## Appendix I
## Related work

**Long-horizon Robot Planning.** Addressing long-horizon planning in robotics has been a persistent challenge. Rule-based methods [38], [39], such as Planning Domain Definition Language (PDDL) [40], attempted to solve task and motion planning sequentially, however, the planned task may not feasible when the domain knowledge is incomplete. The task and motion planning (TAMP) approach [41], addresses this by simultaneously determining symbolic actions and low-level motions. While these methods excel in *verifying* task and motion plans during planning, their *generalization* to new environments and tasks as well as their efficiency on replanning are constrained. In addressing multi-stage planning challenges, many works focus on learning task plans from input task specifications, leveraging reinforcement learning (RL) and imitation learning (IL) techniques. For example, Behavior-1K [42] employs RL to acquire semantics and physical manipulation skills, often benefiting from classical motion planners and simplifying assumptions. However, it's important to note that these learning-based techniques demand significant domain expertise for reward engineering and rely on large datasets for task learning [43]. While they adeptly *react* to environmental uncertainties by iteratively updating policies based on observations, their zero-shot generalization across multi-stage tasks remains a persistent challenge.

**Robot Control with Physically Grounded Language Models.** Recent advancements in LLMs have resulted in their adoption in robot planning, leveraging their natural language capabilities and common-sense reasoning for generating robot task and motion plans[6], [44]. Notably, LLMs have been applied to planning multi-stage tasks [45], [2], by utilizing LLMs to improve sample efficiency in reinforcement learning. ProgPrompt and Code-As-Policies are among the early approaches using code-writing LLMs to generate code for robot policies [45], [46]. Language models with a verifier have been used for generating long-horizon tasks in an iterative prompting technique with lower error rates [47]; however, there is no guarantee that the output task plan can be executed. To overcome that shortcoming, SayPlan used LLMs to reason over scene graphs and generate plans across large environments, using online replanning to ensure scene graph constraints were not violated [48]. Toward grounding the language belief with visual and motor control feedback, several works have equipped embodied agents with vision-language models for solving planning problems [49], [3], [14], [15]. Table I provides a detailed comparison of REPLAN with related works.

**Language to Reward Shaping.** In contrast to approaches that map natural task descriptions to robot actions and subsequently to rewards, an alternative approach seeks to directly infer rewards from natural language inputs, addressing the challenge of reward engineering [50]. This language-driven reward-shaping approach has demonstrated utility in various domains, including negotiation [22] and gaming [51], facilitating desired behavior learning through RL. [52] introduce a visuo-language model that generates robot motion policy reward on goal text and raw pixel observations, in a manner similar to [33], enabling zero-shot prompting for unseen scenarios. [10] employs an iterative prompting method using an LLM to link user task specifications and robot motion through reward functions. While excelling in motion generation with minimal data, their approach falls short in handling long-horizon multistage tasks and lacks real-time environment feedback, necessitating user intervention for adaptation. [9] extended the previous work for robot reward policy refinement by requiring substantial human involvement and Pythonic observation from the environment. Both of these methods struggle with open-ended problems and multi-stage tasks. To mitigate these limitations, our work autonomously performs long-horizon tasks and adapts to execution outcomes by leveraging motor control and raw visual feedback.

**Online Replanning Based on Sensory Feedback.** Recent works have explored online robotic adaption for recovering from errors during task execution. CAPE queries an LLM for a possible corrective action if a precondition is not met [53]. DROC introduces a framework that can provide a robot with low-level and high-level corrections, which are then stored in a knowledge base that can be retrieved later [13]. However, the corrections must be provided by humans. DoReMi uses an LLM to generate constraints that must be met during a given subtask that the robot is executing, and then queries a VLM to ensure those preconditions are met [12]. However, this framework has to rely on constraints that an LLM generates and only probes the VLM for a yes-or-no answer, which may miss unexpected obstacles, and does not allow for tasks which have ambiguous solutions. The framework also queries a VLM every $t$ seconds, which may increase latency depending on the VLM used. LLM-POP uses sensor feedback to obtain missing information from the environment about object states, and utilizes the new information to update its plan [16]. REFLECT uses audio and video sensors to hierarchically generate captions for object states, and then uses an LLM to reason upon possible failure reasons if a subtask is not accomplished [11]. However, this method requires predefining caption states and sensor labels. Furthermore, none of these methods allow for task completion where the task solution is ambiguous (except for CAPE and REFLECT) or where the task requires causal reasoning, and none output unconstrained robot rewards directly.

## Appendix II
## Environment Details

### A. Wooden cabinet scene

A room where there is a yellow cube placed on the floor beside a wooden cabinet. There is a red bar holding the handles of the wooden cabinet closed. The doors of the cabinet cannot be opened without removing the bar. We implemented three

**TABLE III:** Features in REPLAN benchmark tasks.

| **Feature** | *Cabinet-Open* | *Cabinet-Closed* | *Cabinet-Blocked* | *Cabinet-Locked* | *Cubes-Color* | *Cubes-Blocked* | *Kitchen-Explore* | *Composite-Explore* |
|---|---|---|---|---|---|---|---|---|
| Multi-step planning | × | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ |
| Visual feedback | × | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Causal reasoning | × | × | ✓ | ✓ | × | ✓ | ✓ | ✓ |
| Exploration | × | × | × | × | × | × | ✓ | ✓ |

tasks in this scene:

`CabinetOpen`*: Place the yellow cube in the wooden cabinet (easy mode).* – The robot must pick up the yellow cube and place it in the cabinet, which is open. This is a simple 1-step task that evaluates the Low-Level Planner for single motion planning.

`CabinetClosed`*: Place the yellow cube in the wooden cabinet (hard mode).* – This task is the same as the previous task, but now the wooden cabinet doors are closed. This task requires two steps: 1) opening the door and 2) placing the cube inside the cabinet. The High-Level Planner is assessed to plan a sequence of actions and pass them to the Low-Level Planner for motion generation.

`CabinetBlocked`*: Place the yellow cube inside the wooden cabinet (expert mode).* – The challenge with this task is that the robot must identify that it cannot open the wooden cabinet because there is a bar across the handles of the door. After removing the bar, the robot can open the cabinet door and finish the task. This task is challenging because it requires vision to identify that the door cannot be opened, followed by replanning to remove the item blocking the door.

### *B. Kitchen environment scene*

A kitchen that contains a cabinet with two doors, a microwave, a kettle, and a green apple.

`KitchenExplore`*: Find the green apple.* – A green apple was hidden in the microwave is not visible to the robot at the start of the scene. The robot must search for the apple in the kitchen. There is an additional challenge where the kettle is blocking the microwave door from being opened, and so to open the door, the robot must first remove the kettle. Same as `CabinetBlocked`, `Kitchen-Explore` also requires both vision and replanning to solve the task, but it has an additional challenge because the goal requires open-ended exploration (it is unclear where the apple is), which requires replanning at a high level.

### *C. Wooden cabinet and lever scene*

A room containing a wooden cabinet, a blue block, and a lever that controls whether the cabinet door is locked or unlocked.

`CabinetLocked`*: Remove the blue cube from the cabinet.* – Just as with tasks 1-3, this task requires the robot to open a cabinet. There is no physical obstruction preventing the cabinet from being opened; however, the cabinet is locked. The cabinet becomes unlocked once a lever close to the cabinet is pulled. Thus, after (unsuccessfully) trying to open the cabinet door, the robot must reason that it should pull the lever first and then open the door.

### *D. Coloured cubes scene*

A room containing a small red crate and two cubes (one is yellow and the other is red).

