# Conditional Multi-Stage Failure Recovery for Embodied Agents

**Youmna Farag** **Svetlana Stoyanchev** **Mohan Li** **Simon Keizer** **Rama Doddipatla**
Cambridge Research Laboratory, Toshiba Europe Ltd, Cambridge, UK
{youmna.farag, svetlana.stoyanchev, mohan.li,
simon.keizer, rama.doddipatla}@toshiba.eu

## Abstract

Embodied agents performing complex tasks are susceptible to execution failures, motivating the need for effective failure recovery mechanisms. In this work, we introduce a conditional multi-stage failure recovery framework that employs zero-shot chain prompting. The framework is structured into four error-handling stages, with three operating during task execution and one functioning as a post-execution reflection phase. Our approach utilises the reasoning capabilities of LLMs to analyse execution challenges within their environmental context and devise strategic solutions. We evaluate our method on the TfD benchmark of the TEACH dataset and achieve state-of-the-art performance, outperforming a baseline without error recovery by 11.5% and surpassing the strongest existing model by 19%.

## 1 Introduction

In embodied AI settings, autonomous agents are required to perform complex tasks within environments such as homes or offices. In these settings, Large Language Models (LLMs) have been employed to decompose natural language instructions (e.g., *make breakfast*) into a plan of executable actions (e.g., *pick_up(Cup)*), with the objective of ensuring successful task completion (Huang et al., 2022b; Ahn et al., 2022; Huang et al., 2022a; Wang et al., 2023b). Prior research has explored generating plans that are robust to failures through few-shot prompting, utilizing an underlying memory of demonstrations (Zhao et al., 2023; Song et al., 2023; Wang et al., 2023a; Sarch et al., 2023, 2024; Fu et al., 2024). However, the initial LLM-generated plan does not inherently ensure successful task completion, as (a) the plan may contain errors, such as missing or incorrect steps, and (b) the agent may encounter unforeseen challenges within the environment that are difficult to anticipate in the planning phase. Thus, grounding the plan within the environmental context and integrating error recovery mechanisms are essential for enabling the agent to adapt and re-plan to address execution challenges.

This motivates incorporating feedback from the environment for more robust planning and error recovery. For example, existing approaches incorporated visual information represented through image embeddings (Pashevich et al., 2021; Singh et al., 2022; Brohan et al., 2023; Driess et al., 2023; Sarch et al., 2023) or structured scene descriptions[1] (Min et al., 2021; Zhang et al., 2022; Liang et al., 2023; Singh et al., 2023; Kim et al., 2023; Liu et al., 2023b). Other work used human feedback to correct the agent's behavior (Abramson et al., 2022; Huang et al., 2022b; Philipov et al., 2024). Another form of feedback involves verifying action preconditions, which can either be explicitly encoded within the execution module, requiring domain-specific expertise (Zheng et al., 2022b; Zhang et al., 2022; Sarch et al., 2023; Fu et al., 2024), or learned via reinforcement learning methods (Ahn et al., 2022). However, prior research has not systematically examined the structured process of error recovery or devised strategic frameworks for how agents should handle execution challenges.

We propose a Conditional Multi-stage Failure Recovery (CMFR) approach for embodied agents that uses zero-shot chain prompting.[2] Chain prompting decomposes a complex task into a sequence of interdependent prompts, where the output of one prompt serves as input for the next (Wu et al., 2022). Our method leverages LLMs to assess execution challenges in the given environmental context and devise strategic solutions. CMFR is structured across four distinct stages utilised both during and after task execution. Our approach stands out by leveraging the reasoning abilities of

[1] such as the list of observed objects along with their properties and locations

[2] https://github.com/Youmna-H/CMFR_TEACH



*Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pages 200–227

LLMs in a zero-shot manner without relying on external modules, such as example-based memory or domain-specific precondition checks.

We evaluate the CMFR approach on the TEACH benchmark (Padmakumar et al., 2022), which encompasses a diverse set of long-horizon household tasks. Our experimental results demonstrate that our CMFR approach enhances task success by 11.5%, achieving state-of-the-art results and surpassing existing models by a significant margin. In addition, we integrate and evaluate an LLM-based object search mechanism that leverages object locations mentioned in task dialogues. Finally, we conduct an ablation study to highlight the contribution of each stage in CMFR. Our approach provides a structured framework for reasoning about and overcoming execution challenges, contributing to research on embodied agents designed to assist humans in household tasks.

