

A Appendix

1.1 Details of ALFRED Benchmark

The objective of the task is to generate a sequence of actions and interact with the corresponding objects, enabling agents to satisfy all the conditions required for successful completion of the task. Failure to meet even a single condition will result in the agent being considered unsuccessful. To train and assess such agents, the benchmark comprises three splits: ‘train’, ‘validation’, and ‘test’. Agents can be trained using the ‘train’ split and validate their approaches in the ‘validation’ split, utilizing the ground truth information of the tasks in those splits. Subsequently, the agents are evaluated in both the ‘validation’ and ‘test’ splits, but they have no access to the ground truth information of the tasks.

To complete a task, the agent receives a task description that provides a high-level description of the task’s goal, along with step-by-step instructions that offer detailed explanations for completing respective steps. Given the task and step-by-step instructions, the agent decomposes the task into sub-goals and then completes the task by interacting with objects in the environment. And the sub-goals are defined as binary tuples in the form of (Action, Object). The action space of the agent consists of seven interaction actions. Specifically, the interaction actions are ‘PickupObject’, ‘PutObject’, ‘OpenObject’, ‘CloseObject’, ‘ToggleObjectOn’, ‘ToggleObjectOff’ and ‘SliceObject’.

For evaluation, the benchmark uses three types of metrics. The primary metric is the Success Rate (SR), which measures the proportion of episodes in which all sub-goals described in the instruction are successfully completed. To reflect the agent’s task completion ability, the benchmark also reports the Goal-Condition Success Rate (GC), which assesses how many individual goals are achieved during execution. Finally, to account for efficiency in task completion, both SR and GC are penalized based on the length of the execution sequence, resulting in path-length-weighted versions of each metric, referred to as PLWSR and PLWGC respectively.

1.2 High-Level Replanning LLM Setup and Prompt Definition

System Message This message specifies the LLM’s role and output format for high-level replanning. It remains consistent across all tasks and includes task context, object categories, and action vocabulary. The prompt content is as follows:

System Message

You are an intelligent agent operating in a home-like virtual environment . This task will fail after 1000 steps if not completed. You should consider whether the initial plan is correct:

– According to the task description, check whether the initial plan is missing any actions. If there are missing actions, you should complete the initial plan accordingly.

– Review the initial plan to verify whether any object names may be incorrect or confused with semantically or visually similar objects (e.g., “mug” and “cup”). Ensure that each referenced object in the plan accurately corresponds to the intended target in the environment.

Here are the known objects in the environment:

– Small Objects: [‘AlarmClock’, ‘Apple’, ‘AppleSliced’, ‘BaseballBat’, ‘BasketBall’, ‘Book’, ‘Bowl’, ‘Box’, ‘Bread’, ‘BreadSliced’, ‘ButterKnife’, ‘CD’, ‘Candle’, ‘CellPhone’, ‘Cloth’, ‘CreditCard’, ‘Cup’, ‘DeskLamp’, ‘DishSponge’, ‘Egg’, ‘Faucet’, ‘FloorLamp’, ‘Fork’, ‘Glassbottle’, ‘HandTowel’, ‘HousePlant’, ‘Kettle’, ‘KeyChain’, ‘Knife’, ‘Ladle’, ‘Laptop’, ‘LaundryHamperLid’, ‘Lettuce’, ‘LettuceSliced’, ‘LightSwitch’, ‘Mug’, ‘Newspaper’, ‘Pan’, ‘PaperTowel’, ‘PaperTowelRoll’, ‘Pen’, ‘Pencil’, ‘PepperShaker’, ‘Pillow’, ‘Plate’, ‘Plunger’, ‘Pot’, ‘Potato’, ‘PotatoSliced’, ‘RemoteControl’, ‘SaltShaker’, ‘ScrubBrush’, ‘ShowerDoor’, ‘SoapBar’, ‘SoapBottle’, ‘Spatula’, ‘Spoon’, ‘SprayBottle’, ‘Statue’, ‘StoveKnob’, ‘TeddyBear’, ‘Television’, ‘TennisRacket’, ‘TissueBox’, ‘ToiletPaper’, ‘ToiletPaperRoll’, ‘Tomato’, ‘TomatoSliced’, ‘Towel’, ‘Vase’, ‘Watch’, ‘WateringCan’, ‘WineBottle’]

