

APPENDIX A RELATED WORK

Large language models and vision-language models.

Recent developments in various domains and applications have been greatly influenced by the substantial progress achieved through large language models (LLMs) and vision language models (VLMs) [8, 39, 40, 4, 7, 1, 41, 22, 42, 23]. While these models can already solve various tasks in a zero-shot manner [1], well-designed prompts still serve an important role in further eliciting more advanced capabilities. As demonstrated by Brown et al. [4], few-shot prompting can match or surpass the performance of fine-tuning on LLMs. Additionally, other prompting techniques [50, 57, 19, 55] have been proposed to improve LLMs’ reasoning capabilities. Although these methodologies may initially appear enigmatic, they have been empirically validated to consistently demonstrate scalability across various models [4, 7, 1, 54]. Apart from the language prompts, recent vision models [18, 58, 21] are capable of supporting visual prompts, including points, boxes, masks, and texts. Such visual prompts exhibit greater diversity in both form and content, harnessing vision-language models for various application scenarios like perception [53], image editing [44] and reasoning [51].

To build an autonomous agent capable of making decisions in an unstructured environment, the incorporation of robust visual reasoning capabilities becomes imperative. Although the current generation of VLMs cannot seamlessly engage in zero-shot reasoning, they can still be effectively harnessed across a diverse spectrum of tasks. Prior work SoM [51] draws visual marks on the objects in an image using numbers and segmentation masks, and demonstrates the mark-based visual prompting scheme unleashes the reasoning capabilities of GPT-4V, such as object counting and inferring spatial relationship. Different from SoM [51], we want to leverage GPT-4V for open-vocabulary robot manipulation. We represent each manipulation phase with a set of affordance points and motion waypoints. Instead of using object segmentation masks, we use keypoints and grid cells as visual prompts as shown in Fig. 1, and then query GPT-4V to choose the affordance points and motion waypoints from the visual marks, followed by executable point-based motion plans.

Foundation models in robotics. Building upon the successes of large language models, the field of robotics is currently witnessing a rising interest in utilizing LLMs in various application scenarios. LLMs is capable of generating high-level task plan in natural language [2, 16, 15, 5], executable programs [25, 45, 49] or value function [17, 26] for low-level behaviours, environmental reward and feedback for reinforcement learning [28, 20, 30, 56]. However, these approaches necessitate the conversion of both the task and the observations into the textual form. While this can be easily accomplished in simulated environments with ground-truth object state, tasks in real world require the utilization of robust and precise perception modules. To perform open-ended navigation and manipulation in real world [46, 5, 10],

open-vocabulary vision foundation models [18, 58, 21, 27, 34, 24] are often used to extract visual scene information before converting the observation to textual form. However, the process of converting an image into text may result in the loss of important details, such as shape and geometric information. With the recent advancements in vision language models (VLMs), it is now conceivable that the role of state estimation combined with language models can be replaced by a single, powerful VLM, such as GPT-4V, which is leveraged by MOKA. Hu et al. [14] utilizes GPT-4V to get semantic language plans and convert each language plan to a set of pre-defined low-level skills. Different from Hu et al. [14], we represent manipulation affordance and motion using a set of points, and query GPT-4V to select the corresponding points (e.g. grasping point, function point) in each task, which can be directly translated to low-level point-based motions. While preserving the visual reasoning capabilities of VLMs, MOKA provides a framework with more general and flexible low-level motion, which doesn’t require prior knowledge about a specific task compared to pre-defined skills.

Affordance reasoning for robotic control. The psychologist James J. Gibson, along with Eleanor J. Gibson, introduced the concept of an affordance [11], which refers to the ability to perform a certain action with an object in a given environment. Much of the research related to affordances has concentrated on predicting how to interact with objects, as demonstrated by [47, 12, 3]. In the field of robot manipulation, keypoints are often used to provide compact information about the environment and the objects [6, 9, 48, 31, 33, 32, 38], representing the affordance in a structured way. Among these works, the most related one to ours is KETO [38], which predicts affordance and functional keypoints on the tool objects by learning from interactions with a trajectory optimization formulation as in Manuelli et al. [32]. In contrast to KETO [38], our keypoints selection procedure doesn’t require any model training. We design an automated process to annotate keypoints as visual marks on a 2D image, and leverage the broad knowledge from GPT-4V to select affordance and motion keypoints for manipulation.

