{S} \times A \rightarrow \boldsymbol{S}$
  - Reward function: $\boldsymbol{S} \times A \rightarrow \mathbb{R}$
- LLMs/VLMs are typically used to model the **reward functions** based on the language goal.

Reward Specification with LLMs / VLMs

- Translating language goal to $\mathbf{R(S, A)}$ -- why might this be a good idea?

Reward Specification with LLMs / VLMs

- Translating language goal to $\mathbf{R(S, A)}$ -- why might this be a good idea?
  - Language goal is often underspecified: what does it mean by “tidying up a room”?

K. Wyrobek, E. Berger, H.F.M. Van der Loos, and K. Salisbury. ICRA 2008.

Reward Specification with LLMs / VLMs

- Translating language goal to $\mathbf{R(S, A)}$ -- why might this be a good idea?
  - Language goal is often underspecified: what does it mean by "tidying up a room"?
  - Reward is typically more about desired state instead of desired actions

K. Wyrobek, E. Berger, H.F.M. Van der Loos, and K. Salisbury. ICRA 2008.

Reward Specification with LLMs / VLMs

- Translating language goal to $\mathbf{R(S, A)}$ -- why might this be a good idea?
  - Language goal is often underspecified: what does it mean by "tidying up a room"?
  - Reward is typically more about desired state instead of desired actions
  - This type of knowledge should be within the "interpolated space" of LLMs/VLMs

K. Wyrobek, E. Berger, H.F.M. Van der Loos, and K. Salisbury. ICRA 2008.

Reward Specification with LLMs / VLMs

- Key question: what should be the state representation ($\boldsymbol{S}$)?
  - Recall:
    - Reward function: $\boldsymbol{S} \times A \rightarrow \mathbb{R}$
    - Transition function: $\boldsymbol{S} \times A \rightarrow \boldsymbol{S}$

Reward Specification with LLMs / VLMs

- Key question: what should be the state representation ($\boldsymbol{S}$)?
  - Recall:
    - Reward function: $\boldsymbol{S} \times A \rightarrow \mathbb{R}$
    - Transition function: $\boldsymbol{S} \times A \rightarrow \boldsymbol{S}$
  - Such that:
    - Reward is extractable from LLMs/VLMs
    - There needs to be an associated transition function that can support the desired tasks (such that we can use the reward function to generate actions)

Reward Specification with LLMs / VLMs

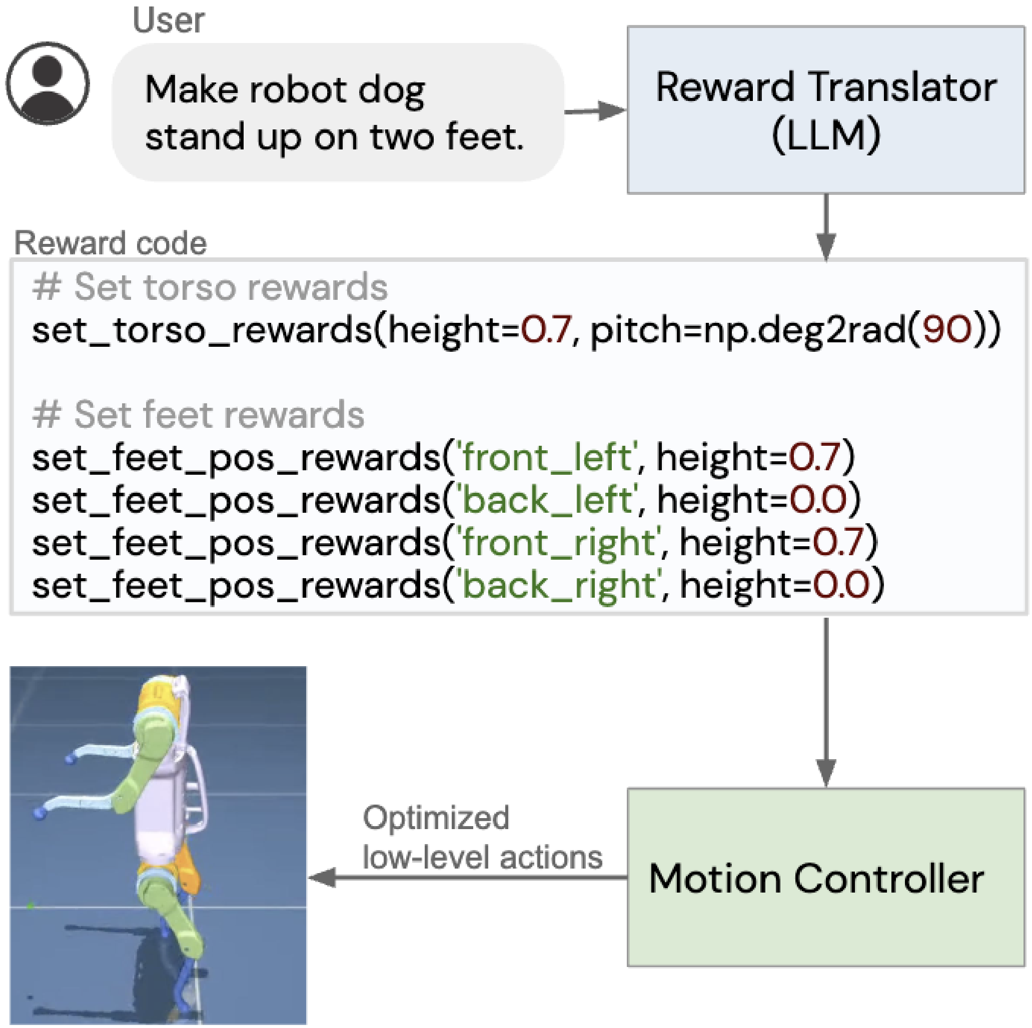
- Case studies:
  - Language to Rewards & Eureka
  - VoxPoser
  - ReKep

Reward Specification with LLMs / VLMs

- Case studies:
  - Language to Rewards & Eureka:
    - **State Representation**: the simulator state (e.g., rigid-body poses, articulation, velocities)
    - **Transition Function**: simulator
    - **Action Space**: joint space commands
  - VoxPoser
  - ReKep