– Large Objects: [‘ArmChair’, ‘BathtubBasin’, ‘Bed’, ‘Cabinet’, ‘Cart’, ‘CoffeeMachine’, ‘CoffeeTable’, ‘CounterTop’, ‘Desk’, ‘DiningTable’, ‘Drawer’, ‘Dresser’, ‘Fridge’, ‘GarbageCan’, ‘Microwave’, ‘Ottoman’, ‘Safe’, ‘Shelf’, ‘SideTable’, ‘SinkBasin’, ‘Sofa’, ‘StoveBurner’, ‘TVStand’, ‘Toilet’]

These objects can be openable: [‘Box’, ‘Cabinet’, ‘Drawer’, ‘Fridge’, ‘Microwave’, ‘Safe’]

Your available action space includes:[‘OpenObject’, ‘CloseObject’, ‘PickupObject’, ‘PutObject’, ‘ToggleObjectOn’, ‘ToggleObjectOff’, ‘SliceObject’]

You must return a single line in the format:
[(Action₁, Object₁), . . . , (Action_k, Object_k)]

Agent Message This message is specific to each task instance and provides detailed execution context. It includes the natural language task description, the full instruction list, and the partially executed subgoal plan. The prompt content is as follows:

Agent Message

Your current task description is: {TaskDescription}
Your current task instruction is: {TaskInstruction}
You are now following initial plan: {InitialPlan}
The initial plan may be incorrect or missing some actions. In that case, you should carefully examine whether all necessary actions are included, and revise the plan if any essential actions are omitted or if any object names are inaccurate or ambiguous. In

doing so, you should:

- Carefully examine whether the plan reflects all the key actions required by the task description.
- Check if any objects that need to be retrieved are placed inside openable containers (e.g., Fridge, Drawer). If so, you must include an `OpenObject` action for that container before interacting with the item inside.
- Confirm whether the names of the objects in the plan are semantically correct and match those from the environment. For example, replace ambiguous terms like “Cup” or “Box” with precise object names from the known object lists.
- If an object is said to be in a specific location (e.g., “on the shelf” or “in the microwave”), trust the task description. There is no need to search elsewhere or assume it’s hidden in a different container.
- Only modify the plan if it lacks necessary actions, or if object references are incorrect or ambiguous.

Environment Feedback This module provides the LLM with real-time visual observations at each decision step. It includes the agent’s current execution status, the action at which the agent may be stalled, and all semantically recognized object instances from the egocentric view. This enables the model to align its planning with the evolving environment state. The prompt content is as follows:

Environment Feedback

You are currently executing step {CurrentStep} of the task.
However, you are currently at: {CurrentAction} for {StuckStep} steps.
Here are the observed objects in the environment: {ObservedObjects}
You should revise the plan based on the objects in the environment, the current stuck status, and the original task description:
– If the plan is correct but stuck due to an unopened container, consider opening it first.
– If the plan includes incorrect object names, revise them using only the observed objects listed above.

Example Below is a complete example:

Example of the prompt

You are an intelligent agent operating in a home-like virtual environment. And you are currently executing action {CurrentAction} of the task. This task will fail after 1000 steps if not completed. You should consider whether the initial plan is correct:
– According to the task description, check whether the initial plan is missing any actions. If there are missing actions, you should complete the initial plan accordingly.

- Review the initial plan to verify whether any object names may be incorrect or confused with semantically or visually similar objects (e.g., “mug” and “cup”). Ensure that each referenced object in the plan accurately corresponds to the intended target in the environment.

