

# Appendix

## Table of Contents

<b>7 Approach Details</b>	<b>12</b>
7.1 Active Preference Learning	12
7.2 Task Planner	12
7.3 Real Robot System	13
<b>8 Experiments Details</b>	<b>14</b>
8.1 Active Preference Learning Experiments Details	14
8.2 Task Planner Real Robot Experiments	14
<b>9 Prompts</b>	<b>16</b>
9.1 Active Preference Learning Prompts	16
9.2 Task Planner Prompts	37
9.3 Baseline Prompts	41

## 7 Approach Details

### 7.1 Active Preference Learning

**Detailed Algorithm.** Algorithm 1 shows the detailed workflow of APRICOT’s active preference learning module. We note that, to expedite computation, we generate the candidate questions before entering the main active preference learning loop. For each candidate preference pair  $(\theta_i, \theta_j)$ , the question-generating LLM also receives the initial state  $s_0$ , the task  $\mathcal{T}$ , the candidate plans corresponding to the pair  $\xi_i, \xi_j$ , and each plan’s rewards given one of the candidate preferences  $(\mathcal{R}(\xi_i, \theta_i), \mathcal{R}(\xi_i, \theta_j))$  and  $(\mathcal{R}(\xi_j, \theta_i), \mathcal{R}(\xi_j, \theta_j))$ . During the main loop, once that question is asked, it is removed from the list of candidate questions.

**Hyperparameters.** We set the number of candidate preferences to  $N = 5$  and the number of questions to generate for each preference pair to  $M = 2$ , so we generate 20 questions in total. For the terminating function, which determines when the preference prior  $P(\theta)$  is sufficient, we set the terminating threshold to 0.07.

The components that require an LLM prompt are: generating candidate preferences, calculating the reward of a plan given a preference, generating candidate questions, estimating the likelihood of an answer given a preference and a question. All LLM calls use temperature of 0.7. We use gpt-4 for “generating candidate preferences” because this task requires reasoning about the similarities and differences across multiple demonstrations and proposing a diverse set of preferences that are consistent with the demonstrations. We use gpt-3-turbo for “estimating the likelihood of answer” because this task calls the LLM the most number of times ( $N \times M \times \binom{N}{2}$  times), and this task has a simple input-output space (inputs include a preference to roleplay and a yes-or-no question to answer; outputs include reasoning and yes/no answer). We use gpt-4o for the rest of the tasks. Prompts can be found in Appendix 9.1.

### 7.2 Task Planner

**Beam Search.** Recall that APRICOT’s task planner first uses an LLM to output a semantic plan given a task  $\mathcal{T}$  (a list of objects to put away) and an initial state  $s_0$  (a dictionary of objects initially at each semantic location in the fridge). Then, it uses a beam search to find the XYZ placement location for each object by sequentially following the semantic plan.

---

**Algorithm 1** APRICOT LLM-based Bayesian Active Preference Learning

---

**Input:** Demonstrations  $\mathcal{D}$ , Initial State  $s_0$ , Task  $\mathcal{T}$ , Terminating Function  $f(P(\theta))$   
**LLM Prompts:** LLM to generate candidate preferences  $P^{\text{gen-pref}}$ , LLM to generate plans  $P^{\text{plan}}$ , LLM to generate questions  $P^{\text{gen-q}}$   
**Output:** Best plan  $\xi_i$ , Corresponding preference of the best preference  $\theta_i$   
*// Generate candidates*  
Generate candidate preferences  $\{\theta_i\}_{i=1}^N: \theta \sim P^{\text{gen-pref}}(\theta|\mathcal{D})$   
Initialize Prior  $P(\theta_i) = \frac{1}{N}$   
Generate candidate plans  $\{\xi_i\}_{i=1}^N: \xi \sim P^{\text{plan}}(\xi|s_0, \mathcal{T}, \theta_i), i = 1, \dots, N$   
Construct  $\mathcal{Q}$  by generating  $M$  questions for all preference pairs:  $q \sim P^{\text{gen-q}}(q|\theta_i, \theta_j)$   
*// While not exist a plan  $\xi$  that has sufficiently high rewards given all preferences in  $\{\theta_i\}_{i=1}^N$*   
**while** not  $f(P(\theta))$  **do**  
    Find the best question:  $q^* \leftarrow \arg \max_{q \in \mathcal{Q}} H[P(\theta)] - \sum_o^{\{\text{yes}, \text{no}\}} \sum_i^N P(o|q, \theta_i) H[P(\theta_i|q, o)]$   
    Query the user with  $q^*$  and gets answer  $o_h$   
    Remove  $q^*$  from  $\mathcal{Q}$   
    Update prior:  $P(\theta) = P(\theta|q = q^*, o = o_h)$   
**Return**  $\xi_i$  that satisfies the Terminating Function  $f$ , the corresponding  $\theta_i$

---

448 The world model contains a mapping from a semantic location (e.g. “left side of top shelf”) to a  
449 range of XYZ coordinates that corresponds to that region. With this semantic-location-to-geometric-  
450 range mapping, the beam search samples candidate XYZ placement location for each object based  
451 on its planned semantic place location. Specifically, it samples in XY space since objects can be  
452 placed anywhere length-wise and depth-wise on a shelf. Z location is based on the height of the  
453 shelf because we do not consider stacking objects atop one another. The best candidates are those  
454 where the placement is geometrically feasible (no fridge or object collisions). These candidates are  
455 maintained in the beam search nodes.

456 The beam search continues until all objects are placed into the fridge. An optimal plan is one where  
457 all objects acquire a geometrically-feasible, collision-free XYZ placement location. If the best plan  
458 in the search only places a subset of objects, the semantic plan is assumed to be geometrically  
459 infeasible and the LLM is reprompted with information about which objects lack a feasible XYZ  
460 placement location.

461 **Hyperparameters.** For semantic plan generation, we use gpt-4-turbo with a temperature of  
462 0.7. For the beam search, we maintain 10 best candidates and sample 10 evenly-spaced points with  
463 the location regions. For feedback retries, we allow up to 4 attempts. Prompts can be found in  
464 Appendix 9.2.

### 465 7.3 Real Robot System

466 **RL Policy Training.** The policy `pick(<obj>)` and `place(<loc>)` can both be abstracted as the  
467 problem of given a goal XYZ position and the current robot joint state, output a sequence of actions  
468 that allows the gripper to reach the goal straight on. We simulate the robot’s kinematics and include  
469 the  $L_1$  norm between the goal and current positions as the observation space. We define the robot’s  
470 teleoperation commands as the discrete action space. To guide the agent to learn a suitable grasp  
471 policy, we implemented a four-phase reward function:

- 472 1. Align the Gripper horizontally: The goal object should align in between the robot’s gripper  
473 (X coordinate).
- 474 2. Align the Gripper height: The gripper should align with the goal height (Z coordinate).
- 475 3. Extend the gripper: The gripper should extend to pick/place the object (Y coordinate).
- 476 4. Goal Achievement: A reward of +1 is given if the robot successfully reaches the goal.

$$r = \begin{cases} -\Delta X + \max(\Delta Y) + \max(\Delta Z), & \Delta X \geq \epsilon_x \\ -\Delta Z - \Delta X + \max(\Delta Y), & \Delta Z \geq \epsilon_z, \Delta x \geq \epsilon_x \\ -\Delta Y - \Delta Z - \Delta X, & \Delta Y \geq \epsilon_y, \Delta Z \geq \epsilon_z, \Delta x \geq \epsilon_x \\ 1, & \Delta Y < \epsilon_y, \Delta Z < \epsilon_z, \Delta x < \epsilon_x \end{cases} \quad (5)$$

With this reward function, we train a Proximal Policy Optimization agent using the implementation from [44], to predict actions that will lead to a suitable grasp on the goal position.

## 8 Experiments Details

### 8.1 Active Preference Learning Experiments Details

**Benchmark Dataset.** The dataset contains 100 test cases, each containing a ground-truth preference, two demonstrations with before and after state of user putting objects into fridge, one scenario to solve. Each demonstration has 3 objects initially in the fridge and 4 to place into the fridge. Meanwhile, each scenario has 4 objects initially in the fridge and 6 to place into the fridge. The objects in the fridge can be categorized into 5 categories: fruits, vegetables, condiments, dairy products, juice and soft drinks. The test cases are also grouped into 5 categories, where each categories contain 20 test cases: specific location, general location, relative position, subcategory exceptions, and conditionals. Fig 3 shows examples of each category.

The test cases “specific location” require each category to be placed at a specific location in the fridge (e.g. “fruit on the left side of top shelf”), while the rest of the categories have special requirements on a subset of the categories and only specific location requirements on the remaining categories. For the 20 “general location” test cases, half of them require only one category to be at a general location (e.g. “fruits on the left side of the fridge”), and the other half require two categories to be at a general location (e.g. “fruits on the left side of the fridge, and vegetables on the right side of the fridge). In addition, for the 20 “relative position” test cases, half of them require one category to be together with no specific location (e.g. “fruits should be placed together regardless of what shelf they are on”), and the other half require two categories to be together (e.g. “fruits and vegetables should be placed together regardless of what shelf they are on.”)

**Setup.** The prompts for the baseline approaches are in Appendix 9.3. We run each approach on a test case once. We make APRICOT generate  $N - 1 = 4$  candidate preferences and add the preference generated by Non-Interactive as the fifth candidate preference. We also make Cand+LLM-Q/A have the same set of candidate preferences and candidate plans and APRICOT for a given test case.

### 8.2 Task Planner Real Robot Experiments

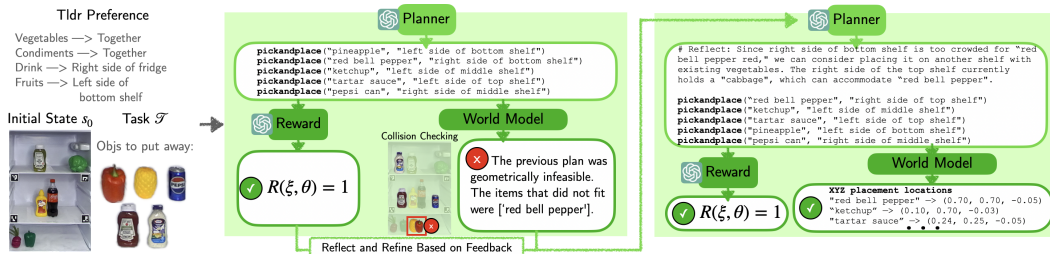


Figure 7: Example of APRICOT’s task planner reflecting and refining its plan based on feedback.

#### 8.2.1 Setup

For all real robot experiments, we use a 6-DoF Stretch Robot RE1 [45] as the mobile manipulator. A static RGB-D camera (ZED-2i [46]) is mounted to have a direct view of the fridge. A table is placed

to the left of the fridge, on which objects that need to be put away by the robots are placed in a row with no occlusions. Before the experiments, we have mapped the environment and determined the location of the table and the fridge in the map.

Images from the static camera are used by the perception module to track the current state of the world for the task planner and to convert user demonstrations into text-based demonstrations. The point cloud from the static camera is used by the world model to collision-check potential XYZ placement location. Meanwhile, the Stretch Robot’s onboard RGB-D camera is used by the pick policy to identify the grasp position of an object.

### 8.2.2 APRICOT Plan Refinement Example

Fig. 7 shows an example of APRICOT’s task planner refining its plan based on feedback. When the LLM generates the semantic plan, because it does not have knowledge about the environmental constraints of the fridge, it decides to place both the pineapple and the red bell pepper on the left side of the bottom shelf, which would satisfy the user’s preference. However, only one object can be placed on the left side of the bottom shelf. Given this plan, although the reward function determines that the semantic plan maximally satisfies the preference  $\mathcal{R}(\xi, \theta) = 1$ , the world model determines that red bell pepper will have a collision when being placed on the left side of the bottom shelf next to the pineapple. This feedback gets included in the input for the LLM, so the LLM is able to reflect on why its current plan is geometrically infeasible and determine an alternative semantic placement location for the red bell pepper (“right side of the top shelf”). The new plan satisfies the preference by maximizing the reward and is geometrically feasible by converting the semantic plans into XYZ placement locations for all the objects.

