Text2Reward: Dense Reward Generation with Language Models for Reinforcement Learning

Xu Tengye

TEXT2REWARD: REWARD SHAPING WITH LANGUAGE MODELS FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

8 8 6 6

INTRODUCTION

- Reward shaping

  Design **reward functions** that guide an agent towards desired behaviors more efficiently

- Traditional RL

  manually designing rewards based on expert intuition and heuristics

  - time-consuming, demands expertise and can be sub-optimal.

- Inverse reinforcement learning

  - necessitates a large amount of high-quality trajectory data

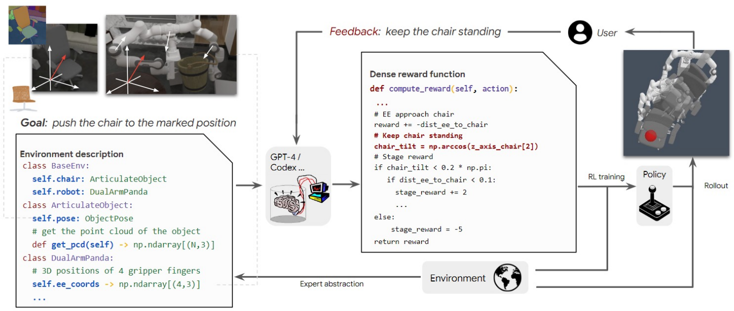
- Preference Learning

  - requires human-annotated preference data

Data-free Automates the generation and shaping of dense reward function

Text2Reward

- Given a goal described in natural language, TEXT2REWARD **generates shaped dense reward functions** as an executable program grounded in a compact representation of the environment
- **zero-shot and few-shot** dense reward generation can achieve **similar or better task success rates** and convergence speed than expert-written reward codes
- allow **iterative refinement** with human feedback
- Real robot experiments

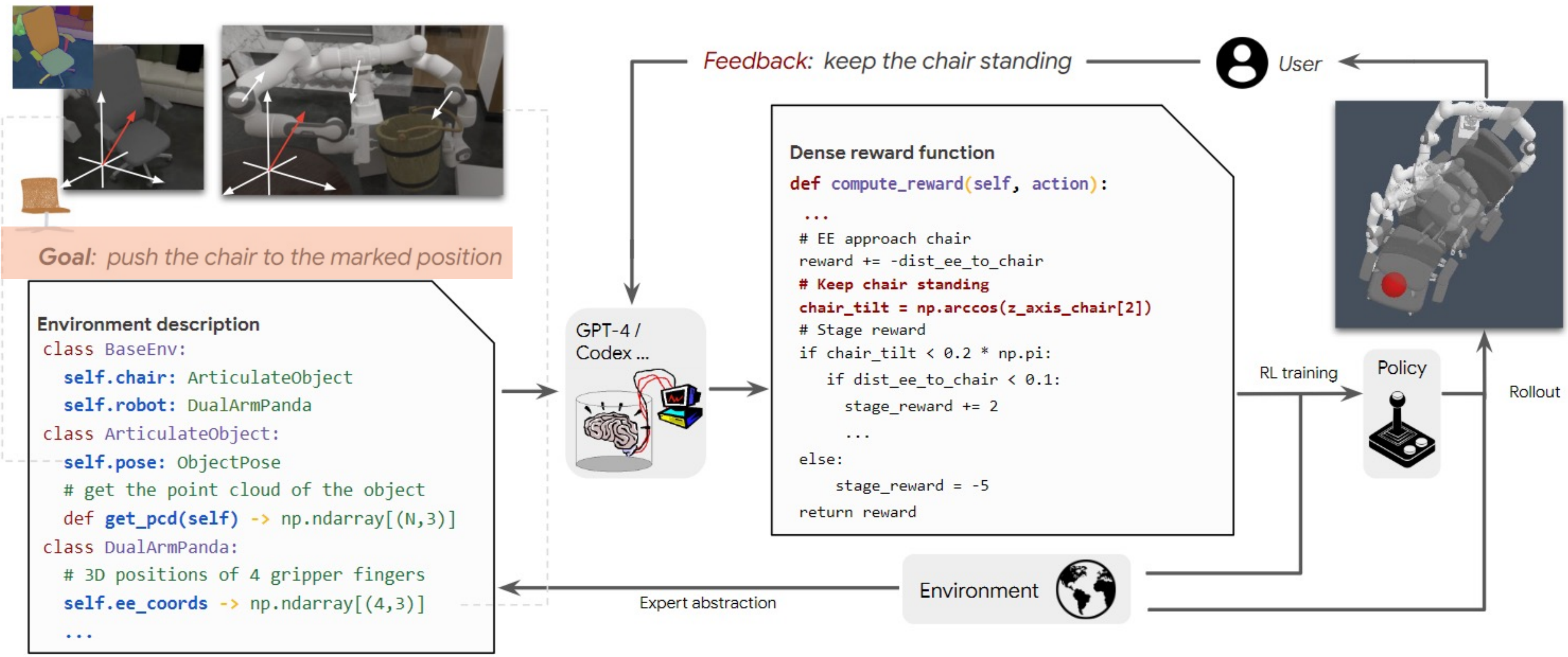


shaped dense reward

- **Task completion rewards**
  - sparse and delayed

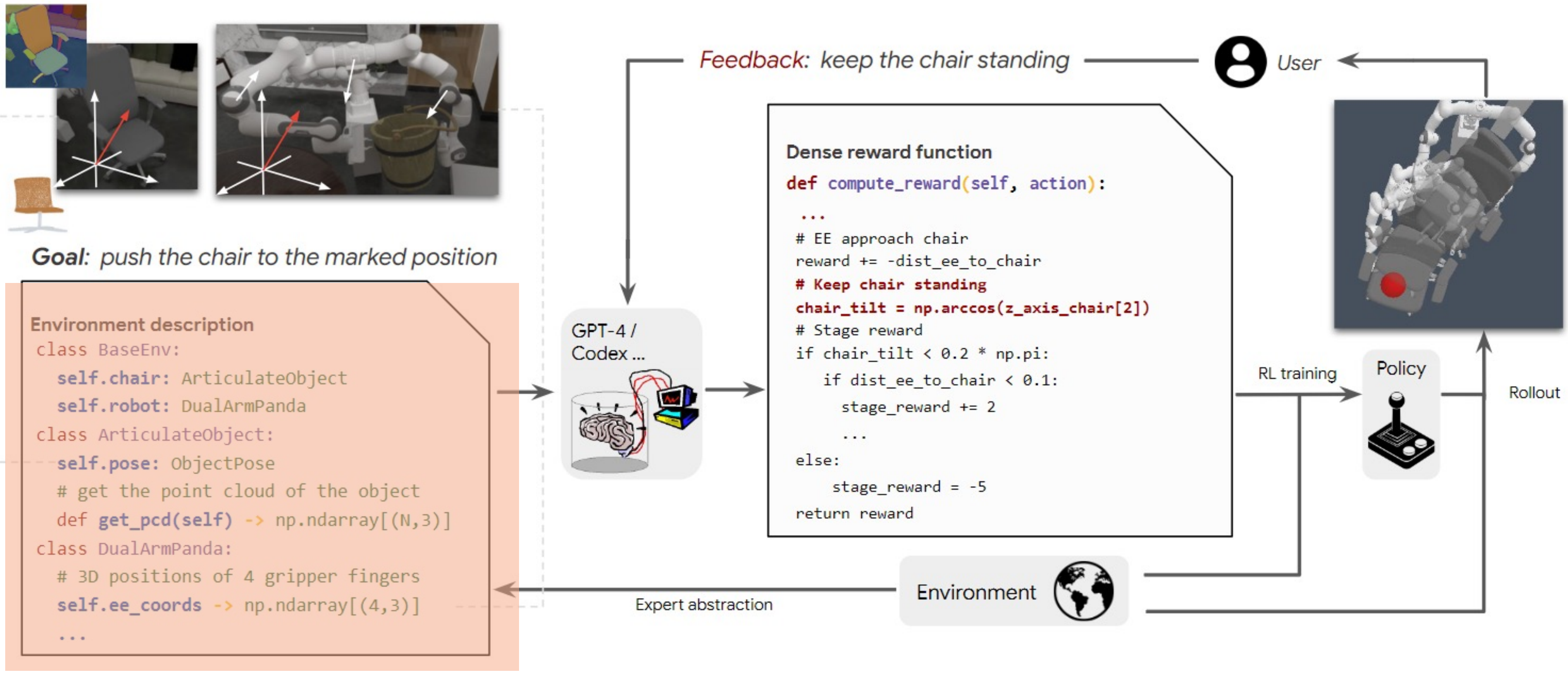
- A **shaped dense reward** function
  - It encourages key intermediate steps and regularization that help achieve the goal.
  - It can take different functional forms at each timestep, instead of being constant across timesteps or just at the end of the episode.

