white basket, green basket, brown basket, white bowl, red bowl, green bowl, red cup, green cup, white cup, white casserole, green casserole, black casserole, green cushion, yellow cushion, blue cushion, yellow dumbbell, green dumbbell, blue dumbbell, blue jug, orange jug, white jug, blue towel, red towel, green towel, grey hat, yellow hat, white plate, green plate, pink plate, black shoe, blue shoe, orange shoe, grey toy, blue toy, black toy, red vase, blue vase, pink vase, black teapot, white teapot, yellow teapot

Table 3. Full list of object categories used for ASK-TO-ACT task.

A USE OF LLM FOR WRITING

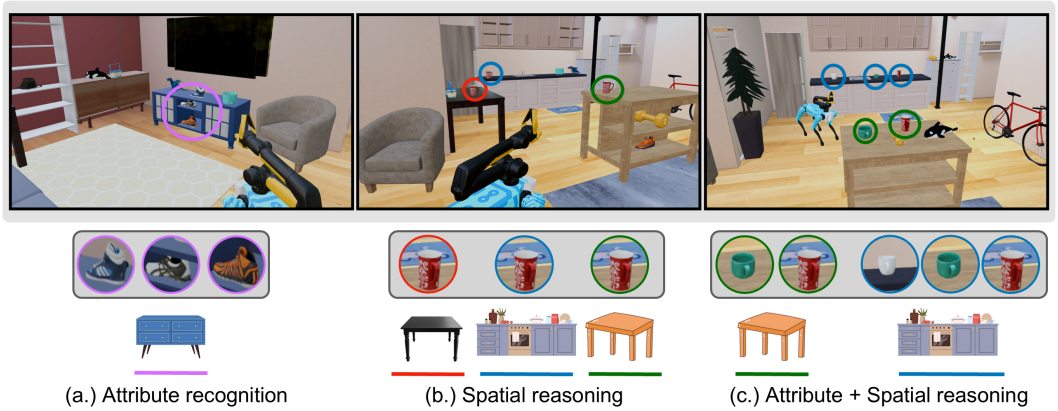
We use LLMs to assist with specific aspects of paper writing. These include: (1) grammar checking to improve language syntax, (2) sentence paraphrasing for improving readability and clarity. We use LLMs to write parts of Sections 1, 3, 4, and 5. Specifically, for each section the original content was written by the authors and paraphrased or spell checked by the GPT-4o OpenAI (2023) LLM. Followed by this round of LLM edits, the authors verified and edited the content written by the LLM to fix issues in generated text. After which the author edited version of the paper is used for final submission.

B DATASET

In Fig. 5, we show additional examples of different types of ambiguities in our ASK-TO-ACT dataset. Specifically we show 3 additional examples for:

- **Attribute Recognition**: The agent must reason about appearance-based attributes such as color and object category to disambiguate the object user is looking for (e.g., a red bowl and white bowl as shown in Fig. 5 (a)).
- **Object Size**: The agent must distinguish between objects of different sizes but similar appearance and location to disambiguate the object user is looking for (e.g., a large red bowl vs. a small red bowl as shown in Fig. 5 (b)) .
- **Attribute/Spatial Reasoning and Object Size**: A combination of attribute and spatial reasoning with object size, where the agent must account for both appearance or location and size to effectively disambiguate the target object the user is looking for. For the example shown in Fig. 5 (c), the agent needs to reason about different colored shoes (blue, white, and orange shoe) which are located on the blue cabinet and the sofa and some of those have different size *e.g.* large blue shoe on blue cabinet *vs.* small blue shoe on sofa and large white shoe on sofa *vs.* small white shoe on blue cabinet.

Object Categories. We present the full list of object categories used for ASK-TO-ACT dataset in Tab. 3.