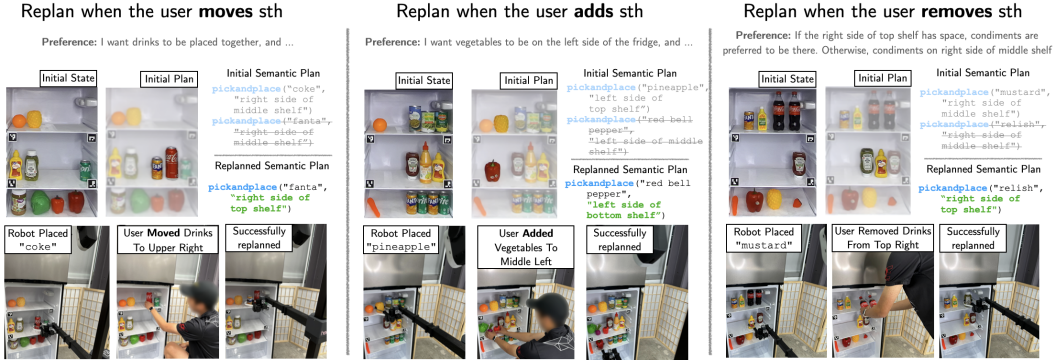


Figure 8: Examples of how APRICOT’s task planner can replan and handle different ways a user could affect and change the environment.

### 8.2.3 APRICOT Replanning Example

Fig. 8 show all 3 examples of how APRICOT’s task planner is able to replan and handle changes in the environment.

**Replanning after additions.** The user’s preference requires vegetables to be placed on the left side of the fridge. The planner initially plans to place the bell pepper on the left side of the middle shelf. However, the user disrupts the scene and adds various vegetables there. APRICOT’s closed-loop planner adjusts to its plan by placing the bell pepper on the left side of the bottom shelf which still has space.

**Replanning after removals.** The user’s preference requires placing condiments on the right of the top shelf if that space is empty. If that location is full, condiments can be placed on the right of the middle shelf. The planner initially plans to place the mustard and relish on the right side of the middle shelf since the right side of the top shelf is filled with coke bottles. However, the user disrupts the scene and removes the bottles. APRICOT’s closed-loop planner adjusts to this by placing the relish in the freed space to best satisfy the user’s preference.

## 8.2.4 Perception System Performance

To quantify the performance of our perception module, we evaluate it on the 9 real-world scenarios used in the task planner real-robot experiments.

**Baselines.** Recall that APRICOT’s perception module (GD+CLIP) uses Grounding-Dino [42] to detect the objects in the fridge and CLIP [43] to label the detected object name with an object name. Compared to this approach, we define the baseline GD-Only, which runs Grounding-DINO, given each object label from the list of all possible grocery items.

**Metrics.** We define two metrics. The **percentage of correctly identified object** calculates the average ratio between the detected objects that have the correct object label over the expected number of objects. The **number of incorrectly detected objects** calculates the average number of incorrect labels, so lower is better.

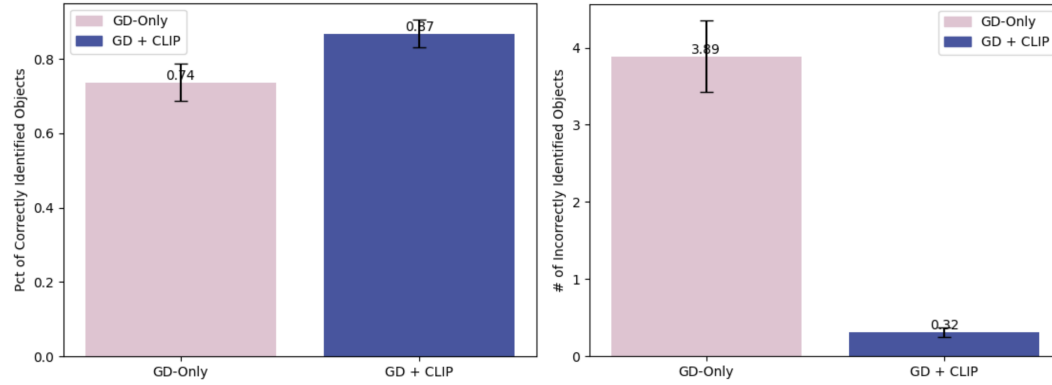


Figure 9: Result of perception module on real-robot scenarios. We also report the standard error in the bar plot.

**Results.** In Fig. 9, we observe that, for objects that are actually in the fridge, GD+CLIP is able to assign more correct labels (87.0%) compared to GD-Only (74.0%). In addition, GD-Only detects more objects incorrectly and has much more false positives compared to GD+CLIP. Because GD-Only calls Grounding-DINO for each possible object label, it repeatedly labels the same region in the image with different object labels. This behavior is undesirable because APRICOT’s task planner relies on the perception system to exactly detect the right number of objects in the fridge and assign each detected object with the correct label. GD-Only would cause the task planner to hallucinate what objects are currently in the fridge.

## 8.2.5 Active Preference Learning Result for Real Robot Scenario

Similar to the benchmark dataset for active preference learning experiments, for each scenario and ground-truth preference in the real robot experiments, we define two corresponding demonstrations that each consist of 3 objects initially in the fridge and 4 objects that the user puts away. We show the visual demonstration and the ground-truth preference in Fig. 10. We report the performance of APRICOT’s active preference learning module in Table. 1.

# 9 Prompts

## 9.1 Active Preference Learning Prompts

The prompts for APRICOT’s active preference learning modules are the following:

- Generate Candidate Preferences Given Demonstrations (Sec. 9.1.1)
- Generate a Plan Given a Preference, an Initial Condition, and a Task (Sec. 9.2)
- Calculate the Reward Given a Preference and a Plan (Sec. 9.1.2)

	Scenario	Preference Accuracy	Number of Queries
Easy	1	1.00	0.00
	2	0.00	5.00
	3	1.00	0.00
Medium	1	0.00	0.00
	2	1.00	1.00
	3	0.00	0.00
Hard	1	1.00	0.00
	2	1.00	2.00
	3	1.00	1.00
Overall		0.56	1.00

Table 1: APRICOT active preference learning module’s performance on real-world scenarios. Note that the difficulty level is based on how difficult the task is for the task planner (i.e. the number of objects initially in the fridge and the number of objects to put away.)

- Generate Candidate Preferences Given a Pair of Preferences (Sec. 9.1.3)
- Answer a Question Given a Preference (Sec. 9.1.4)

### 9.1.1 Generate Candidate Preferences

The LLM first analyzes the demonstrations in detail before proposing candidate preferences that can explain user behaviors in the demonstrations. We use gpt-4 for these prompts because this is a complex task.

```

----- System Message -----
You are an assistant who sees someone demonstrating how the fridge is
↪ organized and analyzes patterns in the demonstrations so that you can
↪ infer the potential preferences that the user might have.
----- Instruction -----
# Input
You are given 2 demonstrations that show the before and after when a set
↪ of objects gets put into the fridge. For each demonstration:
- "Objects that got put away" describes the objects that the user will
↪ demonstrate how they would like to put in the fridge.
- "Initial state of the fridge" describes the objects that are initially
↪ in the fridge before the user starts the demonstration.
- "Final state of the fridge" describes what the fridge looks like after
↪ the demonstration. All the objects in "Objects that got put away"
↪ should be in the fridge now.

# Goal
Your goal is to fill out the reasoning worksheet below that analyze the
↪ demonstrations and help you infer the preferences that the user could
↪ have.

The demonstrations are complementary, and they are shown by the user with
↪ the same preference. Thus, even though they are different, their
↪ difference is due to the difference in the initial state of the fridge
↪ and the objects that got put away. They are not contradictory, and
↪ there exist preferences that can explain why the 2 demonstrations are
↪ the way they are.

# Instructions and useful information
## Specific locations in the fridge

```



	Demonstrations				Ground-Truth Preference	Scenario Initial State	Objs to Put Away
	$s_0^1$	$s_T^1$	$s_0^2$	$s_T^2$			
Easy Scenarios					I prefer condiments to be put next to each other, and drinks to be next to each other.		Mustard Bottle Sprite Can
					I prefer fruits to be on the left side of the fridge, and drinks to be on the right side of the middle shelf.		Pineapple Juice bottle
					I prefer my drinks to be placed together next to existing drinks on the same shelf, regardless of which shelf or which side of the shelf they are on.		Coke Bottle Fanta Can
Medium Scenarios					I prefer fruits and vegetables to be together on one shelf. Dairy products should be on the left side of the fridge, and drinks should be on the right side of the fridge.		Sprite Can Pineapple Cheese Pepsi Can
					Fruits should be together on the left side of a shelf, and vegetables should be together on the left side of another shelf. Condiments can be on the right side of the middle shelf.		Red Bell Pepper Pineapple Ketchup Mustard bottle
					I would like condiments to be placed together and drinks to be on the left side of the fridge. If the left side of the top shelf has too many objects, canned and preservable food and be on the left side of the top shelf. Otherwise, they should be on the right side of the bottom shelf.		Relish Tartar Sauce Sprite Can Canned food
Hard Scenarios							Coke Bottle Relish Black Pepper Orange juice bottle Fanta can
					I want vegetables to be placed next to existing vegetables on the same shelf regardless of what shelf or what side it is. Fruits should be on the left side of the bottom shelf. I want drinks to be on the right side of the fridge. I want condiments to be placed next to existing condiments on the same shelf regardless of what shelf or what side it is.		Red Bell Pepper Ketchup Tartar Sauce Pineapple Pepsi Can
							Mustard Bottle Fanta Can Sprite Can Apple Juice Pineapple

Figure 10: Real world demonstrations and scenarios.

```

610 The fridge can be segmented into 3 shelves (top shelf, middle shelf,
611 ↪ bottom shelf); each shelf has 2 sides (left side, right side). The
612 ↪ fridge has 6 specific locations:
613 - left side of the top shelf, right side of the top shelf
614 - left side of the middle shelf, right side of the middle shelf
615 - left side of the bottom shelf, right side of the bottom shelf
616
617 ## General locations in the fridge
618 A general location contains multiple specific locations. There are 5
619 ↪ general locations:
620 - "left side of fridge" contains: "left side of top shelf", "left side
621 ↪ of middle shelf", or "left side of bottom shelf"
622 - "right side of fridge" contains: "right side of top shelf", "right
623 ↪ side of middle shelf", or "right side of bottom shelf"
624 - "top shelf" contains: "left side of top shelf", "right side of top
625 ↪ shelf"
626 - "middle shelf" contains: "left side of middle shelf", "right side of
627 ↪ middle shelf"
628 - "bottom shelf" contains: "left side of bottom shelf", "right side of
629 ↪ bottom shelf"
630
631 ## Details about the preferences that you need to output
632 A preference is a short paragraph that specifies requirements for each
633 ↪ category of grocery items. There must be at least one requirement for
634 ↪ each category. The type of requirement for each category can be
635 ↪ different. The categories are: "Fruits", "Vegetables", "Juice-and-soft
636 ↪ -drinks", "Dairy-Products", and "Condiments".
637
638 The requirement needs to be one of the following:
639 - **Type-1. Specific Locations.** These represent that the object must
640 ↪ place at this specific location. The options are:
641 - "left side of top shelf"
642 - "right side of top shelf"
643 - "left side of middle shelf"
644 - "right side of middle shelf"
645 - "left side of bottom shelf"
646 - "right side of bottom shelf".
647 - **Type-2. General Locations.** These are vaguer locations that contain
648 ↪ multiple specific locations. The options are:
649 - "left side of fridge": which means that the user is ok if the object
650 ↪ is placed on "left side of top shelf", "left side of middle shelf",
651 ↪ or "left side of bottom shelf"
652 - "right side of fridge": which means that the user is ok if the object
653 ↪ is placed on "right side of top shelf", "right side of middle shelf",
654 ↪ or "right side of bottom shelf"
655 - "top shelf": which means that the user is ok if the object is either
656 ↪ placed on the "left side of top shelf" or "right side of top shelf"
657 - "middle shelf": which means that the user is ok if the object is
658 ↪ either placed on the "left side of middle shelf" or "right side of
659 ↪ middle shelf"
660 - "bottom shelf": which means that the user is ok if the object is
661 ↪ either placed on the "left side of bottom shelf" or "right side of
662 ↪ bottom shelf"
663 - **Type-3. Relative Positions.** The options are:
664 - "<category> must be placed together next to existing <category>
665 ↪ regardless of which shelf they are on.": This means that the user
666 ↪ only cares that a category of objects are placed together next to
667 ↪ existing objects of the same type. The user does not care which
668 ↪ specific shelf or which side of the shelf the objects are placed on.