APPROACH



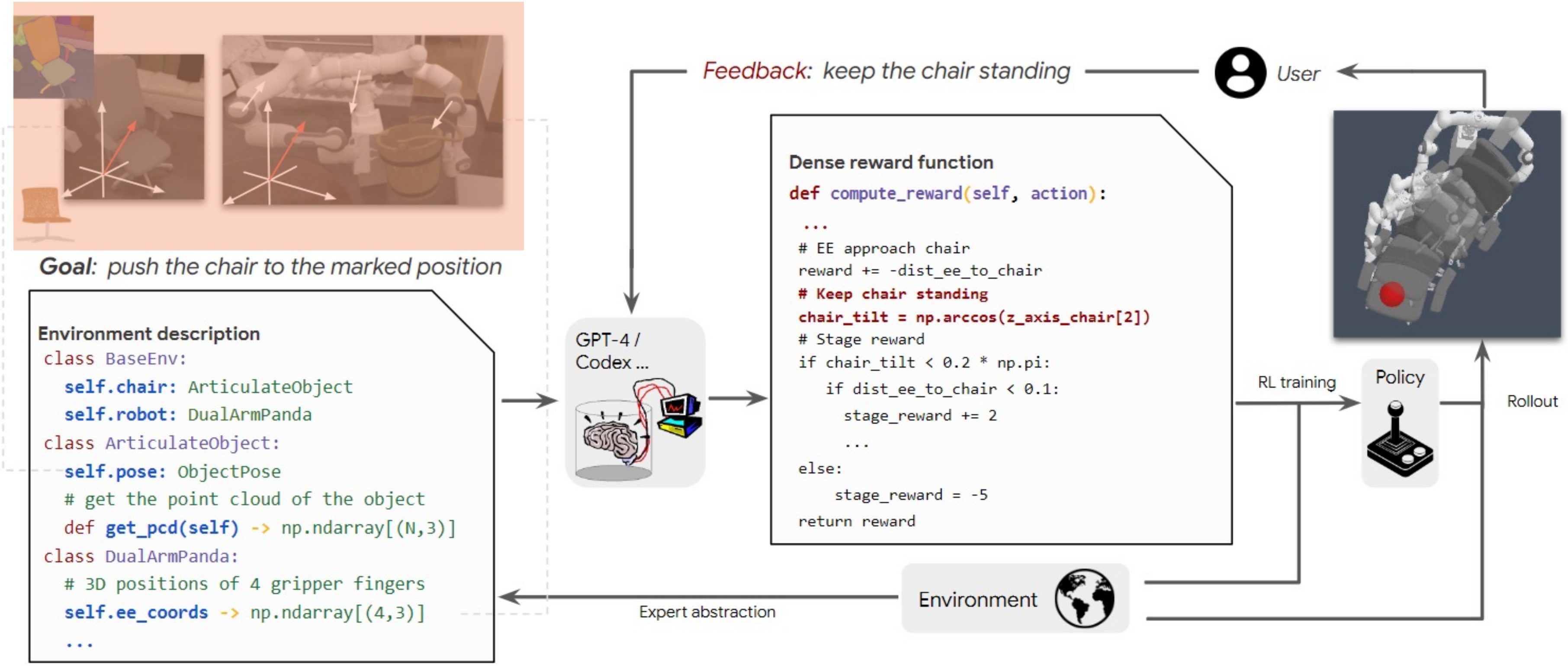
- **Instruction**
  - natural language sentence
  - It can be provided by the user, or it can be one of the subgoals for a long-horizon task, planned by the LLM

APPROACH



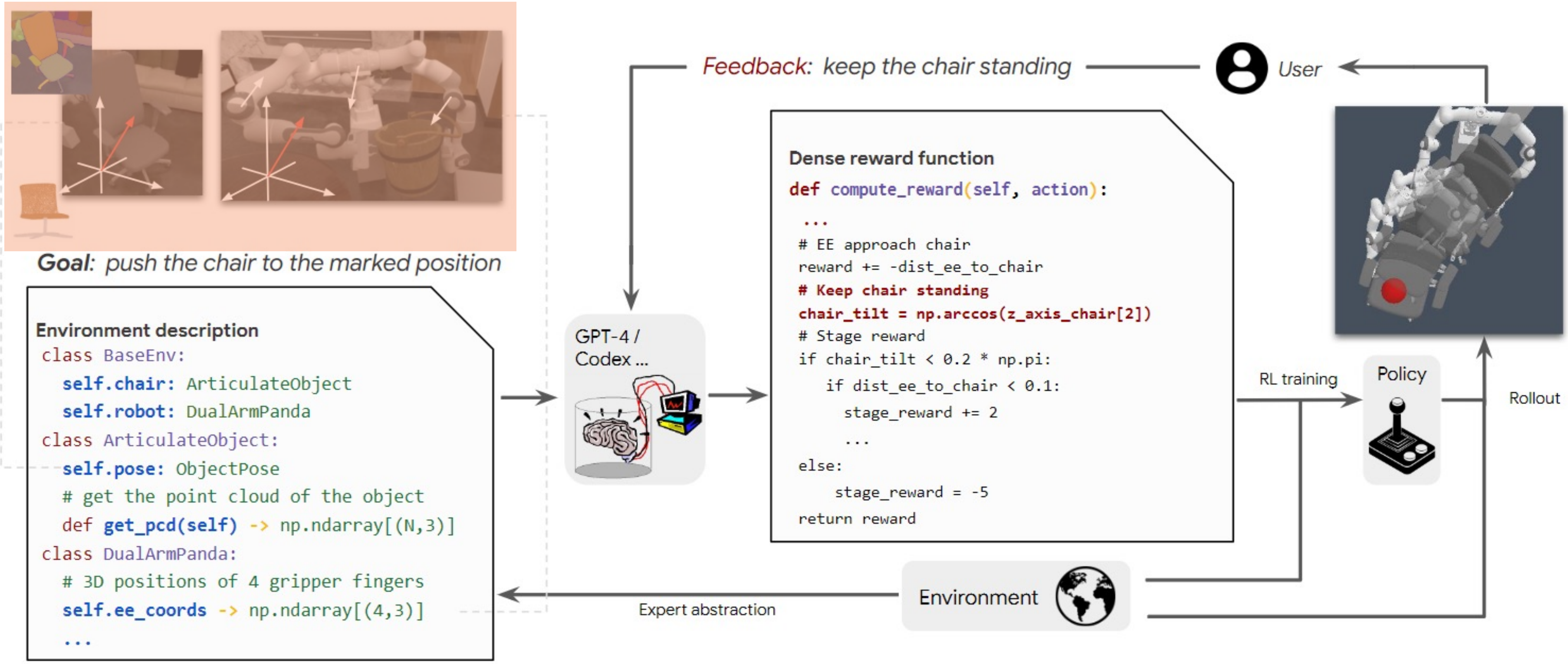
- **Environment abstraction**
  - a compact representation in <u>Pythonic</u> style
  - general, reusable prompts