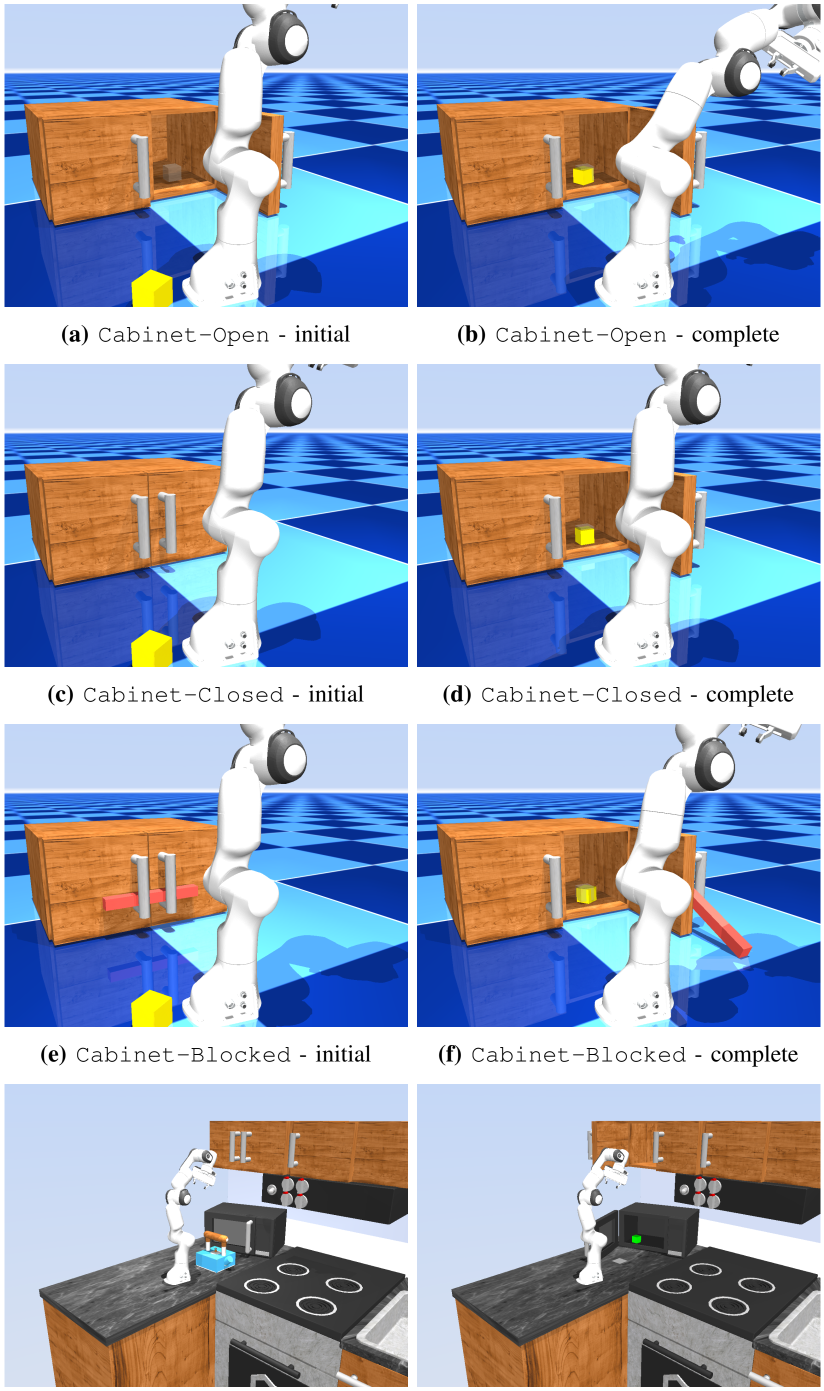
`CubesColor`*: Place the cube with the same colour as the crate on the crate.* – In this task, the robot has to identify the cube with the same colour as the crate and place it on the crate.

`CubesBlocked`*: Blocking cube.* – The robot is given the colour of a cube it must put on the crate. However, there is already a cube on the crate with a different colour and the crate can only hold one cube at a time. The robot must remove the cube that is already on the crate before placing the target one.

`CompositeExplore`*: Weight sensor* – The robot is asked to open a sliding cabinet. The cabinet is locked by a weight sensor, which can be activated by putting a weight on top. To do this, the robot needs first to explore the scene to find the weight.

The instructions we use for each task are listed below:

| *Environment* | *Instruction* |
|---|---|
| Cabinet-{Open, Closed, Blocked} | move the yellow_cube to target_position inside the wooden_cabinet |
| Kitchen-Explore | find the green_cube |
| Cabinet-Locked | find the blue_cube |
| TwoCube-Color | place the cube with the same color as the crate on the crate |
| TwoCube-Blocked | place the red cube on the crate |
| Composite-Explore | open the stone_cabinet. The weight sensor lock can be unlocked by putting the red_cube on it. |



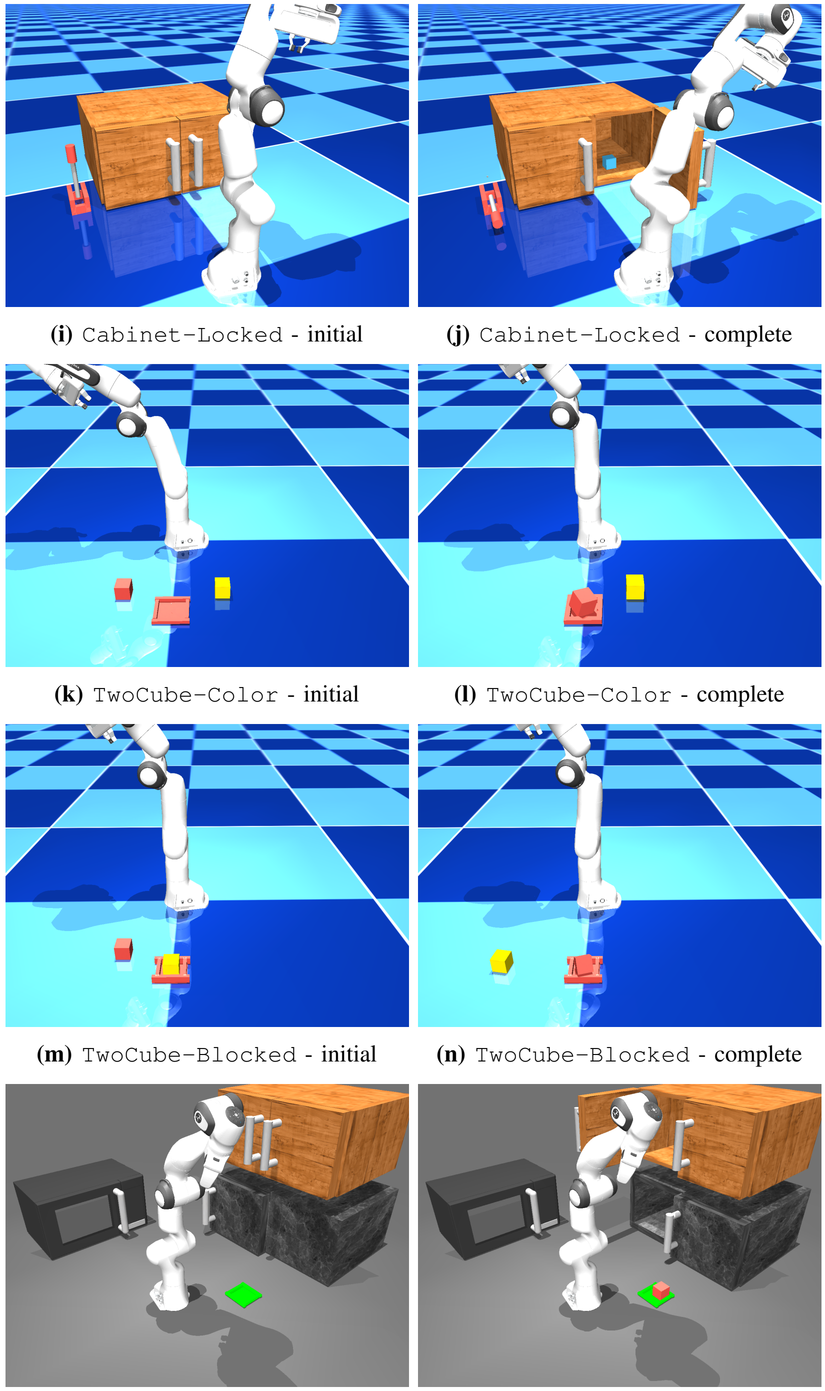
**(a)** `Cabinet-Open` - initial **(b)** `Cabinet-Open` - complete

**(c)** `Cabinet-Closed` - initial **(d)** `Cabinet-Closed` - complete

**(e)** `Cabinet-Blocked` - initial **(f)** `Cabinet-Blocked` - complete

**(g)** `Kitchen-Explore` - initial **(h)** `Kitchen-Explore` - complete

**Fig. II.1:** Initial and final scenes of the tested environments.



(i) `Cabinet-Locked` - initial (j) `Cabinet-Locked` - complete

(k) `TwoCube-Color` - initial (l) `TwoCube-Color` - complete

(m) `TwoCube-Blocked` - initial (n) `TwoCube-Blocked` - complete

(o) `Composite-Explore` - initial (p) `Composite-Explore` - complete

**Fig. II.1:** Initial and final scenes of the tested environments.

## Appendix III
## Prompts

In Figure III.0, we show a example rollout of RePLan solving the `cabinet-closed` task In Figure III.1, we also show a prompt roadmap. Furthermore, in Figures III.2-III.19 we show step-by-step all the prompts we use for the Planners, Perceiver, and Verifier. We show the prompts as the robot would receive them while excuting a task. The prompts are coloured according to the module it comes from – **LLM Planners: blue, VLM Perceiver: pink, Verifier: gray**.