```



669     ↪ For example: "fruits must be placed together next to existing fruits  
670     ↪ regardless which shelf they are on."  
671     - "<category> must be placed on the same shelf next to <another category  
672     ↪ of objects>, and which specific shelf does not matter.": This means  
673     ↪ that the user only cares that a category of objects are placed  
674     ↪ together on the same shelf next to another category of objects. It  
675     ↪ does not matter which specific shelf the objects are on. For example:  
676     ↪ "fruits must be placed on the same shelf next to condiments, and  
677     ↪ which specific shelf does not matter."

678  
679 In addition to giving specific requirements for each category of grocery  
680     ↪ items, sometimes you may choose to add additional requirements. The  
681     ↪ options are:

682     - **\*\*Type-4. Exception For Attribute\*\***  
683     - "<category> needs to be placed at <specific location 1>, but <  
684     ↪ attribute of category> needs to be placed at <specific location 2>.":  
685     ↪ This means that the user has a different specific requirement for  
686     ↪ object with a certain attribute compared to the main category. An  
687     ↪ attribute includes a subcategory of the object, the size/weight of  
688     ↪ the object, a specific feature of the object, etc. For example, "  
689     ↪ Dairy product needs to be placed at the right side of top shelf, but  
690     ↪ cheese needs to be placed at left side of middle shelf." Here, "dairy  
691     ↪ product" is the main category, and "cheese" is the attribute, which  
692     ↪ is a specific type of dairy product. Another example is, "Fruits  
693     ↪ needs to be placed at the left side of middle shelf, but big fruits  
694     ↪ needs to be placed at right side of bottom shelf." Here, "fruits" is  
695     ↪ the main category, and "big fruit" is the attribute, which is about  
696     ↪ the size of the fruit.  
697     - **\*\*Type-5. Conditional On Space\*\***  
698     - "If there are less than <N> objects at <primary specific location>,  
699     ↪ I want <category> to be placed at <primary specific location>. Else  
700     ↪ , I want <category> to be placed at <second choice specific location  
701     ↪ >." : This means that there is a maximum number of objects that can be  
702     ↪ placed at top choice specific location. If that top choice specific  
703     ↪ location does not have less than N number of objects, the user wants  
704     ↪ the a category of objects to be placed at the second choice specific  
705     ↪ location. For example, "if there are less than 3 objects at the right  
706     ↪ side of top shelf, I want dairy products to be placed at right side  
707     ↪ of top shelf. Else, I want dairy products to be placed at left side  
708     ↪ of middle shelf."

709  
710  
711 Below is the reasoning worksheet and output that you must follow. <>  
712     ↪ contains parts that you must fill out and instructions on how to fill  
713     ↪ out that part.

714  
715 # Reasoning worksheet  
716 ## (Fruits) Reasoning about fruits  
717 ### (Fruits.A) Summarize where fruits are in the demonstrations  
718 - In demonstration 1:  
719     - Fruits initially in the fridge:  
720         - <You must write down an unordered list of fruits that are initially  
721         ↪ in the fridge. For each fruit, you must write down the specific  
722         ↪ locations it occupy, and what other grocery items are next to it>  
723     - Fruits that got placed in the fridge:  
724         - <You must write down an unordered list of fruits that are initially  
725         ↪ in the fridge, and the specific locations they occupy>  
726     <You must copy the same format and follow the same process for  
727     ↪ demonstration 2.>

```

728
729 ### (Fruits.B) When fruits are in the fridge, are they only appearing in
730 ↪ one specific location? If that is the case, what is that specific
731 ↪ location?
732 <You must look at the summary you wrote in (Fruits.A). You must carefully
733 ↪ analyze each fruit's location in verbose detail. If you answer yes,
734 ↪ you must write down the specific location that fruits are placed at.>
735
736 ### (Fruits.C) Only if you answered no in (Fruits.B), you should answer
737 ↪ this section. When fruits appear at different locations in the fridge,
738 ↪ you must group the fruits based on their specific locations they
739 ↪ appear at.
740 <Based on the summary you wrote in (Fruits.A), you must write down a json
741 ↪ (a dictionary) that map each specific location to fruits at that
742 ↪ specific location>
743 '''json
744 {
745     "demonstration 1": {
746         "<specific_location_1>": [<You must write down the fruits at
747         ↪ specific_location_1 as a list of strings>],
748         "<specific_location_2>": [<You must write down the fruits at
749         ↪ specific_location_2 as a list of strings>],
750         ... <You must do this for each specific location that appeared in your
751         ↪ summary in (Fruits.A)>
752     },
753     <You must do this for each demonstration.>
754 }
755 '''
756
757 ### (Fruits.D) Potential preferences for fruits
758 #### (Fruits.D.1) **Type-1. Specific Locations.**
759 <Based on section (Fruits.B), what are some specific locations that the
760 ↪ user might like the fruits to be placed at?>
761
762 #### (Fruits.D.2) **Type-2. Group Locations.**
763 <Based on the description of group locations explained at **Type-2.
764 ↪ General Locations.** and the specific locations in mentioned in (
765 ↪ Fruits.A) and (Fruits.C), what are some general locations that the
766 ↪ user might like the fruits to be placed at? For example, if the
767 ↪ specific locations is "left side of top shelf", there are two possible
768 ↪ general locations for fruits: "top shelf" or "left side of the fridge
769 ↪ ". You must write down your reasoning in verbose detail, and you must
770 ↪ specify general locations that fruits might be in.>
771
772 #### (Fruits.D.3) **Type-3. Relative Positions.**
773 <Based on the section (Fruits.A), what type of objects are fruits
774 ↪ typically placed next to? What kind of objects do you think the user
775 ↪ might like the fruits to be placed next to? You must write down your
776 ↪ reasoning in verbose detail.>
777
778 #### (Fruits.D.4) **Type-4. Exception For Attribute.**
779 <Only if your answer in (Fruits.B) is no, you should proceed to answer
780 ↪ this section. Based on the json in (Fruits.C), for each specific
781 ↪ location, what kind of attribute can you identify the fruits at that
782 ↪ specific location share? You must write down your reasoning in verbose
783 ↪ detail.>
784
785 #### (Fruits.D.5) **Type-5. Conditional On Space.**

```

```

786 <Only if your answer in (Fruits.B) is no, you should proceed to answer
787 ↪ this section. Based on the json in (Fruits.C), for each specific
788 ↪ location, what kind of conclusion can you make about the number of
789 ↪ objects there? What is a primary location that the user likes to put
790 ↪ fruits at? What is a secondary location that the user would put fruits
791 ↪ at? What is the condition for user to go from putting fruits at the
792 ↪ primary location to the secondary location?>
793
794 <You must now copy from ## (Fruits) Reasoning about fruits to #### (
795 ↪ Fruits.D.5) **Type-5. Conditional On Space.** and repeat the reasoning
796 ↪ process for the rest of the categories: ## (Vegetables) Reasoning
797 ↪ about vegetables, ## (Juice-and-soft-drinks) Reasoning about juice and
798 ↪ soft drinks, ## (Dairy-Products) Reasoning about dairy products, ## (
799 ↪ Condiments) Reasoning about condiments. You must write down the
800 ↪ potential preferences for all 5 categories. You cannot omit any
801 ↪ category and you must write down detailed reasoning for all of them. >
802

```

```

803 ----- System Message -----
804
805 You are an assistant who sees an analysis of the demonstrations that the
806 ↪ user have provided and propose potential preferences that the user might
807 ↪ have.
808 ----- Instruction -----
809
810 # Input
811 You are given a reasoning worksheet written in markdown format that has
812 ↪ already carefully analyzed the demonstrations that the user has
813 ↪ provided to represent their personal preference. You must pay close
814 ↪ attention to this reasoning worksheet.
815
816 # Goal
817 Your goal is to generate <num_preferences> fundamentally different and
818 ↪ diverse preferences that are consistent with the reasoning worksheet
819 ↪ and explain what the user want.
820
821 # Instructions and useful information
822 ## Specific locations in the fridge
823 The fridge can be segmented into 3 shelves (top shelf, middle shelf,
824 ↪ bottom shelf); each shelf has 2 sides (left side, right side). The
825 ↪ fridge has 6 specific locations:
826 - left side of the top shelf, right side of the top shelf
827 - left side of the middle shelf, right side of the middle shelf
828 - left side of the bottom shelf, right side of the bottom shelf
829
830 ## Details about the preferences that you need to output
831 A preference is a short paragraph that specifies requirements for each
832 ↪ category of grocery items. There must be at least one requirement for
833 ↪ each category present in the demonstrations. The type of requirement
834 ↪ for each category can be different. The categories are: "Fruits", "
835 ↪ Vegetables", "Juice-and-soft-drinks", "Dairy-Products", and "
836 ↪ Condiments".
837 If no objects from a given category are present in any of the
838 ↪ demonstrations, you should omit that category from your planning. Do
839 ↪ not randomly choose a rule. Your preferences should also not re-
840 ↪ explain what the categories are or include.
841
842 The requirement needs to be one of the following:
843 - **Type-1. Specific Locations.** These represent that the object must
844 ↪ place at this specific location. The options are:
845 - "left side of top shelf"
846 - "right side of top shelf"

```

```

846 - "left side of middle shelf"
847 - "right side of middle shelf"
848 - "left side of bottom shelf"
849 - "right side of bottom shelf".
850 - **Type-2. General Locations.** These are vaguer locations that contain
851 ↪ multiple specific locations. The options are:
852 - "left side of fridge": which means that the user is ok if the object
853 ↪ is placed on "left side of top shelf", "left side of middle shelf",
854 ↪ or "left side of bottom shelf"
855 - "right side of fridge": which means that the user is ok if the object
856 ↪ is placed on "right side of top shelf", "right side of middle shelf",
857 ↪ or "right side of bottom shelf"
858 - "top shelf": which means that the user is ok if the object is either
859 ↪ placed on the "left side of top shelf" or "right side of top shelf"
860 - "middle shelf": which means that the user is ok if the object is
861 ↪ either placed on the "left side of middle shelf" or "right side of
862 ↪ middle shelf"
863 - "bottom shelf": which means that the user is ok if the object is
864 ↪ either placed on the "left side of bottom shelf" or "right side of
865 ↪ bottom shelf"
866 - **Type-3. Relative Positions.** The options are:
867 - "<category> must be placed together next to existing <category>"
868 ↪ regardless of which shelf they are on.": This means that the user
869 ↪ only cares that a category of objects are placed together next to
870 ↪ existing objects of the same type. The user does not care which
871 ↪ specific shelf or which side of the shelf the objects are placed on.
872 ↪ For example: "fruits must be placed together next to existing fruits
873 ↪ regardless which shelf they are on."
874 - "<category> must be placed on the same shelf next to <another category
875 ↪ of objects>, and which specific shelf does not matter.": This means
876 ↪ that the user only cares that a category of objects are placed
877 ↪ together on the same shelf next to another category of objects. It
878 ↪ does not matter which specific shelf the objects are on. For example:
879 ↪ "fruits must be placed on the same shelf next to condiments, and
880 ↪ which specific shelf does not matter."
881
882 In addition to giving specific requirements for each category of grocery
883 ↪ items, sometimes you may choose to add additional requirements. The
884 ↪ options are:
885 - **Type-4. Exception For Attribute**
886 - "<category> needs to be placed at <specific location 1>, but <
887 ↪ attribute of category> needs to be placed at <specific location 2>."":
888 ↪ This means that the user has a different specific requirement for
889 ↪ object with a certain attribute compared to the main category. An
890 ↪ attribute includes a subcategory of the object, the size/weight of
891 ↪ the object, a specific feature of the object, etc. For example, "
892 ↪ Dairy product needs to be placed at the right side of top shelf, but
893 ↪ cheese needs to be placed at left side of middle shelf." Here, "dairy
894 ↪ product" is the main category, and "cheese" is the attribute, which
895 ↪ is a specific type of dairy product. Another example is, "Fruits
896 ↪ needs to be placed at the left side of middle shelf, but big fruits
897 ↪ needs to be placed at right side of bottom shelf." Here, "fruits" is
898 ↪ the main category, and "big fruit" is the attribute, which is about
899 ↪ the size of the fruit.
900 - **Type-5. Conditional On Space**
901 - "If there are less than <N> objects at <primary specific location>,
902 ↪ I want <category> to be placed at <primary specific location>. Else
903 ↪ , I want <category> to be placed at <second choice specific location
904 ↪ >."": This means that there is a maximum number of objects that can be