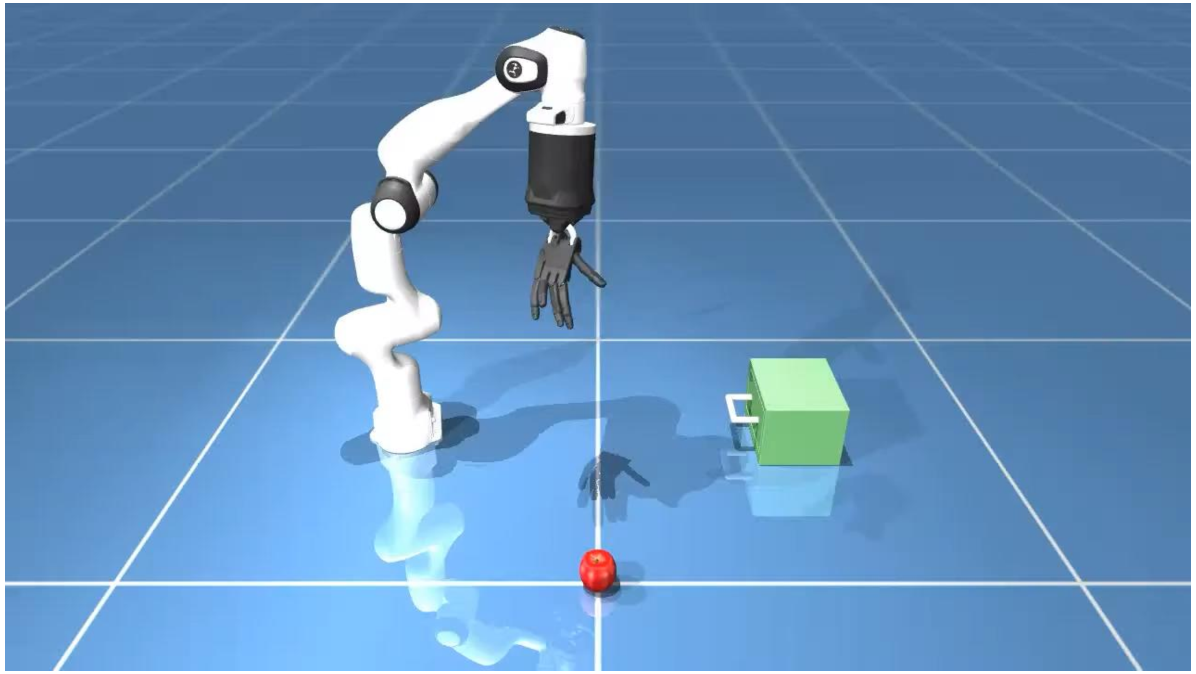
Language to Rewards

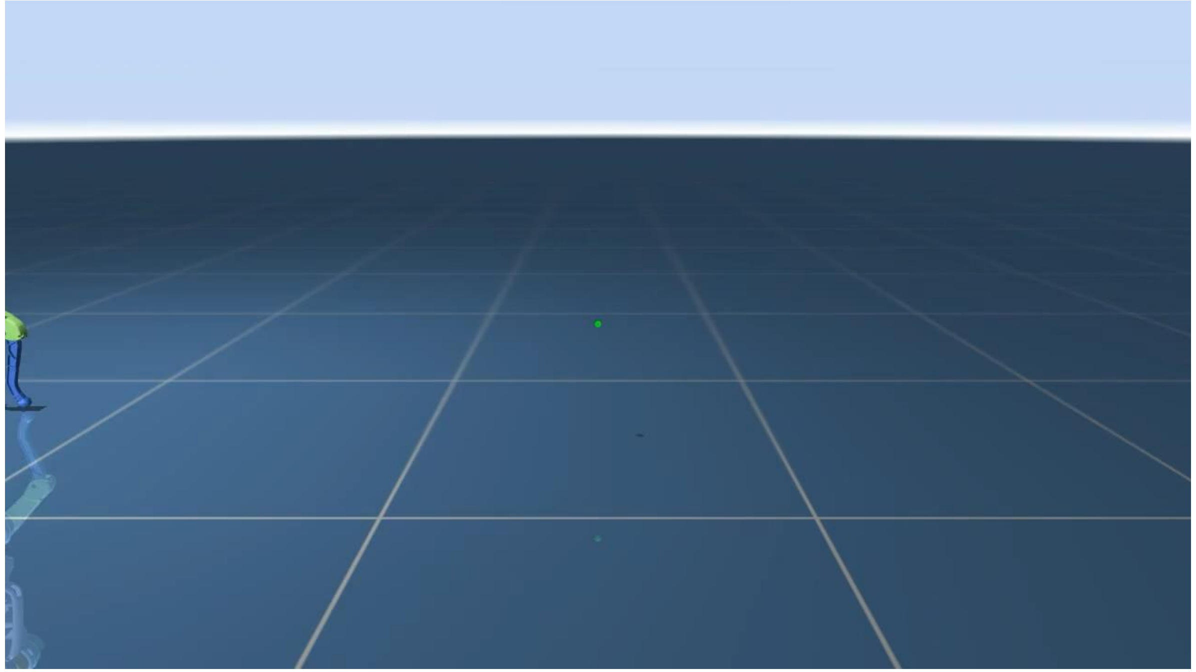
- **Key Idea**:
  - Start with open-ended language goal
  - Provide a set of basic reward APIs:
    - “set_feet_pos_reward”
    - “set_l2_distance_reward”
    - “set_obj_orientation_reward”
    - ...
  - LLMs compose full reward function
  - Perform sampling/planning with simulator as the transition function to obtain actions



Language to Rewards. Yu*, Gileadi* et al. CoRL 2023.

Language to Rewards





Language to Rewards. Yu*, Gileadi* et al. CoRL 2023.

Eureka

Environment Code

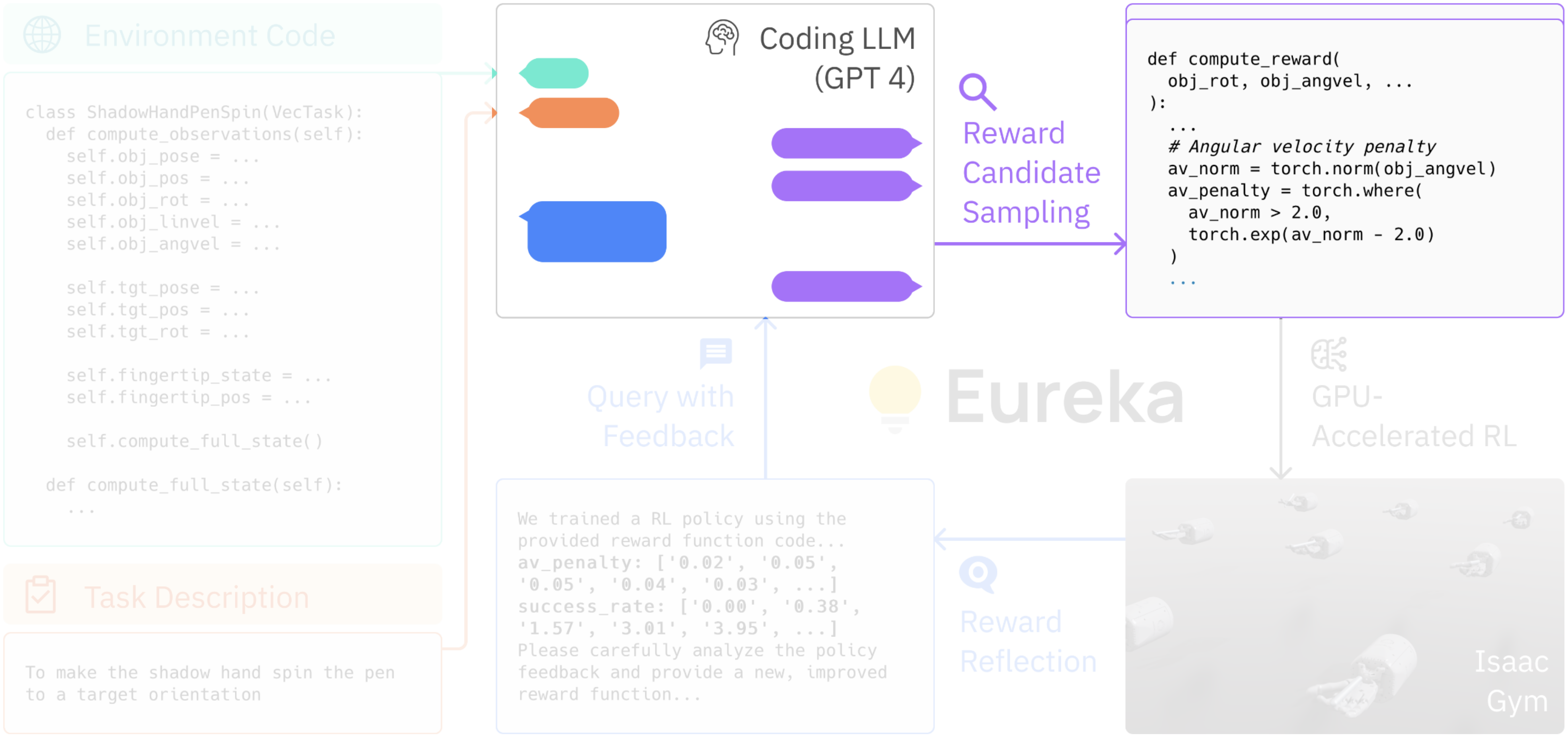
```
class ShadowHandPenSpin(VecTask):
  def compute_observations(self):
    self.obj_pose = ...
    self.obj_pos = ...
    self.obj_rot = ...
    self.obj_linvel = ...
    self.obj_angvel = ...

    self.tgt_pose = ...
    self.tgt_pos = ...
    self.tgt_rot = ...

    self.fingertip_state = ...
    self.fingertip_pos = ...

    self.compute_full_state()

  def compute_full_state(self):
    ...
```

Task Description

```
To make the shadow hand spin the pen
to a target orientation
```

