

## A APPENDIX

### A.1 REPRODUCIBILITY STATEMENT

Our primary experiments employ the following models: 1. For the Large Language Model planner, we utilize GPT-3.5-turbo-0613. 2. In the case of the Video Language Model, we employ GIT (Wang et al., 2022). However, since our main focus lies in long-horizon embodied planning rather than visual-to-language generation, readers can also explore the latest and more robust visual language model which may exhibit superior performance.

### A.2 CLIPORT DATASET

As for the benchmark for evaluating our proposed method with the baselines, we use the following datasets from Cliport (Shridhar et al., 2022).

**1. Assembling Kits** **Task:** Precisely place each specified shape in the specified hole following the order prescribed in the language instruction generated by the LLM-based agent at each timestep. This is one of the hardest tasks in the benchmark requiring precise placements of shapes of randomized colors and grounding spatial relationships. Each task instance contains 5 shapes and a kit with randomized poses. **Goal:** assembling all the kits to the corresponding holes. **Success Metric:** The pose of each shape matches the specified hole at the correct timestep. The final score is the total number of shapes that were placed in the correct pose at the correct timestep, divided by the total number of shapes in the scene (always 5).

**2. Align Rope** **Task:** Manipulate a deformable rope to connect its end-points between two corners of a 3-sided square. There are four possible combinations for aligning the rope: “front left tip to front right tip”, “front right tip to back right corner”, “front left tip to back left corner”, and “back right corner to back left corner”. Here ‘front’ and ‘back’ refer to canonical positions on the 3-sided square. The poses of both the rope and 3-sided square are randomized for each task instance. **Objects:** All align-rope instances contain a rope with 20 articulated beads and a 3-sided square. **Success Metric:** The poses of all beads match the line segments between the two correct sides.

**3. Stack Block Pyramid.** **Task:** Build a pyramid of colored blocks in a color sequence specified through the step-by-step language instructions generated by the LLM-based agent. Each task contains 6 blocks with randomized colors and 1 rectangular base, all initially placed at random poses. **Goal:** stack a pyramid by 6 blocks and 1 rectangular base. **Success Metric:** The pose of each block at the corresponding timestep matches the specified location. The final score is the total number of blocks in the correct pose at the correct timestep, divided by the total number of blocks (always 6).

**4. Towers of Hanoi.** **Task:** Move the ring to the specified peg in the LLM-based agent-generated language instruction at each timestep. The sequence of ring placements is always the same, i.e. the perfect solution to three-ring Towers of Hanoi. This task can be solved without using colors by just observing the ring sizes. However, it tests the agent’s ability to ignore irrelevant concepts to the task (color in this case). The task involves precise pick and place actions for moving the rings from peg to peg. **Goal:** Solve the tower of Hanoi via moving 3 rings (small, medium, and big) across 1 peg base. **Success Metric:** The pose of each ring at the corresponding timestep matches the specified peg location. The final score is the total number of correct ring placements, divided by the total steps in the perfect solution (7 for three-ring Towers of Hanoi).

**5. Put Blocks in Bowl.** **Task:** Place all blocks of a specified color in a bowl of the specified color. Each bowl fits just one block and all scenes contain enough bowls to achieve the goal. Each task instance contains several distractor blocks and bowls with randomized colors. The solutions to this task are multi-modal in that there could be several ways to place the blocks specified in the language goal. This task does not require precise placements and mostly tests an agent’s ability to ground color attributes. **Goal:** to place blocks with certain colors on the bowls with certain colors **Success Metric:** All blocks of the specified color are within the bounds of a bowl of the specified color. The final score is the total number of correct blocks in the correct bowls, divided by the total number of relevant color blocks in the scene.

**6. Packing Shapes.** **Task:** Place a specified shape in the brown box. Each task instance contains 1 shape to be picked along with 4 distractor shapes. The shape colors are randomized but have no relevance to the task. This task does not require precise placements and is mostly a test of the agent’s semantic understanding of arbitrary shapes. **Goal:** To pack the required shapes to the brown box **Success Metric:** The correct shape is inside the bounds of the brown box.

**Task:** Sweep the pile of blocks into the specified zone. Each scene contains two square zones: one relevant to the task, another as a distractor. The pile and zones are placed at random poses on the table.