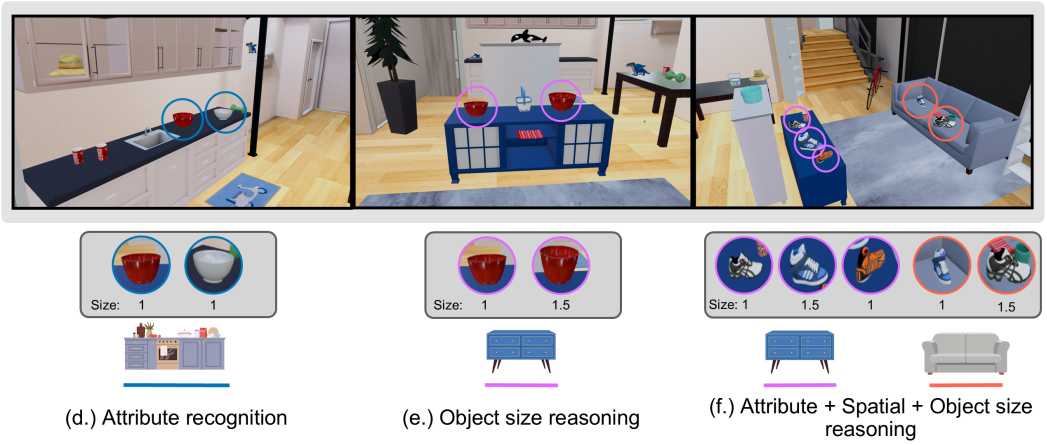


Figure 5. Dataset. Examples of different types of synthetically created ambiguities in ASK-TO-ACT tasks.

C TRAINING DETAILS

Problem Setup. Our problem setup can be formulated as a Partially-Observable Markov Decision Process (POMDP), defined by a tuple $(S, O, A, P, R, p_0, \gamma)$ where S is state space, O is observation space, A is action space, P represents transition dynamics, R is reward function, p_0 is initial state distribution and γ is discount factor. In our setting, O is a combination of responses from the user (for any questions asked) and visual observations, which come from the robots egocentric RGB camera, and provide only partial views of the environment. We consider the extension of including a goal distribution G and the case where the reward is formulated as $R(s, g)$ for $s \in S$ and $g \in G$. We aim to learn a language-conditioned policy $\pi(a|o, g)$ mapping from observation o and task instruction g to an action a that maximizes the sum of discounted rewards $\mathbb{E}_{s_0 \sim p_0, g \sim G} \sum_t \gamma^t R(s_t, g)$.

Training Details. To train our policy using reinforcement learning (RL) we use PPO with Generalized Advantage Estimation (GAE) (Schulman et al., 2016). We use a discount factor γ of 0.99 and set GAE parameter τ to 0.95. We do not use normalized advantages. To parallelize training, we use DD-PPO (Wijmans et al., 2020), an adaptation of PPO (Schulman et al., 2017) for distributed training, with 24 environments each on 8 A40 GPUs for 50 million steps . Each worker collects 24 frames of experience from 24 environments parallely and then performs 2 epochs of PPO update with 2 mini batches in each epoch. Tab. 4 details the default hyperparameters used in all of our training runs.

Per-Step Reward Inference. For per-step reward inference for evaluating questions asked by an agent, we use a vLLM (Kwon et al., 2023) server to host our LLM answer evaluation model using Llama-3.1 8B Instruct (Grattafiori et al., 2024) model. At each rollout step during RL training and policy evaluation, if the agent outputs a 'ask_question' action we send a request to the vLLM server and feed the response back to the policy at the next step. In order to achieve a high throughput we run 8 parallel vLLM servers and randomly send requests to different workers from training and evaluation workers.

Token Normalization for Action Probability and Entropy. When using MLLMs for RL training, each action $a \in A$ at every timestep is represented using a set of tokens $(u_t^1, ..., u_t^m)$. However, the MLLM only outputs token level probabilities which is different from probability of executing an action in the environment as used in RL traditionally. The token-level probability for each action a_t can be represented by:

$$P_{\text{token}}(a_t|s) = \prod_{i=1}^{m} P(u_t^i|s, u_t^1, \ldots, u_k^{i-1}) \tag{3}$$

To compute action probabilities, one naive approach is to take a sum over all token-level probabilities. When training MLLMs with action-space with variable number of tokens across actions one issue in Eq. (3), is that actions with larger number of tokens tend to have lower token-level probabilities, even

Parameter	Value
Number of GPUs	8
Number of environments per GPU	24
Rollout length	24
PPO epochs	2
Number of mini-batches per epoch	2
LR	$2.5e^{-4}$
Optimizer	Adam
Weight decay	0.0
Epsilon	1.0×10^{-5}
PPO clip	0.2
Generalized advantage estimation	True
γ	0.99
τ	0.95
Value loss coefficient	0.5
Max gradient norm	0.2
DDPPO sync fraction	0.6

Table 4. Hyperparameters used for RL finetuning.

though they might be more reasonable to take at some point in the environment. This issue happens because the probability of each token $P(u_t^i|\cdot)$ is always less than 1. This could be problematic in RL optimization especially for the case when the agent needs to output a natural language question for ASK-TO-ACT task, simply because the questions tend to have more tokens than pre-defined skills represented in language. To remedy this issue, we use action length normalization technique normalize the token-level probabilities of the actions with the number of action tokens, which can be defined as:

$$logP(a_t|s) = \sum_{i=1}^{m} logP_{\text{token}}(u_t^i|s, u_t^1, \ldots, u_k^{i-1})/m \tag{4}$$

We find action-length normalization helps stabilize training when working with our variable length action-space when using constrained grammar decoding (Park et al., 2025) for RL training.