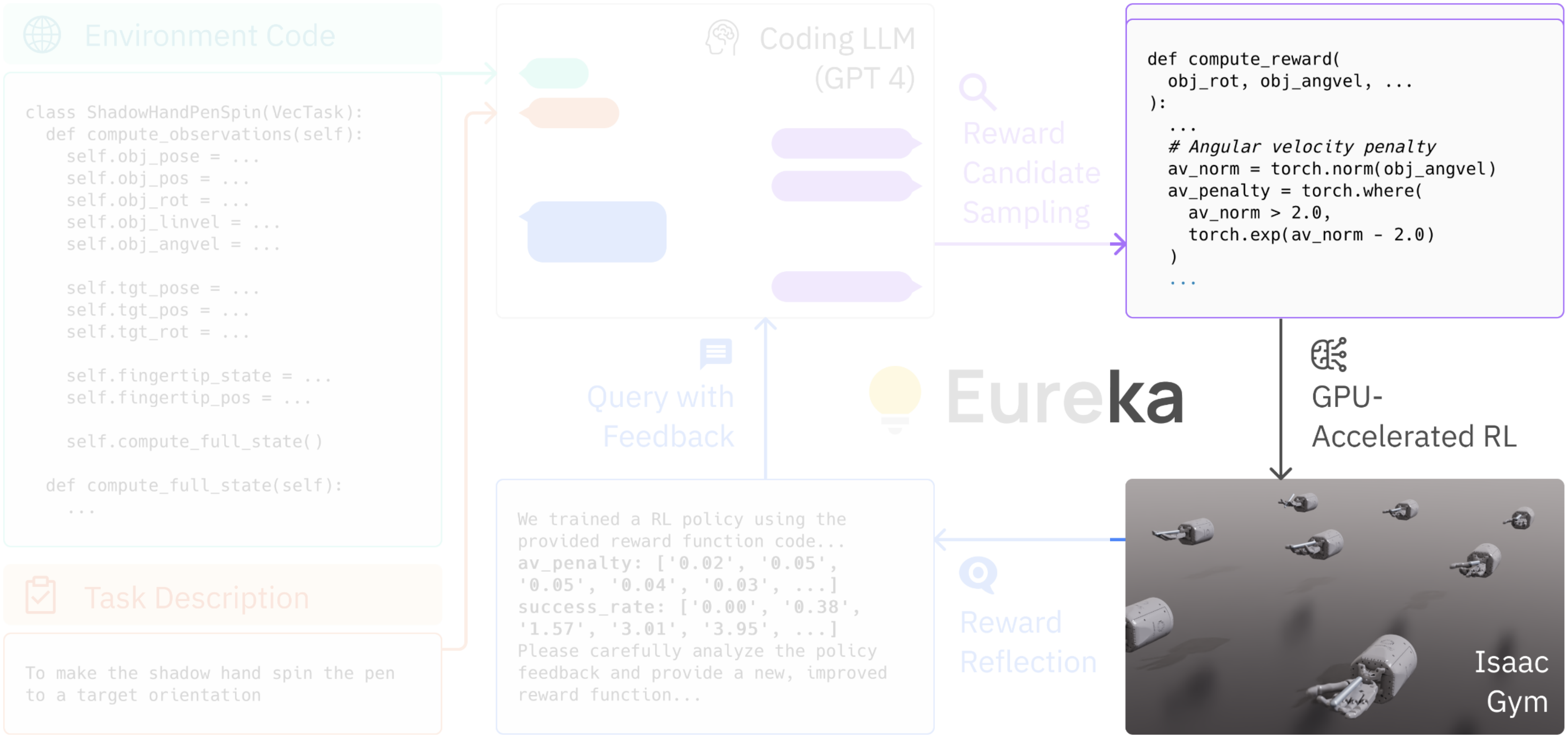

Eureka. Ma et al. ICLR 2024.

Eureka



Eureka. Ma et al. ICLR 2024.

Eureka

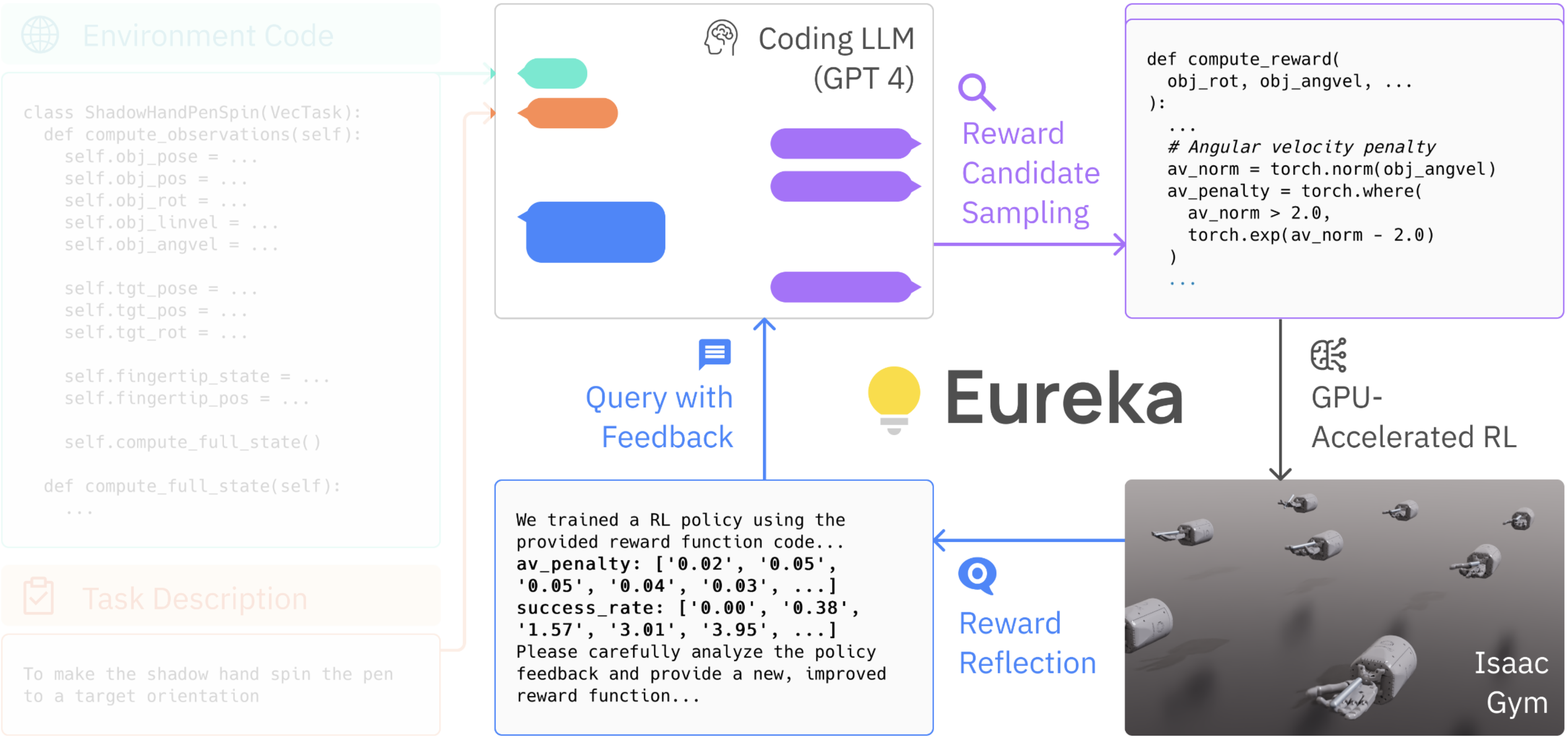


Environment Code
class ShadowHandPenSpin(VecTask):
def compute_observations(self):
self.obj_pose = ...
self.obj_pos = ...
self.obj_rot = ...
self.obj_linvel = ...
self.obj_angvel = ...
self.tgt_pose = ...
self.tgt_pos = ...
self.tgt_rot = ...
self.fingertip_state = ...
self.fingertip_pos = ...
self.compute_full_state()
def compute_full_state(self):
...
Task Description
To make the shadow hand spin the pen to a target orientation
Coding LLM (GPT 4)
Reward Candidate Sampling
def compute_reward(
obj_rot, obj_angvel, ...
):
...
# Angular velocity penalty
av_norm = torch.norm(obj_angvel)
av_penalty = torch.where(
av_norm > 2.0,
torch.exp(av_norm - 2.0)
)
...
Query with Feedback
Eureka
GPU-Accelerated RL
We trained a RL policy using the provided reward function code...
av_penalty: ['0.02', '0.05', '0.05', '0.04', '0.03', ...]
success_rate: ['0.00', '0.38', '1.57', '3.01', '3.95', ...]
Please carefully analyze the policy feedback and provide a new, improved reward function...
Reward Reflection
Isaac Gym

Eureka

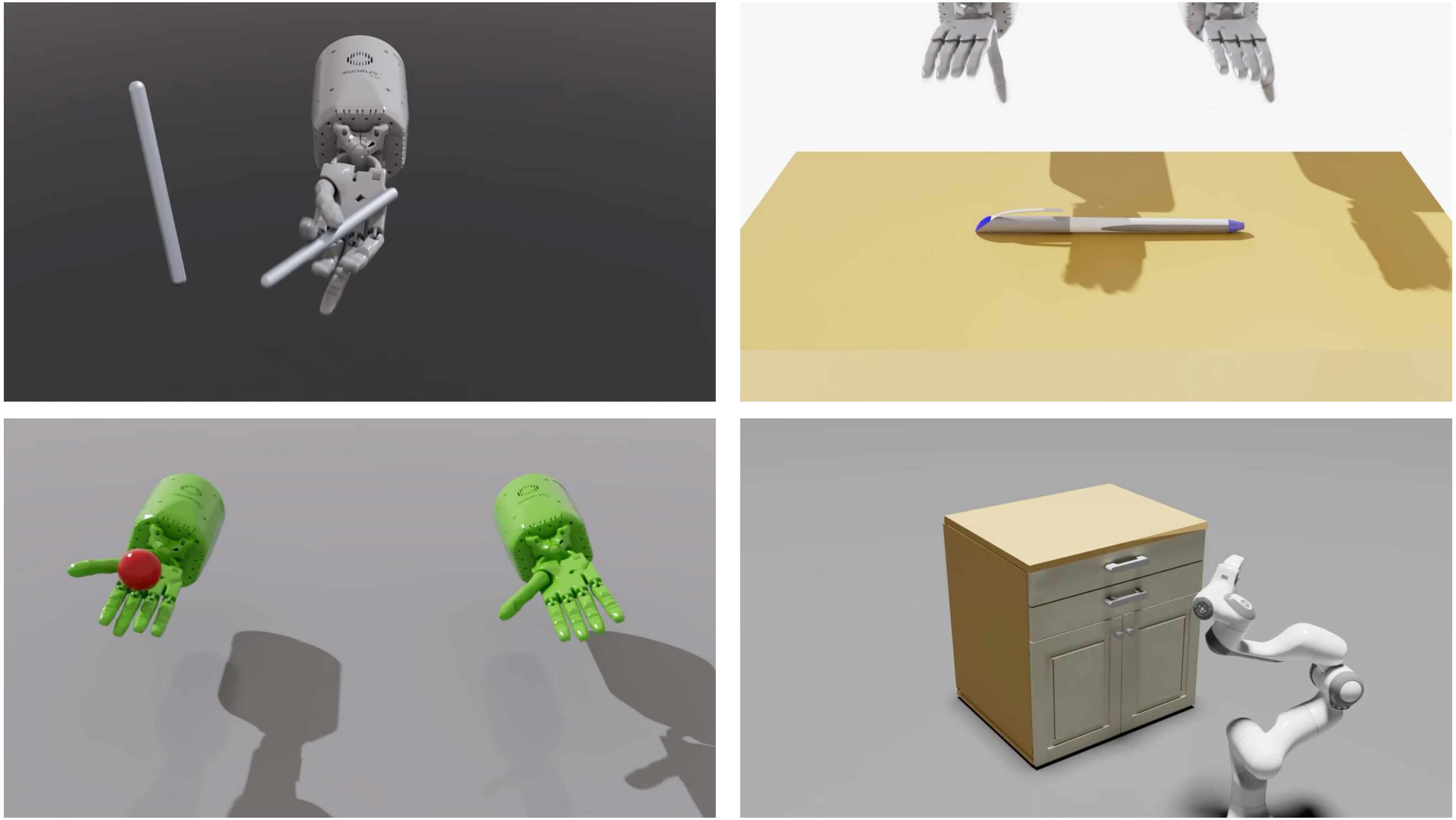
Eureka. Ma et al. ICLR 2024.