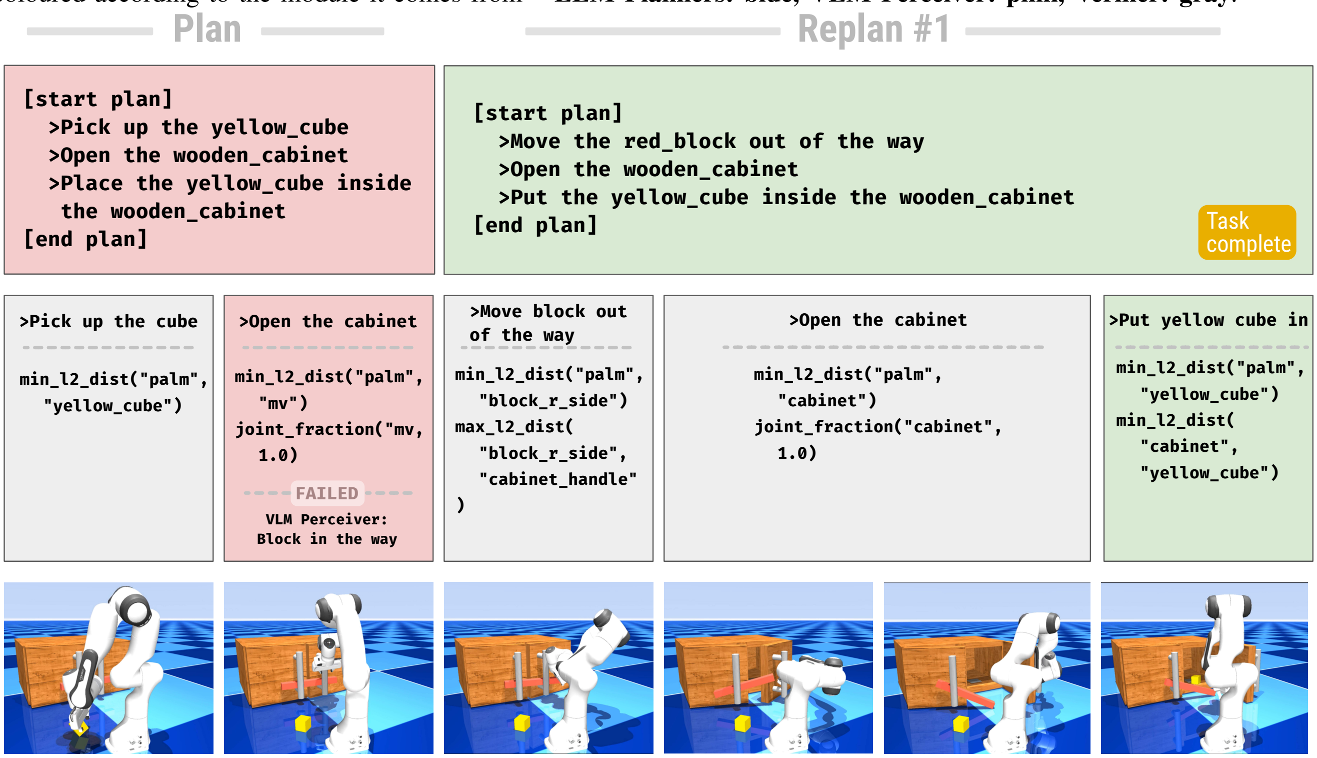



**Fig. III.0:** Rollout of robot solving `cabinet-closed`. The high-level plan is shown in the top row. The second row shows each subtask and the corresponding reward functions generated by the Low-Level Planner, as well as Perceiver feedback. If the subtask fails, its box is colored in red. If the plan is completed and the goal is achieved, its box is green.

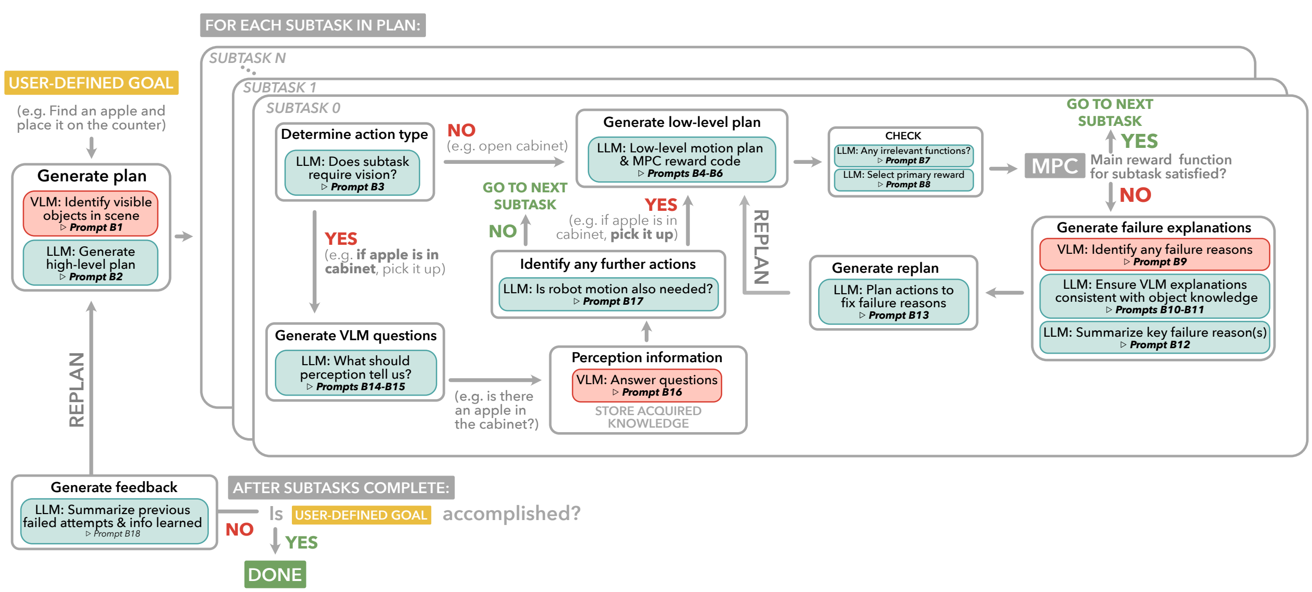



**Fig. III.1:** RePLan prompt roadmap. We show key modules in the pipeline and indicate under each one what prompts are utilized. This is an expanded version of the Figure 3

```
Do you see a(n) {0}?
```

**Fig. III.2:** VLM prompt for perceiving objects in the environment. {0} is an object that the robot knows how to interact with. The VLM is prompted with the list of objects the robot knows how to interact with. If the VLM replies with "yes", that object is added to a list of observed objects.

```
A stationary robot arm is in a location where it sees the following list of objects:

{0}

The robot has the following goal: {1}

Propose high-level, abstract subtasks of what the robot needs to do to {1}. The plan can only use one object.

For example, if the goal is to find a fork, one plan might be:

<thought>To find the fork, I will start by looking inside the drawer.</thought>
[start plan]
    >Open the drawer
    >Look inside the drawer
    >Grab the fork
[end plan]

Rules:
1. You have access to the following objects: {0}. Do not create new objects.
2. Generate a plan that interacts with only one object from the list at a time. Keep it as short as possible.
Most plans should be under 5 steps.
3. Assume that every action is completed successfully.
4. Assume the first thing you try works.
5. Your plan should only propose one way of accomplishing the task.
6. The robot only has one arm and it cannot hold two things at a time. Remember that when you are deciding on
the order of actions.
7. Enclose your thought process with a single pair of tag <thought> and </thought>
8. Enclose your plan with the a single pair of tag [start plan] and [end plan]

{2}
```

**Fig. III.3:** LLM prompt for generating high-level task plans. {0} is the list of objects the robot can see (for example: [cabinet, blue kettle, microwave], {1} is the overall task goal (for example: find the green apple), {2} are previous plans that were attempted but failed (see III.19).

```
A robot was asked to do this action:
    > {0}
If the central verb is related to vision, answer yes.
```

**Fig. III.4:** LLM prompt to determine whether the action that the Planner asked the robot to do involves vision or not. If no, then the Planner is called to generate MPC reward functions (see III.5-III.14). If yes, the Perceiver is called (see III.15-III.18). Examples for {0}: “Compare the color of the left cube with the crate", “Open the microwave".

```
We have a stationary robot arm and we want you to help plan how it should move to perform tasks using the
following template:
[start of description]
The manipulator's palm should move close to {{CHOICE: {0}}}.{1}{2}
[end of description]
Rules:
0. You cannot use one line twice!!!!
1. If you see phrases like [NUM: default_value], replace the entire phrase with a numerical value.
2. If you see phrases like {{CHOICE: choice1, choice2, ...}}, it means you should replace the entire phrase with
 one of the choices listed.
3. If you see [optional], it means you only add that line if necessary for the task, otherwise remove that line.
4. The environment contains {0}. Do not invent new objects not listed here.
5. I will tell you a behavior/skill/task that I want the manipulator to perform and you will provide the full
plan, even if you may only need to change a few lines. Always start the description with [start of description]
and end it with [end of description].
6. You can assume that the robot is capable of doing anything, even for the most challenging task.
7. Your plan should be as close to the provided template as possible. Do not include additional details.
8. Your plan should be as concise as possible. Do not include or make up unncessary tasks.
9. Each object can only be close to or far from one thing.

This is the entire procedure:
{4}

These are the observations we have made so far:
{5}

Create a plan for the following action:
    > {6}
```

**Fig. III.5:** LLM prompt to determine the low-level motion plan for the robot. {0} is the list of objects the robot can interact with. {1} and {2} are modifiers, depending on what type of motion is involved. The Planner is asked to determine whether motion is involved (see III.6. If yes, then {1} becomes: object1={{CHOICE: {0}}} should be {{CHOICE: close to, far from}} object2={{CHOICE: {0}}}. If no, then {1} becomes [optional] object1={{CHOICE: {0}}} should be close to object2={{CHOICE: {0}}}. [optional] object1={{CHOICE: {0}}} should be far from object2={{CHOICE: {0}}}. The modifier {2} is added if there are any joints in the scene that are involved with objects the robot can interact with: [optional] joint={{CHOICE: {3}}} needs to be {{CHOICE: open, closed}}. (where {3} is the list of joints) (adapted from [10]) {4} is the entire plan the robot was given to execute the goal. {5} includes any observations made by the Perceiver (see III.17) using the following format: Q: <question to Perceiver>, A: <answer from Perceiver>. {6} is the action the motion plan should be made for.

```
A robot arm has to do this action:
    > {0}
Does this action necessarily involve relocating an object to a different location that does not involve the
robot arm? Answer with yes or no.
```