APPROACH



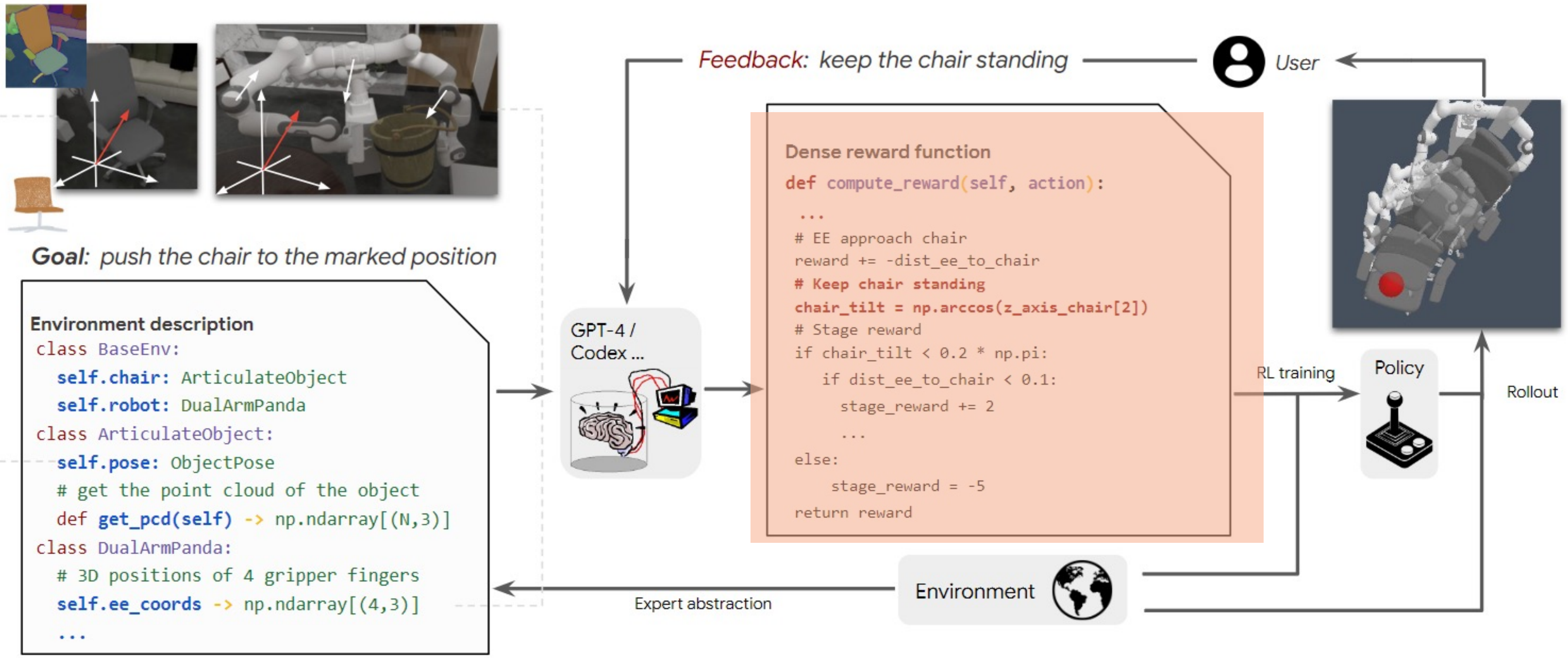
- **Background knowledge**
  - provide functions , e.g., NumPy/SciPy functions , and its usage examples

APPROACH



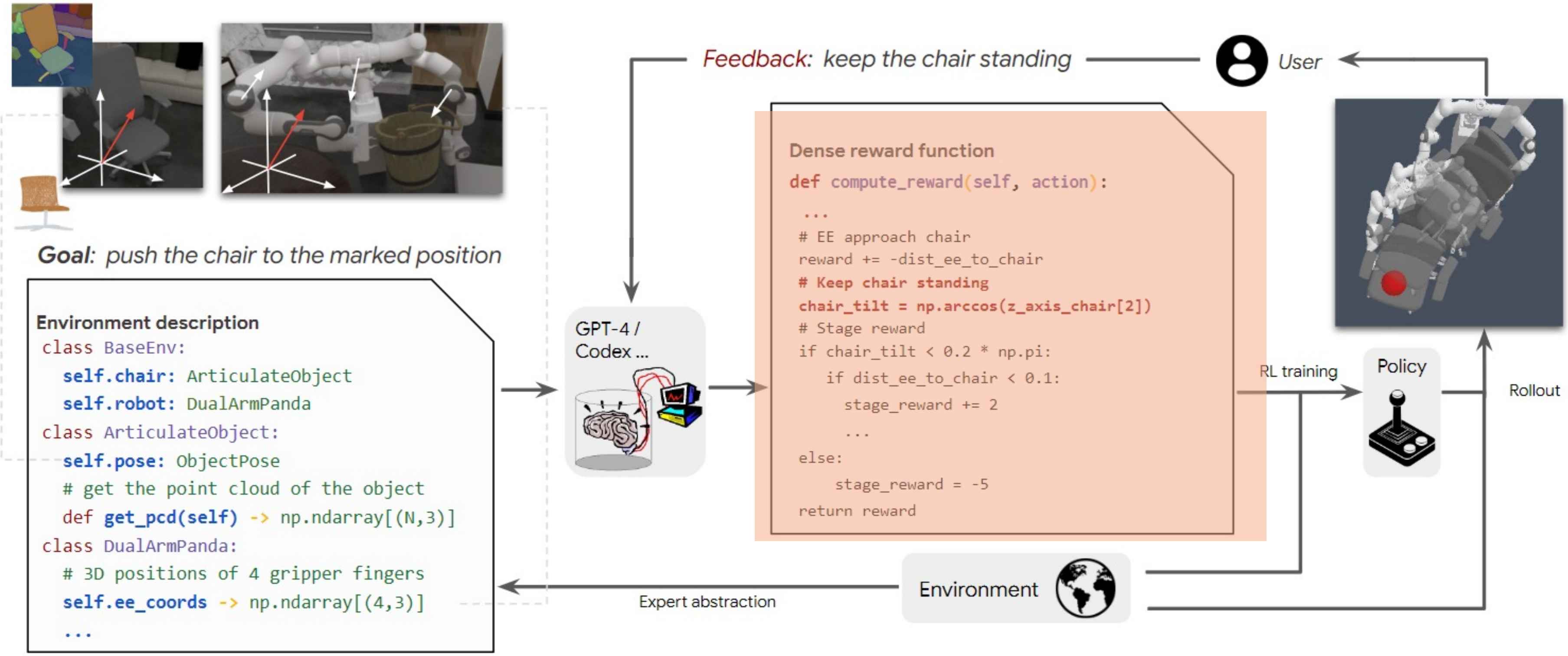
- **Few-shot examples**
- a pool of pairs of instructions and verified reward codes.
- Sentence-T5 embeddings -> encode the instructions
- for a new instructions -> retrieve the top-k similar instructions and concatenate the instruction-code pairs as few-shot examples.

APPROACH



- **Reward code**
- focus on the reward code given its <u>interpretability</u>.

APPROACH



- **Reducing error with code execution**
- execute the code in the code interpreter.
- decreases error rates from 10% to near zero.

Prompt

You are an expert in robotics, reinforcement learning and code generation. We are going to use a Franka Panda robot to complete given tasks. The action space of the robot is a normalized

'Box(-1, 1, (7,), float32)'. Now I want you to help me write a reward function for reinforcement learning. I'll give you the attributes of the environment. You can use these class attributes to write the reward function.