Eureka



Eureka. Ma et al. ICLR 2024.

Eureka



Eureka. Ma et al. ICLR 2024.

Language to Rewards & Eureka

- **Take-away**: While it's challenging for LLMs to directly predict low-level actions, specifying reward can be a promising path of utilizing their knowledge even with a low-level action space.

Language to Rewards & Eureka

- **Take-away**: While it's challenging for LLMs to directly predict low-level actions, specifying reward can be a promising path of utilizing their knowledge even with a low-level action space.
- Similarities:
  - Both use simulator state as the state representation for reward specification
    - **Pros**: Allow the direct use of simulator as the transition function
    - **Cons**: Real2Sim and Sim2Real present a significant challenge
- Differences:
  - Language to Rewards uses MPC for action generation.
  - Eureka uses RL for action generation.

Reward Specification with LLMs / VLMs

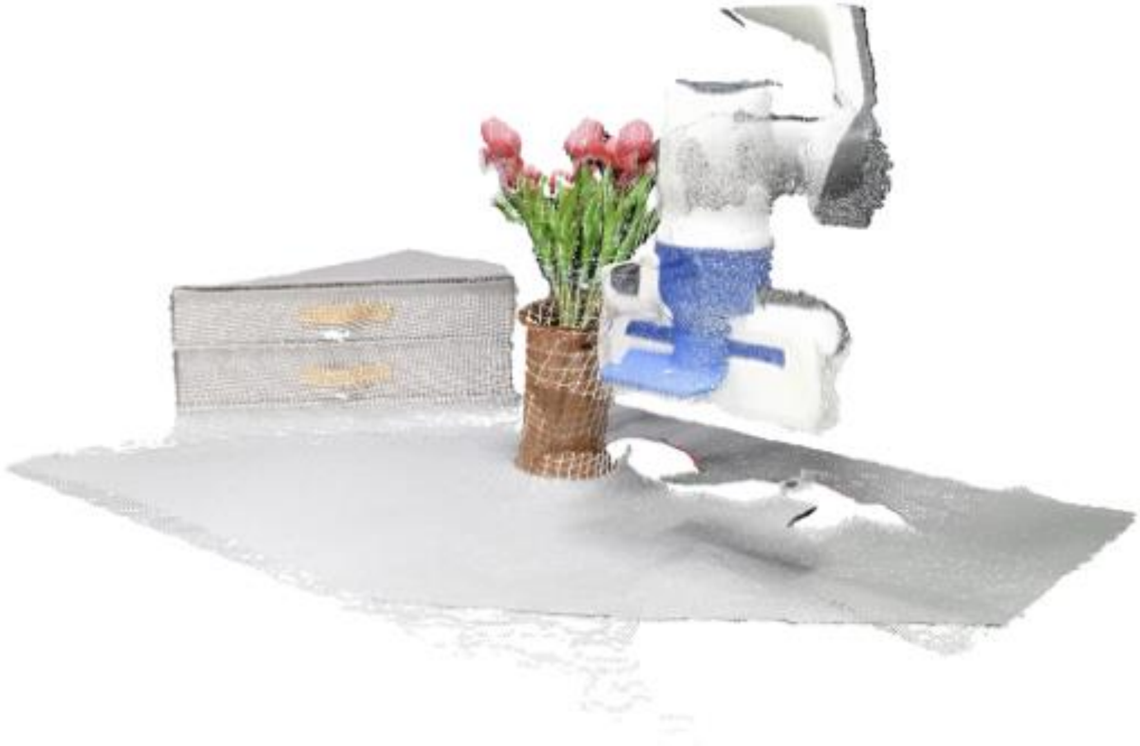
- Case studies:
  - Language to Rewards & Eureka
  - VoxPoser:
    - **State Representation**: 3D voxels of workspace
    - **Transition Function**: Robot only
    - **Action Space**: end-effector pose
  - ReKep

VoxPoser

- **Key Idea**: Given a language goal, LLMs and VLMs can write code to iteratively assign reward values to different locations in a 3D voxel grid that represents robot workspace.

VoxPoser

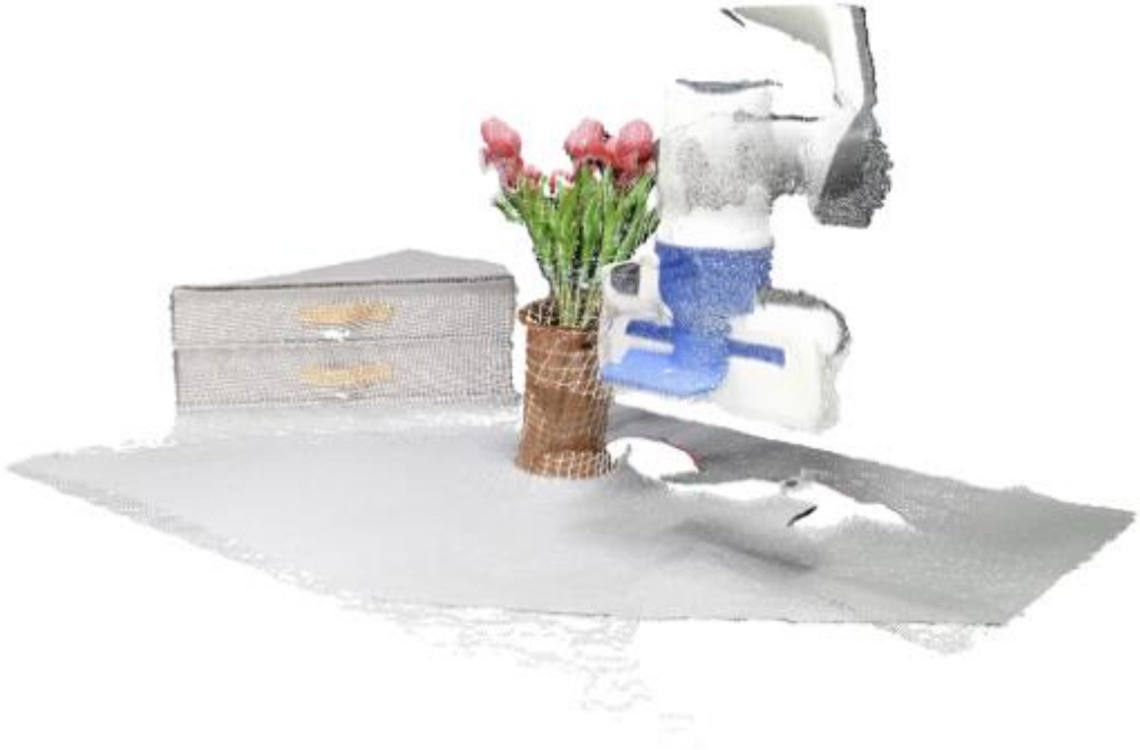
- **Key Idea**: Given a language goal, LLMs and VLMs can write code to iteratively assign reward values to different locations in a 3D voxel grid that represents robot workspace.



Open the top drawer.

VoxPoser

- **Key Idea**: Given a language goal, LLMs and VLMs can write code to iteratively assign reward values to different locations in a 3D voxel grid that represents robot workspace.



Open the top drawer.

LLM Output

```
def value_map():
  msize = (100,100,100)
  map = np.zeros(msize)
  handles = detect('handle')
  k = lambda x: x.pos[2]
  handles.sort(key=k)
  top_handle = handles[-1]
  x,y,z = top_handle.pos
  map[x,y,z] = 1
  return smooth(map)
```

VoxPoser. Huang et al. CoRL 2023.