## 2 Related Work

**Embodied AI** A substantial body of research has focused on developing embodied agents capable of translating natural language instructions into executable actions, leveraging various simulators and benchmarks designed to support such tasks (Kolve et al., 2017; James et al., 2020; Manolis Savva* et al., 2019; Shridhar et al., 2020; Padmakumar et al., 2022; Zheng et al., 2022a; Gao et al., 2022; Li et al., 2023a). Some approaches have focused on fine-tuning multimodal models to encode linguistic and visual inputs for predicting low-level actions (Anderson et al., 2018; Shridhar et al., 2020; Ku et al., 2020; Min et al., 2021; Pashevich et al., 2021; Singh et al., 2022; Brohan et al., 2022; Padmakumar et al., 2022; Zheng et al., 2022b; Shridhar et al., 2022; Zheng et al., 2022a; Driess et al., 2023; Brohan et al., 2023). Other work has leveraged prompting techniques to use LLMs as planners for embodied tasks (Huang et al., 2022a,b; Ahn et al., 2022; Liang et al., 2023; Wang et al., 2023b; Singh et al., 2023; Liu et al., 2023b; Song et al., 2023; Sarch et al., 2023, 2024; Fu et al., 2024). While LLMs excel at reasoning and decomposing complex tasks into actionable steps, they require environmental grounding to address execution challenges. To achieve this, various approaches have been proposed to integrate environmental feedback into LLM-based planning. Some studies utilize perception models or ground-truth simulator data to generate scene descriptions, which are then incorporated into LLM prompts (Huang et al., 2022b; Wang et al., 2023b; Song et al., 2023; Singh et al., 2023; Liang et al., 2023; Liu et al., 2023b). Others employ vision-language models to classify scene images based on predefined failure categories (Sarch et al., 2023, 2024; Fu et al., 2024), while additional research explores learning affordance functions through reinforcement learning (Ahn et al., 2022).

**Chain Prompting** Prior research has applied chain prompting to various tasks including summarization (Zhang et al., 2023; Sun et al., 2024), information extraction (Kwak et al., 2024), classification (Trautmann, 2023), and language generation (Firdaus et al., 2023; Maity et al., 2024). However, to the best of our knowledge, chain prompting has not been explored in the context of embodied AI for enabling agents to address execution challenges. Moreover, our proposed approach employs a conditional chain prompting mechanism, where the activation of each stage is contingent upon the output of the preceding stage.

## 3 Problem Definition

Task-driven embodied agents that chat (TEACH) (Padmakumar et al., 2022) is a dataset focused on long-horizon tasks in household environments. It comprises over 3,000 gameplay episodes built on top of AI2-THOR simulator (Kolve et al., 2017). An episode consists of a human–human interactive dialogue between a *Commander* that has oracle information about the task and a *Follower* (agent) that tries to complete the task by navigating and interacting with objects in the simulated environment. TEACH comprises 12 household tasks with varying granularity (Appendix D).[3] Furthermore, TEACH includes three benchmarks: (1) Trajectory from Dialog (TfD): where given the full dialogue history of the task, the agent predicts the sequence of actions that completes the task successfully. (2) Execution from Dialogue History (EDH): where the task dialogue in TfD is segmented into sessions and the agent is asked to predict the actions that lead to the next session, and (3) Two-Agent Task Completion (TATC) where both the commander and follower are modeled to perform the task.

In this work, we focus on the TfD benchmark

[3]For example, some tasks such as *Prepare Breakfast* includes other tasks such as *Make Coffee* or *Make a Plate of Toast.*