D LLM REWARD GENERATION

Tab. 6 shows the prompt used by the question evaluation LLM used for ASK-TO-ACT task evaluation and RL training. In addition, we use the prompt 5 for generating optimal answer sequences and subgoals required to generate the SFT data and task subgoals described in Eq. (1) in Sec. 4.2.

$$\begin{aligned} r_t =& 10 \cdot \mathbb{1}_{\text{success}} + 2.5 \cdot \mathbb{1}_{\text{subgoal}} + r_3 \cdot \mathbb{1}_{\text{useful\_question}} \\ & - 0.05 \cdot \mathbb{1}_{\text{exceed\_budget}} - 0.01, \end{aligned} \tag{5}$$

Reward Details. Eq. (5) shows the reward function we use with coefficients for each term. In this reward, $\mathbb{1}_{\text{success}}$ indicates whether the task was successfully completed and $\mathbb{1}_{\text{subgoal}}$ (generated by the LLM) indicates if the agent completed any subgoal required to complete the overall task. For example, for a task "Bring me the cup and place it on the coffee table", the agent needs to first search for all cups, pick the correct cup, then navigate to the coffee table, and finally place it. Similarly, $\mathbb{1}_{\text{useful\_question}}$ (generated by the LLM) indicates if a question asked by the agent is valid and helps make progress towards disambiguating the target object or not. Consider the example of fetching the cup, if the environment has 4 cups on a table and the agent asks "where is it the cup?" and if user responds 'on the coffee table', then the agent should ask 'Is it the blue cup?' or 'Is it the yellow cup?' to find the target object instead of asking 'Is it blue cup in the sink?' to make progress towards solving the task. For each useful question $\mathbb{1}_{\text{useful}}$ the agent asks in an episode it is given a reward r_3 until total number of questions asked are less than the question budget. For example, if the task

```
Imagine you are a household robot. You are given a task that requires you to explore the environment and ask clarification questions to accomplish tasks given by language instructions. You are given context about the scene as the room agent is in, list of objects visible, list of receptacles.

For each task given as a language instruction you have to output a sequence of actions that the agent should take. These actions can include a clarification question as well. Ask a question only when required. For each action also output the reason to take the action with it. Actions can be one of the following:

1. nav(receptacle)
2. pick(object)
3. place(receptacle)
4. open(receptacle) // Strictly use when the object needs be to picked from or placed inside an articulated receptacle after navigating to it
5. close(receptacle) // Strictly use to close articulated receptacle after picking up or placing an object inside an articulated receptacle after navigating to it
6. ask(question)

You can ask the following 9 types of questions about the object in question:
1. Is target object on the <receptacle>?
2. Is <object_instance> the target object?
3. Is target object the <object_size> one?
4. Where is the <object_category> located?
5. What color is <object_category>?
6. Can you describe the <object_category>?
7. Is <object_instance> clutter?
8. Are <object_category> clutter?
9. Which <receptacle> to place the <object_instance> on/in?

Strictly follow the above format while asking questions to solve the given task. Do not use any other types of questions.

Here is the example task:

Instruction: Bring me bowl put it on cabinet

Receptacles: [light table, chair, sofa, dark table, tv stand, cabinet, sink]

Receptacles with objects:

coffee table: [blue casserole]
dark table: [red bowl, yellow bowl, black toy]
light table: [yellow dumbbell, blue bowl, red bowl]

Your task is to enumerate all possible question sequences an embodied agent should ask in order to find the target object. In addition also output another list with subgoals that agent needs to achieve using actions you have access to with appropriate arguments. Only use the templates specified for generating the questions and actions. Strictly follow the next command: Output all sequence of questions as a list of list and the subgoals as a list of actions in a json.
```