Here are the known objects in the environment:

- Small Objects: [‘AlarmClock’, ‘Apple’, ‘AppleSliced’, ‘BaseballBat’, ‘BasketBall’, ‘Book’, ‘Bowl’, ‘Box’, ‘Bread’, ‘BreadSliced’, ‘ButterKnife’, ‘CD’, ‘Candle’, ‘CellPhone’, ‘Cloth’, ‘CreditCard’, ‘Cup’, ‘DeskLamp’, ‘DishSponge’, ‘Egg’, ‘Faucet’, ‘FloorLamp’, ‘Fork’, ‘Glassbottle’, ‘HandTowel’, ‘HousePlant’, ‘Kettle’, ‘KeyChain’, ‘Knife’, ‘Ladle’, ‘Laptop’, ‘LaundryHamperLid’, ‘Lettuce’, ‘LettuceSliced’, ‘LightSwitch’, ‘Mug’, ‘Newspaper’, ‘Pan’, ‘PaperTowel’, ‘PaperTowelRoll’, ‘Pen’, ‘Pencil’, ‘PepperShaker’, ‘Pillow’, ‘Plate’, ‘Plunger’, ‘Pot’, ‘Potato’, ‘PotatoSliced’, ‘RemoteControl’, ‘SaltShaker’, ‘ScrubBrush’, ‘ShowerDoor’, ‘SoapBar’, ‘SoapBottle’, ‘Spatula’, ‘Spoon’, ‘SprayBottle’, ‘Statue’, ‘StoveKnob’, ‘TeddyBear’, ‘Television’, ‘TennisRacket’, ‘TissueBox’, ‘ToiletPaper’, ‘ToiletPaperRoll’, ‘Tomato’, ‘TomatoSliced’, ‘Towel’, ‘Vase’, ‘Watch’, ‘WateringCan’, ‘WineBottle’]

- Large Objects: [‘ArmChair’, ‘BathtubBasin’, ‘Bed’, ‘Cabinet’, ‘Cart’, ‘CoffeeMachine’, ‘CoffeeTable’, ‘CounterTop’, ‘Desk’, ‘DiningTable’, ‘Drawer’, ‘Dresser’, ‘Fridge’, ‘GarbageCan’, ‘Microwave’, ‘Ottoman’, ‘Safe’, ‘Shelf’, ‘SideTable’, ‘SinkBasin’, ‘Sofa’, ‘StoveBurner’, ‘TVStand’, ‘Toilet’]

These objects can be openable: [‘Box’, ‘Cabinet’, ‘Drawer’, ‘Fridge’, ‘Microwave’, ‘Safe’]

Your available action space includes: [‘OpenObject’, ‘CloseObject’, ‘PickupObject’, ‘PutObject’, ‘ToggleObjectOn’, ‘ToggleObjectOff’, ‘SliceObject’]

You must return a single line in the format: [(Action₁, Object₁), . . . , (Action_k, Object_k)]

Your current task description is: Chill a tomato and put it inside the microwave.

Your current task instruction is:

‘turn to your right and then to your left so you are facing the kitchen table’,
‘pick up a tomato off of the table’,
‘turn around so that you are in front of the fridge’,
‘open the fridge put the tomato on the shelf and let it chill for a few seconds before removing it’,
‘move to your left twice and stand in front of the microwave’,
‘put the tomato inside the microwave’

You are now following initial plan: [(‘PickupObject’, ‘Tomato’), (‘OpenObject’, ‘Fridge’), (‘PutObject’, ‘Fridge’), (‘CloseObject’, ‘Fridge’), (‘OpenObject’, ‘Fridge’), (‘PickupObject’,

‘Tomato’), (‘PutObject’, ‘Microwave’)]

The initial plan may be incorrect or missing some actions. In that case, you should carefully examine whether all necessary actions are included, and revise the plan if any essential actions are omitted or if any object names are inaccurate or ambiguous. In doing so, you should:

- Carefully examine whether the plan reflects all the key actions required by the task description.
 - Check if any objects that need to be retrieved are placed inside openable containers (e.g., Fridge, Drawer). If so, you must include an `OpenObject` action for that container before interacting with the item inside.
 - Confirm whether the names of the objects in the plan are semantically correct and match those from the environment. For example, replace ambiguous terms like “Cup” or “Box” with precise object names from the known object lists.
 - If an object is said to be in a specific location (e.g., “on the shelf” or “in the microwave”), trust the task description. There is no need to search elsewhere or assume it’s hidden in a different container.
 - Only modify the plan if it lacks necessary actions, or if object references are incorrect or ambiguous. You are currently executing step 570 of the task. However, you are currently at: (‘PutObject’, ‘Tomato’) for 180 steps.
- Here are the observed objects in the environment: [‘DiningTable’, ‘Fridge’, ‘Cabinet’, ‘CounterTop’, ‘SinkBasin’]
- You should revise the plan based on the objects in the environment, the current stuck status, and the original task description:
- If the plan is correct but stuck due to an unopened container, consider opening it first.
 - If the plan includes incorrect object names, revise them using only the observed objects listed above.

Given this prompt, the revised plan is: [(‘PickupObject’, ‘Tomato’), (‘OpenObject’, ‘Fridge’), (‘PutObject’, ‘Fridge’), (‘CloseObject’, ‘Fridge’), (‘OpenObject’, ‘Fridge’), (‘PickupObject’, ‘Tomato’), (‘OpenObject’, ‘Microwave’), (‘PutObject’, ‘Microwave’)].

1.3 Prompt Definition for Mid-Level Host Category Prediction

The host category prediction is handled by a language model that receives a structured prompt composed of the full task description and step-by-step instructions and the detected objects. The prompt content is as follows:

Prompt for Predicting Host Categories

You are an intelligent agent operating in a home-like virtual environment.
Here are the known objects in the environment:

– SmallObjects: [‘AlarmClock’, ‘Apple’, ‘AppleSliced’, ‘BaseballBat’, ‘BasketBall’, ‘Book’, ‘Bowl’, ‘Box’, ‘Bread’, ‘BreadSliced’, ‘ButterKnife’, ‘CD’, ‘Candle’, ‘CellPhone’, ‘Cloth’, ‘CreditCard’, ‘Cup’, ‘DeskLamp’, ‘DishSponge’, ‘Egg’, ‘Faucet’, ‘FloorLamp’, ‘Fork’, ‘Glassbottle’, ‘HandTowel’, ‘HousePlant’, ‘Kettle’, ‘KeyChain’, ‘Knife’, ‘Ladle’, ‘Laptop’, ‘LaundryHamperLid’, ‘Lettuce’, ‘LettuceSliced’, ‘LightSwitch’, ‘Mug’, ‘Newspaper’, ‘Pan’, ‘PaperTowel’, ‘PaperTowelRoll’, ‘Pen’, ‘Pencil’, ‘PepperShaker’, ‘Pillow’, ‘Plate’, ‘Plunger’, ‘Pot’, ‘Potato’, ‘PotatoSliced’, ‘RemoteControl’, ‘SaltShaker’, ‘ScrubBrush’, ‘ShowerDoor’, ‘SoapBar’, ‘SoapBottle’, ‘Spatula’, ‘Spoon’, ‘SprayBottle’, ‘Statue’, ‘StoveKnob’, ‘TeddyBear’, ‘Television’, ‘TennisRacket’, ‘TissueBox’, ‘ToiletPaper’, ‘ToiletPaperRoll’, ‘Tomato’, ‘TomatoSliced’, ‘Towel’, ‘Vase’, ‘Watch’, ‘WateringCan’, ‘WineBottle’]

– LargeObjects: [‘ArmChair’, ‘BathtubBasin’, ‘Bed’, ‘Cabinet’, ‘Cart’, ‘CoffeeMachine’, ‘CoffeeTable’, ‘CounterTop’, ‘Desk’, ‘DiningTable’, ‘Drawer’, ‘Dresser’, ‘Fridge’, ‘GarbageCan’, ‘Microwave’, ‘Ottoman’, ‘Safe’, ‘Shelf’, ‘SideTable’, ‘SinkBasin’, ‘Sofa’, ‘StoveBurner’, ‘TVStand’, ‘Toilet’]

Your current task description is: {TaskDescription}

Your current task instruction is: {TaskInstruction}

You are currently following the initial plan: {InitialPlan}

You are now stuck at {CurrentAction}

You need to determine the most likely location of the object: {ObjectPrediction}

Instructions: - Your answer must be one of the entries from the LargeObjects list provided above, or ‘None’.