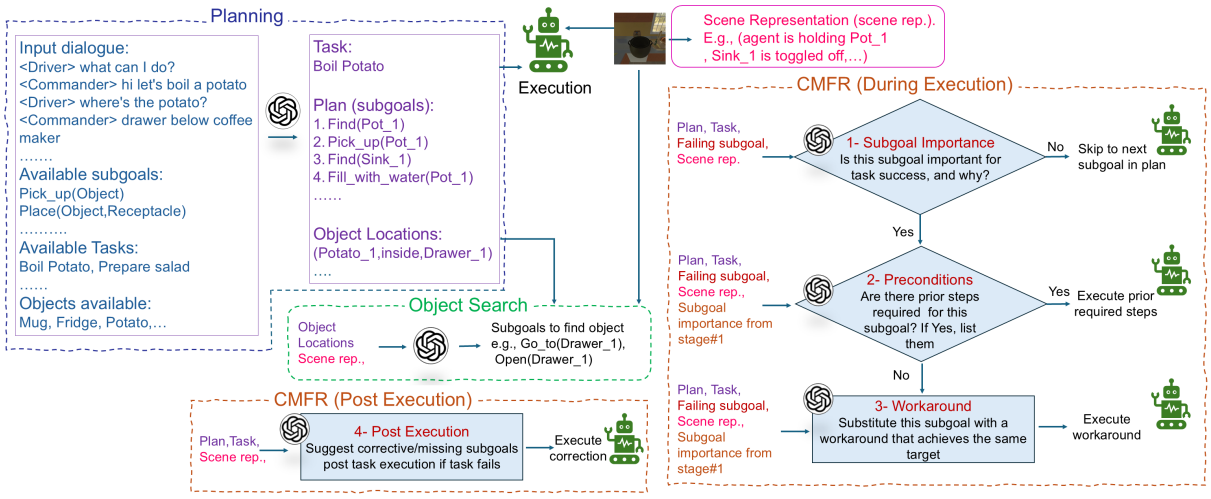



Figure 1: System components: planning, the four CMFR stages, execution, object search and scene representation. The output of one component serves as input for another (e.g., 'Plan' and 'Task' from planning are used in all CMFR stages, and 'subgoal importance' from stage 1 is used as input for stages 2 and 3).

as it poses several challenges. For instance, unlike other datasets that give a single instruction or a high-level task to the agent (Huang et al., 2022a,b; Ahn et al., 2022; Song et al., 2023; Singh et al., 2023; Lin et al., 2023; Liu et al., 2023a), the input in TfD is a noisy dialogue that contains information about the task, making it more challenging for the agent to extract relevant task information and convert that to a sequence of executable steps. Moreover, in TfD, the average number of actions the human agent takes to solve a task is 117, demonstrating the complexity and long-horizon nature of the tasks. For evaluation, we use the following metrics (Shridhar et al., 2020). **Success Rate (SR)** which is the fraction of episodes in which all task goal-conditions are fulfilled.[4] **Goal-Condition Success (GC)** which is the ratio of the completed goal-conditions to those necessary to succeed in the task. **Path Length Weighted (PLW)** where both SR and GC metrics have a path length weighted counterpart which penalises the agent for taking more steps than the human-annotated reference path. More details about the dataset are presented in Appendix A. The task fails if the agent exceeds 1000 actions or 30 failed actions.

## 4 Approach

We depict the components of our approach in Figure 1 and detail them in this section.

[4]All object positions and state changes have to correspond correctly to the task goal-conditions (e.g., the task *Make Coffee* has two goal-conditions: a cup has to be clean and it has to be filled with coffee).

### 4.1 Planning

The initial phase of our system is planning, where the LLM is prompted to generate a plan of the subgoals necessary to succeed at the task. As demonstrated in Figure 1, we prompt the LLM with the input dialogue, list of subgoals/actions the agent is able to execute in the environment, list of TEACH tasks and list of object categories available in AI2-THOR (full lists are included in Appendix D). The LLM is asked to generate a plan using the subgoals and object categories specified in the prompt. Along with generating the plan, the LLM classifies the dialogue into one of the tasks provided in the prompt and extracts object locations if any are mentioned in the dialogue to be used by other system components (Sections 4.3 and 4.5). We include further details about planning in Appendix B and the initial planning prompt in Appendix G Listing 1.

### 4.2 Execution

The subgoals generated by the LLM planner are passed to the Executor module to be executed one by one in the simulated environment. As the agent moves around and interacts with objects to execute the plan, it maintains a memory of the objects observed at each time step along with their properties and locations. Perception models could be used for object detection (Dong et al., 2021) and depth estimation (Bhat et al., 2023). However, as we do not aim in this work to develop or test perception models, we use ground-truth information about objects provided by the simulator (Wang et al., 2023b; Liu et al., 2023b). It is worth noting that working with

simulated environments presents certain challenges particularly in positioning the agent to interact with an object. Following prior research, we employ heuristics to adjust the agent's position before interacting with an object (Padmakumar et al., 2023; Sarch et al., 2023; Fu et al., 2024). However, while these techniques enhance interaction success, they are not entirely fail proof. More details about the executor are included in Appendix C.

### 4.3 Object Search

During execution, the agent looks up its memory to find information about target objects required for interaction. If the object is not found, object search is triggered by prompting the LLM (Appendix G Listing 7) to generate steps to find the object using the object locations generated at the planning stage, if any (as depicted in Figure 1). For instance, if the dialogue mentions that a potato is inside the fridge and this information is extracted by the planner (i.e., generating (Potato_1,inside,Fridge_1), the search module should accordingly generate the steps to locate the potato (e.g., Go_to(Fridge_1), Open(Fridge_1)). This module, therefore, depends on whether (1) the dialogue contains information about object locations and (2) this information was extracted successfully by the planner. Previous work on TEACH have used random search (Zhang et al., 2022), transformer models trained on training data (Zheng et al., 2022b), or the commonsense knowledge of LLMs to locate objects (Sarch et al., 2023; Fu et al., 2024). We do not rely on the commonsense knowledge of LLMs for object search as in TEACH, objects are initialised at random positions and therefore may appear in implausible or nonsensical locations (e.g., a potato being placed in the garbage bin or a saltshaker placed in the sink). That is why we only use object locations mentioned in the dialogue.