Typically, the reward function of a manipulation task is consisted of these following parts:
1. the distance between robot's gripper and our target object
2. difference between current state of object and its goal state
3. regularization of the robot's action
4. [optional] extra constraint of the target object, which is often implied by task instruction
5. [optional] extra constraint of the robot, which is often implied by task instruction
...

```
...

class BaseEnv(gym.Env):
    self.cubeA : RigidObject # cube A in the environment
    self.cubeB : RigidObject # cube B in the environment
    self.cube_half_size = 0.02 # in meters
    self.robot : PandaRobot # a Franka Panda robot

class PandaRobot:
    self.ee_pose : ObjectPose # 3D position and quaternion of robot's end-effector
    self.lfinger : LinkObject # left finger of robot's gripper
    self.rfinger : LinkObject # right finger of robot's gripper
    self.qpos : np.ndarray[(7,)] # joint position of the robot
    self.qvel : np.ndarray[(7,)] # joint velocity of the robot
    self.gripper_openness : float # openness of robot gripper, normalized range in [0, 1]
    def check_grasp(self, obj : Union[RigidObject, LinkObject], max_angle=85) -> bool
        # indicate whether robot gripper successfully grasp an object

class ObjectPose:
    self.p : np.ndarray[(3,)] # 3D position of the rigid object
    self.q : np.ndarray[(4,)] # quaternion of the rigid object
    def inv(self,) -> ObjectPose # return a 'ObjectPose' class instance, which is the inverse
                                 # of the original pose
```

Prompt

```
class ArticulateObject:
    self.pose : ObjectPose # 3D position and quaternion of the articulated object
    self.velocity : np.ndarray[(3,)] # linear velocity of the articulated object
    self.angular_velocity : np.ndarray[(3,)] # angular velocity of the articulated object
    self.qpos : np.ndarray[(K,)] # position of the articulated object joint
    self.qvel : np.ndarray[(K,)] # velocity of the articulated object joint
    def get_pcd(self,) -> np.ndarray[(M,3)] # point cloud of the articulated object surface
                                            # in the world frame
```

```
Additional knowledge:
1. A staged reward could make the training more stable, you can write them in a nested
   if-else statement.
2. `ObjectPose` class support multiply operator `*`, for example: `ee_pose_wrt_cubeA =
   self.cubeA.pose.inv() * self.robot.ee_pose`.
3. You can use `transforms3d.quaternions` package to do quaternion calculation, for example:
   `qinverse(quat: np.ndarray[(4,)])` for inverse of quaternion, `qmult(quat1: np.ndarray[(4,)],
   quat2: np.ndarray[(4,)])` for multiply of quaternion, `quat2axangle(quat: np.ndarray[(4,)])`
   for quaternion to angle.
```

```
I want you to fulfill the following task: {instruction}
1. please think step by step and tell me what does this task mean;
2. then write a function that formats as `def compute_dense_reward(self, action) -> float` and
   returns the `reward : float` only.
3. When write code, you can also add some comments as your thoughts.
```

Generated reward Function

Sample of *zero-shot* generated reward function on MANISKILL2 Pick Cube task:

```
import numpy as np

def compute_dense_reward(self, action):
    # Normalize action
    action = np.clip(action, -1, 1)

    # Calculate distance between gripper and cube
    gripper_pos = self.robot.ee_pose.p
    cube_pos = self.cubeA.pose.p
    dist_gripper_cube = np.linalg.norm(gripper_pos - cube_pos)

    # Calculate distance between cube and goal
    goal_pos = self.goal_position
    dist_cube_goal = np.linalg.norm(goal_pos - cube_pos)

    # Check if the robot is grasping the cube

    grasping_cube = self.robot.check_grasp(self.cubeA)

    # Define reward components
    reward_dist_gripper_cube = -1.0 * dist_gripper_cube
    reward_dist_cube_goal = -1.0 * dist_cube_goal
    reward_grasping_cube = 1.0 if grasping_cube else -1.0

    # Define weights for reward components
    weight_dist_gripper_cube = 0.3
    weight_dist_cube_goal = 0.5
    weight_grasping_cube = 0.2

    # Calculate total reward
    reward = weight_dist_gripper_cube * reward_dist_gripper_cube \
            + weight_dist_cube_goal * reward_dist_cube_goal \
            + weight_grasping_cube * reward_grasping_cube

    # Regularization on action
    reward -= 0.01 * (action ** 2).sum()
```

Prompt-few shot

```
An example:
Tasks to be fulfilled: {instruction}
Corresponding reward function:
```python
{reward_code}
```
```

Generated reward Function

Sample of *few-shot* generated reward function on MANISKILL2 Pick Cube task:

```
import numpy as np

def compute_dense_reward(self, action):
    reward = 0.0

    # Check if the cube is placed at the goal position and the robot is static
    cube_at_goal = np.linalg.norm(self.cubeA.pose.p - self.goal_position) <= self.cube_half_size
    is_robot_static = np.max(np.abs(self.robot.qvel)) <= 0.2

    # If the cube is placed at the goal and the robot is static, return a high reward
    if cube_at_goal and is_robot_static:
        reward += 2.25
        return reward

    # reaching reward, encourages the robot to reach the cube
    gripper_pos = self.robot.ee_pose.p
    obj_pos = self.cubeA.pose.p
    dist_to_obj = np.linalg.norm(gripper_pos - obj_pos)
    reaching_reward = 1 - np.tanh(5 * dist_to_obj)
    reward += reaching_reward

    # grasp reward, encourages the robot to grasp the cube
    is_grasped = self.robot.check_grasp(self.cubeA, max_angle=30)
    if is_grasped:
        reward += 0.25

    # placement reward, encourages the robot to place the cube at the goal
    if is_grasped:
        dist_to_goal = np.linalg.norm(self.cubeA.pose.p - self.goal_position)
        placement_reward = 1 - np.tanh(5 * dist_to_goal)
        reward += placement_reward

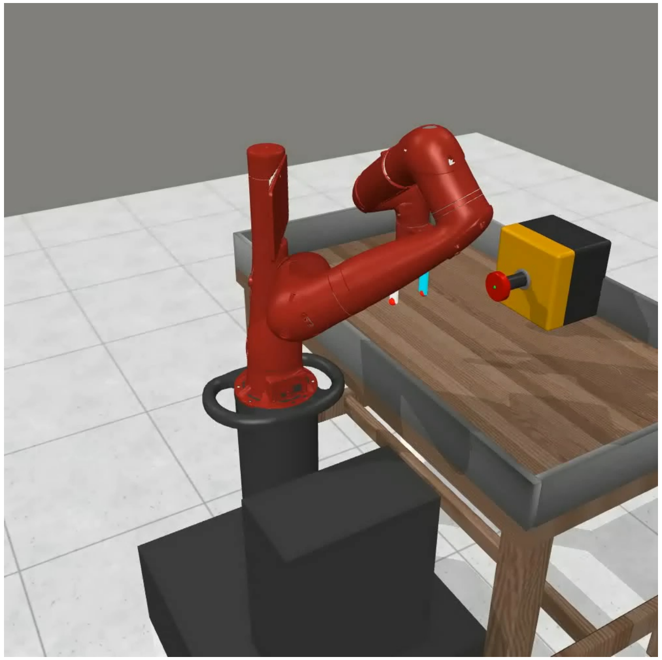
    # regularization term on robot's action
    action_reg = -np.sum(np.square(action)) / len(action)
    reward += 0.1 * action_reg

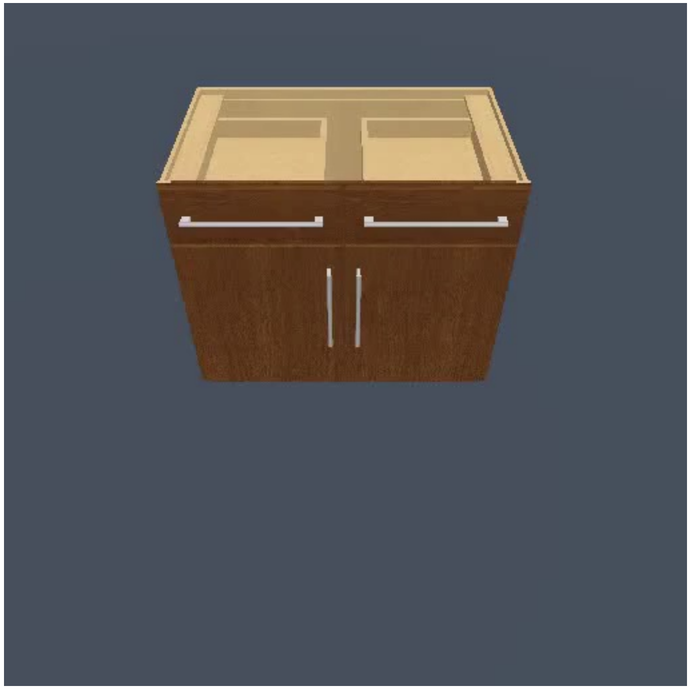
    return reward
```