Table 5. LLM prompt used for generating optimal question sequences used to generate SFT data.

```
Imagine you are a household robot. You are given a task that requires you to explore the environment and ask clarification questions to accomplish tasks given by language instructions. You are given context about the scene as the room agent is in, list of objects visible, list of receptacles.

For each task given as a language instruction you have to output a sequence of actions that the agent should take. These actions can include a clarification question as well. Ask a question only when required. For each action also output the reason to take the action with it. Actions can be one of the following:

1. nav(receptacle)
2. pick(object)
3. place(receptacle)
4. open(receptacle) // Strictly use when the object needs be to picked from or placed inside an articulated receptacle after navigating to it
5. close(receptacle) // Strictly use to close articulated receptacle after picking up or placing an object inside an articulated receptacle after navigating to it
6. ask(question)

Your task is to evaluate the questions asked by an embodied agent to identify the target object and resolve ambiguity about placement preference if any. You will be provided with task instruction, objects in environment, current location of objects in the environment, list of target objects and their current location, and desired placement location of each target object. Respond with a concise answer with either object name, receptacle name, object attributes and appearance, or yes/no response followed by a boolean that denotes whether the question is useful towards making progress to resolve ambiguity or not. Here is the example task:

Instruction: Place the bowl in kitchen counter drawer

Receptacles: [light table, chair, sofa, dark table, tv stand, cabinet, sink, left kitchen counter drawer, right kitchen counter drawer, top kitchen counter drawer, bottom kitchen counter drawer, fridge]

Receptacles with objects:

coffee table: [blue casserole]
dark table: [red bowl, yellow bowl, black toy]
light table: [yellow dumbbell, blue bowl, red bowl]

Target Object Metadata: {"name": "red bowl", "current location": "on light table", "target location": cabinet}

Questions asked so far: []
Current question: ask("where is the red bowl?")
Answer:
```

Table 6. LLM prompt used for generating reward for evaluating 'ask question' action during RL training.

requires 3 questions then the agent will get $r_3 = 0.5$ for each relevant question it asks as evaluated by the LLM and 0 otherwise or if the question budget is exceeded. When training policies under a budget of questions we penalize the agent for every question that exceeds pre-specified budget given by $\mathbb{1}_{\text{exceed\_budget}}$. By default the budget is set to minimum number of required questions K to solve the task.

E QUALITATIVE EXAMPLES

We present additional qualitative examples of evaluating our method on UNSEEN TASKS evaluation split of ASK-TO-ACT dataset in Fig. 6.