### 4.4 Scene Representation

When the agent fails to perform an action, visual information from the environment is crucial to identify the reason and determine the solution. For example, without visual cues, the agent might not realize that an object placement failed because the receptacle is full and needs to be emptied before retrying the action. Therefore, we build a scene representation that stores visual information about the environment (Min et al., 2021; Zhang et al., 2022; Kim et al., 2023; Singh et al., 2023; Liu et al., 2023b) and utilise that for error recovery (as will be elaborated in Section 4.5).

As mentioned in Section 4.2, the agent maintains a memory of observed objects. Scene representation is a pruned version of this memory in order to keep prompts at a reasonable length. Specifically, we only extract from the memory the relevant objects mentioned in the plan and only keep (1) their properties that are relevant to TEACH tasks,[5] (2) the parent objects that contain them and (3) the child objects they enclose, if any. Furthermore, we add information about what object the agent is currently holding in hand, if any. Examples of scene representations are included in Appendix E.

### 4.5 Conditional Multi-stage Failure Recovery

The initial plan generated in 4.1 does not guarantee task success due to missing or incorrect steps in the plan or newly observed input from the environment that must be taken into account. Therefore, the agent may encounter execution failures triggering the need for failure recovery. We propose a conditional multi-stage failure recovery (CMFR) approach to enable the agent to assess its current situation by considering its objectives, progress made thus far, and the surrounding environment (see Figure 1). Accordingly, the agent can formulate an effective strategy to resolve the current situation. CMFR is divided into four stages. The first three stages operate at the subgoal level, addressing subgoal failures as they occur during execution. The final stage functions at the task level when the agent finishes plan execution yet fails at the task. All the stages use zero-shot prompts and are detailed in the remainder of this section. The prompts used for CMFR are added in Appendix G Listings 3, 4, 5 and 6.

**Stage 1: Subgoal Importance** This stage is the entry point to failure recovery and is triggered when the agent fails in executing one of the plan subgoals. Minimizing failures is essential in embodied settings to ensure both efficiency and safety. In TEACH, task evaluation is constrained by a limit of $1,000$ execution steps or 30 failed actions, beyond which the agent is considered unsuccessful. This constraint motivates efficient task completion and effective action execution. Therefore, the agent must avoid redundant actions that do not contribute meaningfully to task completion. For instance, if the agent is trying to prepare coffee and it fails

[5] We only use: is toggled, is sliced, is filled with water, is clean, is open and is cooked.

to clean a mug that is already clean, it should not persist in attempting to resolve this failure but instead proceed to the next step in its plan. To that end, in the first stage of CMFR, the LLM answers the question of whether the current failing subgoal is important for the task and explicates its answer. It is prompted with the task the agent is trying to achieve and the plan (both predicted at the planning stage in Section 4.1), the current subgoal it is failing at and the scene representation (Section 4.4). If the subgoal is marked as important, CMFR goes to the second stage, otherwise, the agent skips this subgoal and moves to the next one in the plan.

**Stage 2: Preconditions** For some actions to succeed, there are preconditions that need to be satisfied in the environment. For instance, the agent needs to be holding a knife before slicing an object and have an empty hand before picking up an object.[6] Previous work has hard-coded those preconditions along with their recovery mechanisms in the executor (Zheng et al., 2022b; Zhang et al., 2022; Sarch et al., 2023; Fu et al., 2024), learned them via reinforcement learning (Ahn et al., 2022) or enumerated them in the prompts (Wang et al., 2023b), which requires specific domain knowledge. We propose another generalizable approach where we leverage the reasoning abilities of LLMs to capture the absence of those preconditions and find solutions by reflecting on the environment and execution history. Specifically, if a failing subgoal is labeled as important by the first stage, it passes to stage 2, where the LLM is prompted with the task, plan of execution, failing subgoal, reason for subgoal importance from the first stage and scene representation. The LLM assesses whether any prerequisite subgoals are missing, generates them then passes them for execution before the original subgoal is attempted again. If no preceding steps are required, error recovery advances to stage 3.