**Fig. III.6:** LLM prompt to determine if relocation is needed in order to determine the motion plan modifier (see III.5.

```
We have a plan of a robot arm with palm to manipulate objects and we want you to turn that into the
corresponding program with following functions:

    def minimize_l2_distance_reward(name_obj_A, name_obj_B)

where name_obj_A and name_obj_B are selected from {0}. This term sets a reward for minimizing l2 distance
between name_obj_A and name_obj_B so they get closer to each other. rest_position is the default position for
the palm when it's holding in the air.

    def maximize_l2_distance_reward(name_obj_A, name_obj_B, distance=0.5)

This term encourages the orientation of name_obj to be close to the target (specified by x_axis_rotation_radians
).

    def execute_plan(duration=2)

This function sends the parameters to the robot and execute the plan for "duration" seconds, default to be 2.

    def set_joint_fraction_reward(name_joint, fraction)

This function sets the joint to a certain value between 0 and 1. 0 means close and 1 means open. name_joint
needs to be select from {1}.

    def reset_reward()

This function resets the reward to default values.
Example plan: To perform this task, the manipulator's palm should move close to object1=faucet_handle. object1
needs to be lifted to a height of 1.0.
This is the first plan for a new task.
Example answer code:

    import numpy as np

    reset_reward()
        # This is a new task so reset reward; otherwise we don't need it
    minimize_l2_distance_reward("palm", "faucet_handle")
    set_joint_fraction_reward("faucet", 1.0)

    execute_plan(4)

Remember:
1. Always format the code in code blocks. In your response execute_plan should be called exactly once at the end
.
2. Do not invent new functions or classes. The only allowed functions you can call are the ones listed above. Do
 not leave unimplemented code blocks in your response.
3. The only allowed library is numpy. Do not import or use any other library.
4. If you are not sure what value to use, just use your best judge. Do not use None for anything.
5. Do not calculate the position or direction of any object (except for the ones provided above). Just use a
number directly based on your best guess.
6. You do not need to make the robot do extra things not mentioned in the plan such as stopping the robot.

The action to perform is {2} and the plan is:
{3}
```

**Fig. III.7:** LLM prompt to generate MPC reward functions. {0} is the list of objects the robot can interact with, to which we also append the word "palm" to represent the robot hand. {1} is the list of object joints. {2} is the high-level action the robot needs to perform and {3} is the motion plan generated from III.5. Adapted from [10].

```
This is a motion plan generated for a robot:

{0}

This is a reward function generated to complete one step in the motion plan:

{1}

The function minimize_l2_distance_reward() refers to bringing two objects close together.
The function maximize_l2_distance_reward() refers to moving two objects further apart.
The function set_joint_fraction_reward() refers to opening or closing an object (0 for closed, 1 for open)
The function set_obj_z_position_reward() specifies the target height of an object.
The function set_obj_orientation_reward() specifies the target rotation of an object.

Which step in the motion plan is the function referring to? Return the step using <step></step> tags. If it does
 not refer to any of them, return <step>-1</step>
```

**Fig. III.8:** The Verifier checks that every reward function generated corresponds to a step in the motion plan. If it does not, the function is removed. {0} is the motion plan generated from Figure III.5 and {1} is one of the generated reward functions (they are looped over individually).

```
A stationary robot arm was asked to do the following motion plan to complete the task '{0}':

{1}

After which step in the motion plan will the task '{0}' be satisfied? First, explain your thought then answer
the step number enclosed with the tag <step> and </step>. Opening a joint can also mean activating it depending
on the context. You must select one. If you think none of the steps does, select the closest one.
```

**Fig. III.9:** The verifier selects which step in the motion plan is considered to be the most important. The reward function generated for that step in the motion plan becomes labelled as the primary reward function. {0} is the action the robot is currently doing and {1} is the motion plan.

```
A robot is in a simulation environment where it can interact with any object like in the real world. The robot
would like to {0} but it cannot. Is there something in this scene preventing that, other than the robot? Assume
the robot can interact with anything. These are the names of the objects in our scene: {1}
In a simulation, a robot wants to {0} but can't. Is anything else, besides the robot, blocking it? Check the
objects in the scene: {1}.
Robot in a simulation wants to {0}, can't. Something else stopping it? Objects in scene: {1}.
A robot can engage with any item. It wants to {0} but can't. Is an object in this scene, apart from the robot,
hindering it? Objects present: {1}
I would like to {0} but I cannot. Is there something in this scene preventing that, other than the robot? These
are the objects in the scene: {1}
I would like to {0} but I am unable to. Is there something in this scene preventing me from doing that? Ignore
the robot. These are the names of the objects: {1}
```

**Fig. III.10:** If the robot is unable to satisfy the primary reward function, the Perceiver is queried on whether there are any obstacles in the scene. The Perceiver is called once for each question (total of 6). Questions 2-4 and 6 were variations generated for Q1 and Q5, respectively, using ChatGPT.

```
We have access to the following objects in our scene: {0}

You are given a sentence describing an image of the scene, but it may have gotten the names of the objects wrong
. Does this sentence contain objects that are not in our scene or get the names of the objects wrong? Start your
 answer with yes or no.

{1}
```

**Fig. III.11:** For every explanation from the Perceiver, the Verifier is called to determine whether the explanation lists objects that do not exist in the scene. {0} is the list of objects in the scene and {1} is the explanation from the Perceiver. If the Verifier answers with 'yes', the explanation is passed to III.12 for object remapping.

```
We have access to the following objects in our scene: {0}

You are given a sentence describing an image of the scene, but got the names of the objects wrong. Rewrite this
sentence using the closest object(s) in our environment:

{1}

Rules:
You can only use objects in the scene. Use your best judgement.
```

**Fig. III.12:** If the Verifier identifies that the Perceiver explanation contains objects that are not listed in the scene, the Verifier rewrites the explanation ({1}) using the closest objects in our scene ({0}).

```
The stationary robot arm would like to {0} but it cannot. Here are possible reasons why based on images of the
scene:

    {1}

Based on the above explanations, what are the top reason(s) why the robot cannot {0}? List each reason on a
separate line, enclosed with the tag <reason> </reason>. Provide up to two reasons. Be as succinct as possible.
You must not include any reasons related to the robot, only reasons related to objects in the scene.
```

**Fig. III.13:** The Planner receives all explanations from the Perceiver ({1}, see III.10 -III.12) and summarizes them into key reasons explaining why the robot could not do the action {0}.

```
<Prompt from III.3>

One or more previous attempts failed. Below are the details.
---------------------------------- attempt #1 ----------------------------------
This attempt failed when executing '\{0\}'.The plan failed because the robot was not able to execute this action
: '\{1\}'. This was identified as a possible reason the action failed: '{2}'.
...
---------------------------------- attempt #R ----------------------------------
...
---------------------------- end of failed attempts ----------------------------
Reminder to propose a different plan than the above failed attempts.
```

**Fig. III.14:** If the robot does not succeed in performing an action, the Planner is able to replan how the robot does the action by providing the failure reason(s) from the Perceiver from the *R* failures reasons. Example for {0}: 'Place the red_cube on the crate'. Example for {1}: 'Place the red_cube on the crate (incidentally the action is the same as the overall goal, but it doesn't have to be)'. Example for {2}: 'The most probable reason why the robot cannot place the red_cube on top of the crate is that the yellow cube is currently on top of the crate, which would prevent the robot from doing so.'

```
You are a robot in the process of executing this plan, with the overall goal to '{0}':

{1}

You are currently performing this action: '{2}'. You have access to a perception model that can answer your
questions related to vision.

{3}

What question do you want to ask the perception model in order to get the answer to '{2}'? You can ask up to two
 questions. You don't have to ask if the information is already sufficient. Avoid asking the vision model to
compare things. Enclose each of your questions with the tag <question> </question>.
```

**Fig. III.15:** If the action requires calling the Perceiver, this prompt is used to determine what questions the Planner wants to ask the Perceiver. {0} is the overall goal, {1} is the high-level plan, {2} is the action the robot is currecntly performing, {3} are the observed objects in the scene.

```
What type of question is this asking perception model: '{0}'? Choose your answer from [OBJECT_PRESENCE, OBJECT_
ATTRIBUTE, NEITHER]
```

**Fig. III.16:** Prompt to field what category of question the Planner wants to ask the Perceiver model. {0} is the output from III.15.

```
<Output from III.15. The names of the objects in our scene are: {0}. {1}
```

**Fig. III.17:** Perceiver query on information about the state of objects in the scene from III.15. States are related to object presence or object attributes. Examples of queries: 'Look for the apple in the cabinet', 'Check the color of the crate'.

```
A robot was tasked to do this plan:

{0}

The robot is currently doing this action: '{1}'.

To do the action, the robot asked a perception model the following questions (Q) and got the answers (A):

{2}

After receiving this answer, has the robot completed the action '{1}'?  Begin your answer with yes or no. If
your answer begins with no, write the remaining action that needs to be completed using <Action></Action> tags.
```

**Fig. III.18:** After the Perceiver has provided information, the Planner is asked to determine whether the action is completed. If not, it generates MPC reward functions to finish the action (see III.5-III.19).

```
<Prompt from III.3>

One or more previous attempts failed. Below are the details.
---------------------------------- attempt #1 ----------------------------------
The proposed plan was:
<thought>{0}</thought>
[start plan]
{1}
[end plan]
The plan failed because {2}.
...
---------------------------------- attempt #P ----------------------------------
...
---------------------------- end of failed attempts ----------------------------
Reminder to propose a different plan than the above failed attempts.
```

**Fig. III.19:** If by executing the plan or the replan and the goal is still not accomplished, the Planner is prompted to generate a new plan using the prompt in Figure III.3. The Planner is allowed to generate a new plan $P$ times. before the task is considered undoable.