Results- manipulation

- **Few-shot outperforming zero-shot**
- **Zero-shot sometimes outperforming few-shot**
  - the <u>quality and relevance </u>of the few-shot examples

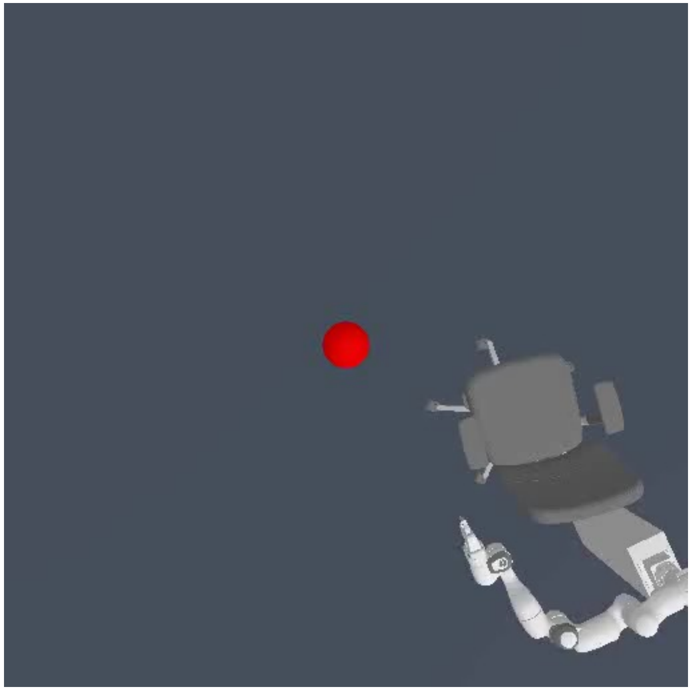
Results- manipulation-zero shot

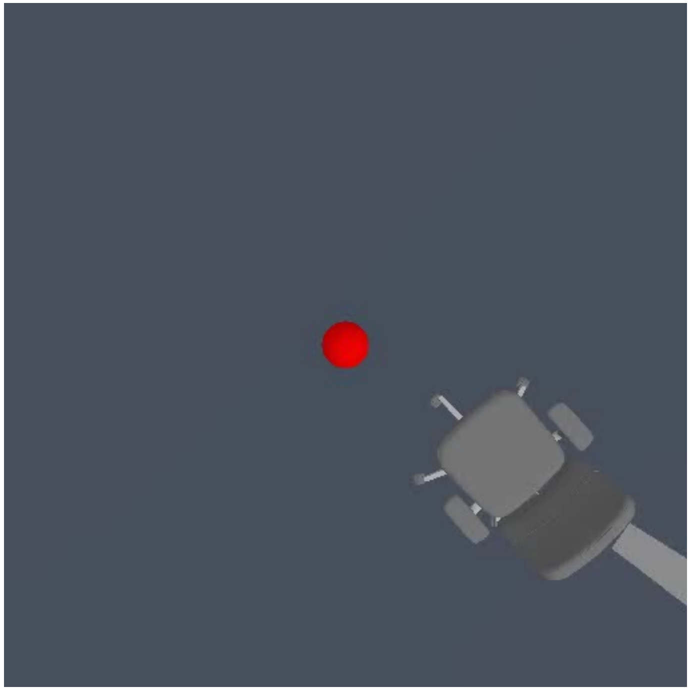




- **Few-shot outperforming zero-shot**
- **Zero-shot sometimes outperforming few-shot**
  - the <u>quality and relevance </u>of the few-shot examples