APPENDIX B EXPERIMENT DETAILS

A. Environment

Our experiments are conducted in a real-world table-top manipulation environment, which involves a 7-DoF Franka Emika robot arm with a 2F-85 Robotiq gripper interacting with various objects on the table at 5Hz. Our tabletop environment has two fixed ZED 2.0 cameras and one ZED mini wrist camera that can take RGBD images.

The *top-down camera*, which is the primary camera used in this paper, provides RGB and the depth images for MOKA and other baseline methods.

In addition, we also set up the front camera and the wrist camera for more complete observation to distill the collected robot experiences to the learned Octo policy [35]. The action space of the robot is 7-dimensional, consisting of the 6DoF

end-effector twist defined in the Cartesian space with an additional dimension of gripper position. Each task can consist of multiple stages, each terminates within a fix horizon of 100 steps.

B. Task Design

We design various table-top manipulation tasks with a diverse set of daily objects. Tab. II shows the language description of the full list of tasks.

| | |
|--------------------|--------------------------------------------------------------------------------|
| Table Wiping | 1. Move the glasses into the glasses case 2. Sweep the trash with the broom |
| Watch Cleaning | 1. Put the watch into the ultrasound cleaner 2. Press the power button |
| Gift Preparation | 1. Put the golden filler in the gift box 2. Put the perfume in the gift box |
| Laptop Packing | 1. Unplug the cable 2. Close the lid of the laptop |
| Fur Removing | 1. Grasp the fur remover 2. Sweep the fur on the sweater with the remover |
| Drawer Closing | 1. Close the drawer |
| Scissor Handling | 1. Grasp the scissor 2. Hand the scissor to a human |
| Flower Arrangement | 1. Grasp the pink roses on the table 2. Insert the roses into the vase |

TABLE II: The language description of all the proposed tasks. Each of the tasks consists of two stages except for the drawer closing task.

We provide the comparative evaluation on the top four tasks (Table Wiping, Laptop Packing, Gift Preparation and Untrasound Cleaning) in the main paper, with additional results of MOKA on other 4 tasks in our supplementary video.

C. Success Checking

To count the failure modes and collect successful trajectories, we first query VLM with task instruction and obtain the point-based motion prediction. If the prediction is correct, it will be executed on the robot. Otherwise, it will be counted as the VLM reasoning failure as mentioned in Sec. III-B. After executing the motion prediction from VLM, the human expert will manually check if the task succeeds or not. If the task is not successfully finished, it will be counted as an execution failure. All the successful trajectories will be saved as demonstrations for policy distillation.

APPENDIX C IMPLEMENTATION DETAILS

We introduce the overview of MOKA in Alg. 1, and the implementation details of each component in the following sections.

A. High-level Task Reasoning

Given the initial observation image s_0 and the language task description l , we first query the VLM \mathcal{M} with the language instruction to produce the decomposed subtask, which contains the structured information for the K subtasks, including

Algorithm 1 MOKA Pipeline

- 1: **Input:** Vision-language Model \mathcal{M} , Task instruction l , text prompt for high-level reasoning p_{high} , text prompt for low-level reasoning p_{low} , initial observation s_0
- 2: Query \mathcal{M} for high-level task reasoning, obtain $y_{high} = \mathcal{M}([p_{high}, l, s_0])$ which decompose the task into N subtasks.
- 3: **for** subtask $k = 0 \cdots N - 1$ **do**
- 4: Get observation s_k from the top-down camera
- 5: Propose keypoint and waypoint candidates and get annotated image $f(s_k)$
- 6: Query \mathcal{M} for low-level motion reasoning, obtain $y_{low}^k = \mathcal{M}([p_{low}, y_{high}^k, f(s_k)])$
- 7: Execute y_{low}^k on the real robot
- 8: **end for**

descriptions of objects and desired motion. We provide the high-level reasoning we used across all the tasks in Tab. III.

The response contains fields “object_grasped”, “object_unattached” and “motion_direction”, which will be used for later stages in our pipeline, such as keypoint proposal and motion prediction.