```

```

905     ↪ placed at top choice specific location. If that top choice specific
906     ↪ location does not have less than N number of objects, the user wants
907     ↪ the a category of objects to be placed at the second choice specific
908     ↪ location. For example, "if there are less than 3 objects at the right
909     ↪ side of top shelf, I want dairy products to be placed at right side
910     ↪ of top shelf. Else, I want dairy products to be placed at left side
911     ↪ of middle shelf."
912
913 Below is the output that you must follow. <> contains parts that you must
914     ↪ fill out and instructions on how to fill out that part. You must only
915     ↪ copy and output what is under # Potential Preference Generation
916     ↪ Process.
917
918 # Potential Preference Generation Process.
919 ## (I) Propose potential preferences based reasoning worksheet
920 <You must write <num_preferences> potential preferences that are diverse
921     ↪ in the requirements for each category. The potential preferences
922     ↪ CANNOT only differ in the wording they have. They must be
923     ↪ fundamentally different with different requirements for at least one
924     ↪ object. >
925 ### (I.1) Potential preference 1
926 #### (I.1.a) What type of requirements are you picking for each category?
927     ↪ You must pick different types of requirements compared to previous
928     ↪ preferences, so how are types that you picked different from previous
929     ↪ ones?
930 <For each category, you must pick at least one or more type of
931     ↪ requirements based on your answers in (Fruits.D), (Vegetables.D), (
932     ↪ Juice-and-soft-drinks.D), (Dairy-Products.D), (Condiments.D). The type
933     ↪ of requirements for each category can be the same, but they do not
934     ↪ have to be the same for all categories. For each category, you must
935     ↪ verbosely explain in detail what is the type of requirement that you
936     ↪ picked and what exactly is the preference for that category. If this
937     ↪ is the first potential preference that you are writing, you do not
938     ↪ need to answer how the types that you picked are different from
939     ↪ previous preferences. However, for subsequent potential preferences
940     ↪ that you will write, you must make sure that you are picking a
941     ↪ different type of requirements for the 5 categories. You must explain
942     ↪ in detail how the types that you pick are different from previous
943     ↪ preferences. You must cite the previous potential preferences by their
944     ↪ number, for example (I.1.b).>
945
946 #### (I.1.b) Preference
947 <Based on the types of requirements that you picked in (I.1.a), your goal
948     ↪ is to write in a paragraph a potential user preference that is
949     ↪ consistent with the demonstrations and the reasoning worksheet. You
950     ↪ must write in natural language and there must be at least one
951     ↪ requirement for each category. You must not make reference to other
952     ↪ potetial preferences or reasoning that you have written. Each
953     ↪ preference be understandable on their own without referring to another
954     ↪ potential preference. >
955
956 <You must now copy from ### (I.1) Potential preference 1 and repeat the
957     ↪ writing process for the remaining <num_preferences_minus_one>
958     ↪ potential preferences. You must write <num_preferences> potential
959     ↪ preferences in total.>
960
961 # (II) Final valid JSON output
962 You must copy the <num_preferences> diverse potential preferences that
963     ↪ you wrote in (II) as a list of strings in a valid json format. You

```



```

964 ↪ must only copy the preference, which must not refer to other
965 ↪ preferences or parts of the writing process. Each preference must
966 ↪ define requirements for all the categories, and it must be
967 ↪ understandable and interpretable on its own without referring to other
968 ↪ potential preferences or other writing that you have done before. Each
969 ↪ preference should You must make sure that when writing each
970 ↪ preference, you do not use single quotes or double quotes or citations
971 ↪ .
972   "'json
973   [
974     "<You must copy the first potential preference in (I.1.b) here. For
975     ↪ subsequent potential preferences, you must not make reference to
976     ↪ previous potential preferences.>",
977     "<You must copy the remaining <num_preferences_minus_one> preferences
978     ↪ here>"
979   ]
980   "'
981

```

## 9.1.2 Calculate the Reward

The LLM is prompted to output a json of yes/no on whether an object's semantic placement location in a plan  $\xi$  satisfies a given preference  $\theta$ . The reward is calculated by the ratio between the sum of the responses (yes is 1 and no is 0) and the total number of objects in the plan. We use gpt-4o for this prompt.

```

987 ----- System Message -----
988 You are roleplaying as a user with a specific preference. Your goal is to
989 ↪ answer whether or not an object placement has satisfied your preference
990 ↪ and the initial configuration of the fridge.
991 ----- Instruction -----
992 You are roleplaying as a user with a specific preference. You are shown
993 ↪ the objects that are already initially in the fridge. The fridge has
994 ↪ different shelves, which are divided into smaller sub-sections. The
995 ↪ objects in a specific section are listed from left to right.
996
997 You are also shown a plan to place an object at a specific part of the
998 ↪ fridge. Your goal is to answer "yes" or "no" on whether the object
999 ↪ placement has satisfied your preference. You must also pay attention
1000 ↪ to objects that are already initially in the fridge because where you
1001 ↪ want objects placed could depend that initial condition.
1002
1003 You must respect the following rules when you make your decision:
1004 - You do not need to worry about the amount of space in the fridge. You
1005 ↪ must not make your decision based on you judging if a shelf is already
1006 ↪ full.
1007 - If the preference for a category is a specific location, you must only
1008 ↪ check if the object is being placed at that specific location. You
1009 ↪ must ignore other objects that are at that specific location.
1010   * For example, consider when your preference is to put drinks together
1011     ↪ regardless of the shelf and to put fruits on the left side of top
1012     ↪ shelf, and the plan has placed a drink on the left side of the top
1013     ↪ shelf then a fruit also on the left side of top shelf. When you
1014     ↪ answer whether the plan of placing the fruit on the left side of top
1015     ↪ shelf, you should answer "yes" because it satisfies the specific
1016     ↪ placement location for fruit.
1017 - If the object placement plan is more general than your preference, that
1018 ↪ plan does not satisfy your preference, and you must reply "no".
1019   * For example, if your preference is to put fruits on the left side of
1020     ↪ the middle shelf, and the plan tries to put a fruit on the middle
1021

```

```

1022     ↪ shelf, the plan is too general, and it will not always satisfy your
1023     ↪ preference. Thus, you must reply "no".
1024 - When your preference has conditionals, you must carefully check if that
1025     ↪ conditional is true first when you make your decision.
1026     * For example, if your preference wants vegetables to be at location 2
1027     ↪ if there are already N objects at location 1, you must check the
1028     ↪ objects that are already initially in the fridge and make sure that
1029     ↪ the condition (N objects at location 1) is true.
1030 - When your preference depends on the initial condition of the fridge,
1031     ↪ you must carefully check what is desirable placement location based on
1032     ↪ the initial condition of the fridge.
1033     * For example, if your preference wants vegetables to be placed together
1034     ↪ , you must check if there are already vegetables in the fridge. If
1035     ↪ there are already vegetables at the right side of top shelf, the
1036     ↪ object placement plan should also place new objects at the right side
1037     ↪ of top shelf.
1038
1039 The preference, the objects already initially in the fridge, the object
1040     ↪ placement plan that you receive will be in markdown format:
1041 # Your preference
1042 ... Your preference will be stated here ...
1043
1044 # Objects already initially in the fridge
1045 ```
1046 ... The objects already initially in the fridge will be stated here as an
1047     ↪ json ...
1048 ```
1049
1050 # Object placement plan
1051 ```
1052 ... The plan will be stated here as a python code: pickandplace(
1053     ↪ object_to_place, placement_location)
1054 ```
1055
1056 You must reply in a valid json format, each item corresponds to one of
1057     ↪ the pickandplace() function in the object placement plan. You must
1058     ↪ only use double quote. You cannot use apostrophe or single quote. See
1059     ↪ below for the format that you must follow.
1060 ```json
1061 [
1062     {
1063         "Reasoning": "{You must not use any more of quotation mark inside
1064             ↪ your reasoning. You must not write apostrophe, double quote, or
1065             ↪ single quote. You must put your reasoning on whether a user with
1066             ↪ your preference would be happy with the object placement given
1067             ↪ the objects already initially in the fridge.}",
1068         "Does this plan satisfy your preference (yes/no)": "{you must only
1069             ↪ reply yes or no}"
1070     }, ...<there should be one dictionary per line of pickandplace()>
1071 ]
1072 ```
1073 ----- Example Input -----
1074 # Your preference
1075 I like putting dairy on the top shelf and vegetables on the left side of
1076     ↪ the bottom shelf.
1077
1078 # Objects already initially in the fridge
1079 ```
1080 {

```

```

1081     "top shelf":
1082     {
1083         "left side of top shelf": [],
1084         "right side of top shelf": ["cheese", "yogurt"]
1085     },
1086     "middle shelf":
1087     {
1088         "left side of middle shelf": [],
1089         "right side of middle shelf": []
1090     },
1091     "bottom shelf":
1092     {
1093         "left side of bottom shelf": ["carrot"],
1094         "right side of bottom shelf": []
1095     }
1096 }
1097 '''
1098
1099 # Object placement plan
1100 '''
1101 pickandplace("oat milk", "right side of top shelf")
1102 pickandplace("whole milk", "right side of middle shelf")
1103 '''
1104 ----- Example Response -----
1105 '''json
1106 [
1107     {
1108         "Reasoning": "Oat milk is a dairy product. I prefer dairy on the
1109         ↪ top shelf. The plan was to put oat milk on the right side of
1110         ↪ top shelf, which falls under top shelf, so the plan
1111         ↪ satisfies my preference.",
1112         "Does this plan satisfy your preference (yes/no)": "yes"
1113     },
1114     {
1115         "Reasoning": "Whole milk is a dairy product. I prefer dairy on
1116         ↪ the top shelf. The plan was to put whole milk on the right
1117         ↪ side of middle shelf, which is not top shelf, so the plan
1118         ↪ does not satisfy my preference.",
1119         "Does this plan satisfy your preference (yes/no)": "no"
1120     }
1121 ]
1122 '''
1123 ----- Example Input -----
1124 # Your preference
1125 I like fruits on the left side of the middle shelf, and vegetables on
1126 ↪ the right side of the middle shelf.
1127
1128 # Objects already initially in the fridge
1129 '''
1130 {
1131     "top shelf":
1132     {
1133         "left side of top shelf": ["whole milk"],
1134         "right side of top shelf": []
1135     },
1136     "middle shelf":
1137     {
1138         "left side of middle shelf": [],
1139         "right side of middle shelf": []

```

```

1140     },
1141     "bottom shelf":
1142     {
1143         "left side of bottom shelf": [],
1144         "right side of bottom shelf": []
1145     }
1146 }
1147 '''
1148
1149 # Object placement plan
1150 '''
1151 pickandplace("apple", "middle shelf")
1152 '''
1153 ----- Example Response -----
1154 '''json
1155 [
1156     {
1157         "Reasoning": "Apple is a fruit. I prefer fruits on the left side
1158         ↪ of the middle shelf. The plan is placing apple in a more
1159         ↪ general location: middle shelf. This means that it could
1160         ↪ potentially place apple not specifically at the left side of
1161         ↪ the middle shelf, so the plan does not satisfy my preference
1162         ↪ .",
1163         "Does this plan satisfy your preference (yes/no)": "no"
1164     }
1165 ]
1166 '''
1167 ----- Example Input -----
1168 # Your preference
1169 I prefer vegetables be placed together. I also want fruits on the middle
1170 ↪ shelf.
1171
1172 # Objects already initially in the fridge
1173 '''
1174 {
1175     "top shelf":
1176     {
1177         "left side of top shelf": [],
1178         "right side of top shelf": []
1179     },
1180     "middle shelf":
1181     {
1182         "left side of middle shelf": [],
1183         "right side of middle shelf": ["cucumber", "carrot"]
1184     },
1185     "bottom shelf":
1186     {
1187         "left side of bottom shelf": [],
1188         "right side of bottom shelf": []
1189     }
1190 }
1191 '''
1192
1193 # Object placement plan
1194 '''
1195 pickandplace("spinach", "right side of top shelf")
1196 '''
1197 ----- Example Response -----
1198 '''json