VoxPoser

- **Key Idea**: Given a language goal, LLMs and VLMs can write code to iteratively assign reward values to different locations in a 3D voxel grid that represents robot workspace.

Open the top drawer.

LLM Output

```
def value_map():
  msize = (100,100,100)
  map = np.zeros(msize)
  handles = detect('handle')
  k = lambda x: x.pos[2]
  handles.sort(key=k)
  top_handle = handles[-1]
  x,y,z = top_handle.pos
  map[x,y,z] = 1
  return smooth(map)
```

VLM

VoxPoser. Huang et al. CoRL 2023.

- **Key Idea**: Given a language goal, LLMs and VLMs can write code to iteratively assign reward values to different locations in a 3D voxel grid that represents robot workspace.

Open the top drawer.

LLM Output

```
def value_map():
  msize = (100,100,100)
  map = np.zeros(msize)
  handles = detect('handle')
  k = lambda x: x.pos[2]
  handles.sort(key=k)
  top_handle = handles[-1]
  x,y,z = top_handle.pos
  map[x,y,z] = 1
  return smooth(map)
```

VoxPoser. Huang et al. CoRL 2023.

VoxPoser

- **Key Idea**: Given a language goal, LLMs and VLMs can write code to iteratively assign reward values to different locations in a 3D voxel grid that represents robot workspace.

Open the top drawer.

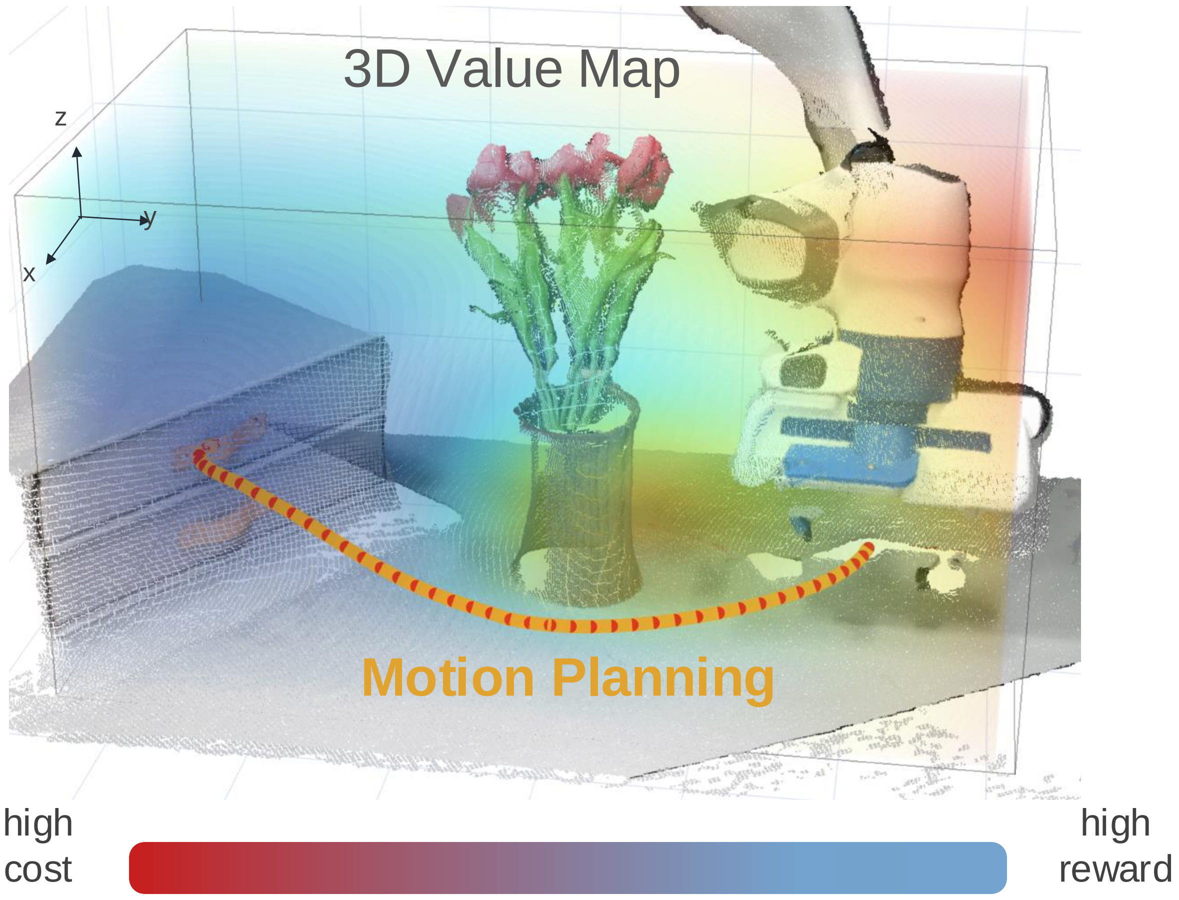
LLM Output

```
def value_map():
  msize = (100,100,100)
  map = np.zeros(msize)
  handles = detect('handle')
  k = lambda x: x.pos[2]
  handles.sort(key=k)
  top_handle = handles[-1]
  x,y,z = top_handle.pos
  map[x,y,z] = 1
  return smooth(map)
```

VoxPoser. Huang et al. CoRL 2023.

VoxPoser

- After obtaining the **3D value maps**, perform **motion planning** to obtain end-effector actions.



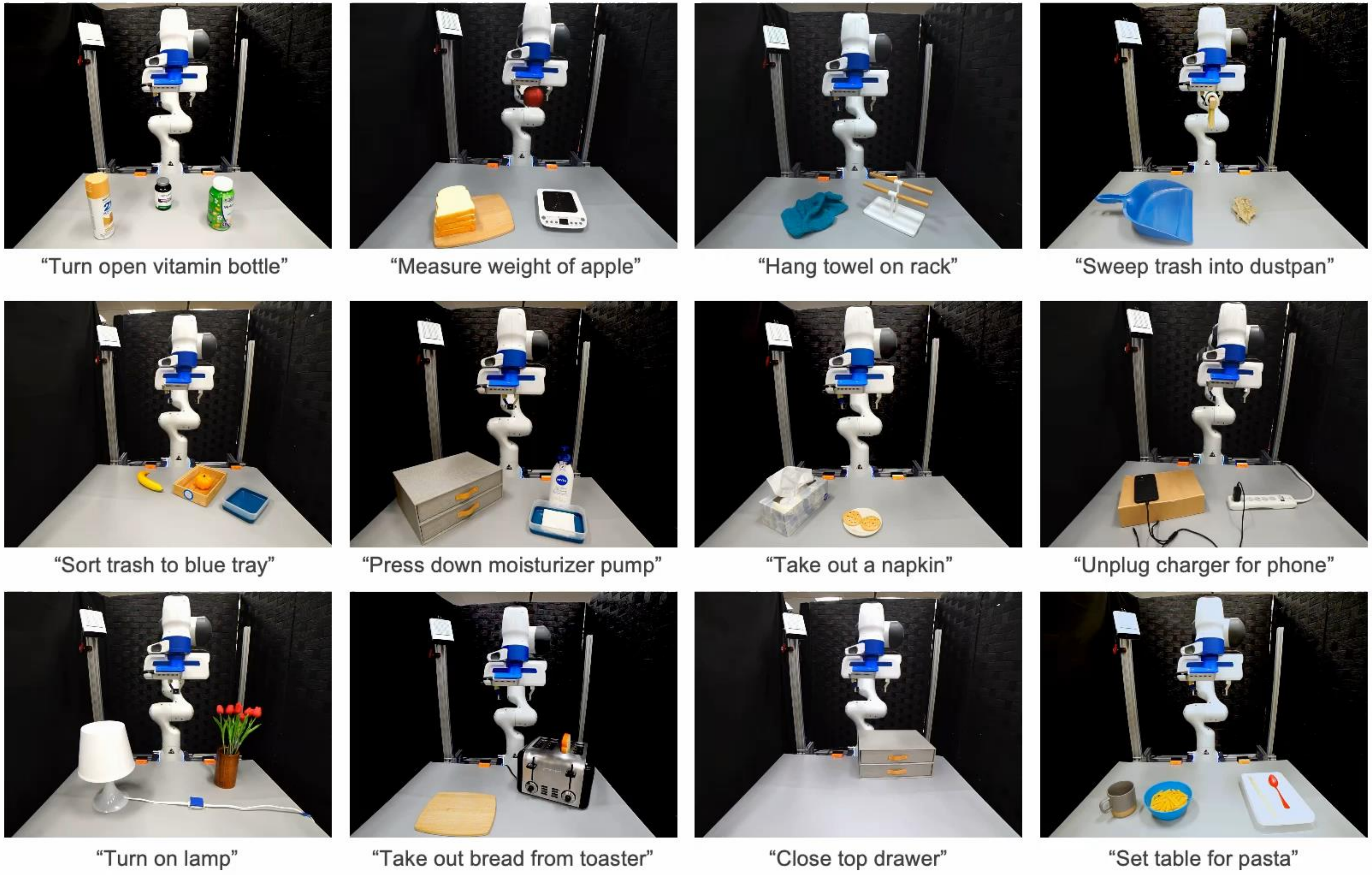
Open the top drawer.