## Appendix IV
## Autonomous Continual Learning with Memory

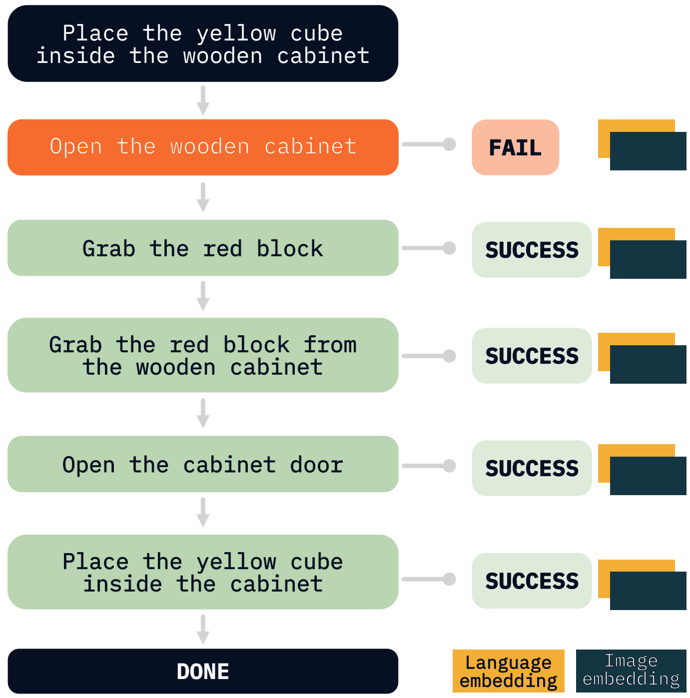
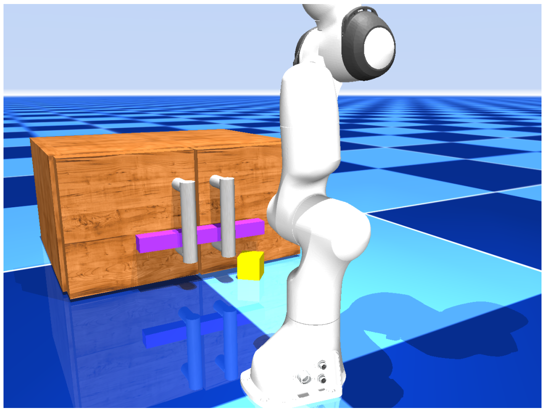


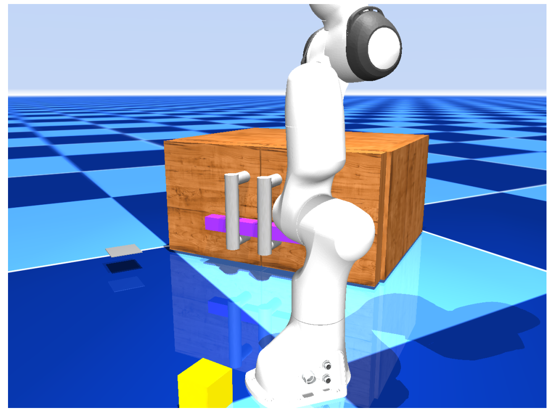

**Fig. IV.1:** Program trace of robot executing CabinetBlocked task, written to memory.

```
# High-Level LLM Planner:
<thought>To move the yellow cube to the target_position inside the wooden_cabinet, the robot will need to pick
up the yellow cube and place it inside the wooden_cabinet.</thought>
[start plan]
    > Approach the yellow_cube
    > Grasp the yellow_cube
    > Approach the wooden_cabinet
    > Open the wooden_cabinet
    > Place the yellow_cube inside the wooden cabinet at the target position
[end plan]

This is a trajectory that the robot previously executed to perform a similar task:

[start plan]
    > Open the wooden_cabinet --> FAILED
    > Grab the red_block --> SUCCEEDED
    > Move the red_block away from the wooden_cabinet --> SUCCEEDED
    > Open the wooden_cabinet --> SUCCEEDED
    > Pick up the yellow_cube --> SUCCEEDED
    > Move the yellow_cube to target_position inside the wooden_cabinet --> SUCCEEDED
[end plan]

You have access to a vision model that can answer visual questions about the scene. Based on this trajectory, is
 there anything you want to ask the vision model? The question should be a ``yes or no" question. Enclose your
question with the tag <question> <\question>. If you don't want to ask anything, reply NONE.

# Questions to the VLM:
<question>Is there any object blocking the wooden_cabinet?<\question>

# VLM reply:
Yes, there is a red object in front of the wooden cabinet that appears to be blocking it.

# Planner plans to move red object away from wooden cabinet.
```

**Fig. IV.2:** Planner using most relevant trajectory from past experiences to help plan next action sequence.



**(a)** CabinetBlocked task with different block colour



**(b)** CabinetBlocked task with different cabinet location and different block colour

**Fig. IV.3**

## APPENDIX V
## ERROR EVALUATION

Here we provide three error cases of REPLAN and their analyses.

*Case 1 (from `Cabinet-Blocked`, Figure V.1(a))* – The robot tried to open the cabinet door but failed and the Perceiver gave a correct diagnosis to remove the bar from the handle. However, when generating the reward functions to remove the bar, the LLM selected the wrong primary reward function, as demonstrated below:

```
reset_reward()
minimize_l2_distance_reward("palm", "red_block_right_side", primary_reward=True)
maximize_l2_distance_reward("red_block_right_side", "target_position_in_cabinet")
execute_plan()
```

The correct primary function should be the second one. As a result, MPC ended prematurely before the robot could remove the bar. The robot was not able to remove the bar in the following steps.

*Case 2 (from `Kitchen-Explore`, Figure V.1(b))* – The robot tried to open the microwave door but failed due to a kettle obstructing the path. The Perceiver gave five diagnoses, of which three claimed that the kettle was blocking the way, one claimed the cabinet door was blocking the way, and one did not give any conclusive diagnosis. The summary LLM concluded that it was the cabinet door that blocked the action. The robot went on to interact with the cabinet and never removed the kettle.

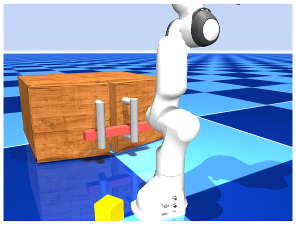
*Case 3 (from `TwoCube-Color`, Figure V.1(c))* – The high-level planner proposed a plan where the first step was *"Identify the cube with the same colour as the crate"*. This was a VLM action. However, the LLM proposed to ask the VLM *"Which cube has the same colour?"*, which was a bit vague, resulting in the VLM answering *"The same color cube is the yellow cube and the yellow cube in the middle of the blue cube group."*. This answer did not provide the necessary information to solve the task. Eventually, the robot put the wrong cube on the crate.

*Case 4 (from `TwoCube-Blocked`, Figure V.1(d))* – After the robot was not able to execute the task "Place the red cube on the crate", the Perceiver was called to help identify any issues. The Perceiver's diagnoses all mentioned that the robot was holding the red cube but did not identify the yellow cube as blocking the crate, and so the Planner's summary of the VLMs diagnoses was: "Based on the given information, the most probable reason why the robot cannot place the red_cube on the crate is because it is currently holding the red cube." However, it's also important to note that `TwoCube-Blocked` used GPT-4V which severly limits the number of ouput tokens from the model, and so a lot of explanations were cut off (for example: "In the image provided, the robot is holding the red cube, which is currently").

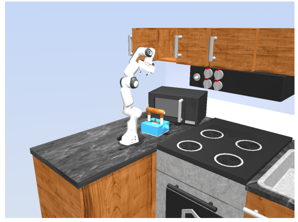
### A. LLM Diagnosis with Ground-truth Data

An alternative way to diagnose errors is to input the simulator ground-truth state of the objects to an LLM and ask the LLM to infer a possible reason. We show a scenario here where the cabinet door is locked by a red bar (from `Cabinet-Blocked`) and test the capability of LLM in this regard. The prompt we use is:

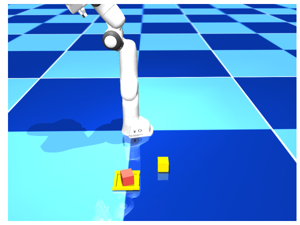
```
A robot is in a simulation environment where it can interact with any object like in the real world. The robot
would like to open the wooden cabinet but it cannot. Is there something in this scene preventing that, other
than the robot? Assume the robot can interact with anything. These are the names of the objects in our scene:
wooden cabinet, red block, yellow cube.

The data of the objects are:
```



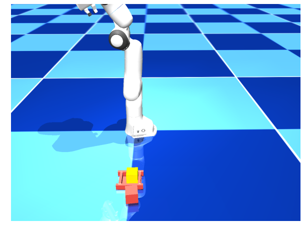
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

**Fig. V.1: Examples of error cases** that occurred in REPLAN include a) failure to remove the bar due to incorrect reward generation (CabinetBlocked); b) failure to identify the reason for the inability to open the microwave door (KitchenExplore); c) vague VLM question about object attributes resulting in incorrect cube selection (CubesColor); d) failure to identify why the robot could not place the red cube in the crate (CubesBlocked).


```
{
    "joints": {
        "wooden_cabinet": 0.16235716765387215
    },
    "objects": {
        "wooden_cabinet_handle": [0.08292539, 0.50163332, 0.21],
        "target_position_in_wooden_cabinet": [0.2,    0.85,    0.0775],
        "red_block_right_side": [0.2158331,  0.55110136, 0.15143364],
        "yellow_cube": [-0.24545329,  0.01631876,  0.03599184],
        "palm": [0.2231031,  -0.00875834,  0.76900653]
    }
}
```


And the response from GPT-4 is:

```
The data does not indicate any object that would prevent the robot from opening the cabinet. There's no
information about any object blocking the cabinet or its handle. The problem could be due to the robot itself as
 it might not possess the necessary functionality to interact with the objects as intended. The cabinet's joint
also has a non-zero value, which might mean it's not locked or stuck. Hence, there is likely a mechanical
limitation with the robot or an issue with its programming.
```

This example demonstrates that LLM is not capable of directly inferring spatial relations of objects through ground-truth data.