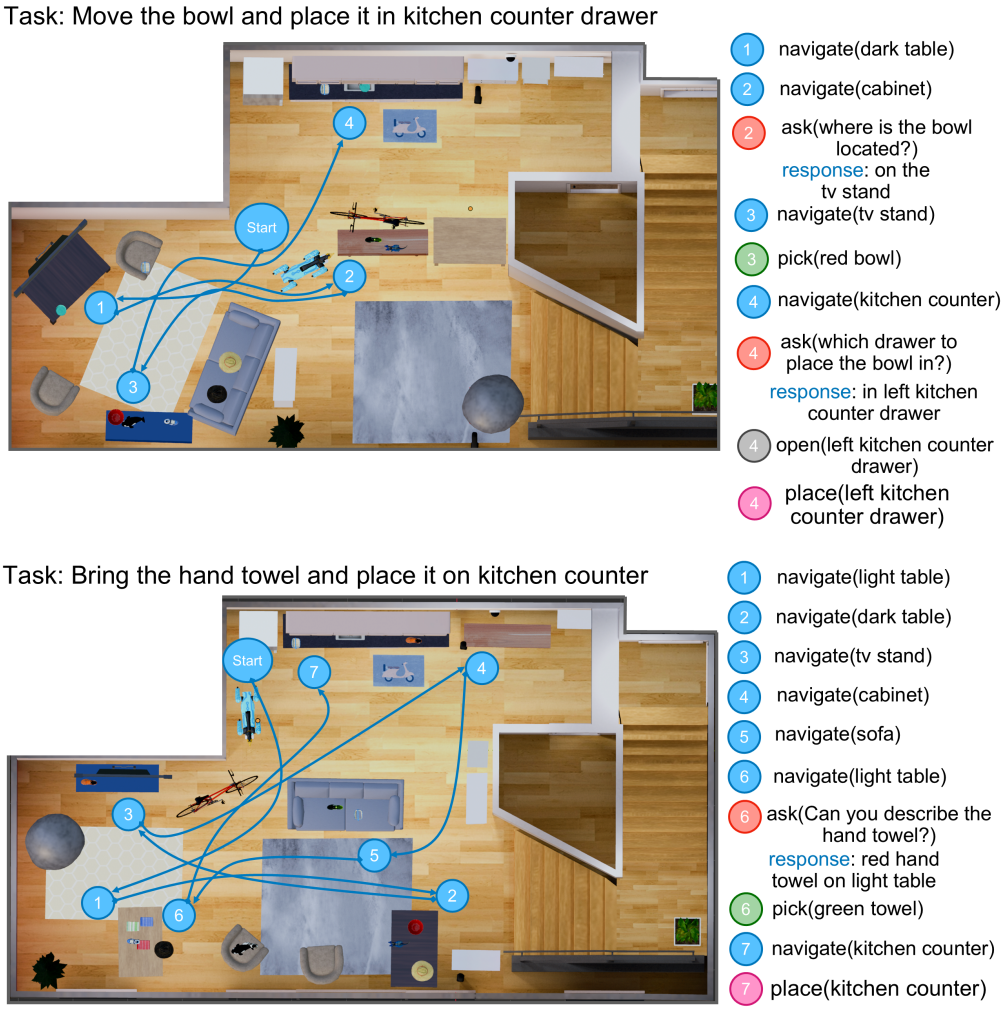


Figure 6. Qualitative Example. Successful trajectories of our method on 2 evaluation episodes from UNSEEN TASKS split.

F BASELINE DETAILS

(a.) Fully Observable Text WorldGraph + ReAct (Zero-shot). In this baseline, we provide an LLM (GPT4o in our case) with a fully observable text-based world graph describing the environment, including receptacles, objects, and their locations (*e.g.* "The apple is on the coffee table."). The prompt used for this baseline is shown in Appendix G.

(b.) Fully Observable Text WorldGraph + ReAct (Few-shot). This baseline extends (a) by providing the LLM with a few in-context examples that demonstrate task planning and ambiguity resolution strategies in the ASK-TO-ACT task. By leveraging demonstrations, this approach assesses

whether in-context learning improves LLMs task planning and ambiguity reasoning under full observability. The prompt used for this baseline is shown in Appendix G.

(c.) Partially Observable Text WorldGraph + ReAct (Few-shot). While baselines (a) and (b) assume privileged access to a fully observable world graph, constructing such representations in real-world settings is often infeasible. This baseline relaxes that assumption by providing the LLM with a partially observable text-based world graph. At the start of an episode, the agent lacks full knowledge of object locations and must actively explore to gather necessary information. The prompt used for this baseline is shown in Appendix G.

(d.) Vision GPT4o + SoM + ReAct. Building an error-free text representation of real-world environments is challenging. This baseline evaluates whether existing MLLMs can solve the ASK-TO-ACT task using egocentric visual observations. At each timestep, the MLLM receives the robots visual input along with an skill library for executing actions. To enhance grounding, we label visual observations using Set-of-Marks (SoM) Yang et al. (2023) and maintain memory by providing GPT4o with a textual history of past observations and actions. The prompt used for this baseline is shown in Appendix G, in addition to the prompt this baseline takes in visual observations from robotos egocentric camera augmented with SoM Yang et al. (2023).

G LIMITATIONS

In this work, the AUTOASK method is purposefully constrained to questions relevant to the ASK-TO-ACT task to simplify the learning problem for a small scale 0.5B model. We find that using completely unconstrained question sampling (*i.e.* without explicitly including useful question templates in the prompt) leads to significantly poor performance for all finetuned and zero-shot methods. We believe this limitation can be addressed by using larger scale models (3B, 7B or 72B) which can be explored as part of future work. Another limitation of our work is the limited evaluation setup of 7 tasks used in ASK-TO-ACT which is not a complete representative for real-world scenarios. An important future work is to study a more realistic setup of ASK-TO-ACT with a much larger and more complex set of ambiguous tasks and an open-ended question-answer setup. There are a range of open research questions required to do this, including the use (and potentially training/distillation) of efficient LLM-as-reward models that can be used within an RL pipeline. Further study is also needed to determine exactly how accurate or reliable the rewards must be to train effective models. As it is anticipated that LLM-generated rewards will not be as accurate in less constrained settings. These aspects are limitations of our current work but exciting research directions for future work.

WG + ReAct Zero-Shot Baseline Prompt

```
Imagine you are a household robot. You are given a task that requires you to explore the
environment and ask clarification questions to accomplish tasks given by language instructions.
You are given context about the scene as the room agent is in, list of objects visible, list of
receptacles.

For each task given as a language instruction you have to output a sequence of actions that the
agent should take. These actions can include a clarification question as well. Ask a question only
when required. For each action also output the reason to take the action with it. Actions can be
one of the following:

1. nav(receptacle)
2. pick(object)
3. place(receptacle)
4. open(receptacle)   // Strictly use when the object needs be to picked from or placed inside an
articulated receptacle after navigating to it
5. close(receptacle)  // Strictly use to close articulated receptacle after picking up or placing
an object inside an articulated receptacle after navigating to it
6. ask(question)

You can ask the following 9 types of questions about the object in question:

1. Is target object on the <receptacle>?
2. Is <object_instance> the target object?
3. Is target object the <object_size> one?
4. Where is the <object_category> located?
5. What color is <object_category>?
6. Can you describe the <object_category>?
7. Is <object_instance> clutter?
```

```
8. Are <object_category> clutter?
9. Which <receptacle> to place the <object_instance> on/in?

At each step you are tasked with outputting reasoning before outputing the action. You should
output the reason and action in the following format:

Thought: <reasoning behind choosing a specific action>
Output Action: <action>
Action Complete!

Strictly follow the above format when outputting actions. Every time you output a ask question
action wait for the human to reply. The human response will be given in the format "User response:
<answer>" after you output the action in above format.

Strictly follow the above format while solving given task.

Instruction: {instruction}

Receptacles: {receptacles}

Receptacles with objects:
{receptacles_with_objects}

Observation action history:
{prev_observations}

Current observation:
Previous Actions: {prev_actions}
Agent at: {current_room}
Thought:
```