LLM Output

```
def value_map():
  msize = (100,100,100)
  map = np.zeros(msize)
  handles = detect('handle')
  k = lambda x: x.pos[2]
  handles.sort(key=k)
  top_handle = handles[-1]
  x,y,z = top_handle.pos
  map[x,y,z] = 1
  return smooth(map)
```

VoxPoser. Huang et al. CoRL 2023.

VoxPoser



VoxPoser. Huang et al. CoRL 2023.

VoxPoser

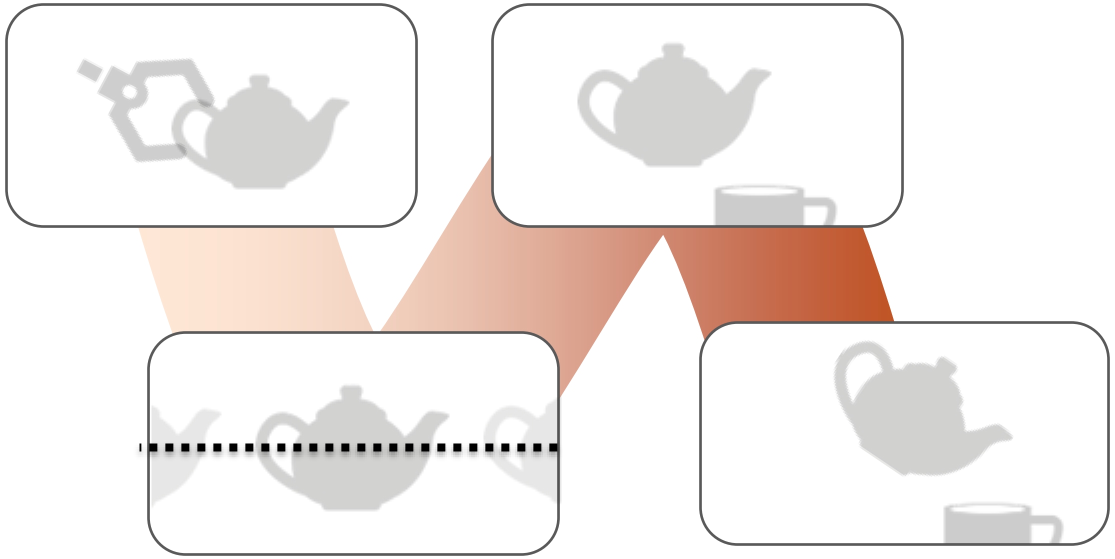
- **Take-away**:
  - LLMs can specify voxel-based reward by using a code interface that can be more applicable to real-world execution
  - Transition model for the environment is challenging to obtain because it needs to work in-the-wild, so only robot transition model is used.
    - **The implication**: only applicable to quasi-static and relatively simple tasks.

VoxPoser. Huang et al. CoRL 2023.

VoxPoser

- **Take-away**:
  - LLMs can specify voxel-based reward by using a code interface that can be more applicable to real-world execution
  - Transition model for the environment is challenging to obtain because it needs to work in-the-wild, so only robot transition model is used.
    - **The implication**: only applicable to quasi-static and relatively simple tasks.
  - The generated reward function does not consider temporal dependencies of actions.

A “pour tea” task



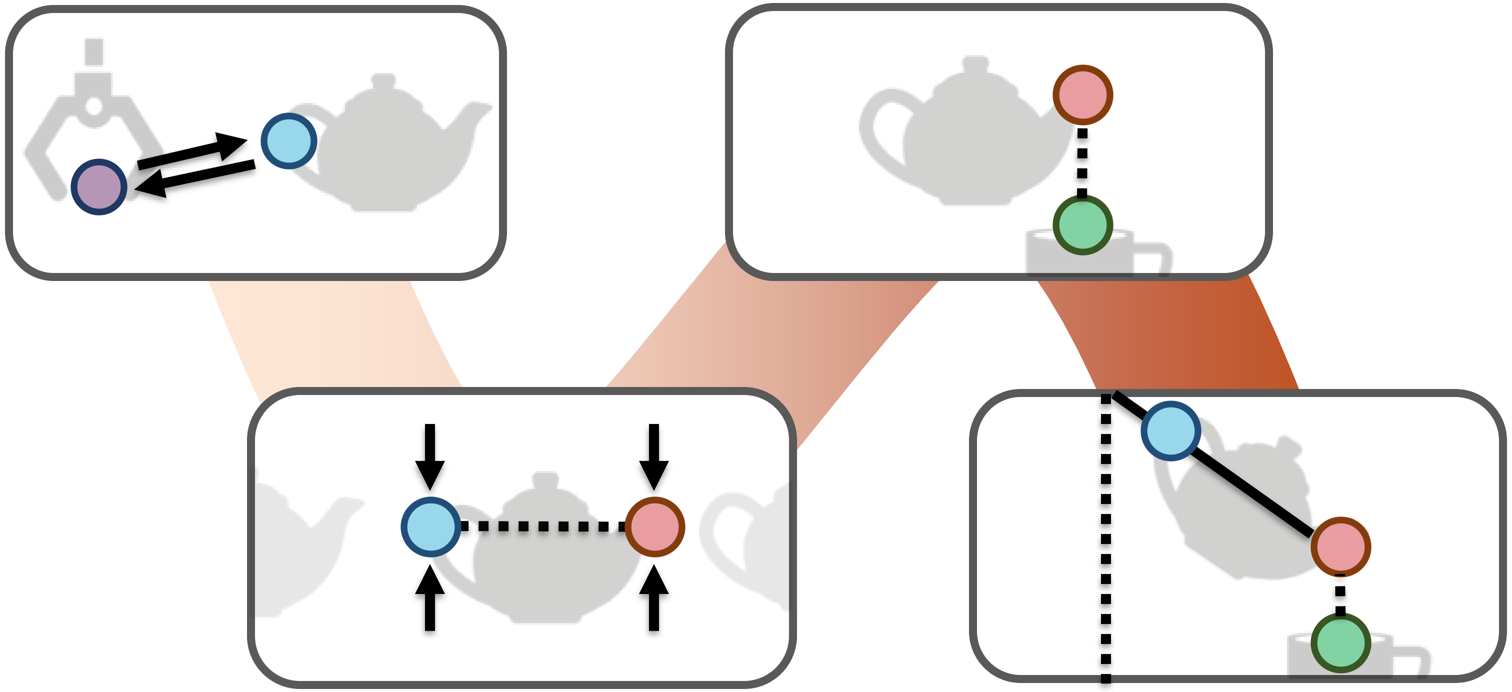
VoxPoser. Huang et al. CoRL 2023.

Reward Specification with LLMs / VLMs

- Case studies:
  - Language to Rewards & Eureka
  - VoxPoser
  - ReKep:
    - **State Representation**: 3D keypoints
    - **Transition Function**: Rigid attachment
    - **Action Space**: End-effector pose

ReKep

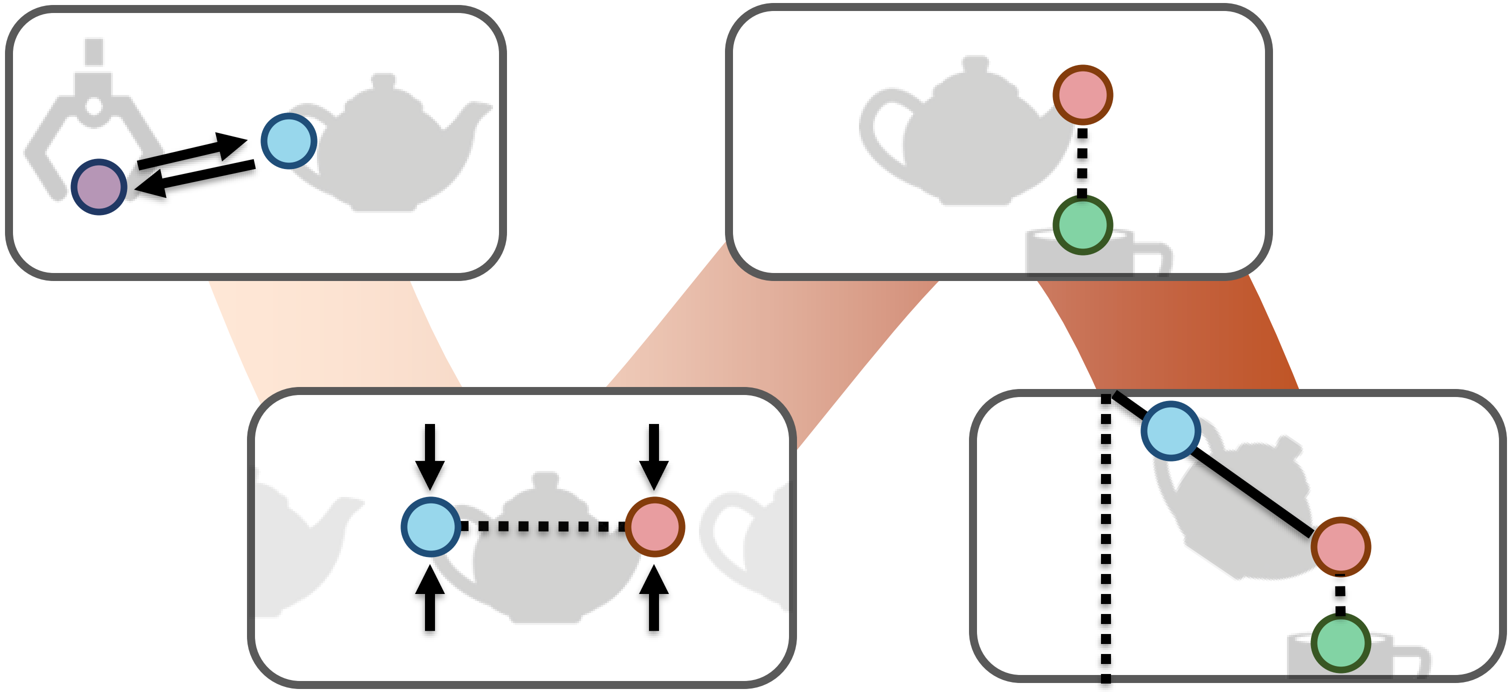
- **Key Idea**:
  - Represent tasks as a sequence of keypoint-based constraint functions.