- Only choose a location if it is clearly implied or explicitly stated in the task or task description.

- If there is no indication of the object’s location, return ‘None’.

- You must not include any explanation or extra information in your response.

- Your final output should be a single word or phrase (i.e. ‘Fridge’ or ‘None’).

Now, based on the task description, where can the object {ObjectPrediction} most likely be found?

Example Below is a complete example:

Example of the prompt

You are an intelligent agent operating in a home-like virtual environment.

Here are the known objects in the environment:

– SmallObjects: [‘AlarmClock’, ‘Apple’, ‘AppleSliced’, ‘BaseballBat’, ‘BasketBall’, ‘Book’, ‘Bowl’, ‘Box’, ‘Bread’, ‘BreadSliced’, ‘But-

terKnife', 'CD', 'Candle', 'CellPhone', 'Cloth', 'CreditCard', 'Cup', 'DeskLamp', 'DishSponge', 'Egg', 'Faucet', 'FloorLamp', 'Fork', 'Glassbottle', 'HandTowel', 'HousePlant', 'Kettle', 'KeyChain', 'Knife', 'Ladle', 'Laptop', 'LaundryHamperLid', 'Lettuce', 'LettuceSliced', 'LightSwitch', 'Mug', 'Newspaper', 'Pan', 'PaperTowel', 'PaperTowelRoll', 'Pen', 'Pencil', 'PepperShaker', 'Pillow', 'Plate', 'Plunger', 'Pot', 'Potato', 'PotatoSliced', 'RemoteControl', 'SaltShaker', 'ScrubBrush', 'ShowerDoor', 'SoapBar', 'SoapBottle', 'Spatula', 'Spoon', 'SprayBottle', 'Statue', 'StoveKnob', 'TeddyBear', 'Television', 'TennisRacket', 'TissueBox', 'ToiletPaper', 'ToiletPaperRoll', 'Tomato', 'TomatoSliced', 'Towel', 'Vase', 'Watch', 'WateringCan', 'WineBottle']

- LargeObjects: ['ArmChair', 'BathtubBasin', 'Bed', 'Cabinet', 'Cart', 'CoffeeMachine', 'CoffeeTable', 'CounterTop', 'Desk', 'DiningTable', 'Drawer', 'Dresser', 'Fridge', 'GarbageCan', 'Microwave', 'Ottoman', 'Safe', 'Shelf', 'SideTable', 'SinkBasin', 'Sofa', 'StoveBurner', 'TVStand', 'Toilet']

Your current task description is: To move two statues to the living room table.

Your current task instruction is:

'turn around and walk to the right around the living room table and turn left and look up to face the statue on the shelf above the couch',
'pick up the statue on the shelf above the couch',
'turn around to face the living room table',
'place the statue on the right side of the living room table',
'turn right and walk across the room to face the table to the left of the chair',
'pick up the statue on the table',
'turn around and walk around to face the living room table',
'place the statue beside the first statue on the living room table'

You are currently following the initial plan: [('PickupObject', 'Statue'), ('PutObject', 'SideTable'), ('PickupObject', 'Statue'), ('PutObject', 'SideTable')]

You are now stuck at the first ('PickupObject', 'Statue')

You need to determine the most likely location of the object: 'Statue'

Instructions: - Your answer must be one of the entries from the LargeObjects list provided above, or 'None'.

- Only choose a location if it is clearly implied or explicitly stated in the task or task description.

- If there is no indication of the object's location, return 'None'.