B. Point-based Affordance Proposal

After obtaining the high-level reasoning results, we can know the objects involved in each subtask. Since VLM cannot directly generate keypoints and waypoints, we need to propose some candidate points and let VLM select corresponding points through a visual question-answering way.

Keypoint proposal. We leverage GroundedSAM [43] to extract segmentation masks conditioned on a text prompt, which is designed to be a string of object names involved in the current subtask (e.g. “trash, broom”). Given an image of current observation and such a text prompt about the involved object names, we first employ GroundingDINO [27] to generate bounding boxes for the objects by conditioning on the text prompt. Later, the annotated boxes given by GroundingDINO [27] serve as the box prompts for SAM [18] to generate corresponding segmentation masks for the objects. We define the keypoints of each object to be a set of boundary keypoints with a center keypoint. To extract them, we sample K points on the contour of each object using farthest point sampling [37], and also include the geometric mean point of the object segmentation mask. We then plot these $K + 1$ keypoints on the 2D image as our keypoint proposals to the VLM.

Waypoint proposal. Unlike keypoints, waypoints are often the points in the free space that are not attached to any objects. To perform waypoint selection, we evenly divide the full image into 5×5 grid tiles, which mark the column as a, b, c, d, e from left to right and rows as $1, 2, 3, 4, 5$ from bottom to top, as shown in Fig. 3. The waypoints will be sampled from the predicted tile from VLM.

The input request contains:

- A string describes the multi-stage task.
- An image of the current table-top environment captured from a top-down camera.
- List of object candidates in the scene.

The output response is a list of dictionaries in the JSON form. Each dictionary specifies the information of a subtask, following the correct order of executing the subtasks to solve the input task. Each dictionary contains these fields:

- **instruction**: A string to describe the subtask in natural language forms.
- **object_grasped**: A string to describe the name of the object that the robot gripper will hold in hand while executing the task (e.g., the object to be picked or used as a tool to interact with other objects). This field can be empty if there is no such object involved in this subtask.
- **object_unattached**: A string to describe the name of the object that the robot gripper will interact with directly or via another object without holding it in hand (e.g., the object to be touched by the tool, the target object where “object_grasped” will be moved onto). This field can be empty if there is no such object involved in this subtask.
- **motion_direction**: A string to describe the direction of the robot gripper motion while performing the task (e.g., “from right to left”, “backward”, “downward”).

TABLE III: The high-level reasoning prompt for all the tasks. It will decompose a multi-stage task into subtasks. The specified output fields can be used for later low-level reasoning stage.

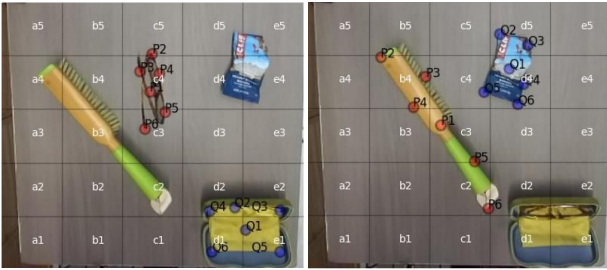


Fig. 3: The affordance proposal for the table wiping task in stage 1 and stage 2. The keypoints are plotted on corresponding objects for different stages based on the high-level reasoning results. The annotated grid cells are used for VLM to select the position of potential waypoints.

C. Low-level Motion Reasoning

After annotating the observation image using object keypoints and grid cells as described in the above section, we can query VLM about low-level motion with annotated image and text prompts. Firstly, we describe the text and image inputs to the VLM using prompt IV.

Then we first explain the definition of keypoints and waypoints using prompt V.

After that, we can specify the output format and further explain the role of each field using prompt VI. We can also provide some step-by-step guidance about how the reasoning procedure should be done (similar to chain-of-thought prompting [50]). Our complete low-level reasoning prompts are covered in Tab. IV, V and VI.

D. Motion Execution

We can decompose a manipulation subtask into an **optional** grasping phase and a manipulation phase. For some tasks like tool using, the robot needs to first grasp an object before making contact with another object. For other tasks where the robot can directly interact with target objects (e.g. pushing, pressing button), the “grasp_keypoint” field will be empty. For such tasks, we will skip the execution of grasping phase.