ReKep. Huang et al. CoRL 2024.

ReKep

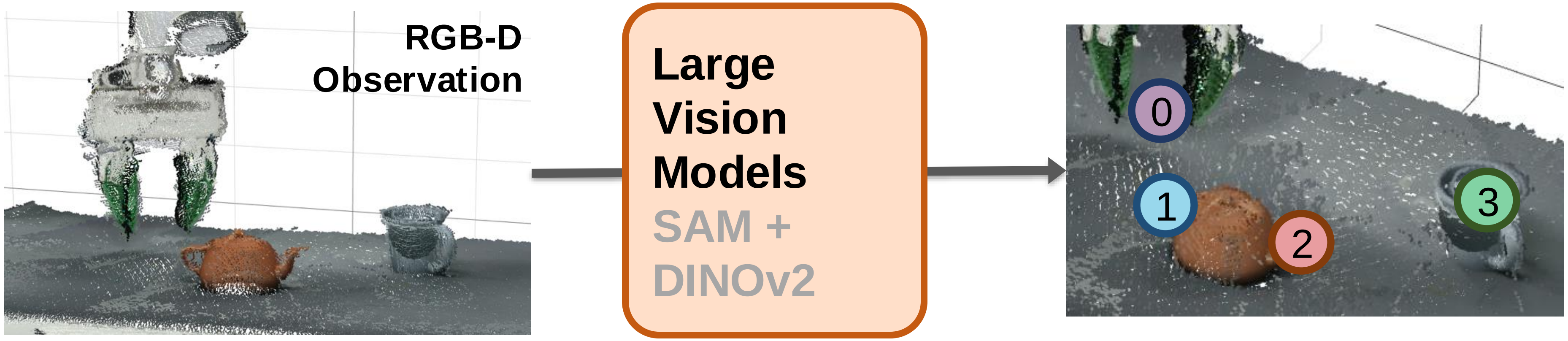
- **Key Idea**:
  - Represent tasks as a sequence of keypoint-based constraint functions.
  - Using 3D keypoints as state ($\boldsymbol{S}$) and end-effector pose as action, we may use a transition model ($\boldsymbol{S} \times \boldsymbol{A} \rightarrow \boldsymbol{S}$) based on rigid attachment assumption (i.e., transform the “attached-to-robot” keypoints by the action).



ReKep. Huang et al. CoRL 2024.

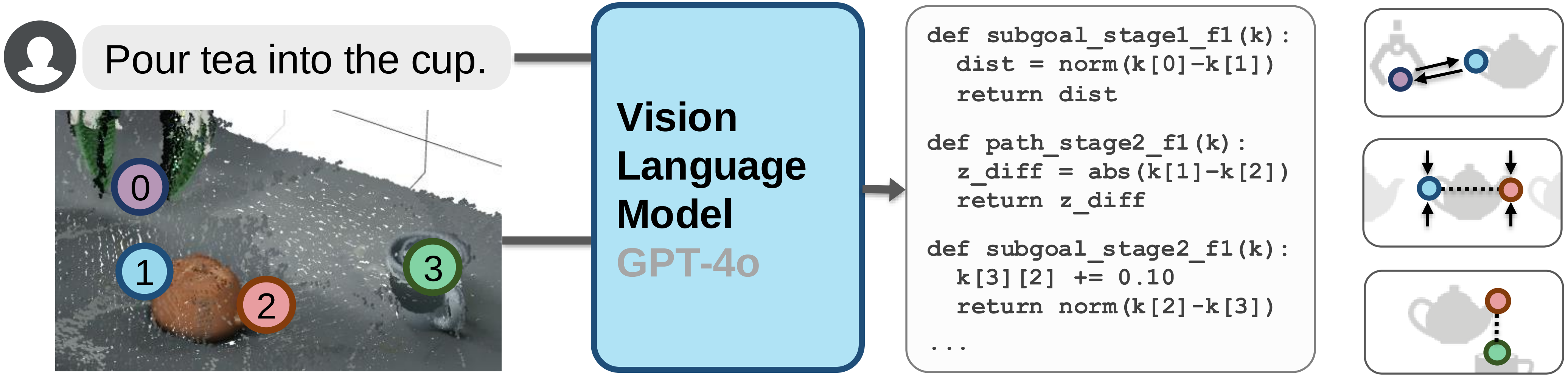
ReKep

- **Step 1**: Obtain a set of semantically meaningful keypoints in the scene



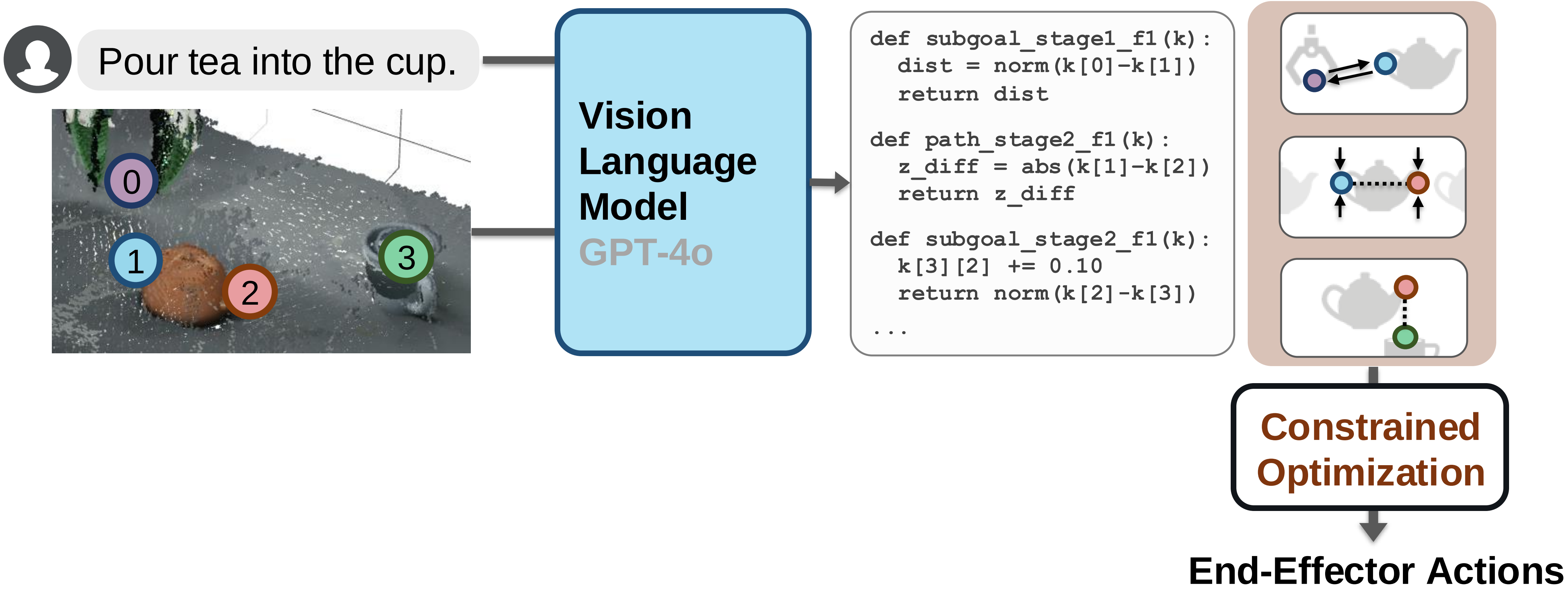
ReKep

- **Step 1**: Obtain a set of semantically meaningful keypoints in the scene
- **Step 2**: Visually prompt VLM to write keypoint-based constraint code.