Results- manipulation

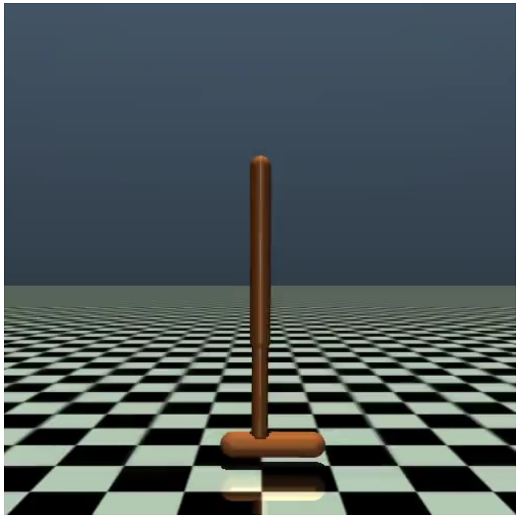


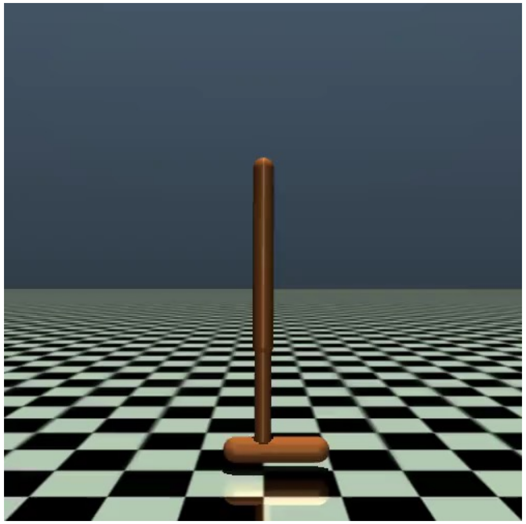


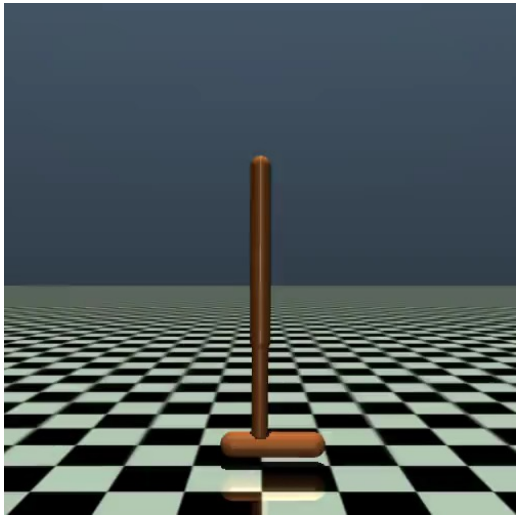
- zero-shot

Few-shot

Results- locomotion-zero shot







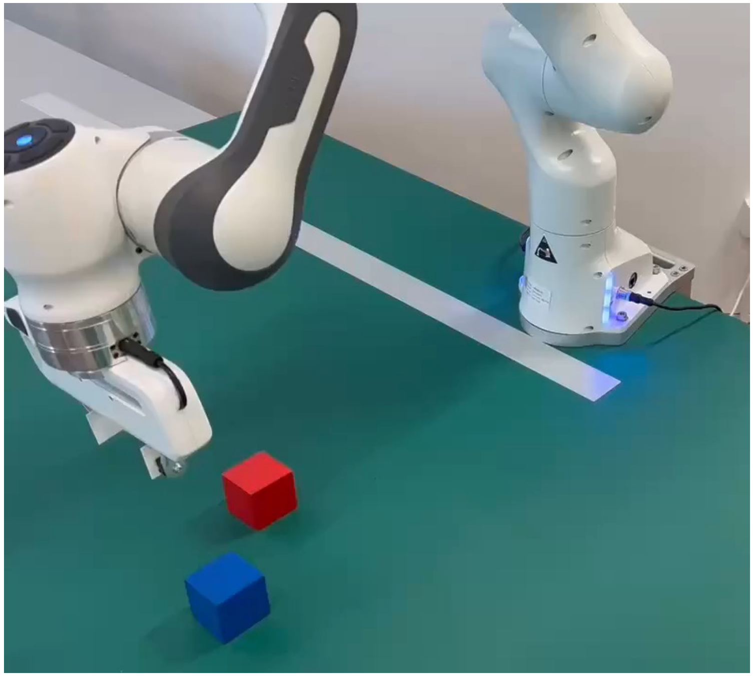
- Front-flip

move

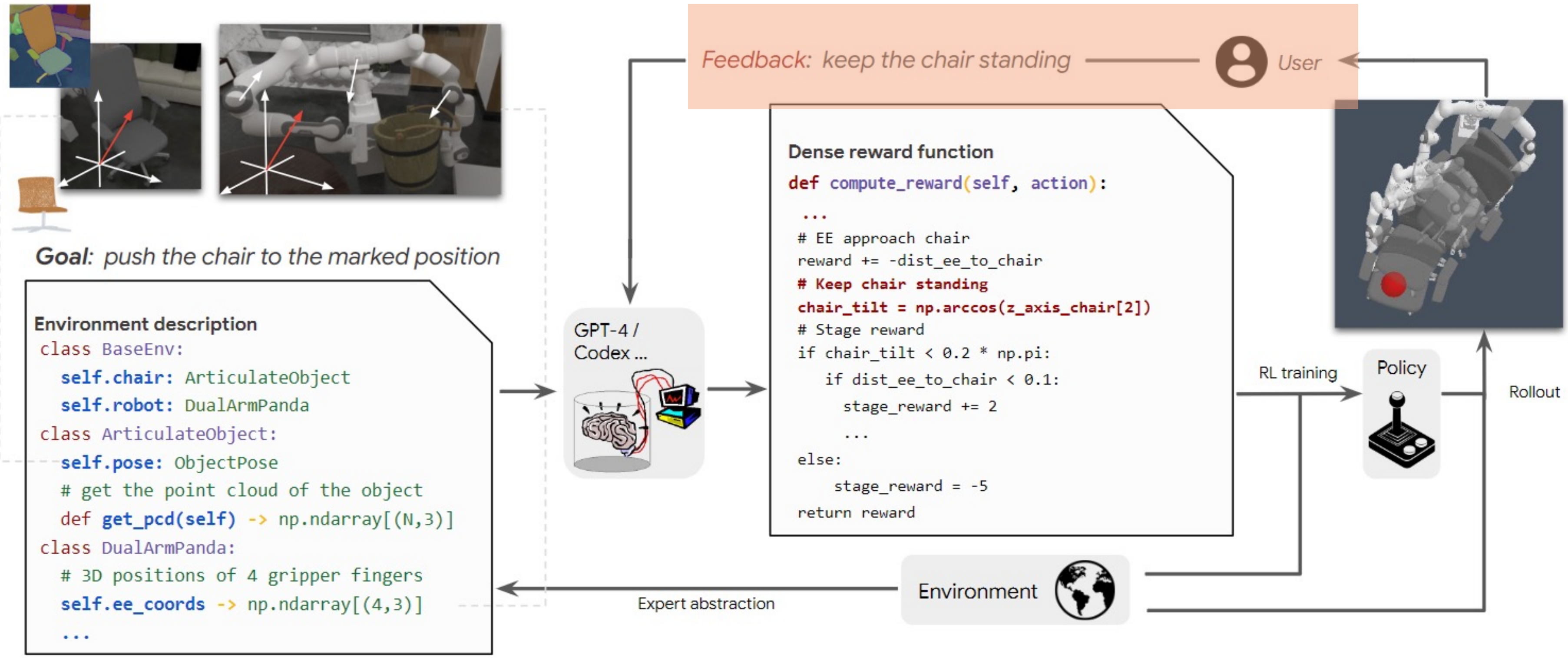
- Back-flip

Results- REAL ROBOT MANIPULATION

- necessitating only minor calibration and the introduction of random noise for sim-to-real transfer.
- a depth camera to get the estimated pose of objects

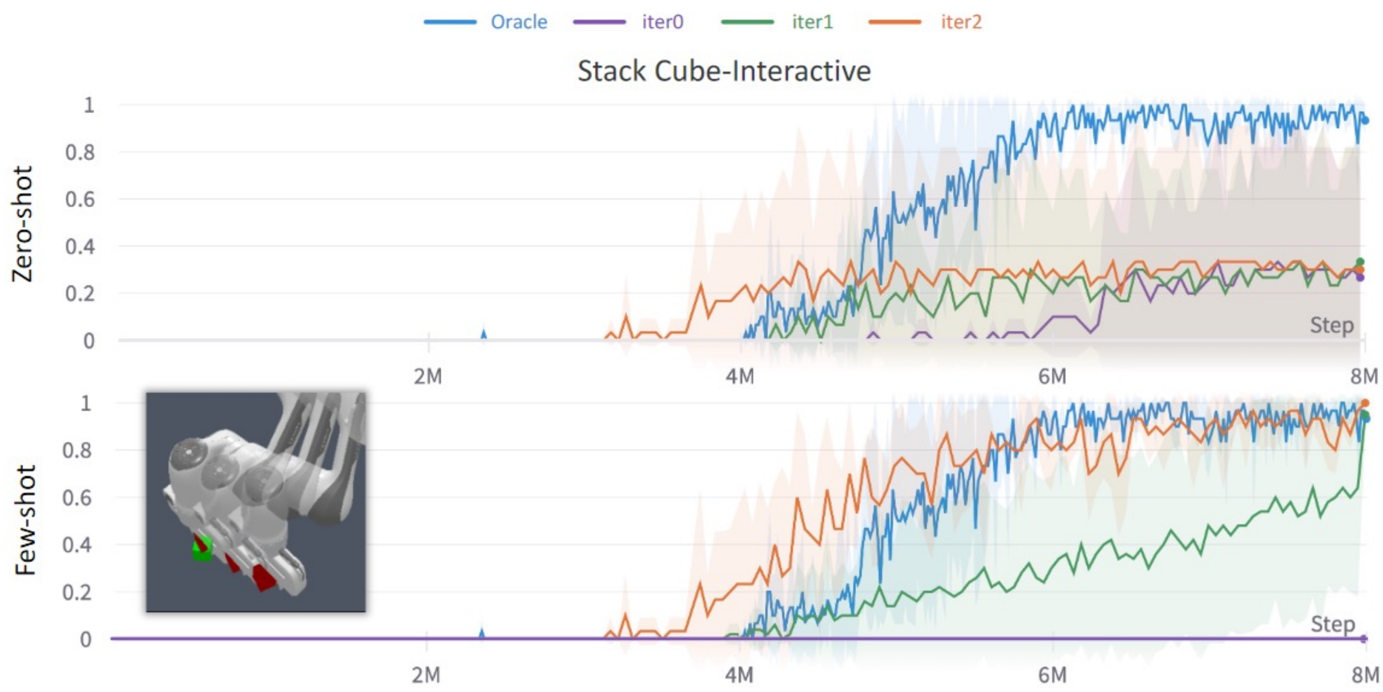


APPROACH



- **Improve reward with human feedback**
- Users then offer critical <u>insights and feedback </u>based on the video, identifying areas of improvement or errors..
- encourages the participation <u>of general users,</u>

Results



- **Improve reward with human feedback**
- Users then offer critical <u>insights and feedback</u> based on the video, identifying areas of improvement or errors..
- encourages the participation <u>of general users,</u>

Other LLM


GPT-4
Code-Llama
Llama-2
Drawer Open
Window Close
Button Press
Sweep Into
Door Close
Handle Press
Door Unlock
Drawer Close
Step
1
0.8
0.6
0.4
0.2
0
200k
400k
600k
800k