Grasping phase. We first sample 30 antipodal 4DoF grasp proposals based on the primary camera’s depth image, which is cropped by the bounding box of **object_grasped**. Here, we use the same antipodal grasp proposal method used in DexNet 2.0 [29]. We then lift the VLM-predicted grasp point from 2D image to the robot frame based on the primary camera’s intrinsic and extrinsic parameters. After that, we can select the nearest grasp proposal based on the lifted grasp point, and execute the 4DoF grasp on the robot.

Manipulation phase. After obtaining the pre-contact tile and post-contact tile, we first sample pre-contact waypoint and post-contact waypoint from the predicted tiles. Since the waypoints are in the free space, its 3D position cannot be determined given its 2D projection on the image. So we also assign the VLM-predicted height to them. We only consider the cases where the waypoints are in same height with target point (e.g. pushing), or above the target point (e.g. placing) in most common table top manipulation scenarios.

We then sequentially move the function keypoint to the pre-contact waypoint, target keypoint and post-contact waypoint to perform a contact motion.

After both phases are successfully done, the robot need to get the motion prediction for the next subtask. In order to obtain a clean and non-occluded image to perform low-level reasoning, we first move the robot to the neural pose, get an observation image from the top-down camera, and then resume the robot. Subsequently, the robot will execute the predicted motion until the multi-stage task is finished.

E. Evaluation

We evaluate MOKA in both zero-shot and in-context learning settings. For the zero-shot setting, we keep all the above prompts **unchanged** but only change the language task description (e.g. “sweep the garbage”, “pick up the perfume”, “press the button”).

For in-context learning settings, we first collect two examples from VLM’s successful predictions in different scenes with scene and object variations. As shown in Fig. 4, MOKA can be improved from such simple and intuitive in-context examples, without intricate prompt engineering.

Please describe the robot gripper’s motion to solve the task by selecting keypoints and waypoints.
The input request contains:

- The task information with these fields:
 - **instruction**: The task in natural language forms.
 - **object_grasped**: The object that the robot gripper will hold in hand while executing the task.
 - **object_unattached**: The object that the robot gripper will interact with either directly or via another object without holding it in hand.
 - **motion_direction**: The motion direction of the robot gripper or the in-hand object while performing the task.
- An image of the current table-top environment captured from a top-down camera, annotated with a set of visual marks:
 - **candidate keypoints on object_grasped**: Red dots marked as $P[i]$ on the image.
 - **candidate keypoints on object_unattached**: Blue dots marked as $Q[i]$ on the image.
 - **grid for waypoints**: Grid lines that uniformly divide the images into tiles. The grid equally divides the image into columns marked as a, b, c, d, e from left to right and rows marked as 1, 2, 3, 4, 5 from bottom to top.

TABLE IV: The description about image and text inputs in the low-level motion reasoning stage.

The motion consists of an optional grasping phase and a manipulation phase, specified by **grasp_keypoint**, **function_keypoint**, **target_keypoint**, **pre_contact_waypoint**, and **post_contact_waypoint**.

The definitions of these points are:

- **grasp_keypoint**: The point on “object_grasped” indicates the part where the robot gripper should hold.
- **function_keypoint**: The point on “object_grasped” indicates the part that will make contact with “object_unattached”.
- **target_keypoint**: If the task is pick-and-place, this is the location where “object_grasped” will be moved to. Otherwise, this is the point on “object_unattached” indicating the part that will be contacted by “function_keypoint”.
- **pre_contact_waypoint**: The waypoint in the free space that the functional point moves to before making contact with the “target_keypoint”.
- **post_contact_waypoint**: The waypoint in the free space that the functional point moves to after making contact with the “target_keypoint”.

TABLE V: The explanation about the definition of keypoints and waypoints.