**5. Separating Piles.** **Task:** Sweep the pile of blocks into the specified zone. Each scene contains two square zones: one relevant to the task, another as a distractor. The pile and zones are placed at random poses on the table. **Objects:** A pile of colored blocks and two squares. **Success Metric:** All blocks are inside the bounds of the specified zone. The final score is the total number of blocks inside the correct zone, divided by the total number of blocks in the scene.

### A.3 LOW-LEVEL POLICY NETWORK

**Code as Policies.** The primitives listed below are used in Cap (Liang et al., 2022). Note that we also used the same control and observation primitives without introducing new functions.

```

1 class LMP_wrapper():
2
3     def __init__(self, env, cfg, render=False):
4         self.env = env
5         self._cfg = cfg
6         self.object_names = list(self._cfg['env']['init_objs'])
7
8         self._min_xy = np.array(self._cfg['env']['coords']['bottom_left'])
9         self._max_xy = np.array(self._cfg['env']['coords']['top_right'])
10        self._range_xy = self._max_xy - self._min_xy
11
12        self._table_z = self._cfg['env']['coords']['table_z']
13        self.render = render
14
15    def is_obj_visible(self, obj_name):
16        return obj_name in self.object_names
17
18    def get_obj_names(self):
19        return self.object_names[:]
20
21    def denormalize_xy(self, pos_normalized):
22        return pos_normalized * self._range_xy + self._min_xy
23
24    def get_corner_positions(self):
25        unit_square = box(0, 0, 1, 1)
26        normalized_corners = np.array(list(unit_square.exterior.coords))[:4]
27        corners = np.array([self.denormalize_xy(corner) for corner in
28                             normalized_corners])
29        return corners
30
31    def get_side_positions(self):
32        side_xs = np.array([0, 0.5, 0.5, 1])
33        side_ys = np.array([0.5, 0, 1, 0.5])
34        normalized_side_positions = np.c_[side_xs, side_ys]
35        side_positions = np.array([self.denormalize_xy(corner) for corner
36                                   in normalized_side_positions])
37        return side_positions
38
39    def get_obj_pos(self, obj_name):
40        # return the xy position of the object in robot base frame
41        return self.env.get_obj_pos(obj_name)[:2]

```

```

41 def get_obj_position_np(self, obj_name):
42     return self.get_pos(obj_name)
43
44 def get_bbox(self, obj_name):
45     # return the axis-aligned object bounding box in robot base frame
46     # (not in pixels)
47     # the format is (min_x, min_y, max_x, max_y)
48     bbox = self.env.get_bounding_box(obj_name)
49     return bbox
50
51 def get_color(self, obj_name):
52     for color, rgb in COLORS.items():
53         if color in obj_name:
54             return rgb
55
56 def pick_place(self, pick_pos, place_pos):
57     pick_pos_xyz = np.r_[pick_pos, [self._table_z]]
58     place_pos_xyz = np.r_[place_pos, [self._table_z]]
59     pass
60
61 def put_first_on_second(self, arg1, arg2):
62     # put the object with obj_name on top of target
63     # target can either be another object name, or it can be an x-y
64     # position in robot base frame
65     pick_pos = self.get_obj_pos(arg1) if isinstance(arg1, str) else arg1
66     place_pos = self.get_obj_pos(arg2) if isinstance(arg2, str) else arg2
67     self.env.step(action={'pick': pick_pos, 'place': place_pos})
68
69 def get_robot_pos(self):
70     # return robot end-effector xy position in robot base frame
71     return self.env.get_ee_pos()
72
73 def goto_pos(self, position_xy):
74     # move the robot end-effector to the desired xy position while
75     # maintaining same z
76     ee_xyz = self.env.get_ee_pos()
77     position_xyz = np.concatenate([position_xy, ee_xyz[-1]])
78     while np.linalg.norm(position_xyz - ee_xyz) > 0.01:
79         self.env.movep(position_xyz)
80         self.env.step_sim_and_render()
81         ee_xyz = self.env.get_ee_pos()
82
83 def follow_traj(self, traj):
84     for pos in traj:
85         self.goto_pos(pos)
86
87 def get_corner_positions(self):
88     normalized_corners = np.array([
89         [0, 1],
90         [1, 1],
91         [0, 0],
92         [1, 0]
93     ])
94     return np.array([self.denormalize_xy(corner) for corner in
95         normalized_corners])
96
97 def get_side_positions(self):
98     normalized_sides = np.array([
99         [0.5, 1],
100         [1, 0.5],
101         [0.5, 0],
102         [0, 0.5]
103     ])
104     return np.array([self.denormalize_xy(side) for side in
105         normalized_sides])