## APPENDIX VI
## ADDITIONAL EXPERIMENTS

### A. VLM Ablation

We conduct a VLM ablation study for Tasks 3-4 using two open-source models (Qwen-VL-Chat7B [32] and Llava-1.5-7B [54]) and GPT-4V[5]. The results are shown in Table IV. We measure (a) their ability to recognize objects, reported as a percentage of the number of object successfully detected in the scene (column 2), (b) their ability to reason about obstacles in the scene, reported as a percentage of times the correct reason was identified out of all the prompts it was given in Figure III.10, and (c) the ability for the LLM to consilidate the outputs in (b) into a single summary reason of why the robot cannot perform an action given its knowledge about the scene. We found that Qwen somtimes struggled with object detection of smaller objects, and so we coupled it with Segment Anything Model (SAM) [55] to first segment the objects in the scene. We found that all models did well with object recognition (except for Qwen when not used with SAM). For object reasoning, Qwen + SAM was able to get the correct scene error in 50-67% of the prompts it was given, and then the LLM was able to summarize the prompts to generate the correct error reason overall. The reason the LLM was able to do this despite the VLM not giving perfect answers was that the remaining VLM answers pertained where the robot was located or a general comment about the objects in the scene. Llava tended to reply that it was unable to reason because the scene was a simulation and not real life. GPT-4V had the best overall performance in all categories, but API calls to it are still restricted. All ablations were repeated over 5 runs.

### B. GPT-4V Experiments

We run the initial High-Level Planner prompt (Prompt III.3) using GPT-4V[6] on initial task scenes to investigate the ability of GPT-4V to find the correct solution in a single step.

[5]https://openai.com/research/gpt-4
[6]https://openai.com/research/gpt-4v-system-card

| | Models | | | |
|---|---|---|---|---|
| Scenarios | Qwen + SAM | Qwen | Llava | GPT-4V |
| VLM object detection | 100% | 66% | 100% | 100% |
| VLM Reasoning | 67% | 0% | 23% | 100% |
| LLM summarization and consistency step | 100% | 0% | 100% | 100% |

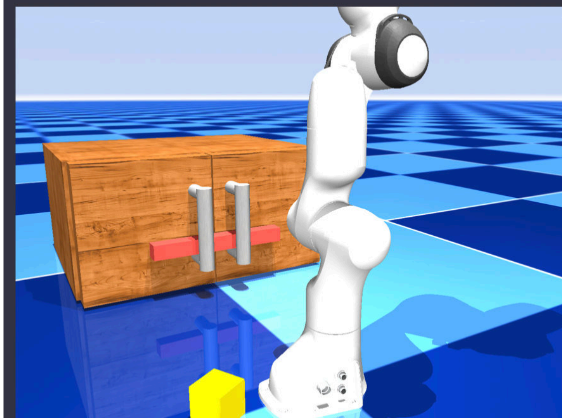
**(a)** `Cabinet-Blocked`

| | Models | | | |
|---|---|---|---|---|
| Scenarios | Qwen + SAM | Qwen | Llava | GPT-4V |
| VLM object detection | 100% | 100% | 100% | 100% |
| VLM Reasoning | 50% | 66% | 40% | 83% |
| LLM summarization and consistency step | 100% | 100% | 20% | 100% |

**(b)** `Kitchen-Explore`

**TABLE IV:** VLM ablation study.

**You**



A stationary robot arm is in a location where it sees the following list of objects:

[wooden cabinet, red block, yellow cube]

The robot has the following goal: move the yellow cube to target position inside the wooden cabinet

Propose high-level, abstract subtasks of what the robot needs to do to move the yellow cube to target position inside the wooden cabinet. The plan can only use one object.

For example, if the goal is to find a fork, one plan might be:

<thought>To find the fork, I will start by looking inside the drawer.</thought>
[start plan]
  >Open the drawer
  >Look inside the drawer
  >Grab the fork
[end plan]

Rules:
1. You have access to the following objects: [wooden cabinet, red block, yellow cube]. Do not create new objects.
2. Generate a plan that interacts with only one object from the list at a time. Keep it as short as possible. Most plans should be under 5 steps.
3. Assume that every action is completed successfully.
4. Assume the first thing you try works.
5. Your plan should only propose one way of accomplishing the task.
6. The robot only has one arm and it cannot hold two things at a time. Remember that when you are deciding on the order of actions.
7. Enclose your thought process with a single pair of tag <thought> and </thought>
8. Enclose your plan with the a single pair of tag [start plan] and [end plan]

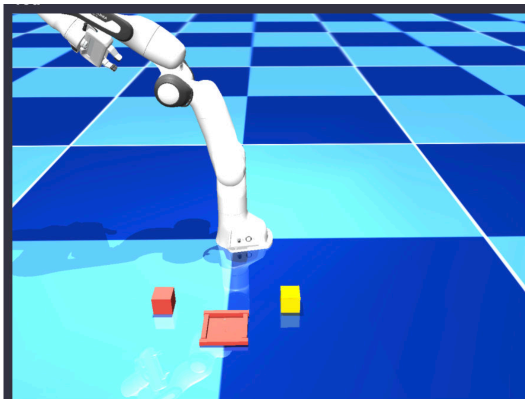
**(a)** Prompt III.3 to GPT-4V.

**ChatGPT**

<thought>To move the yellow cube to the target position inside the wooden cabinet, I need to consider the simplest and most direct method that involves minimal interaction with other objects. Since the robot has only one arm and cannot hold two things at once, the plan should avoid unnecessary handling of the red block, focusing solely on the yellow cube and the cabinet.</thought>
[start plan]
>Approach the wooden cabinet.
>Open the cabinet door.
>Pick up the yellow cube.
>Place the yellow cube inside the cabinet at the target position.
>Close the cabinet door.
[end plan]

**(b)** GPT-4V output.

**Fig. VI.1:** GPT-4V high-level plan for moving the yellow cube inside the wooden cabinet (`Cabinet-Blocked` in Section II-A).



You
Uploaded image
A stationary robot arm is in a location where it sees the following list of objects:

[crate, left cube, right cube]

The robot has the following goal: place the cube with the same color as the crate on the crate

Propose high-level, abstract subtasks of what the robot needs to do to place the cube with the same color as the crate on the crate. The plan can only use one object.
For example, if the goal is to find a fork, one plan might be:

<thought>To find the fork, I will start by looking inside the drawer.</thought>
[start plan]
>Open the drawer
>Look inside the drawer
>Grab the fork
[end plan]

Rules:
1. You have access to the following objects: [crate, left cube, right cube]. Do not create new objects.
2. Generate a plan that interacts with only one object from the list at a time. Keep it as short as possible. Most plans should be under 5 steps.
3. Assume that every action is completed successfully.
4. Assume the first thing you try works.
5. Your plan should only propose one way of accomplishing the task.
6. The robot only has one arm and it cannot hold two things at a time. Remember that when you are deciding on the order of actions.
7. Enclose your thought process with a single pair of tag <thought> and </thought>
8. Enclose your plan with the a single pair of tag [start plan] and [end plan]

**(a)** Prompt III.3 to GPT-4V.

**ChatGPT**
<thought>To achieve the goal of placing the cube with the same color as the crate on the crate, I need to identify which cube matches the crate's color. Once identified, I will use the robot arm to pick up the correct cube and place it on the crate.</thought>

[start plan]
>Identify the cube with the same color as the crate
>Move the robot arm towards the identified cube
>Lower the robot arm to reach the cube
>Grasp the cube securely
>Lift the cube
>Move the robot arm over the crate
>Lower the cube onto the crate
>Release the cube
[end plan]

**(b)** GPT-4V output.

**Fig. VI.2:** GPT-4V high-level plan for placing the cube with the same color as the crate on the crate (`TwoCube-Color` in Section II-D).

Even using GPT-4V as an oracle does not always work. In Figure VI.3, we show a failure case where it is unable to detect that a block is obstructing the cabinet.