The response should be a dictionary in JSON form, which contains:

- **grasp_keypoint**: Selected from candidate keypoints marked as $P[i]$ on the image. This will be empty if and only if object_grasped is empty.
- **function_keypoint**: Selected from candidate keypoints marked as $P[i]$ on the image. This will be empty if and only if object_grasped or object_unattached is empty.
- **target_keypoint**: Selected from keypoint candidates marked as $Q[i]$ on the image. This will be empty if and only if object_unattached is empty.
- **pre_contact_tile**: The tile that the pre-contact waypoint should be in. This is selected from candidate tiles marked on the image.
- **post_contact_tile**: The tile that post-contact waypoint should be in. This is selected from candidate tiles marked on the image.
- **pre_contact_height**: The height of pre-contact waypoint as one of the two options “same” or “above” (same or higher than the height of making contact with target keypoint).
- **post_contact_height**: The height of post-contact waypoints as one of the two options “same” or “above”.
- **target_angle**: Describe how the object should be oriented during this motion in terms of the axis pointing from the grasping point to the function point.

Think about this problem step by step and explain the reasoning steps. First, choose grasp_keypoint, function_keypoint, and target_keypoint on the correct parts of the objects. Next, describe which tile the target_keypoint is located in. Then choose pre_contact_tile, post_contact_tile, pre_contact_height, post_contact_height such that the resultant motion from pre-contact waypoint to target keypoint, then to post-contact waypoint in 3D follows the “motion_direction” input. Remember that the columns are marked as ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ from left to right, and the rows are marked as 1, 2, 3, 4, 5 from bottom to top.

TABLE VI: The output format of the low-level motion reasoning, including some further explanations.

APPENDIX D ADDITIONAL RESULTS

A. Qualitative Analysis

Failure breakdown We analyze the failure cases of MOKA. Trajectories with wrong predictions from the GPT-4V are counted as reasoning failures. The following failure cases are illustrated in the top row of Fig. 5, including grasping the broom upside down due to confusing the the grasp point with the function point, pointing the room to the wrong direction due to wrong target angle, pressing the hinge of the laptop due to the wrong target point. The trajectories with desired VLM prediction but fail to execute successfully are counted as execution failures. The following failure cases are illustrated in the bottom row of Fig. 5. From left to right, the failures

include the gripper narrowly misses the button, the filler getting disassembled in the middle of the trajectory, and the cable slipped through the gripper fingers.

Robustness We analyze the robustness of MOKA with respect to instruction variations, initialization variations, and object variations. Fig. 6 provides a pictorial illustration of MOKA’s predictions under different instructions and observations. Each image is queried with the language prompt on the top within the same column. The examples in the same column share similar initialization object positions and orientations. The example in the first row uses the same set of objects while the second row involves objects of alternative geometries, colors, and materials. Some of these objects are also deformable or transparent. Fig. 6 demonstrates consistent