**Stage 3: Workaround** A failed subgoal that reaches this stage is deemed significant (by stage 1), but there is no evident indication in the environment for any missing preceding steps (as determined by stage 2). As a result, the LLM tries to find a workaround of one or more subgoals to substitute the failing subgoal while achieving the same objective. For instance, the LLM might suggest to use an alternative object if the intended object is not found. In this workaround stage, the LLM is given the same information as in the second stage (the task, plan of execution, failing subgoal, reason for subgoal importance and scene representation).

**Stage 4: Post Execution** The previous three stages happen during execution to assist the agent when it is stuck. They provide a more localized perspective, as the agent focuses on a single step of execution and attempts to complete it successfully. However, even if the agent manages to recover from failures this does not necessarily guarantee task success. This can occur in situations where, for example, the LLM have missed generating important steps in the original plan. To handle such situations, if the agent finishes executing the whole plan and the task is still unsuccessful, it is given one final opportunity to reflect on the task and the environment and identify what is missing to succeed. The prompt in stage 4 consists of the task, plan of execution and scene representation. Unlike the first three stages, this stage takes a more global view on the task, aiming to identify the discrepancy between the task requirements and the current state of the scene.

## 5 Experiments

We assess our approach on both the seen and unseen splits of the TfD benchmark in TEACH and present the evaluation results using the SR, GC, and PWL metrics (Section 3). All experiments use the same initial plans generated by GPT-4o (Hurst et al., 2024), allowing a focused assessment of error recovery. We evaluate four LLMs for CMFR and object search: GPT-4o, o3-mini, Qwen2.5-7B (Yang et al., 2024) and Llama-3.1-8B (Dubey et al., 2024). Our CMFR method is compared against a baseline where no failure recovery is included. We also conduct an experiment where we apply Chain-of-thought (CoT) reasoning (Wei et al., 2022) in error recovery (CoT-GPT-4o).[7]

Additionally, we compare our results to the following previous models. **HELPER** (Sarch et al., 2023) uses GPT-4 for planning, error recovery and object search, supported by a memory that is expanded with successful examples for few-shot prompting. HELPER explicitly encodes subgoal preconditions within the executor module and relies on previously established perception models (Dong

[6] More examples are included in Appendix J.

[7] GPT-4o is prompted using same information provided in CMFR, but asked to perform CoT reasoning to recover from the current failure.

| Model | Seen | | Unseen | |
|---|---|---|---|---|
| | SR (PWL) | GC (PWL) | SR (PWL) | GC (PWL) |
| CMFR-GPT-4o | **36.46** (20.64) | 50.14 (28.30) | **31.20** (19.96) | 44.16 (27.03) |
| CMFR-o3-mini | 35.35 (21.17) | **50.68** (29.93) | 29.90 (17.20) | **45.21** (23.70) |
| CMFR-Qwen2.5-7B | 28.72 (17.80) | 41.92 (25.50) | 24.67 (15.07) | 37.42 (21.78) |
| CMFR-Llama-3.1-8B | 28.72 (18.21) | 42.27 (25.50) | 24.18 (15.17) | 37.57 (23.02) |
| No Failure Recovery | 24.86 (15.61) | 39.33 (25.72) | 22.05 (14.46) | 35.31 (22.55) |
| CoT-GPT-4o | 30.93 (17.11) | 47.34 (26.03) | 27.45 (13.99) | 42.58 (21.16) |
| HELPER* | 17.12 (5.5) | 29.01 (16.4) | - | - |
| DANLI* | 4.41 (2.6) | 15.05 (14.2) | - | - |
| HELPER | 12.15 (1.79) | 18.62 (9.28) | 13.73 (1.61) | 14.17 (4.56) |
| DANLI | 4.97 (1.86) | 10.50 (10.27) | 7.98 (3.20) | 6.79 (6.57) |
| MSI | 12.70 (2.60) | 13.66 (8.72) | 14.54 (3.70) | 10.08 (6.35) |

Table 1: Results on the TEACH TfD benchmark. Results of HELPER, DANLI and MSI are copied from their respective papers, while HELPER* and DANLI* are results of replicating their models with ground-truth perception.

et al., 2021; Bhat et al., 2023). To ensure a fair comparison, we reproduce their results using ground-truth perception on the seen split (**HELPER***). **DANLI** (Zhang et al., 2022) fine-tunes a BART-Large model (Lewis et al., 2020) to predict high-level subgoals that are translated to low-level actions using a PDDL planner (Lamanna et al., 2021). In DANLI, all preconditions and error recovery mechanisms are hardcoded and perception models (Dosovitskiy et al., 2021) are used. We replicate their experiments using ground-truth perception on the seen split (**DANLI***). **MSI** (Fu et al., 2024) builds on top of HELPER and enhances the performance by collecting the agent's experiences and leveraging them later for future task executions.[8]

Furthermore, we conduct experiments using few-shot learning in CMFR, where we incorporate a fixed set of examples across the stages.[9] Finally, we conduct an ablation study to demonstrate the impact of each CMFR stage and the object search module. The main results and the ablations are shown in Tables 1 and 2 respectively.