```

```
101
102 def get_corner_name(self, pos):
103     corner_positions = self.get_corner_positions()
104     corner_idx = np.argmin(np.linalg.norm(corner_positions - pos,
105                                         axis=1))
106     return ['top left corner', 'top right corner', 'bottom left corner',
107            'bottom right corner'][corner_idx]
108
109 def get_side_name(self, pos):
110     side_positions = self.get_side_positions()
111     side_idx = np.argmin(np.linalg.norm(side_positions - pos, axis=1))
112     return ['top side', 'right side', 'bottom side', 'left
113            side'][side_idx]
```

Listing 1: The Cap primitives

#### A.4 LLM PROMPTS

The following text illustrates how we primarily prompt the large language model.

Assume you are a Robotics Agent that can use Tools to solve a task. you are allowed to use the following tools: {tool.description} You are allowed to call the following actions:

1. `video_captioning`, given a high-level instruction you should parse it to low-level instructions by using Python REPL to call this function, use `video_captioning()` as Action Input of Python REPL, do not add additional parameters.
2. `policy_network`, given a low-level instruction you should use the policy network to solve the instruction by using Python REPL to call this function, use `policy_network(num_sub_inst, lang_goal, positive_text, negative_text)` as Action Input of Python REPL. For example, if instructions contain 6 sub-instructions, and the sub-instruction is "put the blue block on the lightest brown block," then you should call `policy_network(6, 'put the blue block on the lightest brown block', 'the blue block is on the lightest brown block', 'the blue block is not on the lightest brown block')` as Action Input of Python REPL.

For each time step, you should use the following format to describe your solution. Do not generate all the steps at once; you should generate the steps one by one:

**Thought:** Comment on what you want to do next

**Action:** The action to take, exactly one element of {tool.names}

**Action Input:** The input to the action, either `video_captioning()` or `policy_network('sub_instruction')`. Do not include Action or Observation in the Action Input.

(Repeat the above Thought/Action/Action Input N times, use it until you are sure you finished the task)

In the first step, you have to use the video caption model to translate the video into text, by using Python REPL to call `video_captioning()` as Action Input of Python REPL. Then you have to use the policy network to solve the sub-instructions one by one, by using Python REPL to call `policy_network('sub_instruction')` as Action Input of Python REPL.