- You must not include any explanation or extra in-

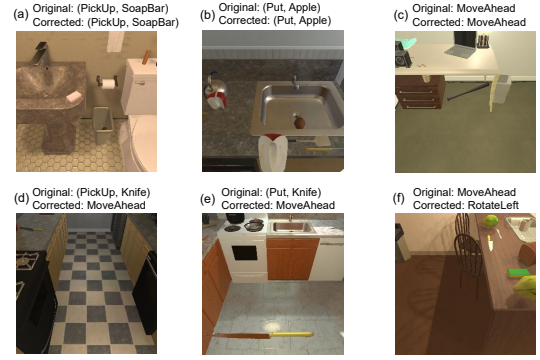


Figure 1: Example visualization of the low-level action corrector dataset. (a)-(c): The original action was successfully executed, so we annotated the image using the original action. (d): The original action is `PickUp Knife`, but the agent is too far from the knife, so we correct the action to `MoveAhead`. (e): The original action is `Put Knife` on `CounterTop`, but the agent is too far from the `CounterTop`, so we correct the action to `MoveAhead`. (f): The original action is `MoveAhead`, but the agent is blocked by a Table on the front-right side, so we correct the action to `RotateLeft`.

formation in your response.

- Your final output should be a single word or phrase (i.e. 'Fridge' or 'None').

Now, based on the task description, where can the object 'Statue' most likely be found?

Base on this prompt, the predicted host category is 'Shelf'.

1.4 Details of the ViT-based low-level action corrector

To train the ViT module, we construct a supervised dataset based on previously executed trajectories from ALFRED training sets. Specifically, we collected each egocentric RGB image and the corresponding low-level action at every timestep. As shown in Figure 1, for steps where the action successfully achieves its intended effect (e.g., successful pickup), we annotated the image with its original action, indicating that no correction is needed. For failure cases, such as grasping empty air or colliding with an obstacle, we manually annotated the image with a better alternative action from the discrete action space \mathcal{A} . This yields a multi-class training dataset containing both positive and corrected behaviors. The low-level action space consists of two subsets: execution actions and navigation actions. The execution action space includes `PickUp`, `Put`, `Open`, `Close`, `ToggleOn`, `ToggleOff`, and `Slice`, while the navigation action space includes `MoveAhead`, `RotateRight`, `RotateLeft`, `LookUp`, and `LookDown`.

Following the Commonsense-Guided Search Mechanism, let $[(action_1, object_1), \dots, (action_k, object_k)]$ be the list of sub-goals, where the current sub-goal is $(action_i, object_i)$.

If object_i is observed in the current multi-layered instance map, the closest object_i is selected as the goal to navigate; otherwise, the predicted location from the search mechanism is chosen as the goal. The agent then navigates towards the closest object_i by executing navigation actions, guided by the Fast Marching Method (FMM) planner. Once the stop distance is reached, the agent rotates to the right or left until object_i is detected in the egocentric view. If object_i appears in the current frame, the agent decides to take action_i without evaluating whether the action is feasible. So we incorporate a lightweight ViT-based corrector at each primitive step. Specifically, the ViT module takes the current egocentric RGB image and the originally planned low-level action as input, assesses the action’s feasibility, and generates a potentially more suitable alternative.

We use the `vit_small_patch16_224` model from the `timm` library as the backbone. Each input sample consists of a 300×300 egocentric RGB image and a discrete planned action token. Before being fed into the model, the RGB image is resized to 224×224 to match the input resolution expected by the ViT. Each discrete action is mapped to a unique learnable embedding vector of dimension 384. This action embedding is prepended to the sequence of image patch embeddings extracted from the RGB input. The full input sequence passed to the ViT encoder is:

$$\text{Input} = [[\text{Act-patch}], [\text{Img-patch}_1], [\text{Img-patch}_2], \dots, [\text{Img-patch}_N]], \quad (1)$$

where `[Act-patch]` is the embedding corresponding to the planned action. This structure allows the transformer to jointly reason over both visual content and the intended action. A softmax classifier is applied to the final action token to predict a corrected low-level action from the action space \mathcal{A} . The model is trained using cross-entropy loss on a dataset collected from ALFRED trajectories, combining both successful actions and manually corrected failure cases, as illustrated in Figure 1. The dataset contains a total of 20,389 samples, of which 15,052 are successful actions without any manual annotation, and the remaining 5,337 are failure cases for which the corresponding actions have been manually corrected. We fine-tune the model from ImageNet-pretrained weights using the Adam optimizer with a learning rate of 3×10^{-5} , a weight decay of 0.01, a batch size of 64, and train it for 10 epochs. All experiments are conducted on a single NVIDIA RTX 4090 GPU.