## 6 Results

Table 1 illustrates the efficacy of the CMFR approach which sets new state-of-the-art results on the TfD benchmark of TEACH on both the seen and unseen data splits. The results show that our model outperforms previous models with a substantial margin even when we incorporate ground-truth perception in those models to match our evaluation setup (HELPER* and DANLI*). However, as discussed in Section 4.2, each of the previous work and our work has included different heuristics in the executor (e.g., for agent positioning) and therefore this comparison should be interpreted with caution. A more indicative comparison that highlights the effectiveness of CMFR is its evaluation against the no failure recovery scenario as both models use the same executor and even start from the same initial LLM-generated plan. This comparison shows that adding CMFR improves the success rate by 11.6% and 9.15% on seen and unseen splits respectively. Additionally, CMFR outperforms CoT reasoning (CoT-GPT-4o)[10] which further demonstrates the strength of our structured prompting approach. The effectiveness of the approach is also validated with other LLMs (Qwen2.5 and Llama-3.1), and although performance drops compared to GPT-4o and o3-mini, it remains superior to the no failure recovery scenario. We detail per-task performance in Appendix F.

We further conduct an ablation study on the seen split to demonstrate the importance of each stage of error recovery (Table 2). Our analysis indicates that performance drops with the removal of each of the four stages with preconditions (stage 2) and post-execution (stage 4) having the greatest impact. When each of those two stages is removed individually, success rate decreases to 32.04%. We also examine ablating both stages simultaneously and find that performance drops to 25.96%, which suggests that the two stages are complementary, each addressing distinct execution challenges. This finding shows that taking a global perspective on the

[8]We copy the results reported in their paper as they do not provide detailed replication instructions.

[9]We use three examples in the first stage, four in the second, five in the third, and four in the final stage.

[10]CoT-GPT-4o also starts from the same initial plan as CMFR and uses same executor and object search modules.

| Model | Seen | |
|---|---|---|
| | SR (PWL) | GC (PWL) |
| CMFR-GPT-4o | 36.46 (20.64) | 50.14 (28.30) |
| w/o stage 1 | 33.70 (19.39) | 47.83 (26.80) |
| w/o stage 2 | 32.04 (19.64) | 45.61 (27.14) |
| w/o stage 3 | 35.35 (21.35) | 49.60 (29.13) |
| w/o stage 4 | 32.04 (18.67) | 46.76 (27.11) |
| w/o stage 2&4 | 25.96 (16.62) | 40.81 (25.94) |
| w/o object search | 31.49 (18.93) | 47.00 (28.14) |
| with few shots | **38.12** (21.71) | **52.62** (29.55) |

Table 2: Results of ablations to different components of CMFR and the addition of few-shot prompting.

situation post-execution can successfully resolve issues that remained unresolved during the execution phase. Table 2 also demonstrates the importance of the object search component as removing it results in $\sim 5\%$ drop in success rate. Finally, performance improves further to $38.12\%$ with few-shot prompting, despite utilizing only 3-5 fixed examples in the prompts.

We further investigate subgoal failure reasons for all the failures that occurred during execution and report the most common reasons in Table 3. The table clearly indicates that most failures are caused by the inability to locate objects or precisely position the agent for interaction (see Appendix H for examples).[11] This finding aligns with the failure analysis conducted by Zheng et al. (2022b). As previously mentioned, and in line with prior research, we employ heuristic-based adjustments to refine the agent's positioning. However, this approach is not entirely reliable, and improper positioning remains a significant factor contributing to interaction failures.

We depict in Figure 2 the fraction of subgoals passed to each CMFR stage during execution. The figure shows that $19.5\%$ of *all subgoals* generated in the initial plan fail and hence are passed to CMFR. In the remainder of this section, we further examine the impact of each stage in CMFR in addition to the importance of object search.