```

```

1199 [
1200     {
1201         "Reasoning": "Spinach is a vegetable. I want vegetables to be
1202             ↪ placed together, so I need to check if there are vegetables
1203             ↪ initially in the fridge. There are vegetables (cucumber,
1204             ↪ carrot) at the right side of middle shelf, so new vegetables
1205             ↪ should also get placed at the right side of middle shelf.
1206             ↪ However, the plan decides to place spinach to the right side
1207             ↪ of the top shelf, which is not right side of middle shelf.
1208             ↪ Thus, this plan does not satisfy my preference.",
1209         "Does this plan satisfy your preference (yes/no)": "no"
1210     }
1211 ]
1212 '''
1213 ----- Example Input -----
1214 # Your preference
1215 If there are less than 2 dairy product on the right side of top shelf,
1216 ↪ you can put dairy products on the right side of top shelf; else, I
1217 ↪ want the rest of the dairy product to be on the right side of middle
1218 ↪ shelf.
1219
1220 # Objects already initially in the fridge
1221 '''
1222 {
1223     "top shelf":
1224     {
1225         "left side of top shelf": [],
1226         "right side of top shelf": ["oat milk"]
1227     },
1228     "middle shelf":
1229     {
1230         "left side of middle shelf": [],
1231         "right side of middle shelf": []
1232     },
1233     "bottom shelf":
1234     {
1235         "left side of bottom shelf": [],
1236         "right side of bottom shelf": []
1237     }
1238 }
1239 '''
1240
1241 # Object placement plan
1242 '''
1243 pickandplace("whole milk", "right side of middle shelf")
1244 '''
1245 ----- Example Response -----
1246 '''json
1247 [
1248     {
1249         "Reasoning": "Whole milk is a dairy product. My preference has a
1250             ↪ condition: whether there are less than 2 dairy products on
1251             ↪ the right side of top shelf, so I must pay close attention to
1252             ↪ the objects initially in the fridge. There is 1 dairy
1253             ↪ product (oat milk), so the condition (less than 2 dairy
1254             ↪ product on the right side of top shelf) is true. My
1255             ↪ preference would want whole milk also on the right side of
1256             ↪ top shelf. However, the plan put whole milk on right side of
1257             ↪ middle shelf, so it does not satisfy my preference.",

```



```

1258         "Does this plan satisfy your preference (yes/no)": "no"
1259     }
1260 ]
1261 '''
1262 ----- Example Input -----
1263 # Your preference
1264 If there are less than 2 dairy product on the right side of top shelf,
1265 ↪ you can put dairy products on the right side of top shelf; else, I
1266 ↪ want the rest of the dairy product to be on the right side of middle
1267 ↪ shelf.
1268
1269 # Objects already initially in the fridge
1270 '''
1271 {
1272     "top shelf":
1273     {
1274         "left side of top shelf": [],
1275         "right side of top shelf": ["oat milk", "whole milk"]
1276     },
1277     "middle shelf":
1278     {
1279         "left side of middle shelf": [],
1280         "right side of middle shelf": []
1281     },
1282     "bottom shelf":
1283     {
1284         "left side of bottom shelf": [],
1285         "right side of bottom shelf": []
1286     }
1287 }
1288 '''
1289
1290 # Object placement plan
1291 '''
1292 pickandplace("cheese", "right side of top shelf")
1293 '''
1294 ----- Example Response -----
1295 '''json
1296 [
1297     {
1298         "Reasoning": "Cheese is a dairy product. My preference has a
1299         ↪ condition: whether there are less than 2 dairy products on
1300         ↪ the right side of top shelf, so I must pay close attention to
1301         ↪ the objects initially in the fridge. There are 2 dairy
1302         ↪ product (oat milk, whole milk), so the condition (less than 2
1303         ↪ dairy product on the right side of top shelf) is false. I
1304         ↪ should look at the else case of my prefernece, which wants
1305         ↪ the cheese to be on the right side of middle shelf. However,
1306         ↪ the plan put cheese on right side of top shelf, so it does
1307         ↪ not satisfy my preference.",
1308         "Does this plan satisfy your preference (yes/no)": "no"
1309     }
1310 ]
1311 '''
1312 ----- Example Input -----
1313 # Your preference
1314 I prefer vegetables be placed on the right side of middle shelf. I want
1315 ↪ fruits on the right side of top shelf, condiments on the left side of
1316 ↪ top shelf.

```

```

1317
1318 # Objects already initially in the fridge
1319 '''
1320 {
1321     "top shelf":
1322     {
1323         "left side of top shelf": [],
1324         "right side of top shelf": []
1325     },
1326     "middle shelf":
1327     {
1328         "left side of middle shelf": [],
1329         "right side of middle shelf": ["cucumber", "carrot", "corn"]
1330     },
1331     "bottom shelf":
1332     {
1333         "left side of bottom shelf": [],
1334         "right side of bottom shelf": []
1335     }
1336 }
1337 '''
1338
1339 # Object placement plan
1340 '''
1341 pickandplace("spinach", "right side of bottom shelf")
1342 '''
1343 ----- Example Response -----
1344 '''json
1345 [
1346     {
1347         "Reasoning": "Spinach is a vegetable. I prefer vegetables on the
1348         ↳ right side of middle shelf. The plan is placing spinach at
1349         ↳ the wrong location: right side of bottom shelf. Thus, this
1350         ↳ plan does not satisfy my preference.",
1351         "Does this plan satisfy your preference (yes/no)": "no"
1352     }
1353 ]
1354 '''
1355 ----- Example Input -----
1356 # Your preference
1357 I prefer vegetables be placed together next to exsiting vegetables
1358 ↳ regardless of which shelf they are on. I want fruits on the right
1359 ↳ side of top shelf.
1360
1361 # Objects already initially in the fridge
1362 '''
1363 {
1364     "top shelf":
1365     {
1366         "left side of top shelf": [],
1367         "right side of top shelf": ["cucumber"]
1368     },
1369     "middle shelf":
1370     {
1371         "left side of middle shelf": [],
1372         "right side of middle shelf": []
1373     },
1374     "bottom shelf":
1375     {

```

```

1376         "left side of bottom shelf": [],
1377         "right side of bottom shelf": []
1378     }
1379 }
1380 """
1381
1382 # Object placement plan
1383 """
1384 pickandplace("spinach", "right side of top shelf")
1385 pickandplace("apple", "right side of top shelf")
1386 """
1387 ----- Example Response -----
1388 """json
1389 [
1390     {
1391         "Reasoning": "Spinach is a vegetable. I want vegetables to be
1392         ↪ placed together, so I need to check if there are vegetables
1393         ↪ initially in the fridge. There are vegetables (cucumber) at
1394         ↪ the right side of top shelf, so new vegetables should also
1395         ↪ get placed at the right side of top shelf. Since the plan put
1396         ↪ spinach at the top shelf, it has satisfied my preference.",
1397         "Does this plan satisfy your preference (yes/no)": "yes"
1398     },
1399     {
1400         "Reasoning": "Apple is a fruit. I prefer fruits on the right
1401         ↪ side of top shelf. The plan is placing apple at the right
1402         ↪ side of top shelf, which matches the requirement and
1403         ↪ satisfies my preference.",
1404         "Does this plan satisfy your preference (yes/no)": "yes"
1405     }
1406 ]
1407 """
1408

```

### 1409 9.1.3 Generate Candidate Questions

1410 The LLM receives as input a pair of preferences, the corresponding plans for these preferences, the  
1411 reward of each plan given these preferences, the initial condition, and the tasks that these plans are  
1412 generated to solve. We use gpt-4o for this prompt.

```

1413 ----- System Message -----
1414 You are an assistant that ask informative yes-or-no questions to try to
1415 ↪ differentiate between two potential preference candidates.
1416 ----- Instruction -----
1417
1418 # Goal
1419 You are trying to learn what is a user's personal preference. You
1420 ↪ receive a new initial state that you are trying to solve, two possible
1421 ↪ candidates of the user's preference, the corresponding task plan
1422 ↪ generated based on the candidate preferences.
1423
1424 Your goal is to analyze the differences between the two candidates and
1425 ↪ their plans and ask <num_questions> informative, effective yes-or-no
1426 ↪ questions such that the user's answers to your questions can reveal
1427 ↪ which candidate is closer to the user's preference.
1428
1429 # Input
1430 You will receive the input in the following format:
1431 # New Initial Condition To Solve
1432 ## Objects initially in the fridge
1433 """json

```

```

1434 ... The objects already initially in the fridge will be stated here as an
1435 ↪ json ...
1436 ""
1437
1438 ## Objects that must be put away into the fridge
1439 ""json
1440 ... The objects that need to be put away into the fridge will be stated
1441 ↪ here as a list ...
1442 ""
1443
1444 # Preference Candidate 1
1445 ... The first candidate will be stated here ...
1446
1447 ## Plan 1 based on Preference Candidate 1
1448 ... The plan will put away all the objects that need to be put away ...
1449
1450 ### Preference Candidate 1's Score on Plan 1
1451 ... Numerical score here. The higher a score, the better Plan 1 is at
1452 ↪ satisfying the preference candidate 1...
1453 ### Preference Candidate 2's Score on Plan 1
1454 ... Numerical score here. The higher a score, the better Plan 1 is at
1455 ↪ satisfying the preference candidate 2 ...
1456
1457 # Preference Candidate 2
1458 ... The second candidate will be stated here ...
1459
1460 ## Plan 2 based on Preference Candidate 2
1461 ... The plan will put away all the objects that need to be put away ...
1462
1463 ### Preference Candidate 1's Score on Plan 2
1464 ... Numerical score here. The higher a score, the better Plan 2 is at
1465 ↪ satisfying the preference candidate 1...
1466 ### Preference Candidate 2's Score on Plan 2
1467 ... Numerical score here. The higher a score, the better Plan 2 is at
1468 ↪ satisfying the preference candidate 2 ...
1469
1470 You must follow these rules when coming up with questions to ask:
1471 - You must analyze the difference in the candidate preferences and their
1472 ↪ plans. The differences will inform you on what question to ask. The
1473 ↪ scores can inform you how much each plan is liked by each preference
1474 ↪ candidate. The scores can help guide you to understand the main
1475 ↪ difference in the candidate preferences.
1476 - You must ask questions about the preferences. You must never ask
1477 ↪ question about that mention the new initial condition to solve or the
1478 ↪ plan that got generated.
1479 - Your questions must just be about what the user prefer. They must be
1480 ↪ standalone so that the user can just look at the question and be able
1481 ↪ to answer that question.
1482 - You must ask effective, useful questions where user's answer will help
1483 ↪ you figure out which preference candidate is better.
1484 - You must ask simple yes or no questions.
1485 - Your questions cannot ask if the user prefers A or B. However, you can
1486 ↪ ask if the user prefers A rather than B.
1487 - Your questions should try to use wording or terminology that are also
1488 ↪ used in the candidate preferences.
1489 - Your question must be about specific categories or attributes under a
1490 ↪ specific category. The categories are for example: ["fruits", "
1491 ↪ vegetables", "dairy product", "condiments", "juice and soft drinks"].
1492 ↪ Some examples of attributes are: "heavy fruits", which is an attribute

```

```

1493   ↪ under "fruits"; "sweet conditments", which is an attribute under "
1494   ↪ condiments"; "soft drinks", which is an attribute under "juice and
1495   ↪ soft drinks".
1496   - Your questions must not be about overall objects or items. You must
1497   ↪ never ask questions like "Do you prefer to items to be ...?" or "Do
1498   ↪ you prefer items of the same category ...?"
1499   - Your questions must not be asking about specific objects. You must
1500   ↪ never ask questions like "Do you prefer chocolate milk to be placed at
1501   ↪ ...?" Instead, you can ask questions about the specific category (e.g
1502   ↪ . "Do you prefer dairy products to be placed at ...?" or specific
1503   ↪ attribute (e.g. "Do you prefer milk to be placed at", where "milk" is
1504   ↪ the specific attribute.)
1505
1506   You must reply in this format:
1507   # Reasoning
1508     You should analyze the differences between preference candidates and
1509   ↪ their plans. You should strategize what questions you would ask here
1510
1511   # Questions
1512     You must put your <num_questions> questions as an unordered list here
1513