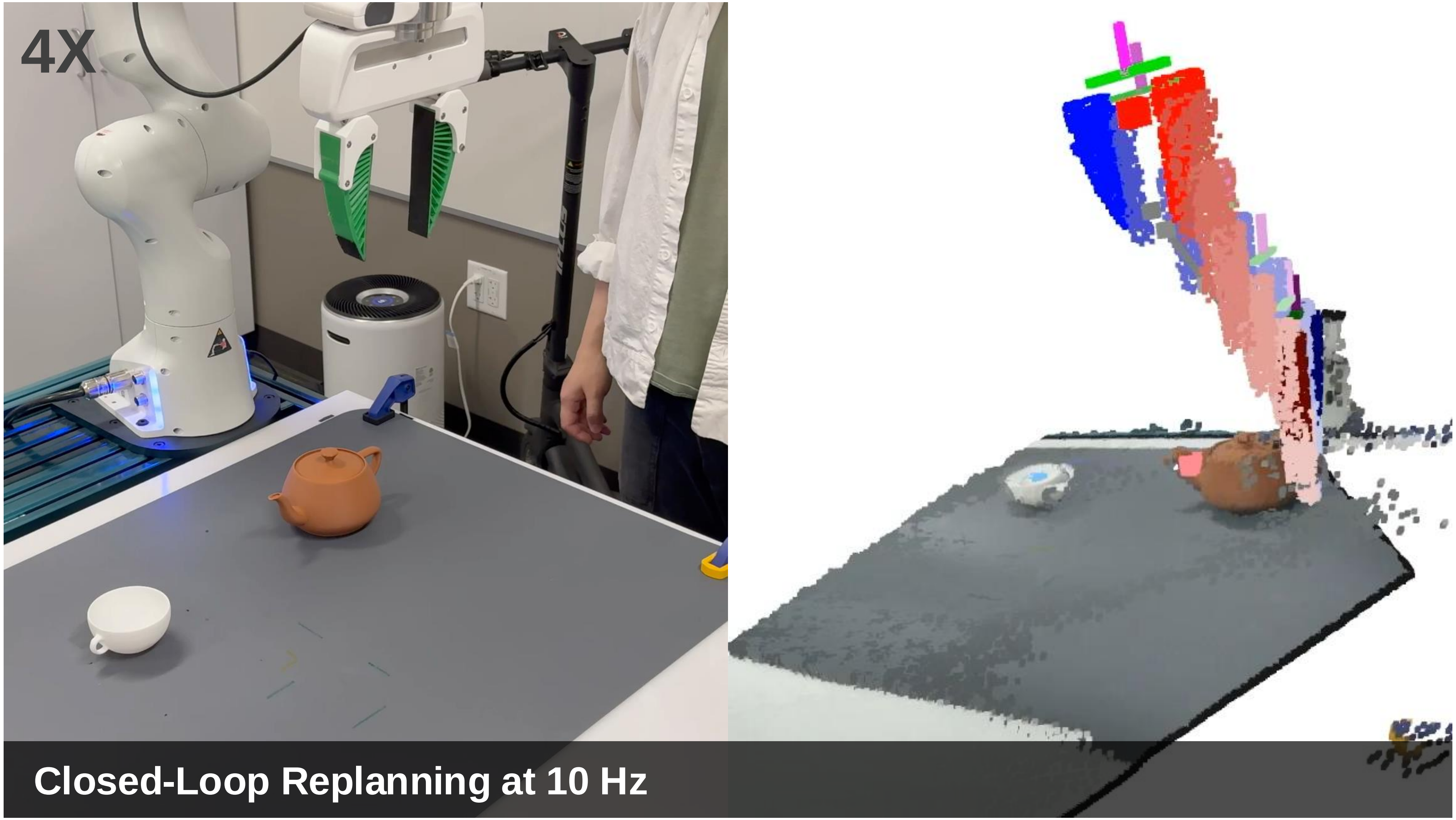
ReKep

- **Step 1**: Obtain a set of semantically meaningful keypoints in the scene
- **Step 2**: Visually prompt VLM to write keypoint-based constraint code.
- **Step 3**: Perform constrained optimization to obtain robot actions.



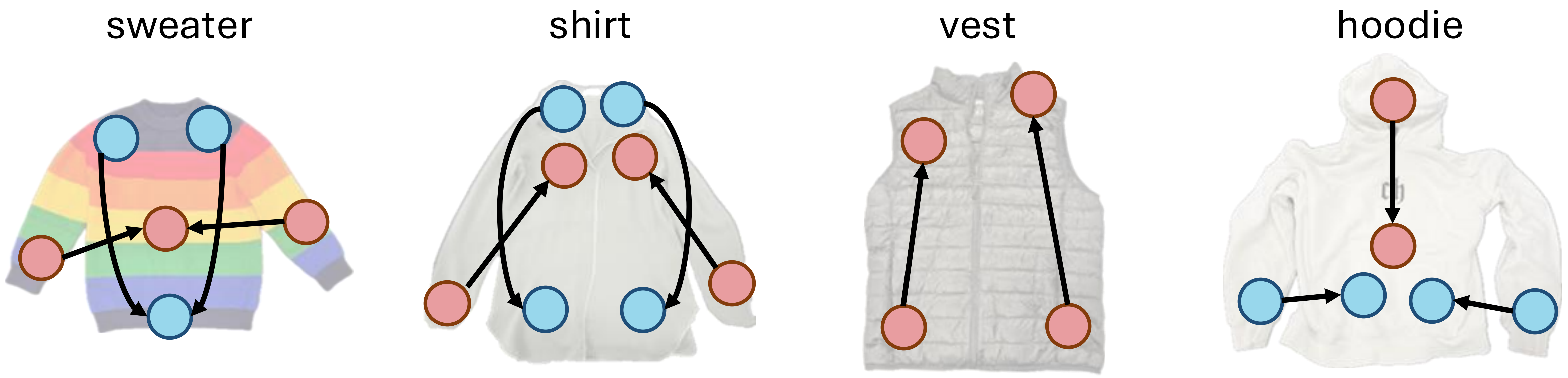
ReKep. Huang et al. CoRL 2024.

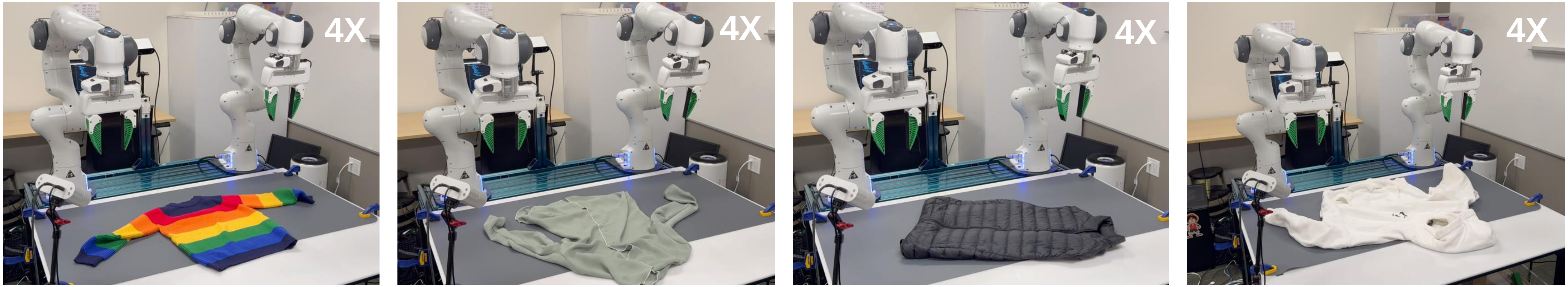
ReKep



ReKep. Huang et al. CoRL 2024.

ReKep





ReKep. Huang et al. CoRL 2024.

Part II Summary

- We can formalize both high-level and low-level action space under the same MDP

Part II Summary

- We can formalize both high-level and low-level action space under the same MDP
- Similar paradigms in both cases for obtaining actions:
  - If expert demos are available, we can directly model $\mathbf{P}(\mathbf{a_t} \mid \mathbf{o_t}, \mathbf{g})$
  - If not, we need a state representation (**S**), reward function (**R**), transition function (**T**). Then we can perform planning to obtain actions.

Part II Summary

- We can formalize both high-level and low-level action space under the same MDP
- Similar paradigms in both cases for obtaining actions:
  - If expert demos are available, we can directly model $\mathbf{P(a_t \mid o_t, g)}$
  - If not, we need a state representation (**S**), reward function (**R**), transition function (**T**). Then we can perform planning to obtain actions.
- Only for high-level actions: LLMs may be used without finetuning, but certain challenges remained.

Part II Summary

- We can formalize both high-level and low-level action space under the same MDP
- Similar paradigms in both cases for obtaining actions:
  - If expert demos are available, we can directly model $\mathbf{P(a_t \mid o_t, g)}$
  - If not, we need a state representation (**S**), reward function (**R**), transition function (**T**). Then we can perform planning to obtain actions.
- Only for high-level actions: LLMs may be used without finetuning, but certain challenges remained.
- Low-level actions are significantly more challenging.
  - Modeling reward remains an effective way of integrating the knowledge LLMs/VLMs for action generation.
  - The choice of a state representation is crucial as it impacts how the reward and transition functions are defined and how they can be obtained.