### 6.1 Stage 1: Subgoal Importance

Removing stage 1 reduces the success rate to $33.7\%$, demonstrating its effectiveness in filtering out redundant steps. This stage is particularly important in scenarios where execution failures incur penalties, such as in TEACH evaluation, where exceeding 30 failures results in task failure. Further analysis reveals that subgoal importance assessment decreases the proportion of games failing due to exceeding this threshold from $8\%$ to $6\%$, further emphasizing its role in minimizing unnecessary failures and improving overall task success. Interestingly, we observe a notable difference in subgoal importance assessment across different LLMs. For example, the GPT-4o-based method classifies $66\%$ of subgoals as important, while Qwen2.5 identifies only $25\%$. Llama-3.1 falls between the two, marking $42\%$ of failed subgoals as important (see Appendix I). A possible explanation for this discrepancy is that all the methods rely on initial plans generated by GPT-4o, which inherently considers its own plans effective unless external environmental information changes that.

### 6.2 Stage 2: Preconditions

Stage 2 plays a pivotal role in failure recovery by ensuring that preconditions for subgoals are satisfied. We find that $93\%$ of subgoals that reach this stage are labeled as having unmet preconditions and accordingly corrective steps are generated. Additionally, $14\%$ of subgoals addressed by this stage succeed. We investigate a sample of 30 subgoals where precondition steps were executed yet failure persisted after stage 2 and find that: (1) in only four cases, the LLM failed to predict the correct preconditions based on scene information (e.g., not recognizing the need to empty a full receptacle before placing a new object); (2) in 16 cases, failure was due to not finding the required object, which should be resolved through object search or the next workaround stage; (3) in 10 cases, the LLM correctly identified precondition actions but failed in execution, despite no apparent reason in scene representation. Such failures could be attributed to agent positioning challenges or obstacles not explicitly represented in scene representation. Future work could explore integrating spatial reasoning into LLM-based systems to improve error recovery in such scenarios.

### 6.3 Stage 3: Workaround

Stage 3 has the least impact on performance, with success rate dropping by only $1\%$ when removed. This is expected, as (a) it is the least frequently triggered stage, occurring in only $1\%$ of total subgoals (Figure 2), and (b) it is more challenging to think of a workaround that would replace an action but achieves the same goal than to suggest the prerequisite steps for this action. Our analysis

[11] Positioning challenges include being at an incorrect distance or angle for object interaction, as well as encountering obstacles that impede access.

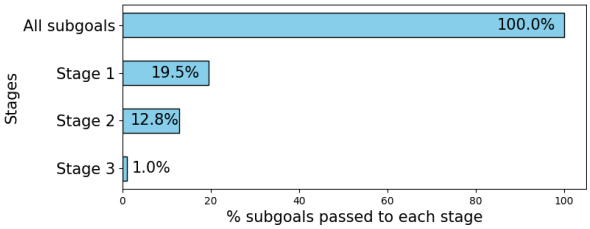



Figure 2: Subgoals that reach each stage of CMFR, during execution, from 'All subgoals' generated in the initial plan.

| Failure Reason | Frequency |
|---|---|
| Object not found | 41.1% |
| Positioning | 32.2% |
| pick-up an object while already holding another object | 4.7% |
| place an object while robot hand is empty | 3.4% |
| slice an object while not holding a knife | 2.1% |
| Others | 16.5% |

Table 3: Most common failure reasons for all the subgoals executed in the seen split games.

reveals that 51% of the subgoals reaching stage 3 failed due to the inability to locate a required object, prompting the workaround stage to propose alternative objects.[12] For instance, stage 3 proposed using apples instead of lettuce for a salad or placing all remote controls on a sofa when no chair was observed.[13] While such substitutions may be reasonable in some real-world scenarios, they do not align with the strict object-type constraints of the TEACH evaluation. In the cases where the agent fails to interact with the object due to imprecise positioning, the workaround stage suggests an alternative object of the same type (e.g., using another knife for slicing or selecting a different lettuce slice if the first was inaccessible). In rare cases, the agent generates creative but impractical solutions, such as using a spatula to pick up a potato slice or a dish sponge for cleaning in the absence of a sink. While these solutions demonstrate reasoning ability, they are not applicable within the constraints of the TEACH environment and therefore do not lead to task success.

### 6.4 Stage 4: Post Execution

The final stage plays a critical role by serving as a post-execution reflection phase, enabling the agent to assess task outcomes, interpret the environment, and identify missing steps necessary for successful completion. Analysis of cases where success was achieved only after this stage reveals that in 63% of those cases, the LLM recovered by generating previously omitted steps (e.g., recognizing the need to clean kitchenware before use) or by incorporating objects observed during execution but absent from the initial plan. In the remaining 37% of cases, success was achieved by re-executing steps from the original plan based on environmental feedback, such as repeating a *Place(Object,Receptacle)* action if the object was not found in the receptacle at the end of execution.