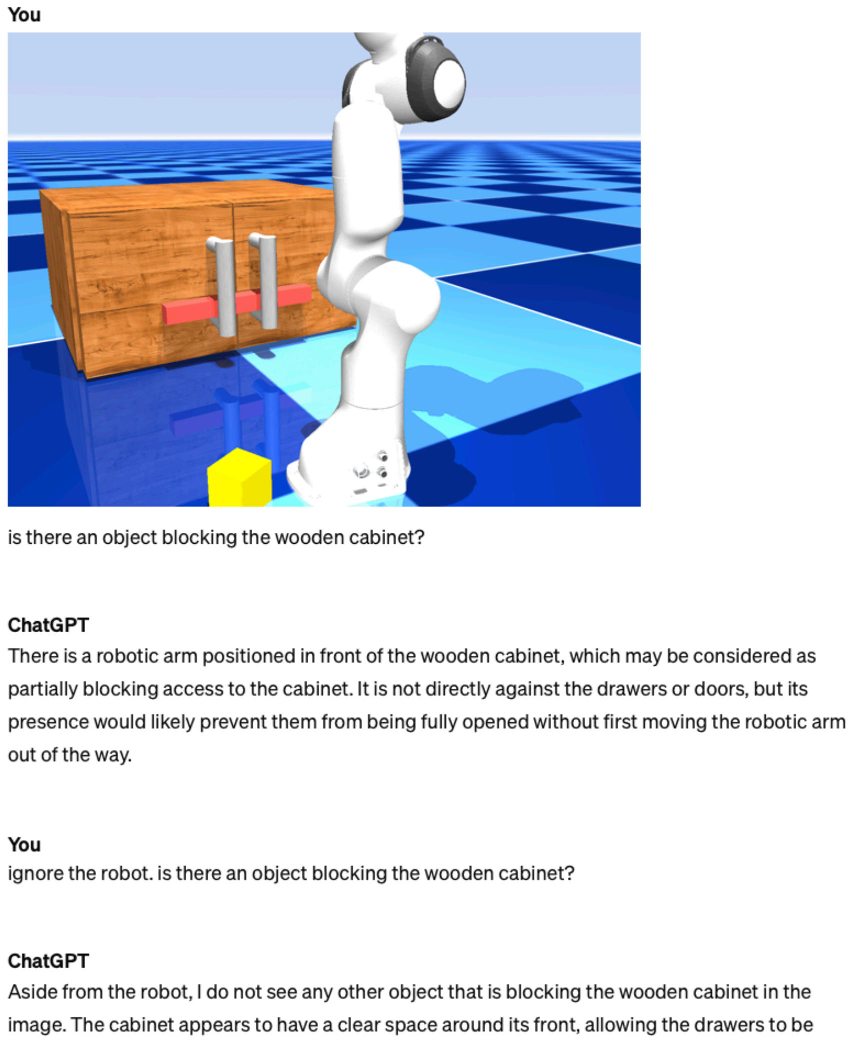



**Fig. VI.3:** Prompt to GPT-4V about obstructions in scene.

### C. *Proactive prompting*

Instead of only engaging the VLM Perceiver when an action fails, we experiment with *proactively* querying the VLM to determine if there exist any obstacles before task execution since proactively finding obstacles is more efficient than identifying them in retrospect. To investigate this, we enabled REPLAN to take cropped images of any objects involved in a given subtask and queried the VLM Perceiver:

```
 A robot is about to perform the action '{0}'. Is there anything preventing the robot from doing this action?
The names of the object in the scene are: {1}. Start your answer with yes or no. Ignore anything related to the
robot.
```

**Fig. VI.4:** Perceiver prompt on proactively identifying scene obstacles.

We intergrated this step into REPLAN and evaluated it on two tasks: CabinetBlocked and KitchenExplore, running each task 5 times. For CabinetBlocked, REPLAN was able to proactively identify the block obstructing the cabinet **3/5** times. For KitchenExplore, REPLAN what able to proactively identify the kettle obstructing the microwave **2/5** times.

We speculate that this is because the question is very general and the VLM might not know what to look for: identifying all types of prevention methods is intractable (it was shown in DoReMi [12] that an LLM asks limited precondition queries) and also the VLM might not know if something is an obstacle or not until after the robot interacts with the object in question. However, once the robot attempts the action and fails, the VLM has more context because it knows that the task failed. In light of these results, we believe that incorporating both proactive querying and replanning makes the system more robust: if the preconditions are not caught, the replanning system still exists to make sure the system is robust and the robot can recover from unexpected obstacles.

Below are sample VLM responses. The first response is a success case where it correctly identified the obstruction proactively, while the second case is where it was not able to:

[SUCCESS CASE]
Q: A robot is about to perform the action 'Open the wooden cabinet'. Is there anything preventing the robot from doing this action? The names of the object in the scene are: wooden cabinet, yellow cube, red block. Start your answer with yes or no. Ignore anything related to the robot.
A: Yes, there is a red block that appears to be wedged between the two handles of the wooden cabinet, which would prevent the doors from being opened without first removing the block.

[FAILURE CASE]
Q: A robot is about to perform the action 'Open the wooden cabinet'. Is there anything preventing the robot from doing this action? The names of the object in the scene are: wooden cabinet, yellow cube, red block. Start your answer with yes or no. Ignore anything related to the robot.
A: No, there is nothing preventing the robot from opening the wooden cabinet based on the objects named in the scene. The yellow cube and red block do not appear to be obstructing the cabinet doors.

### D. Comparison with existing methods

While prior methods appear general in their framing, they have only been experimentally evaluated on tasks that are easy to solve: those tasks require fewer steps, provide full state or privileged information, necessitate human involvement, and/or do not require complex multi-stage reasoning. We compare how the CabinetBlocked and CabinetLocked tasks are solved by REPLAN, as well as Inner Monologue [15], SayCan [14], DROC [13], and LLM-POP [16].

**TABLE V:** Plan comparisons of various methods for cabinet-opening step in CabinetBlocked task. Note that we speculate the plans for Inner Monologue, SayCan, and DROC based on the framework descriptions from their papers.

| **REPLAN** | **Inner Monologue** [15] | **SayCan** [14] | **DROC** [13] | **LLM-POP** [16] |
|---|---|---|---|---|
| **(1) Can I open the door?** No.<br>*(2)* REPLAN.*Planner engages* REPLAN.*Perceiver to ask for possible reasons*<br>*(3)* REPLAN.*Perceiver: "There is a red block blocking the cabinet handles."*<br>*(4)* REPLAN.*Planner generates plan for removing the block.* | **Can I open the door?** No.<br>*Maybe my low-level skill was wrong, I will select a new skill and try to open the door again, or I will get information from a human about how to solve the task.* | **Can I open the door?** No.<br>*If I was told that the task failed and I have a value function for "Remove the red block" in my database, maybe I would plan to remove the red block.* | **Can I open the door?** No.<br>*If I have previously seen a similar task I can use information from my knowledge base; else I need a human to correct me.* | **Can I open the door?** No.<br>*Output from GPT-4 based on prompt in Appendix A.1 in [16]: FAIL Task is not finished. Based on the [History], the force sensor readings indicate the cabinet door is not open, suggesting the action to open the cabinet failed. This failure might be due to insufficient force applied or improper alignment of the gripper relative to the cabinet handle. Adjusting the force or the gripper's pose could potentially resolve the issue. (Note that even if replacing force sensors with perception sensor, the perception sensor does not provide causal information).* |

**TABLE VI:** Plan comparisons of various methods for cabinet-opening step in CabinetLock task. Note that we speculate the plans for Inner Monologue, SayCan, and DROC based on the framework descriptions from their papers.

| **REPLAN** | **Inner Monologue** [15] | **SayCan** [14] | **DROC** [13] | **LLM-POP** [16] |
|---|---|---|---|---|
| **(1) Can I open the door?** No. *(2)* REPLAN.*Planner engages* REPLAN.*Perceiver to ask for possible reasons* *(3)* REPLAN.*Perceiver: "There is nothing obstructing the cabinet."* *(4)* REPLAN.*Planner: Maybe the door is locked, I will pull the lever first and then try to open the door again.* | **Can I open the door?** No. *Maybe my low-level skill was wrong, I will select a new skill and try to open the cabinet again, or I will get information from a human about how to solve the task.* | **Can I open the door?** No. *If I was told that the task failed and I have a value function for "Pull the lever" in my database, maybe I would plan to pull the lever.* | **Can I open the door?** No. *If I have previously seen a similar task, I can use information from my knowledge base; else I need a human to correct me.* | **Can I open the door?** No. *Output from GPT-4 based on prompt in Appendix A.1 in [16]: FAIL Task is not finished. Based on the [History], the force sensor readings indicate the cabinet door is not open, suggesting the action to open the cabinet failed. This failure might be due to insufficient force applied or improper alignment of the gripper relative to the cabinet handle. Adjusting the force or the gripper's pose could potentially resolve the issue. (Note that even if replacing force sensors with perception sensor, the planner still did not reason to interact with the lever even though it was provided as an object in the Task Description.)* |