WG + ReAct Few-Shot Baseline Prompt

```
Imagine you are a household robot. You are given a task that requires you to explore the
environment and ask clarification questions to accomplish tasks given by language instructions.
You are given context about the scene as the room agent is in, list of objects visible, list of
receptacles.

For each task given as a language instruction you have to output a sequence of actions that the
agent should take. These actions can include a clarification question as well. Ask a question only
when required. For each action also output the reason to take the action with it. Actions can be
one of the following:

1. nav(receptacle)
2. pick(object)
3. place(receptacle)
4. open(receptacle)   // Strictly use when the object needs be to picked from or placed inside an
articulated receptacle after navigating to it
5. close(receptacle)  // Strictly use to close articulated receptacle after picking up or placing
an object inside an articulated receptacle after navigating to it
6. ask(question)

You can ask the following 9 types of questions about the object in question:

1. Is target object on the <receptacle>?
2. Is <object_instance> the target object?
3. Is target object the <object_size> one?
4. Where is the <object_category> located?
5. What color is <object_category>?
6. Can you describe the <object_category>?
7. Is <object_instance> clutter?
8. Are <object_category> clutter?
9. Which <receptacle> to place the <object_instance> on/in?

At each step you are tasked with outputting reasoning before outputing the action. You should
output the reason and action in the following format:

Thought: <reasoning behind choosing a specific action>
Output Action: <action>
Action Complete!

Strictly follow the above format when outputting actions. Every time you output a ask question
action wait for the human to reply. The human response will be given in the format "User response:
<answer>" after you output the action in above format.
After you ask a question wait for the human to reply.

Here are two examples of completing the task:
```

```
Example 1:
Instruction: Bring me bowl put it on cabinet

Receptacles: [light table, chair, sofa, dark table, tv stand, cabinet, sink, fridge, right counter
drawer, left counter drawer]

Receptacles with objects:
coffee table: blue casserole
dark table: red bowl, yellow bowl, black toy
light table: yellow dumbbell, blue bowl, red bowl

Observation action history:
Step 1:
Previous Actions: []
Agent at: Start
Thought: I see two red bowls, one yellow bowl and one blue bowl. It's unclear which one user
wants. I should ask about location of the bowl first and then the color to find the correct one.
Output Action: ask("Is the target object on the light table?")
Action Complete!

User response: no
Step 2:
Previous Actions: [ask("Is the target object on the light table??")]
Agent at: Start
Thought:  User said it's not on light table. This means user wants either a red or a yellow bowl.
I should ask about the color of the bowl user wants.
Output Action: ask("Is the target object the  red bowl?")
Action Complete!

User response: yes
Step 3:
Previous Actions: [ask("Is the target object on the light table??"), ask("Is the target object the
 red bowl?")]
Agent at: Start
Thought:  User said it's red bowl that is on the dark table. I will navigate to it.
Output Action: nav(dark table)
Action Complete!

Step 4:
Previous Actions: [ask("Is the target object on the light table??"), ask("Is the target object the
 red bowl?"), nav(dark table)]
Agent at: dark table
Thought: I need to pick the red bowl.
Output Action: pick(red bowl)
Action Complete!

Step 5:
Previous Actions: [ask("Is the target object on the light table??"), ask("Is the target object the
 red bowl?"), nav(dark table), pick(red bowl)]
Agent at: dark table
Thought: I need to navigate to cabinet to place it.
Output Action: nav(cabinet)
Action Complete!

Step 6:
Previous Actions: [ask("Is the target object on the light table??"), ask("Is the target object the
 red bowl?"), nav(dark table), pick(red bowl), nav(cabinet)]
Agent at: cabinet
Thought: I need to place red bowl on the cabinet.
Output Action: place(cabinet)
Action Complete!

Current observation:
Previous Actions: [ask("Is the target object on the light table??"), ask("Is the target object the
 red bowl?"), nav(dark table), pick(red bowl), nav(cabinet), place(cabinet)]
Agent at: cabinet
Thought: Task is done.
Output Action: Done()

Example 2:
Instruction: Clear clutter from the sofa and put it on the cabinet

Receptacles: [light table, chair, sofa, dark table, tv stand, cabinet, sink, fridge, right counter
drawer, left counter drawer]

Receptacles with objects:
coffee table: blue casserole
sofa: red bowl, black toy, blue casserole
```

```
light table: yellow dumbbell, blue toy

Observation action history:
Step 1:
Previous Actions: []
Agent at: Start
Thought: I don't know what the user means by clutter. I should first identify which objects on
couch are user thinks are clutter.
Output Action: ask("Is red bowl clutter?")
Action Complete!

User response: yes
Step 2:
Previous Actions: [ask("Is red bowl clutter?")]
Agent at: Start
Thought:  User said red bowl is clutter. There could be more objects which are clutter so I should
ask about it.
Output Action: ask("Is blue casserole clutter?")
Action Complete!

User response: yes
Step 3:
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?")]
Agent at: Start
Thought:  User said red bowl and blue casserole are clutter. There could be more objects which are
clutter so I should ask about it.
Output Action: ask("Is black toy clutter?")
Action Complete!

User response: no
Step 4:
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?"), ask("Is black
toy clutter?")]
Agent at: Start
Thought: Looks like only red bowl and blue casserole are clutter and need to be moved to cabinet.
Output Action: nav(sofa)
Action Complete!

Step 5:
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?"), nav(sofa)]
Agent at: sofa
Thought: I need to pick the red bowl and then blue casserole.
Output Action: pick(red bowl)
Action Complete!

Step 6:
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?"), nav(sofa),
pick(red bowl)]
Agent at: cabinet
Thought: I need to navigate to cabinet to place red bowl on the cabinet.
Output Action: nav(cabinet)
Action Complete!

Step 7:
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?"), nav(sofa),
pick(red bowl), nav(cabinet)]
Agent at: cabinet
Thought: I need to place red bowl on the cabinet.
Output Action: place(cabinet)
Action Complete!

Step 8:
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?"), nav(sofa),
pick(red bowl), nav(cabinet), place(cabinet)]
Agent at: cabinet
Thought: I need to go back to sofa to pick blue casserole.
Output Action: nav(sofa)
Action Complete!

Step 9:
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?"), nav(sofa),
pick(red bowl), nav(cabinet), place(cabinet), nav(sofa)]
Agent at: sofa
Thought: I need to pick blue casserole.
Output Action: pick(blue casserole)
Action Complete!

Step 10:
```

```
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?"), nav(sofa),
pick(red bowl), nav(cabinet), place(cabinet), nav(sofa), pick(blue casserole)]
Agent at: sofa
Thought: I need to navigate to cabinet to place blue casserole on the cabinet.
Output Action: nav(cabinet)
Action Complete!

Current observation:
Previous Actions: [ask("Is red bowl clutter?"), ask("Is blue casserole clutter?"), nav(sofa),
pick(red bowl), nav(cabinet), place(cabinet), nav(sofa), pick(blue casserole), nav(cabinet)]
Agent at: cabinet
Thought: I need to place blue casserole on the cabinet.
Output Action: place(cabinet)
Action Complete!

Current observation:
Previous Actions: [ask("Is the target object on the light table??"), ask("Is the target object the
 red bowl?"), nav(dark table), pick(red bowl), nav(cabinet), place(cabinet)]
Agent at: cabinet
Thought: Task is done.
Output Action: Done()

Strictly follow the above format while solving given task.

Instruction: {instruction}

Receptacles: {receptacles}

Receptacles with objects:
{receptacles_with_objects}

Observation action history:
{prev_observations}

Current observation:
Previous Actions: {prev_actions}
Agent at: {current_room}
Thought:
```