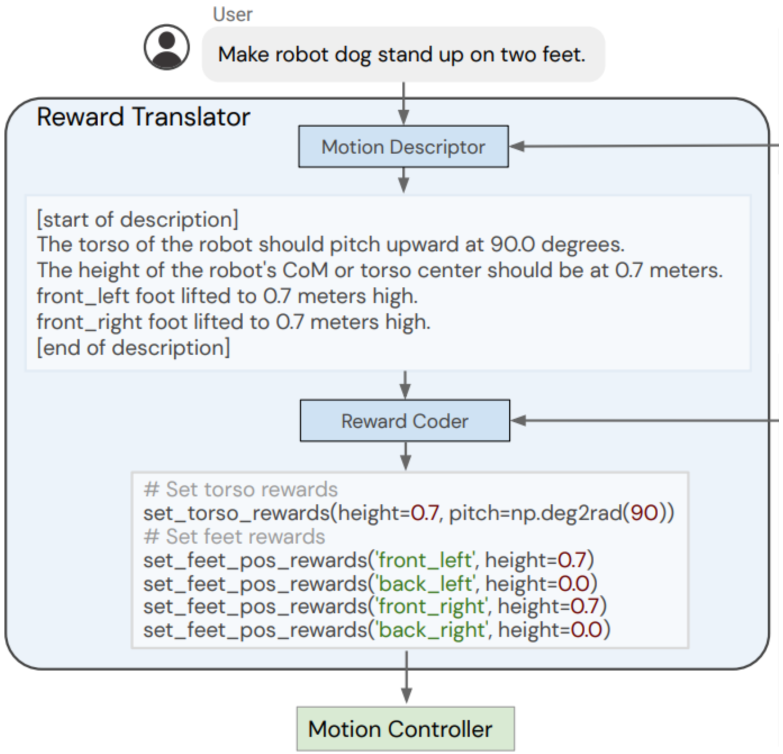
Comparing work

Language to Rewards for Robotic Skill Synthesis

Wenhao Yu[*], Nimrod Gileadi[*], Chuyuan Fu[†], Sean Kirmani[†], Kuang-Huei Lee[†],
Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever,
Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang,
Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, Fei Xia
Google DeepMind

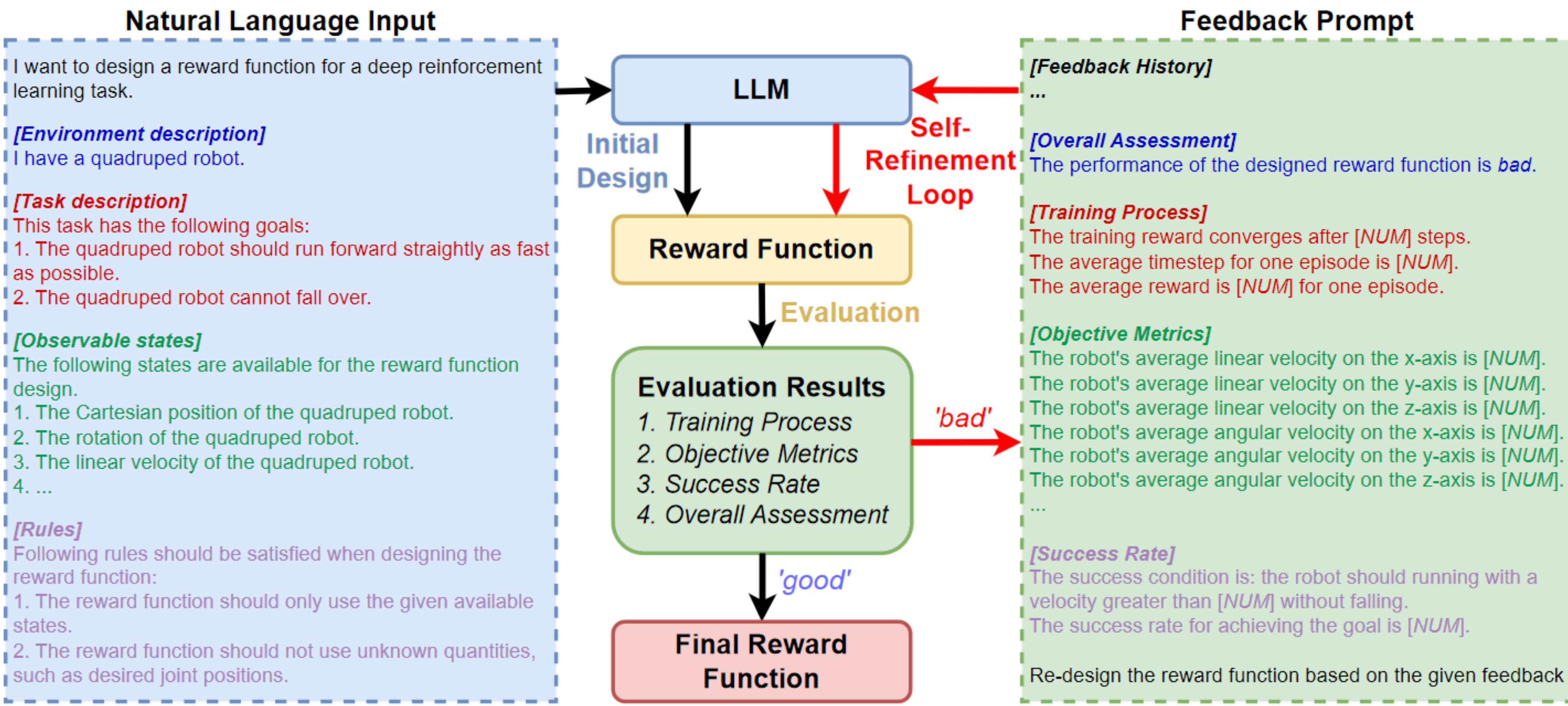
Rewards form
suitable for use with MPC

$$R(\mathbf{s},\mathbf{a}) = -\sum_{i=0}^{M} w_i \cdot \mathrm{n}_i\big(r_i(\mathbf{s},\mathbf{a},\psi_i)\big),$$



Comparing work

Self-Refined Large Language Model as Automated Reward Function Designer for Deep Reinforcement Learning in Robotics



Comparing work

Iteration 0 (Initial)

Reward Function:

$$R = r_{v_x} + r_{\text{bal}} + r_y + r_{\text{rot}}$$

$$r_{v_x} = v_x, r_{\text{bal}} = \begin{cases} 1, & p_z \geq 0.5, \\ 0, & \text{otherwise}, \end{cases}$$

$$r_y = 1 - |v_y|, r_{\text{rot}} = 1 - |\theta_{\text{yaw}}|$$

Evaluation:

$$\text{SR} = 10\%, g_{v_x} = 2.52, g_{v_y} = -0.11,$$
$$g_{v_z} = -0.23, g_{p_z} = 0.61, g_{\text{action}} = 2.36,$$
$$g_{\dot\theta_{\text{roll}}} = -0.07, g_{\dot\theta_{\text{pitch}}} = -0.01, g_{\dot\theta_{\text{yaw}}} = -0.01$$

Iteration 1

Reward Function:

$$R = 2 * r_{v_x} + r_{\text{bal}} + r_{yz} + r_{\text{rot}} + r_{\text{action}}$$

$$r_{v_x} = v_x, r_{\text{bal}} = \begin{cases} 1, & p_z \geq 0.5, \\ 0, & \text{otherwise}, \end{cases}$$

$$r_{yz} = 1 - |v_y| - |v_z|, r_{\text{rot}} = 1 - |\theta_{\text{yaw}}|$$

$$r_{\text{action}} = 1 - \frac{\sum_{i=1}^{n} |a_i|}{n} * \frac{1}{2.36}$$

Evaluation:

$$\text{SR} = 90\%, g_{v_x} = 3.23, g_{v_y} = -0.12,$$
$$g_{v_z} = -0.22, g_{p_z} = 0.59, g_{\text{action}} = 2.74,$$
$$g_{\dot\theta_{\text{roll}}} = -0.1, g_{\dot\theta_{\text{pitch}}} = -0.02, g_{\dot\theta_{\text{yaw}}} = -0.01$$

Iteration 2 (final)

Reward Function:

$$R = 2 * r_{v_x} + r_{\text{bal}} + r_{yz} + r_{\text{rot}} + r_{\text{action}}$$

$$r_{v_x} = v_x, r_{\text{bal}} = \begin{cases} 1, & p_z \geq 0.5, \\ 0, & \text{otherwise}, \end{cases}$$

$$r_{yz} = 1 - 2 * (|v_y| + |v_z|), r_{\text{rot}} = 1 - |\theta_{\text{yaw}}|$$

$$r_{\text{action}} = 1 - \frac{\sum_{i=1}^{n} |a_i|}{n} * \frac{1}{2.74}$$

Evaluation:

$$\text{SR} = 98\%, g_{v_x} = 3.76, g_{v_y} = -0.11,$$
$$g_{v_z} = -0.21, g_{p_z} = 0.6, g_{\text{action}} = 2.67,$$
$$g_{\dot\theta_{\text{roll}}} = -0.04, g_{\dot\theta_{\text{pitch}}} = -0.01, g_{\dot\theta_{\text{yaw}}} = 0.01$$