```

#### 1514 9.1.4 Answering a Question Given a Preference

1515 The LLM is asked to roleplay a user who has a preference  $\theta$  answering the question  $q$ . We use  
 1516 gpt-4o for this prompt. This prompt is used to:

- 1517 • Estimate the likelihood  $P(o|q, \theta)$ : We use OpenAI's API, which outputs the log-likelihood  
 1518 of outputting a specific token, to extract the log-likelihood of the LLM outputting "yes" or  
 1519 "no".
- 1520 • Mimic the user's answer with ground-truth preference: When the interactive approaches  
 1521 need to query the user, the LLM is supplied with the ground-truth preference (not known  
 1522 to the approaches) to answer the question that an approach asks.

```

1523 ----- System Message -----
1524 You are roleplaying as a user with a specific preference. Your goal is to
1525 ↪ answer yes-or-no question based on the preference of the user whom you
1526 ↪ are roleplaying. When you put your answer in the json, you must only
1527 ↪ write "yes" or "no"
1528 ----- Instruction -----
1529 You are roleplaying as a user with a specific preference. You are asked
1530 ↪ one yes-or-no question, and you goal is answer "yes" or "no" based on
1531 ↪ your preference.
1532
1533 The preference and the questions that you receive will be in markdown
1534 ↪ format:
1535 # Your preference
1536   Your preference will be stated here
1537
1538 # Question for you to answer
1539   The question will be stated here as an unordered list
1540
1541 You must follow these rules when answering:
1542 - When you write down your reasoning, you must not use apostrophy, single
1543 ↪ quote, or double quote.
1544 - You must only answer "yes" or "no". Even if it does not apply to your
1545 ↪ preference or you are not sure, you must answer "yes" or "no". You
1546 ↪ cannot answer anything else. You must not write "N/A" or "na" or "n/a
1547 ↪ ". You must only output "yes" or "no".
1548

```



```

1549 - If the question is asking whether the user prefers a specific thing,
1550 ↪ but the user's preference is more general, the user will likely
1551 ↪ respond no because they do not care about the specific thing and they
1552 ↪ only care about the general requirement.
1553 - If the question is asking whether the user prefers a general thing, but
1554 ↪ the user's preference is more specific, the user will likely respond
1555 ↪ no because the general thing can include cases that violate the user's
1556 ↪ specific requirement.
1557
1558 You must reply in a json, which includes your reasoning process and
1559 ↪ answer to a question. You must follow this format:
1560 {
1561   "Reasoning": "{put your reasoning on how a user with your preference
1562   ↪ would answer to this question. You must not use single quote or
1563   ↪ double quote inside your reasoning. }",
1564   "Answer (yes/no)": "{you must only reply yes or no}"
1565 }
1566 ----- Example Input -----
1567 # Your preference
1568 Fruits should be placed on left side of the fridge.
1569
1570 # Questions for you to answer
1571 - Do you prefer fruits to be placed on the left side of the top shelf?
1572 ----- Example Response -----
1573 {
1574   "Reasoning": "The user prefers fruits to be on the left side of the
1575   ↪ fridge, which is a general requirement. The user does not
1576   ↪ specifically prefer fruits to be on the left side of the top shelf,
1577   ↪ so the user would answer no.",
1578   "Answer (yes/no)": "no"
1579 }
1580 ----- Example Input -----
1581 # Your preference
1582 Fruits should be placed on left side of the fridge.
1583
1584 # Questions for you to answer
1585 - Is it important to you to have fruits specifically on the left side of
1586 ↪ the middle shelf rather than just on the right left of the fridge?
1587 ----- Example Response -----
1588 {
1589   "Reasoning": "The user prefers fruits to be on the left side of the
1590   ↪ fridge, which is a general requirement and the user will not care
1591   ↪ about the specific shelf that fruits are placed on as long as it is
1592   ↪ the left side of the fridge. Thus, it is not important for fruits
1593   ↪ to be on the left side of the middle shelf, and the user would
1594   ↪ answer no.",
1595   "Answer (yes/no)": "no"
1596 }
1597 ----- Example Input -----
1598 # Your preference
1599 Fruits should be placed on left side of the top shelf.
1600
1601 # Questions for you to answer
1602 - Do you prefer fruits to be placed on the left side of the fridge?
1603 ----- Example Response -----
1604 {
1605   "Reasoning": "The user prefers that fruits are on the left side of the
1606   ↪ top shelf, which is specific. Left side of the fridge includes
1607   ↪ left side of the middle shelf and left side of the bottom shelf,

```

```

1608     ↪ which does not match preference of the user. The user would answer
1609     ↪ no.",
1610     "Answer (yes/no)": "no"
1611 }
1612 ----- Example Input -----
1613 # Your preference
1614 Fruits should be placed on the left side of the middle or bottom shelf.
1615
1616 # Questions for you to answer
1617 - Do you prefer to place fruits next to other fruits already in the
1618   ↪ fridge?
1619 ----- Example Response -----
1620 {
1621     "Reasoning": "The user prefers that fruits are placed at specific
1622     ↪ locations (left side of the middle or bottom shelf). The question
1623     ↪ is more explicitly asking if the user prefers fruits to be placed
1624     ↪ next to other fruits already in the fridge, which does not have
1625     ↪ specfic locations requirement, so the user would say no.",
1626     "Answer (yes/no)": "no"
1627 }
1628 ----- Example Input -----
1629 # Your preference
1630 Fruits should be placed on the left side of the middle or bottom shelf.
1631
1632 # Questions for you to answer
1633 - Do you prefer to have similar items grouped together on the same shelf
1634   ↪ ?
1635 ----- Example Response -----
1636 {
1637     "Reasoning": "The user prefers that fruits are placed at specific
1638     ↪ locations (left side of the middle or bottom shelf). The question
1639     ↪ is implying that the user only cares about similar items being on
1640     ↪ the same shelf without specific requirements on which shelf and
1641     ↪ which side of the shelf, so the user would answer no.",
1642     "Answer (yes/no)": "no"
1643 }
1644 ----- Example Input -----
1645 # Your preference
1646 I want fruits to be placed together next to fruits that are already in
1647   ↪ the fridge.
1648
1649 # Questions for you to answer
1650 - Do you prefer to place fruits next to other fruits already in the
1651   ↪ fridge?
1652 ----- Example Response -----
1653 {
1654     "Reasoning": "The user prefers that fruits are placed together next to
1655     ↪ fruits that are already in the fridge. The question also asks if
1656     ↪ the user prefers to place fruits next to other fruits already in
1657     ↪ the fridge, so the answer is yes.",
1658     "Answer (yes/no)": "yes"
1659 }
1660 ----- Example Input -----
1661 # Your preference
1662 Fruits should be placed on the left side of the middle shelf.
1663
1664 # Questions for you to answer
1665 - Do you prefer if fruits are placed specifically on the middle shelf?
1666 ----- Example Response -----

```

```

1667 {
1668   "Reasoning": "The user prefers that fruits are placed at specific
1669   ↪ locations (left side of the middle shelf). The question is asking
1670   ↪ about a more general requirement because middle shelf is more
1671   ↪ general than the left side of middle shelf. Based on the rules
1672   ↪ above, the general location contains locations that violate the
1673   ↪ user's preference (right side of the middle shelf), so answer is no
1674   ↪ .",
1675   "Answer (yes/no)": "no"
1676 }
1677 ----- Example Input -----
1678 # Your preference
1679 Fruits should be placed on the left side of the middle shelf. Vegetables
1680 ↪ should be on the right side of middle shelf.
1681
1682 # Questions for you to answer
1683 - Do you prefer to have items of the same category placed together on
1684 ↪ the same shelf?
1685 ----- Example Response -----
1686 {
1687   "Reasoning": "The user prefers that fruits are placed at specific
1688   ↪ locations (left side of the middle shelf) and vegetables are also
1689   ↪ placed at specific locations (right side of the middle shelf). The
1690   ↪ question is asking if there is any category of objects that needs
1691   ↪ to be placed together on the same shelf. Since the requirements for
1692   ↪ fruits is about a specific location instead of putting fruits
1693   ↪ together, and the requirements for vegetables is also about a
1694   ↪ specific location instead of putting vegetables together, the
1695   ↪ answer to this question is no",
1696   "Answer (yes/no)": "no"
1697 }
1698 ----- Example Input -----
1699 # Your preference
1700 Fruits should be placed on the left side of the middle shelf. Condiments
1701 ↪ on the right side of the top shelf. Vegetables should be placed
1702 ↪ together next to other existing vegetables.
1703
1704 # Questions for you to answer
1705 - Do you prefer to have items of the same category placed together on
1706 ↪ the same shelf?
1707 ----- Example Response -----
1708 {
1709   "Reasoning": "The user prefers that fruits are placed at specific
1710   ↪ locations (left side of the middle shelf), condiments are placed at
1711   ↪ specific locations (right side of the top shelf), and vegetables
1712   ↪ need to be placed together. The question asks if there is any
1713   ↪ category of objects that needs to be placed together on the same
1714   ↪ shelf. In this preference, vegetables need to be placed together on
1715   ↪ the same shelf. Since there exists one category that are placed
1716   ↪ together on the same shelf, the answer to this question is yes",
1717   "Answer (yes/no)": "yes"
1718 }

```

## 1720 9.2 Task Planner Prompts

1721 The LLM prompt will generate a semantic plan that satisfies the given user preferences. When  
1722 it receives feedback, it will reflect on its plan and revise its original plan. We use gpt-4-turbo