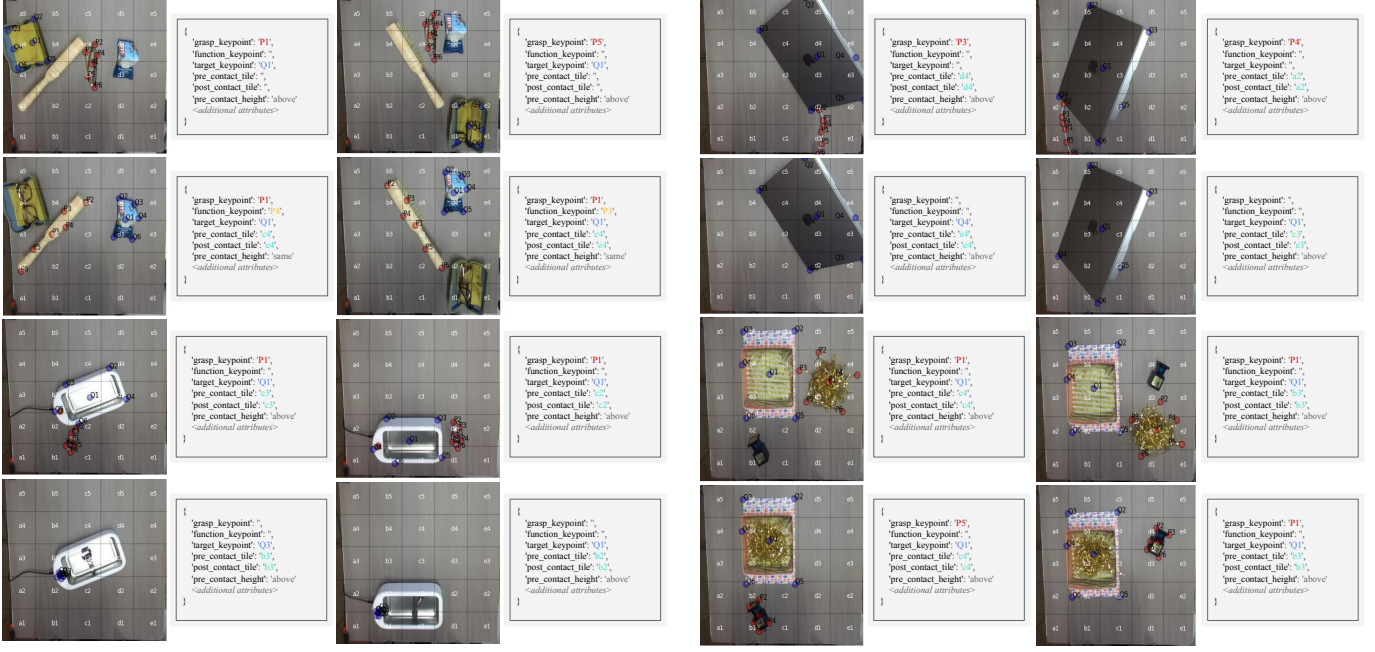


Fig. 4: The in-context examples used in MOKA which are collected under object pose variations.



Fig. 5: **Failure analysis.** We breakdown the failure cases of the variants of our approach using zero-shot prediction, distilled policies, and in-context learning. We breakdown the failures into reasoning failures (caused by errors in the affordance prediction) and execution failures (caused by low-level motions).

point predictions within rows and columns, indicating that MOKA is robust to the changes of the language instructions, initialization, and objects for the same task.

B. Ablation Study

We design the following ablative studies to understand the effectiveness of different design options in MOKA.

- MOKA w/o hierarchy: we can skip the high-level task reasoning but directly ask for low-level motion reasoning from GPT-4V. Here all the objects in the scene will be

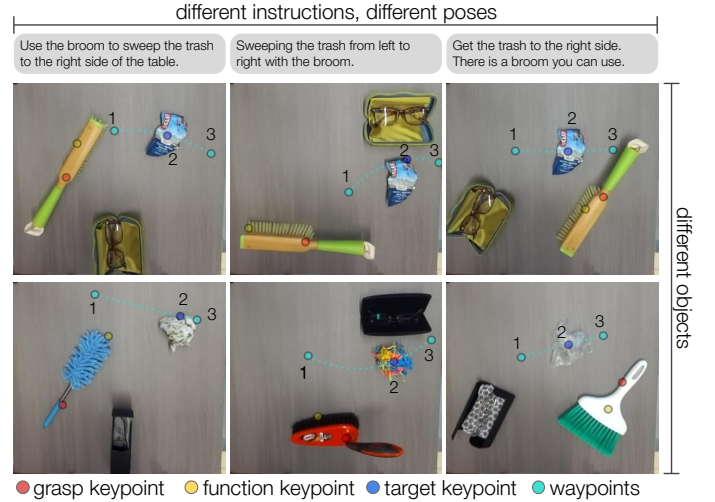


Fig. 6: **Robustness analysis.** We analyze MOKA’s robustness with respect to various instructions, initial arrangements, and objects. Each column in the image uses the same language instruction and similar initial arrangements of objects. The two rows involve different objects.

annotated with keypoints, and VLM is queried to generate point-based affordance directly from the annotated image.

- MOKA w/o description of points: we can remove the description of the definition of keypoints and waypoints in Tab. V.
- MOKA w/o chain-of-thought prompting: we can remove the step-by-step guidance at the last paragraph in the prompt in Tab. VI.

The results of our ablative studies are illustrated in Tab. VII.

For each task, we report the number of reasoning successes rate out of 10 trials. The results demonstrate that our method can obtain consistent improvements from all the above prompting designs. Specifically, **MOKA w/o hierarchy** decreases the performance by a large margin, which indicates the importance of subtask decomposition. **MOKA w/o keypoint description** and **MOKA w/o CoT** can preserve most of the performance, but still worse than MOKA on most of the tasks. As illustrated in Fig. 7, VLM can make various mistakes in terms of keypoint and waypoint predictions. Specifically, for MOKA w/o hierarchy, the VLM can make mistakes about subtask ordering, which can cause a complete failure of the task.