## *E. Comparison with TAMP Experiments*

To compare the RePLan with a TAMP framework, we use Planning Domain Definition Language (PDDL) [56] to define the domain of Cabinet Tasks 1-3 in Table II-D in Section II as follows:

```
(define (domain pick-place-domain)
  (:requirements :strips :typing :negative-preconditions :conditional-effects)

  ;; Define the object and its possible locations
  (:types
      object
      location
      conf
      robot
      door cabinet cube block - object
      area
      remove_area cabinet_area - area

  )
  ;; define constants
  (:constants
      cube_loc cabinet_loc block_loc remove_loc init_loc door_loc open_door_loc - location
      init_conf robot_conf_cube robot_conf_block robot_conf_cabinet robot_conf_remove robot_conf_door open_door_conf - conf
      robot - robot
      door - door
      cabinet - cabinet
      cube - cube
      block - block
      remove_area - remove_area
      cabinet_area - cabinet_area
  )

  ;; Define predicates
  (:predicates
    (at ?obj - object ?loc - location)  ; the Object at location loc
    (grasped ?obj -object)     ; the object is grasped
    (at_conf ?conf -conf)               ; the robot is at conf configuration
    (rob_at_loc ?loc -location)    ; the robot is location loc
    (is_free ?rob - robot)      ; the robot hand is free
    (in ?loc -location ?area -area)   ; sampling and certifying the loc location is inside an area type
    (is_closed ?door - door)        ; the door is closed
    (is_blocked ?door)             ; the door is blocked
    (is_moveable ?init_conf -conf ?final_loc -location ?final_conf - conf)
    ; certifying the robot can move from the initial condition to the goal location/pose with the sampled final configuration

  )

    ;; Define actions
  (:action remove
      :parameters (?block - block ?door - door ?rob - robot ?init_conf ?final_conf - conf ?init_loc ?final_loc - location)
      :precondition (and
          (is_blocked ?door)
          (grasped ?block)
          (not (is_free ?rob))
          (at_conf ?init_conf )
          (at ?block ?init_loc)
          (rob_at_loc ?init_loc)
          (in ?final_loc remove_area)
          (is_moveable ?init_conf ?final_loc ?final_conf )
      )
      :effect (and
          (rob_at_loc ?final_loc )
          (not (rob_at_loc ?init_loc))
          (at ?block ?final_loc)
          (at_conf ?final_conf )
          (is_free ?rob)
          (not (grasped ?block))
          (not (is_blocked ?door))
          (not (at_conf ?init_conf ))
      )
  )

      (:action place
      :parameters (?cube - cube ?door - door ?rob - robot ?init_conf ?final_conf - conf ?init_loc ?final_loc - location)
      :precondition (and
          (not (is_closed ?door))
          (not (is_blocked ?door))
          (grasped ?cube)
          (not (is_free ?rob))
          (at_conf ?init_conf )
          (at ?cube ?init_loc)
```

```
        (rob_at_loc ?init_loc)
        (in ?final_loc cabinet_area)
        (is_moveable ?init_conf ?final_loc ?final_conf )
    )
    :effect (and
        (rob_at_loc ?final_loc )
        (at ?cube ?final_loc)
        (at_conf ?final_conf )
        (not (at_conf ?init_conf ))
        (is_free ?rob)
        (not (grasped ?cube))
    )
)


(:action pick
  :parameters (?init_conf ?final_conf -conf ?obj - object ?loc - location ?rob -robot)
  :precondition (and
    (at_conf ?init_conf )
    (at ?obj ?loc)
    (is_free ?rob )
    (is_moveable ?init_conf ?loc ?final_conf )
    )
  :effect (and
    (at_conf ?final_conf)
    (not (at_conf ?init_conf))
    (rob_at_loc ?loc)
    (not (is_free ?rob))
    (grasped ?obj)
  )
)

(:action open
    :parameters (?init_conf -conf ?door -door ?rob -robot )
    :precondition (and
      (is_closed ?door)
      (not (is_blocked ?door))
      (grasped ?door)
      (not (is_free ?rob))
      (at_conf ?init_conf )
      (is_moveable ?init_conf open_door_loc open_door_conf )
    )
    :effect (and
      (not (is_closed ?door))
      (not (grasped ?door))
      (is_free ?rob)
      (not (at_conf ?init_conf ))
      (at_conf open_door_conf)
      (rob_at_loc open_door_loc)
      (at door open_door_loc)
      )
    )

   (:action is_moveable_cube
       :parameters ()
       :precondition (and
         (not (is_closed door))
         (not (is_blocked door))
       )
       :effect (is_moveable robot_conf_cube cabinet_loc robot_conf_cabinet)
   )

)
```

**Snippet 1:** PDDL domain definition for Tasks 1-3.

Tasks 1-3 problem definition are written as follows:

```

(define (problem pick-place-problem)
  (:domain pick-place-domain)

  ;; Define objects
  (:objects

  )

  ;; Define initial state
  (:init
    (at cube cube_loc)
    (at door door_loc)
    (at block block_loc)
    (at_conf init_conf)
    (rob_at_loc init_loc)
    (in remove_loc remove_area)
    (in cabinet_loc cabinet_area)
    (is_free robot)
    (is_moveable init_conf cube_loc robot_conf_cube)
    (is_moveable open_door_conf cube_loc robot_conf_cube)
    (is_moveable robot_conf_block remove_loc robot_conf_remove)
    (is_moveable robot_conf_remove door_loc robot_conf_door)
    (is_moveable init_conf block_loc robot_conf_block)
    (is_moveable init_conf door_loc robot_conf_door)
    (is_moveable robot_conf_door open_door_loc open_door_conf)

    ;====================================================================
    ;; commenting the following two initial conditions can change the robot behavior greatly in terms of task plan
    ;; easy mode (Task 1): comment both of the following lines [comment (is_closed door) and (is_blocked door)]
    ;; hard mode (Task 2): comment the second condition [(is_blocked door)]
    ;; expert mode (Task 3): keep both of the following conditions uncommented.
    ;====================================================================

    ; door is closed at the begining
    (is_closed door)
    ; door is blocked at the begining
    (is_blocked door)


    )

  ;; Define goal
  (:goal
    (and
        ;; only picking the cube
        ; (grasped block)

        ;; picking and placing the cube inside the cabinet
        (at cube cabinet_loc)
          ; (at block remove_loc)
          ; (grasped door)


    )
  )
)
```

**Snippet 2:** PDDL problem definition for Cabinet Tasks 1-3.

The cabinet task problem is solved using the lpg-td [57] solver from the planutils library[7].

As seen in Snippets 1 and 2, even for three cabinet tasks 1-3, domain and problem definitions require careful and laborious attention. Task-solving details are outlined in Snippets with blue and olive colours. While our method discovers information through interaction and reasoning over perceiver's feedback, the PDDL solver relies on ground truth (highlighted in blue) and rules (example in olive) provided by the user for problem resolution.

For the same task, PDDLStream [58] offers an alternative using a Task and Motion Planning (TAMP) framework. Rather than human-grounded truth information, a motion planning framework certifies predicates through *streams*. However, this requires user-defined rules for success or failure and a motion planner. Solving long-horizon problems with PDDLStream may become computationally expensive [59].

RePlan can robustly solve long-horizon multi-stage problems through interaction with the environment and reasoning based on the perceiver's feedback. This capability enables RePlan to uncover underlying rules without the need for an additional domain description and ground truth information.

[7] https://github.com/AI-Planning/planutils

## Appendix VII
## Real-Robot Implementations

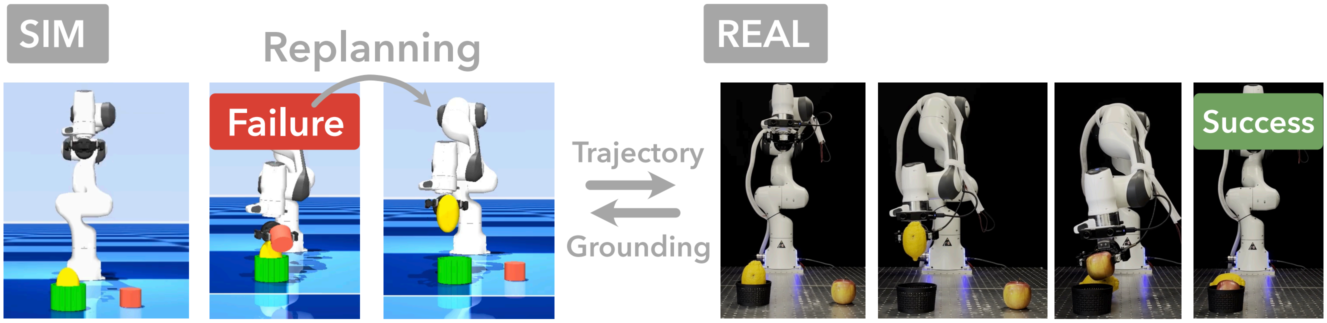



**Fig. VII.1: Robot experiment**. The robot is tasked with placing an apple inside the bowl, but it has to figure out that the lemon must first be removed. The full experiment trajectory can be seen in our video.

To ensure safety and robustness in real-robot experiments, we add the following supplementary terms to our MPC objectives:

- *A penalty for moving the bowl.* This term is added to prevent the robot from contacting with the bowl, which can potentially cause damage.
- *A penalty for rotating the manipulated object.* This term is added to prevent the robot from hitting the manipulated object, which will cause the object to roll.
- *A penalty for closing the gripper when the manipulated object is not nearby.* This term is added for two concerns: (1) Same with the rotating penalty, this term also contributes to not hitting the object. (2) Due to the sim-to-real gap including simulation inaccuracies and modeling inaccuracies, the timing of getting to a good position for grasping might be different in simulation and in the real world. This term improves the consistency of grasping.