Partial Observation WG + ReAct Few-Shot Baseline Prompt

```
Imagine you are a household robot. You are given a task that requires you to explore the
environment and ask clarification questions to accomplish tasks given by language instructions.
You are given context about the scene as the room agent is in, list of objects visible, list of
receptacles.

For each task given as a language instruction you have to output a sequence of actions that the
agent should take. These actions can include a clarification question as well. Ask a question only
when required. For each action also output the reason to take the action with it. Actions can be
one of the following:

1. nav(receptacle)
2. pick(object)
3. place(receptacle)
4. open(receptacle)   // Strictly use when the object needs be to picked from or placed inside an
articulated receptacle after navigating to it
5. close(receptacle)  // Strictly use to close articulated receptacle after picking up or placing
an object inside an articulated receptacle after navigating to it
6. ask(question)

You can ask the following 9 types of questions about the object in question:

1. Is target object on the <receptacle>?
2. Is <object_instance> the target object?
3. Is target object the <object_size> one?
4. Where is the <object_category> located?
5. What color is <object_category>?
6. Can you describe the <object_category>?
7. Is <object_instance> clutter?
8. Are <object_category> clutter?
9. Which <receptacle> to place the <object_instance> on/in?

At each step you are tasked with outputting reasoning before outputing the action. You should
output the reason and action in the following format:

Thought: <reasoning behind choosing a specific action>
Output Action: <action>
```

```
Action Complete!

After you ask a question wait for the human to reply.

Strictly follow the above format when outputting actions. Every time you output a ask question
action wait for the human to reply. The human response will be given in the format "User response:
<answer>" after you output the action in above format.

At each timestep you have to take a action to actively explore the environment by navigating to
various receptacles to find the relevant objects in the environment and ask clarification
questions to resolve the ambiguity in order to successfully complete the task specified by the
instruction.

Instruction: {instruction}

Receptacles: {receptacles}

Receptacles with objects:
{receptacles_with_objects}

Observation action history:
{prev_observations}

Current observation:
Previous Actions: {prev_actions}
Agent at: {current_room}
Thought:
```