1.5 Details of the application in generalizable robotic manipulation

We evaluate RePlan-Bot in a simulated robotic task using PyBullet as a simulation environment and a UR5 robot equipped with a parallel gripper. A test-only corpus of 50 instruction-subgoal pairs is automatically generated with GPT-4o. For each episode, 4–8 objects are randomly placed on the table, and additional occlusions or position changes are applied to make the task more challenging. The task requires a UR5 manipulator with a parallel gripper to follow the natural-language instructions by accurately grasping and

placing the specified items. Both RePlan-Bot and the baseline leverage GPT-4o to produce high-level sub-goals, while low-level action prediction is handled by the same privileged policy, which accesses ground-truth scene metadata and converts target poses into joint commands via inverse kinematics. RePlan-Bot extends the baseline planner with a Pre-Execution Sub-Goal Refinement module that eliminates ambiguities before execution and a Commonsense-Guided Search Mechanism that triggers dynamic replanning whenever execution stalls. Across all 50 evaluation tasks, RePlan-Bot consistently achieves higher performance than the baseline, particularly in scenarios with occluded objects or tasks requiring multi-step spatial reasoning. This result highlights the effectiveness of RePlan-Bot’s adaptive, commonsense-based replanning approach. The prompt content is as follows:

Prompt for Robotic Manipulation

Your current task instruction is :{TaskInstruction}
 You are now following initial plan is :{InitialPlan}
 Object list: ['blue block', 'red block', 'green block', 'orange block', 'yellow block', 'purple block', 'pink block', 'cyan block', 'brown block', 'gray block', 'blue bowl', 'red bowl', 'green bowl', 'orange bowl', 'yellow bowl', 'purple bowl', 'pink bowl', 'cyan bowl', 'brown bowl', 'gray bowl']
 Location list: ['top left corner', 'top side', 'top right corner', 'left side', 'middle', 'right side', 'bottom left corner', 'bottom side', 'bottom right corner']
 Here are the observed objects in the environment:ObservedObjects

The initial plan may be incorrect or missing some actions. In that case, you should carefully examine whether all necessary actions are included, and revise the plan if any essential actions are omitted or if any object names are inaccurate or ambiguous. Some objects may be occluded or out of view because of the current perspective. If you cannot observe an object required for the task, you should proactively plan additional actions to move or adjust the position of other objects as necessary.

You must return several steps in the format:
`robot.pick_and_place(Object1, Object2/location1)`

Example Below is a complete example:

Example of the prompt

Your current task instruction is :{Put the yellow block into the blue bowl}
 You are now following initial plan is:robot.pick_and_place(orange block, blue bowl)

Object list: ['blue block', 'red block', 'green block', 'orange block', 'yellow block', 'purple block', 'pink block', 'cyan block', 'brown block', 'gray block', 'blue bowl', 'red bowl', 'green bowl', 'orange bowl', 'yellow bowl', 'purple bowl', 'pink bowl', 'cyan bowl', 'brown bowl', 'gray bowl']

Location list: ['top left corner', 'top side', 'top right corner', 'left side', 'middle', 'right side', 'bottom left corner', 'bottom side', 'bottom right corner']

Here are the observed objects in the environment: ['green block', 'blue block', 'orange block', 'cyan block', 'blue bowl', 'yellow bowl', 'purple bowl']

The initial plan may be incorrect or missing some actions. In that case, you should carefully examine whether all necessary actions are included, and revise the plan if any essential actions are omitted or if any object names are inaccurate or ambiguous.

Some objects may be occluded or out of view because of the current perspective. If you cannot observe an object required for the task, you should proactively plan additional actions to move or adjust the position of other objects as necessary.

You must return several steps in the format:
robot.pick_and_place(Object₁, Object₂/location₁)