1723 because we empirically find that gpt-4o tends to make more mistakes (typos) when generating the  
1724 code.

```
1725 ----- System Message -----
1726
1727 You are an assistant that comes up with a plan for putting items into a
1728 ↪ fridge given a list of items and a human's preference.
1729 ----- Instruction -----
1730 You must analyze a human's preferences and then come up with a plan to
1731 ↪ put items into a fridge.
1732
1733 You will receive the following as input:
1734 Optional[Feedback]: ...
1735 Objects: ...
1736 Locations: ...
1737 Initial State: ...
1738 Preference: ...
1739
1740 where
1741 - "Feedback" appears if the previous plan was geometrically infeasible.
1742   - It will contain the items that did not fit in the previous plan.
1743   - It is possible that the item could fit but the pickandplace function
1744     ↪ call was incorrect (e.g. misspelled item or location)
1745 - "Objects" is a list of items that need to be placed in the fridge.
1746 - "Locations" is a list of locations in the fridge.
1747 - "Initial State" is a dictionary whose keys are the locations in the
1748   ↪ fridge and the values are a list of items currently in that location.
1749 - "Preference" is a description of the human's preferences for where
1750   ↪ they like things in a fridge.
1751   - The preference should always be satisfied, even when attempting to
1752     ↪ place items that did not fit in the previous plan.
1753
1754 You must respond in the following format:
1755 # Reflect: ...
1756 # Reasoning: ...
1757 pickandplace(item1, location1)
1758 pickandplace(item2, location2)
1759 ...
1760
1761 where
1762 - "Reflect" should contain reasoning about the previous plan if it was
1763   ↪ geometrically infeasible.
1764   - Reflect on what went wrong and how you plan to fix it.
1765   - The plan must abide by the human's preference.
1766 - "Reasoning" should contain the reasoning for your plan.
1767   - Reason about how best to place the items in the fridge based on the
1768     ↪ human's preference.
1769   - If the reflection involves repositioning items, ensure that the
1770     ↪ human's preference is still satisfied.
1771 - "pickandplace(item, location)" is a function call that places the item
1772   ↪ in the location in the fridge.
1773   - This is your plan of action.
1774
1775 Each time you are prompted to generate a new plan, the fridge is reset to
1776 ↪ its initial state. Use this
1777 to your advantage to come up with a better plan. It is absolutely
1778 ↪ necessary to satisfy the human's
1779 preference; geometric infeasibility only suggests that objects should be
1780 ↪ placed in different locations
1781 that still satisfy the preference.
```

```

1782 ----- Example Input -----
1783 Objects: ["milk", "cheese", "apple", "orange"]
1784 Locations: ["top shelf", "left side of top shelf", "right side of top
1785 ↪ shelf", "middle shelf", "left side of middle shelf", "right side of
1786 ↪ middle shelf", "bottom shelf", "left side of bottom shelf", "right
1787 ↪ side of bottom shelf"]
1788 Initial State: {}
1789 Preference: "I like putting dairy on the top shelf and fruits on the
1790 ↪ right side of middle shelf."
1791 ----- Example Response -----
1792 # Reasoning: Milk and cheese are dairy product, which based on the
1793 ↪ preference needs to be on the top shelf. Apples and oranges are
1794 ↪ fruits, which needs to be on the right side of middle shelf.
1795 pickandplace("milk", "top shelf")
1796 pickandplace("cheese", "top shelf")
1797 pickandplace("apple", "right side of middle shelf")
1798 pickandplace("orange", "right side of middle shelf")
1799 ----- Example Input -----
1800 Objects: ["milk", "cheese", "apple", "orange"]
1801 Locations: ["top shelf", "left side of top shelf", "right side of top
1802 ↪ shelf", "middle shelf", "left side of middle shelf", "right side of
1803 ↪ middle shelf", "bottom shelf", "left side of bottom shelf", "right
1804 ↪ side of bottom shelf"]
1805 Initial State: {"left side of middle shelf": ["peach", "cherries"]}
1806 Preference: "Fruits must be placed next to other exisiting fruits
1807 ↪ regardless of which shelf they are on. Dairy products on the be the
1808 ↪ right side of bottom shelf. "
1809 ----- Example Response -----
1810 # Reasoning: Apples and oranges are fruits, which needs to be placed
1811 ↪ next to other exisiting fruits. Existing fruits in the fridge are "
1812 ↪ peach" and "cherries", which are at the left side of middle shelf, so
1813 ↪ apple and oranges should be placed there as well. Milk and cheese
1814 ↪ are diary product, which needs to be placed on the right side of
1815 ↪ bottom shelf.
1816 pickandplace("apple", "left side of middle shelf")
1817 pickandplace("orange", "left side of middle shelf")
1818 pickandplace("milk", "right side of bottom shelf")
1819 pickandplace("cheese", "right side of bottom shelf")
1820 ----- Example Input -----
1821 Objects: ["milk", "cheese", "apple", "orange"]
1822 Locations: ["top shelf", "left side of top shelf", "right side of top
1823 ↪ shelf", "middle shelf", "left side of middle shelf", "right side of
1824 ↪ middle shelf", "bottom shelf", "left side of bottom shelf", "right
1825 ↪ side of bottom shelf"]
1826 Initial State: {"right side of top shelf": ["cabbage", "corn"]}
1827 Preference: "Fruits must be placed next to other exisiting vegetables
1828 ↪ regardless of which shelf they are on. Dairy products on the be the
1829 ↪ right side of bottom shelf. "
1830 ----- Example Response -----
1831 # Reasoning: Apples and oranges are fruits, which needs to be placed
1832 ↪ next to other exisiting vegetables. Existing vegetables in the fridge
1833 ↪ are "cabbage" and "corn", which are at the right side of top shelf,
1834 ↪ so apple and oranges should be placed there as well. Milk and cheese
1835 ↪ are diary product, which needs to be placed on the right side of
1836 ↪ bottom shelf.
1837 pickandplace("apple", "right side of top shelf")
1838 pickandplace("orange", "right side of top shelf")
1839 pickandplace("milk", "right side of bottom shelf")
1840 pickandplace("cheese", "right side of bottom shelf")

```



```

1841 ----- Example Input -----
1842 Objects: ["milk", "cheese", "apple", "orange"]
1843 Locations: ["top shelf", "left side of top shelf", "right side of top
1844 ↪ shelf", "middle shelf", "left side of middle shelf", "right side of
1845 ↪ middle shelf", "bottom shelf", "left side of bottom shelf", "right
1846 ↪ side of bottom shelf"]
1847 Initial State: {"left side of middle shelf": ["peach", "cherries"]}
1848 Preference: "Most dairy product should be placed on the left side of top
1849 ↪ shelf, but cheese product should be placed on the right side of
1850 ↪ middle shelf. Fruits should be placed on the left side of middle
1851 ↪ shelf."
1852 ----- Example Response -----
1853 # Reasoning: Cheese is a cheese product, so it needs to be placed at
1854 ↪ right side of middle shelf. Milk is a dairy product, so it should be
1855 ↪ placed on the left side of top shelf. Apples and oranges are fruits,
1856 ↪ which needs to be on the left side of middle shelf.
1857 pickandplace("cheese", "right side of middle shelf")
1858 pickandplace("milk", "left side of top shelf")
1859 pickandplace("apple", "left side of middle shelf")
1860 pickandplace("orange", "left side of middle shelf")
1861 ----- Example Input -----
1862 Objects: ["milk", "cheese", "apple", "orange"]
1863 Locations: ["top shelf", "left side of top shelf", "right side of top
1864 ↪ shelf", "middle shelf", "left side of middle shelf", "right side of
1865 ↪ middle shelf", "bottom shelf", "left side of bottom shelf", "right
1866 ↪ side of bottom shelf"]
1867 Initial State: {"right side of bottom shelf": ["yogurt", "butter"]}
1868 Preference: "If the right side of bottom shelf has less than 3 items,
1869 ↪ dairy products can be placed there. Else, you must place them at the
1870 ↪ left side of top shelf. Fruits should be placed on the left side of
1871 ↪ middle shelf."
1872 ----- Example Response -----
1873 # Reasoning: Right side of bottom shelf can fit one more dairy product,
1874 ↪ so I will put milk there. Since now there are 3 items at the right
1875 ↪ side of bottom shelf, I must put cheese on the left side of top shelf
1876 ↪ . Apples and oranges are fruits, which needs to be on the left side
1877 ↪ of middle shelf.
1878 pickandplace("milk", "right side of bottom shelf")
1879 pickandplace("cheese", "left side of top shelf")
1880 pickandplace("apple", "left side of middle shelf")
1881 pickandplace("orange", "left side of middle shelf")
1882 ----- Example Input -----
1883 Objects: ["milk", "cheese", "apple", "melon"]
1884 Locations: ["top shelf", "middle shelf", "bottom shelf"]
1885 Initial State: {
1886     "top shelf": ["yogurt", "butter"],
1887     "middle shelf": ["watermelon", "pizza box"]
1888 }
1889 Preference: "I don't want any of the fridge shelves to be too crowded."
1890 ----- Example Response -----
1891 pickandplace("milk", "top shelf")
1892 pickandplace("cheese", "top shelf")
1893 pickandplace("apple", "middle shelf")
1894 pickandplace("melon", "middle shelf")
1895 ----- Example Input -----
1896 Feedback: The previous plan was geometrically infeasible. The items that
1897 ↪ did not fit were ["melon", "apple"].
1898 Objects: ["milk", "cheese", "apple", "melon"]
1899 Locations: ["top shelf", "middle shelf", "bottom shelf"]

```

```

1900     Initial State: {
1901         "top shelf": ["yogurt", "butter"],
1902         "middle shelf": ["watermelon", "pizza box"]
1903     }
1904     Preference: "I don't want any of the fridge shelves to be too crowded."
1905     ----- Example Response -----
1906     # Reflect: The melon and apple did not fit in the previous plan. This
1907     ↪ must mean that the middle shelf is too crowded.
1908     # Reasoning: Instead of placing the apple and melon on the middle shelf,
1909     ↪ they can be placed on the bottom shelf which is empty.
1910     pickandplace("milk", "top shelf")
1911     pickandplace("cheese", "top shelf")
1912     pickandplace("apple", "bottom shelf")
1913     pickandplace("melon", "bottom shelf")

```

### 1915 9.3 Baseline Prompts

#### 1916 9.3.1 Cand+LLM-Q/A Prompts

1917 Cand+LLM-Q/A use the same prompts as APRICOT to generate candidate preferences (Sec 9.1.1) and  
1918 plans (Sec 9.2). Below is the prompt that it used to determine whether to terminate, what is the best  
1919 question to ask if not terminating, and what is the best preference out of the candidate preference  
1920 list if terminating. We use gpt-4o.

```

1921     ----- System Message -----
1922     You are a helpful, thoughtful assistant whose task is to select the user's
1923     ↪ preference from the demonstrations that the user provides and asking
1924     ↪ users for yes/no clarification question.
1925     ----- Instruction -----
1926     # Input
1927     You are giving a list of preferences, a new initial condition of the
1928     ↪ fridge that your generated preference will get used at, and chat log
1929     ↪ of what you have asked so far.
1930
1931     ## Preferences
1932     You are given a list of preferences that show how the user wants their
1933     ↪ fridge organized.
1934     - A preference is a short paragraph that specifies requirements for each
1935     ↪ category of grocery items.
1936     - There will be at least one requirement for each category. The type of
1937     ↪ requirement for each category can be different.
1938
1939     ## New initial state of the fridge
1940     This is the new initial state that your preference will get used to
1941     ↪ determine how a new set of objects will get placed into the fridge and
1942     ↪ satisfy the user's preference.
1943
1944     ## Objects to put away
1945     This is the list of object that will get put away in this new initial
1946     ↪ state of the fridge.
1947
1948     ## Chat log
1949     This is a json that contains a history of your thought and your
1950     ↪ conversation with the user.
1951     - When the "role" is "thought", this is your internal reasoning about
1952     ↪ what you were going to do. The user never sees your thought.
1953     - When the "role" is "assistant", this is the yes/no clarification
1954     ↪ question that you ask the user about.
1955
1956

```

```

1957 - When the "role" is "user", this is the yes/no answer that the user
1958 ↪ provided to your yes/no clarification question.
1959
1960 ## Can you still ask question?
1961 There is a maximum number of questions that you can ask. This field will
1962 ↪ tell you if you can still ask questions or if you must terminate and
1963 ↪ generate your best guess at the user's preference.
1964
1965 # Goal
1966 Your goal is to select the best of the given preference in how the
1967 ↪ grocery items should be placed into the fridge. You can do this
1968 ↪ inference by analyzing the given preferences and asking users yes/no
1969 ↪ clarification questions. Doing this process, you can choose between 2
1970 ↪ actions:
1971     (1) Ask the user a yes/no clarification question
1972     (2) Select the index of the best preference from the list and terminate
1973
1974 # Instructions and useful information
1975 ## Set up of the fridge
1976 The fridge can be segmented into 3 shelves (top shelf, middle shelf,
1977 ↪ bottom shelf); each shelf has 2 sides (left side, right side):
1978 - left side of the top shelf, right side of the top shelf
1979 - left side of the middle shelf, right side of the middle shelf
1980 - left side of the bottom shelf, right side of the bottom shelf
1981
1982 ## Concrete steps that you must follow
1983 To successfully determine the user's preference, you must do the
1984 ↪ following steps.
1985 ### Step 1: Reasoning about what actions to do
1986 You can choose between 2 actions:
1987     (1) Ask the user a yes/no clarification question
1988     (2) Select the best preference, output its index, and terminate
1989
1990 You must follow these rules when you are deciding what to do:
1991 - If you are still uncertain about the preference, you should ask an
1992 ↪ informative yes/no clarification question.
1993 - If you are 100% confident about the preference, you can select which
1994 ↪ you think the right preference is and terminate.
1995
1996 ### Step 2: Take the action that you decided to do
1997 You must reply in a valid json format:
1998 ```json
1999 {
2000     "terminate? (yes/no)": "<you must only reply yes or no>",
2001     "reasoning": "<you must write down the reasoning behind either why you
2002 ↪ generated a specific question (if you choose the first action) or why
2003 ↪ you think the user's preference is what you are generating (if you
2004 ↪ choose the second action)>",
2005     "question": "<you must only fill out this string if you chose action (1)
2006 ↪ ask yes/no clarification question. Otherwise, you must leave this as
2007 ↪ an empty string.>",
2008     "index_of_best_preference": "<you must only fill out this if you chose
2009 ↪ action (2) select index of best preference. Otherwise, you must leave
2010 ↪ this as null.>"
2011 }
2012 ```
2013
2014 ##### If you chose (1) Ask the user a yes/no clarification question in
2015 ↪ Step 3