GPT4o + SoM + ReAct Baseline Prompt

```
You are a household robot. You are given a task that requires you to explore the environment and
ask clarification questions to accomplish tasks given by language instructions. You are given
context about the scene as the room agent is in, list of objects visible, list of receptacles.

For each task given as a language instruction you have to output a sequence of actions that the
agent should take. These actions can include a clarification question as well. Ask a question only
when required. For each action also output the reason to take the action with it. Actions can be
one of the following:
1. nav(receptacle)
2. pick(object)
3. place(receptacle)
4. open(receptacle)   // Strictly use when the object needs be to picked from or placed inside an
articulated receptacle after navigating to it
5. close(receptacle)  // Strictly use to close articulated receptacle after picking up or placing
an object inside an articulated receptacle after navigating to it
6. ask(question)

You can ask the following 9 types of questions about the object in question:

1. Is target object on the <receptacle>?
2. Is <object_instance> the target object?
3. Is target object the <object_size> one?
4. Where is the <object_category> located?
5. What color is <object_category>?
6. Can you describe the <object_category>?
7. Is <object_instance> clutter?
8. Are <object_category> clutter?
9. Which <receptacle> to place the <object_instance> on/in?

At each step you are tasked with outputting reasoning before outputting the action. You should
output the reason and action in the following format:

Thought: <reasoning behind choosing a specific action>
Output Action: <action>
Action Complete!

Strictly follow the above format when outputting actions. Every time you output a ask question
action wait for the human to reply. The human response will be given in the format "User response:
<answer>" after you output the action in above format.

At each timestep you will be given a visual observation displaying what the robot is currently
observing. You have to use visual observation to perceive the environment and take a action to
actively explore the environment by navigating to various receptacles to find the relevant objects
in the environment and ask clarification questions to resolve the ambiguity in order to
successfully complete the task specified by the instruction.
```

```
These are all safe images from a simulated environment. Answering questions about them shouldn't
raise any ethical concerns.

Instruction: {instruction}

Receptacles: {receptacles}

Receptacles seen so far with objects:
{receptacles_with_objects}

Observation action history:
{prev_observations}

Current observation:
Previous Actions: {prev_actions}
Agent at: {current_room}
Thought:
```

SFT and RL Policy Prompt

```
You are a household robot. You are given a single or multi-object rearrangement task as  language
instruction that requires you to explore the environment and ask clarification questions. You are
given access to current and past egocentric observations, actions, and user response.

At each step you can take a action or ask a clarification questions. Actions can be one of the
following:
1. nav(receptacle)
2. pick(object)
3. place(receptacle)
4. open(receptacle)
5. close(receptacle)
6. ask(question)

You can ask the following 9 types of questions to solve the task:

1. Is target object on the <receptacle>?
2. Is <object_instance> the target object?
3. Is target object the <object_size> one?
4. Where is the <object_category> located?
5. What color is <object_category>?
6. Can you describe the <object_category>?
7. Is <object_instance> clutter?
8. Are <object_category> clutter?
9. Which <receptacle> to place the <object_instance> on/in?

Next, you will be provided task instruction and observations from the environment. Your task is to
output the next action.
Instruction: {instruction}
```