Weight/Parameter Adjustment

Reward Component Adjustment

Comparing work

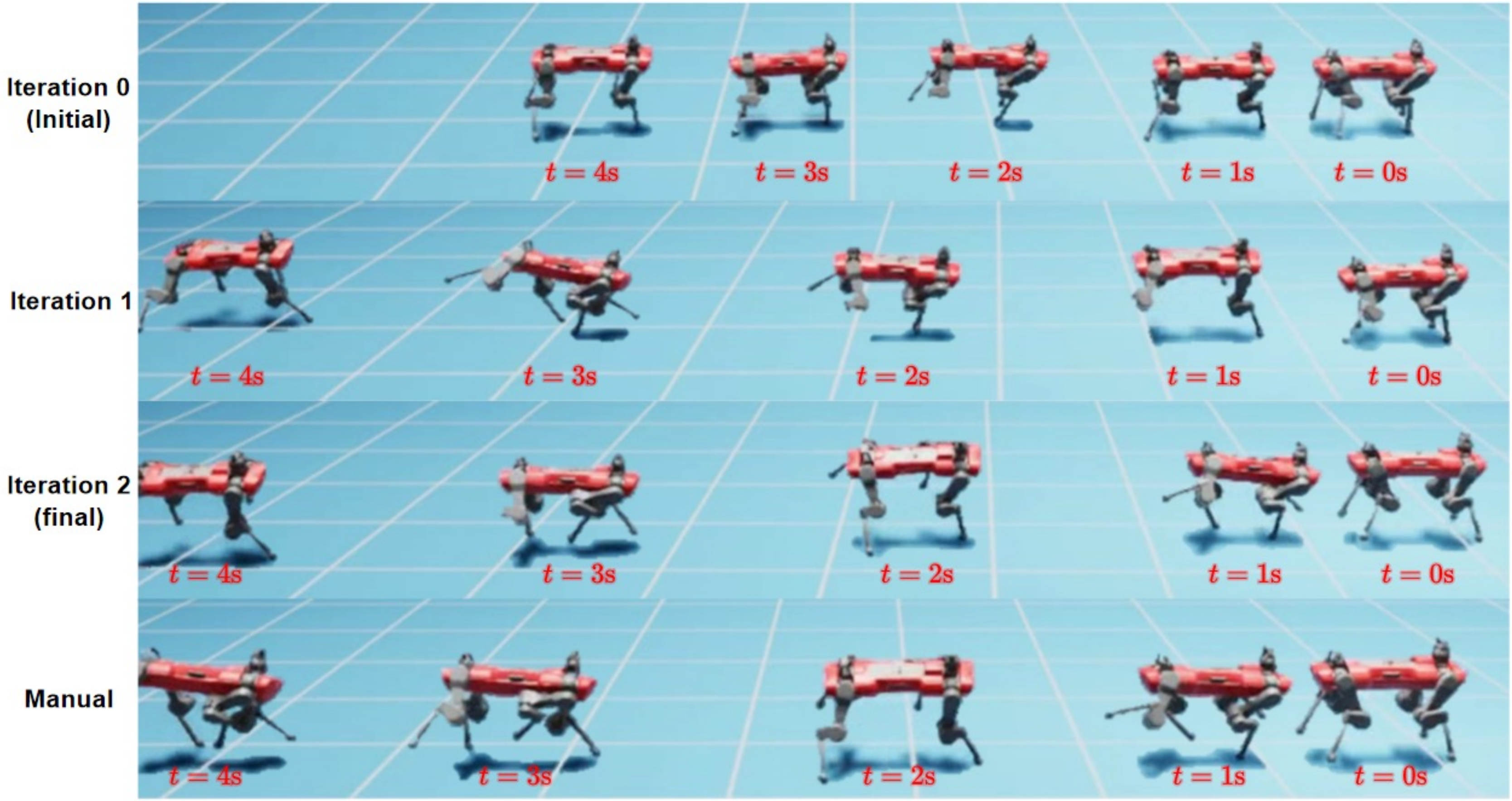


Table 1: Success rates of different reward functions and the number of self-refinement iterations (Iter.) used for R_{Refined}.

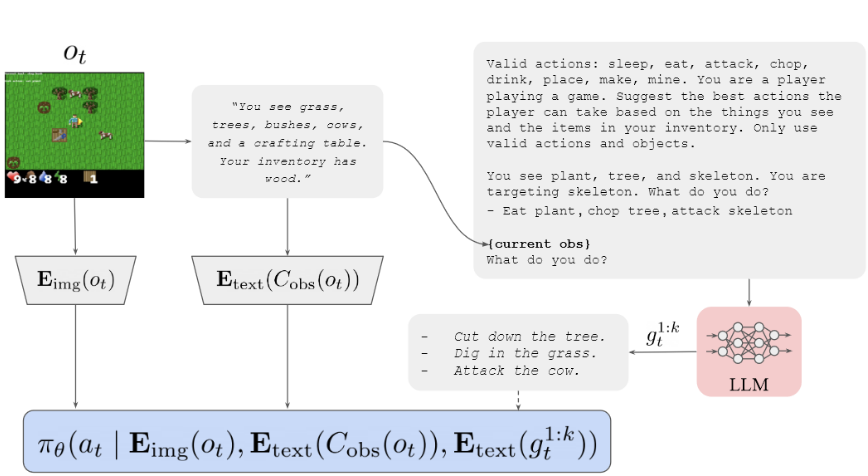
Robotic System	Task	Success Rate SR			Iter.
		R_{Initial}	R_{Refined}	R_{Manual}	
Manipulator	Ball Catching	100%	100%	100%	0
	Ball Balancing	100%	100%	98%	0
	Ball Pushing	0%	93%	95%	5
Quadruped	Velocity Tracking	0%	96%	92%	3
	Running	10%	98%	95%	2
	Walking to Target	0%	85%	80%	5
Quadcopter	Hovering	0%	98%	92%	2
	Wind Field	0%	100%	100%	4
	Velocity Tracking	0%	99%	91%	3

Comparing work

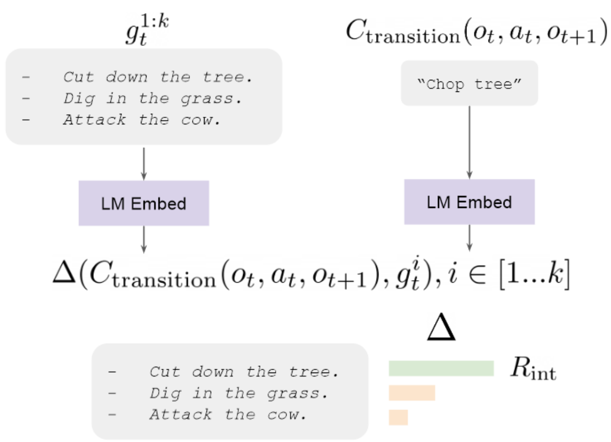
Guiding Pretraining in Reinforcement Learning with Large Language Models

Yuqing Du [*1] Olivia Watkins [*1] Zihan Wang [2] Cédric Colas [3 4] Trevor Darrell [1] Pieter Abbeel [1]
Abhishek Gupta [2] Jacob Andreas [3]

PMLR



(a) Policy parametrization for ELLM. We optionally condition on embeddings of the goals $E_{\text{text}}(g_t^{1:k})$ and state $E_{\text{text}}(C_{\text{obs}}(o_t))$.



(b) LLM reward scheme. We reward the agent for the similarity between the captioned transition and the goals.

Thanks !

浙江大學
ZHEJIANG UNIVERSITY