**Begin! Instruction:** {instruction}

**Thought:** {previous\_responses}

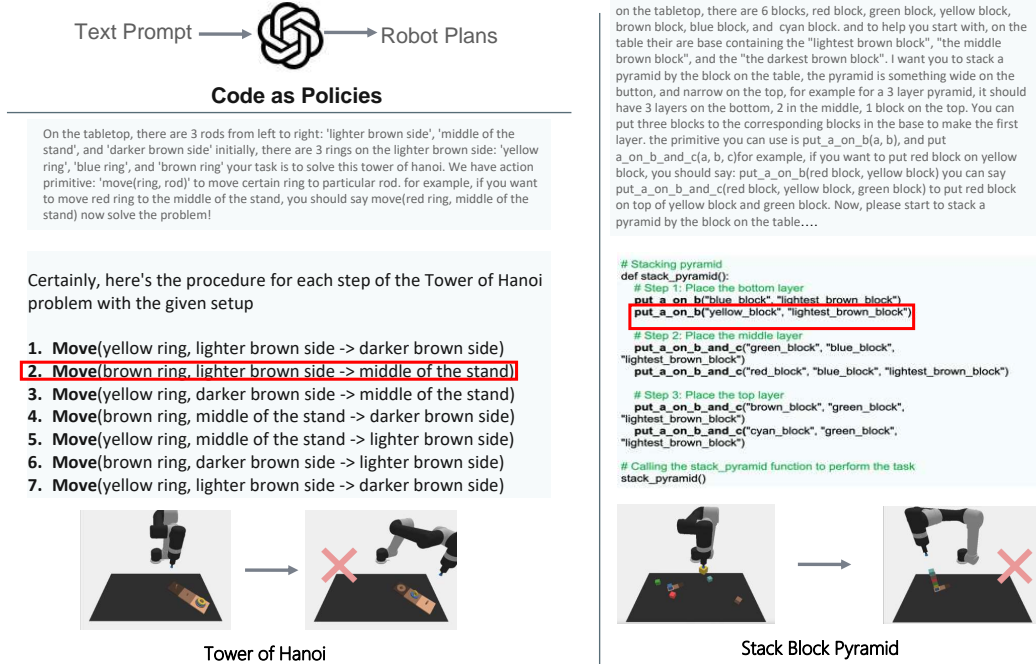


Figure 4: Cap (Liang et al., 2022) hallucination on the stack-block-pyramid task and tower of Hanoi task, the red boxes indicate the places where the agent starts to make faults.

#### A.5 ADDITIONAL RESULTS

##### LLM hallucinates on long-horizon planning.

Figure 4 illustrates how Cap fails on the long-horizon planning tasks: Tower-of-Hanoi and Stack-Block-Pyramid. The red boxes indicate the step LLM starts to hallucinate.

**Example of MultiReAct solve Stack-Block-Pyramid task.** The following text illustrates how MultiReAct generated the ReAct trace to solve the stack-block-pyramid puzzle.

```
1 Thought:
2 I will start by using the video caption model to translate the video
  into text.
3 Action: Python REPL
4 Action Input: video_captioning()
5 Feedback: 1. put the cyan block on the lightest brown block 2. put the
  yellow block on the middle brown block 3. put the blue block on the
  darkest brown block 4. put the green block on the cyan and yellow
  blocks 5. put the brown block on the yellow and blue blocks 6. put
  the red block on the green and brown blocks.
6
7 Thought:
8 Now that I have the low-level instructions, I will use the policy
  network to solve each sub-instruction one by one.
9 Action: Python REPL
10 Action Input: policy_network(6, "put the cyan block on the lightest
  brown block", "the cyan block is on the lightest brown block", "the
  cyan block is not on the lightest brown block")
11 Reward: the reward is 0.16666666666666666
12
13 Thought:
14 The reward has increased compared to the last step, so I am on the right
  track. Now I will move on to the next sub-instruction.
15 Action: Python REPL
```

```

16 Action Input: policy_network(6, "put the yellow block on the middle
    brown block", "the yellow block is on the middle brown block", "the
    yellow block is not on the middle brown block")
17 Reward: the reward is 0.3333333333333333
18
19 Thought:
20 The reward has increased compared to the last step, so I am on the right
    track. Now I will move on to the next sub-instruction.
21 Action: Python REPL
22 Action Input: policy_network(6, "put the yellow block on the middle
    brown block", "the yellow block is on the middle brown block", "the
    yellow block is not on the middle brown block")
23 Reward: the reward is 0.3333333333333333
24
25 Thought:
26 The reward remains unchanged, which means my solution is not correct. I
    need to retry the last step.
27 Action: Python REPL
28 Action Input: policy_network(6, "put the blue block on the darkest brown
    block", "the blue block is on the darkest brown block", "the blue
    block is not on the darkest brown block")
29 Reward: the reward is 0.5
30
31 Thought:
32 The reward has increased again, so I will continue to the next
    sub-instruction.
33 Action: Python REPL
34 Action Input: policy_network(6, "put the green block on the cyan and
    yellow blocks", "the green block is on the cyan and yellow blocks",
    "the green block is not on the cyan and yellow blocks")
35 Reward: the reward is 0.6666666666666666
36
37 Thought:
38 The reward has increased again, so I will continue to the next
    sub-instruction.
39 Action: Python REPL
40 Action Input: policy_network(6, "put the brown block on the yellow and
    blue blocks", "the brown block is on the yellow and blue blocks",
    "the brown block is not on the yellow and blue blocks")
41 Reward: the reward is 0.8333333333333333
42
43 Thought:
44 The reward has increased again, so I will continue to the final
    sub-instruction.
45 Action: Python REPL
46 Action Input: policy_network(6, "put the red block on the green and
    brown blocks", "the red block is on the green and brown blocks",
    "the red block is not on the green and brown blocks")
47 Reward: the reward is 0.9999999999999999

```

Listing 2: The procedure of how MultiReAct solve stack block pyramid problem