```

```

2016 You must follow these rules when coming up with questions to ask:
2017 - You must ask simple yes or no questions.
2018 - Your questions cannot ask if the user prefers A or B. However, you can
2019 ↪ ask if the user prefers A rather than B.
2020 - Your questions must not be about overall objects or items. You must
2021 ↪ never ask questions like "Do you prefer to items to be ...?" or "Do
2022 ↪ you prefer items of the same category ...?"
2023 - Your question must be about specific categories or type of objects. The
2024 ↪ categories are for example: ["fruits", "vegetables", "dairy product",
2025 ↪ "condiments", "juice and drinks"]
2026 - Your output must have "index_of_best_preference": null
2027
2028 Remember that you must reply in a valid json format:
2029 ```json
2030 {
2031   "terminate? (yes/no)": "no",
2032   "reasoning": "<you must write down the reasoning behind why you
2033   ↪ generated a specific question (because you choose the first action)
2034   ↪ >",
2035   "question": "<you must fill out this string because you chose action (1)
2036   ↪ ask yes/no clarification question.>",
2037   "index_of_best_preference": null
2038 }
2039 ```
2040
2041 ##### If you chose (2) Select the best preference index
2042 You must follow these rules.
2043 - You must now terminate so you should answer "yes" under the key "
2044 ↪ terminate? (yes/no)"
2045 - You must output the index of the best preference, with 0 referring to
2046 ↪ the first listed preference
2047
2048 Remember that you must reply in a valid json format:
2049 ```json
2050 {
2051   "terminate? (yes/no)": "yes",
2052   "reasoning": "<you must write down the reasoning behind why you think
2053   ↪ the user's preference is what you are selecting (because you choose
2054   ↪ the second action)>",
2055   "question": "",
2056   "index_of_best_preference": <you must output a 0-indexed integer
2057   ↪ corresponding to the preference you picked from the given preference
2058   ↪ list because you chose action (2) select best preference.>
2059 }
2060 ```

```

### 2062 9.3.2 LLM-Q/A Prompts

2063 LLM-Q/A only uses the prompt below to determine whether to terminate, determine what is the  
2064 best question to ask if not terminating, and generate the preference based on demonstrations and its  
2065 queries with the user if terminating. We use gpt-4o.

```

2066 ----- System Message -----
2067 You are a helpful, thoughtful assistant whose task is to select the user's
2068 ↪ preference from the demonstrations that the user provides and asking
2069 ↪ users for yes/no clarification question.
2070 ----- Instruction -----
2071 # Input

```

```

2073 You are giving a list of preferences, a new initial condition of the
2074 ↪ fridge that your generated preference will get used at, and chat log
2075 ↪ of what you have asked so far.
2076
2077 ## Preferences
2078 You are given a list of preferences that show how the user wants their
2079 ↪ fridge organized.
2080 - A preference is a short paragraph that specifies requirements for each
2081 ↪ category of grocery items.
2082 - There will be at least one requirement for each category. The type of
2083 ↪ requirement for each category can be different.
2084
2085
2086 ## New initial state of the fridge
2087 This is the new initial state that your preference will get used to
2088 ↪ determine how a new set of objects will get placed into the fridge and
2089 ↪ satisfy the user's preference.
2090
2091 ## Objects to put away
2092 This is the list of object that will get put away in this new initial
2093 ↪ state of the fridge.
2094
2095 ## Chat log
2096 This is a json that contains a history of your thought and your
2097 ↪ conversation with the user.
2098 - When the "role" is "thought", this is your internal reasoning about
2099 ↪ what you were going to do. The user never sees your thought.
2100 - When the "role" is "assistant", this is the yes/no clarification
2101 ↪ question that you ask the user about.
2102 - When the "role" is "user", this is the yes/no answer that the user
2103 ↪ provided to your yes/no clarification question.
2104
2105 ## Can you still ask question?
2106 There is a maximum number of questions that you can ask. This field will
2107 ↪ tell you if you can still ask questions or if you must terminate and
2108 ↪ generate your best guess at the user's preference.
2109
2110 # Goal
2111 Your goal is to select the best of the given preference in how the
2112 ↪ grocery items should be placed into the fridge. You can do this
2113 ↪ inference by analyzing the given preferences and asking users yes/no
2114 ↪ clarification questions. Doing this process, you can choose between 2
2115 ↪ actions:
2116 (1) Ask the user a yes/no clarification question
2117 (2) Select the index of the best preference from the list and terminate
2118
2119 # Instructions and useful information
2120 ## Set up of the fridge
2121 The fridge can be segmented into 3 shelves (top shelf, middle shelf,
2122 ↪ bottom shelf); each shelf has 2 sides (left side, right side):
2123 - left side of the top shelf, right side of the top shelf
2124 - left side of the middle shelf, right side of the middle shelf
2125 - left side of the bottom shelf, right side of the bottom shelf
2126
2127 ## Concrete steps that you must follow
2128 To successfully determine the user's preference, you must do the
2129 ↪ following steps.
2130 ### Step 1: Reasoning about what actions to do
2131 You can choose between 2 actions:

```

```

2132 (1) Ask the user a yes/no clarification question
2133 (2) Select the best preference, output its index, and terminate
2134
2135 You must follow these rules when you are deciding what to do:
2136 - If you are still uncertain about the preference, you should ask an
2137 ↪ informative yes/no clarification question.
2138 - If you are 100% confident about the preference, you can select which
2139 ↪ you think the right preference is and terminate.
2140
2141 ### Step 2: Take the action that you decided to do
2142 You must reply in a valid json format:
2143 ```json
2144 {
2145   "terminate? (yes/no)": "<you must only reply yes or no>",
2146   "reasoning": "<you must write down the reasoning behind either why you
2147 ↪ generated a specific question (if you choose the first action) or why
2148 ↪ you think the user's preference is what you are generating (if you
2149 ↪ choose the second action)>",
2150   "question": "<you must only fill out this string if you chose action (1)
2151 ↪ ask yes/no clarification question. Otherwise, you must leave this as
2152 ↪ an empty string.>",
2153   "index_of_best_preference": "<you must only fill out this if you chose
2154 ↪ action (2) select index of best preference. Otherwise, you must leave
2155 ↪ this as null.>"
2156 }
2157 ```
2158
2159 #### If you chose (1) Ask the user a yes/no clarification question in
2160 ↪ Step 3
2161 You must follow these rules when coming up with questions to ask:
2162 - You must ask simple yes or no questions.
2163 - Your questions cannot ask if the user prefers A or B. However, you can
2164 ↪ ask if the user prefers A rather than B.
2165 - Your questions must not be about overall objects or items. You must
2166 ↪ never ask questions like "Do you prefer to items to be ...?" or "Do
2167 ↪ you prefer items of the same category ...?"
2168 - Your question must be about specific categories or type of objects. The
2169 ↪ categories are for example: ["fruits", "vegetables", "dairy product",
2170 ↪ "condiments", "juice and drinks"]
2171 - Your output must have "index_of_best_preference": null
2172
2173 Remember that you must reply in a valid json format:
2174 ```json
2175 {
2176   "terminate? (yes/no)": "no",
2177   "reasoning": "<you must write down the reasoning behind why you
2178 ↪ generated a specific question (because you choose the first action)
2179 ↪ >",
2180   "question": "<you must fill out this string because you chose action (1)
2181 ↪ ask yes/no clarification question.>",
2182   "index_of_best_preference": null
2183 }
2184 ```
2185
2186 #### If you chose (2) Select the best preference index
2187 You must follow these rules.
2188 - You must now terminate so you should answer "yes" under the key "
2189 ↪ terminate? (yes/no)"

```

```

2190 - You must output the index of the best preference, with 0 referring to
2191 ↪ the first listed preference
2192
2193 Remember that you must reply in a valid json format:
2194 ```json
2195 {
2196   "terminate? (yes/no)": "yes",
2197   "reasoning": "<you must write down the reasoning behind why you think
2198   ↪ the user's preference is what you are selecting (because you choose
2199   ↪ the second action)>",
2200   "question": "",
2201   "index_of_best_preference": <you must output a 0-indexed integer
2202   ↪ corresponding to the preference you picked from the given preference
2203   ↪ list because you chose action (2) select best preference.>
2204 }
2205 ```
2206

```

### 2207 9.3.3 Non-Interactive Prompts

2208 Non-Interactive uses the prompt below to generate one preference from demonstrations, and it  
 2209 uses the same prompt as APRICOT to generate the task plan. We use gpt-4 because this is a complex  
 2210 task to reason about all the demonstrations.

```

2211 ----- System Message -----
2212 You are an assistant who sees someone demonstrating how the fridge is
2213 ↪ organized and summarizes that person's preference.
2214 ----- Instruction -----
2215
2216 # Input
2217 You are given 2 demonstrations that show the before and after when a set
2218 ↪ of objects gets put into the fridge. For each demonstration:
2219 - "Objects that got put away" describes the objects that the user will
2220 ↪ demonstrate how they would like to put in the fridge.
2221 - "Initial state of the fridge" describes the objects that are initially
2222 ↪ in the fridge before the user starts the demonstration.
2223 - "Final state of the fridge" describes what the fridge looks like after
2224 ↪ the demonstration. All the objects in "Objects that got put away"
2225 ↪ should be in the fridge now.
2226
2227 # Goal
2228 Your goal is to generate one preferences that are consistent with the
2229 ↪ demonstrations and explain what the user want.
2230
2231 # Instructions and useful information
2232 ## Specific locations in the fridge
2233 The fridge can be segmented into 3 shelves (top shelf, middle shelf,
2234 ↪ bottom shelf); each shelf has 2 sides (left side, right side). The
2235 ↪ fridge has 6 specific locations:
2236 - left side of the top shelf, right side of the top shelf
2237 - left side of the middle shelf, right side of the middle shelf
2238 - left side of the bottom shelf, right side of the bottom shelf
2239
2240 ## Details about the preferences that you need to output
2241 A preference is a short paragraph that specifies requirements for each
2242 ↪ category of grocery items. There must be at least one requirement for
2243 ↪ each category. The type of requirement for each category can be
2244 ↪ different. The categories are: "Fruits", "Vegetables", "Juice-and-soft
2245 ↪ -drinks", "Dairy-Products", and "Condiments".
2246
2247 The requirement needs to be one of the following:

```

```

2248 - **Type-1. Specific Locations.** These represent that the object must
2249 ↪ place at this specific location. The options are:
2250   - "left side of top shelf"
2251   - "right side of top shelf"
2252   - "left side of middle shelf"
2253   - "right side of middle shelf"
2254   - "left side of bottom shelf"
2255   - "right side of bottom shelf".
2256 - **Type-2. General Locations.** These are vaguer locations that contain
2257 ↪ multiple specific locations. The options are:
2258   - "left side of fridge"
2259   - "right side of fridge"
2260   - "top shelf"
2261   - "middle shelf"
2262   - "bottom shelf"
2263 - **Type-3. Relative Positions.** The options are:
2264   - "<category> must be placed together next to existing <category>"
2265   ↪ regardless of which shelf they are on."
2266   - "<category> must be placed on the same shelf next to <another category"
2267   ↪ of objects>, and which specific shelf does not matter."
2268
2269 In addition to giving specific requirements for each category of grocery
2270 ↪ items, sometimes you may choose to add additional requirements. The
2271 ↪ options are:
2272 - **Type-4. Exception For Attribute**
2273   - "<category> needs to be placed at <specific location 1>, but <"
2274   ↪ attribute of category> needs to be placed at <specific location 2>."
2275   ↪ An attribute includes a subcategory of the object, the size/weight of
2276   ↪ the object, a specific feature of the object, etc.
2277 - **Type-5. Conditional On Space**
2278   - "If there are less than <N> objects at <primary specific location>,"
2279   ↪ I want <category> to be placed at <primary specific location>. Else
2280   ↪ , I want <category> to be placed at <second choice specific location"
2281   ↪ >."
2282
2283 Now, you need to generate 1 preference as a jsons.
2284 - Each preference must be in natural language.
2285 - Each preference must contain at least one placement requirement for
2286 ↪ each category of objects: "fruits", "vegetables", "drinks", "dairy
2287 ↪ product", and "condiments".
2288
2289 You must refer to the demonstrations and output the preferences in this
2290 ↪ format
2291   `` `json
2292   {
2293     "reasoning": "<You must explain your thought process behind generating"
2294     ↪ this preference.>",
2295     "preference": "<You must write the preference in natural language here>"
2296   }
2297   `` `
2298

```

2299