C. Additional Tasks

For our four additional tasks, please refer to our website.

| | Table Wiping | | Watch Cleaning | | Gift Preparation | | Laptop Packing | |
|-------------------------------|--------------|------------|----------------|------------|------------------|------------|----------------|------------|
| | Subtask I | Subtask II | Subtask I | Subtask II | Subtask I | Subtask II | Subtask I | Subtask II |
| MOKA | 0.7 | 0.7 | 0.8 | 1.0 | 1.0 | 0.8 | 0.6 | 0.8 |
| MOKA w/o hierarchy | 0.1 | 0.1 | 0.2 | 0.1 | 0.2 | 0.2 | 0.0 | 0.1 |
| MOKA w/o keypoint description | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 0.6 | 0.4 | 0.8 |
| MOKA w/o CoT | 0.5 | 0.4 | 0.6 | 0.8 | 0.9 | 0.6 | 0.4 | 0.6 |

TABLE VII: Ablation studies on different prompt designs. Across 4 tasks, each consists of 2 subtasks, MOKA consistently benefits from the three prompt designs.

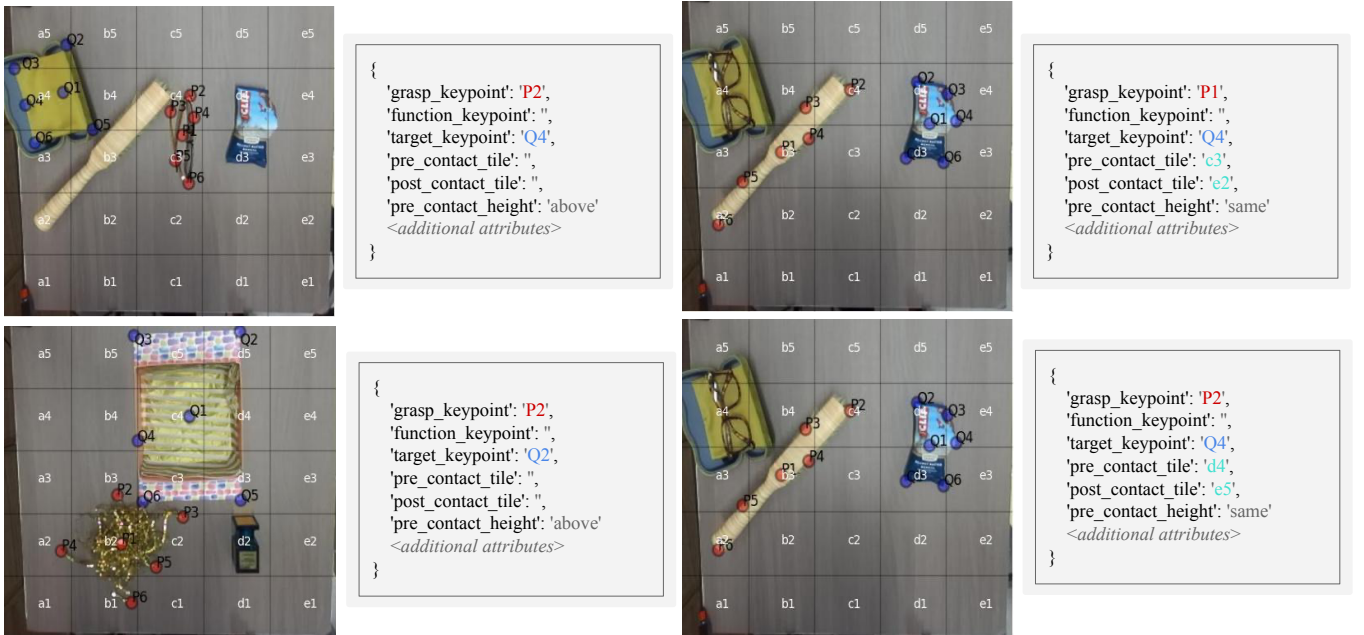


Fig. 7: Qualitative results of ablation studies. Without keypoint description or chain-of-thought prompting, the model can make mistakes in keypoint prediction (left column) and waypoint prediction (right column).