### 6.5 Object Search

The removal of the object search component results in $\sim 5\%$ decrease in success rate, highlighting its significance within the system. While the initial planning stage is expected to generate steps for locating objects if they are mentioned in the dialogue (e.g., directing the agent to open the fridge if it is mentioned that the target potato is inside), it is susceptible to mistakes and omitting necessary actions. Consequently, additional prompting is required to locate unobserved objects. Furthermore, the object search component exemplifies the interdependence of system modules, as it relies on the object locations produced by the initial planning phase.

To further assess the impact of object search, we analyze the games that failed and find that when object search is not utilized, 26% of those games fail as a result of the inability to locate required objects. In contrast, this percentage decreases to 17% when object search is employed. These findings, along with the data presented in Table 2, highlight the critical role of object search in our system while also indicating that it remains a performance bottleneck. As previously noted, TEACH presents an additional challenge, as objects may be initialized in illogical positions, limiting the effectiveness of common-sense reasoning for object retrieval. Future improvements could involve integrating human interaction or incorporating a specialized system with expert knowledge of the task environment to enhance object search capabilities in such cases.

## 7 Conclusion

We presented a conditional multi-stage failure recovery framework for embodied agents, achieving state-of-the-art performance on the TEACH TfD benchmark with 36% success rate, which further improves to 38% with few-shot prompting. Through an ablation study, we demonstrated the

[12] This result is in line with our findings in Table 3.

[13] The task was to place all remote controls on one chair.

importance of each stage within the framework, identifying the preconditions stage and the post-execution stage as the most critical for effective error recovery. Our findings also showed the importance of object search, highlighting object localization as a key performance bottleneck that requires further investigation. For future work, we aim to explore the integration of spatial reasoning to enhance error recovery and improve task success rates.

## Limitations

Our work has the following limitations:

**Simulated Environments** We use AI2-THOR which simplifies manipulation actions and abstracts away from physics. Applying our approach to real-world environments necessitates incorporating a more fine-grained action space (Brohan et al., 2023). Furthermore, as we discussed in the paper, working with the simulator poses the challenge of accurately positioning the agent for interaction, even with hardcoded movement adjustments. This results in execution failures that are not attributed to planning or error recovery. These challenges underscore the need for either refining the evaluation setup or developing models capable of learning fine-grained motion adjustments.

**TEACH Challenges** We evaluate our approach on the TfD benchmark of TEACH rather than the EDH benchmark as in the latter, the agent is penalised if the state of the environment after execution differs from the reference state achieved by the human follower in the dataset. This suggests that the EDH evaluation lacks precision as any incidental changes made by the human follower in the environment are considered essential and the agent is penalised if it does not replicate the same changes. On the other hand, in TfD, the evaluation specifically targets the task-specific changes that are intrinsic to the task itself. Furthermore, in TEACH, objects are initialised at random locations which limits the ability to use common-sense reasoning to find objects.

**Perception** In our models and in replicating previous work, we used ground-truth perception derived from information provided by the simulator. The incorporation of perception models could potentially lead to a decline in performance. While this study abstracts from the use of perception models, as its primary focus is failure recovery, future research will explore the integration of generalizable perception models (Li et al., 2023b, 2024).

**LLM Cost** Our highest performance was attained using GPT-4o followed by o3-mini, which outperformed other freely available models such as Llama-3.1 and Qwen-2.5. This highlights the continued cost implications associated with utilizing large language models. Ongoing research in LLMs may reduce or eliminate these costs in the future.

## References


Josh Abramson, Arun Ahuja, Federico Carnevale, Petko Georgiev, Alex Goldin, Alden Hung, Jessica Landon, Jirka Lhotka, Timothy Lillicrap, Alistair Muldal, et al. 2022. Improving multimodal interactive agents with reinforcement learning from human feedback. *arXiv preprint arXiv:2211.11602*.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*.

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3674–3683.

Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. 2023. Zoedepth: Zero-shot transfer by combining relative and metric depth. *arXiv preprint arXiv:2302.12288*.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia

Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. 2023. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *arXiv preprint arXiv:2307.15818*.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. 2022. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*.

Bin Dong, Fangao Zeng, Tiancai Wang, Xiangyu Zhang, and Yichen Wei. 2021. Solq: Segmenting objects by learning queries. *Advances in Neural Information Processing Systems*, 34:21898–21909.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Mauzama Firdaus, Gopendra Singh, Asif Ekbal, and Pushpak Bhattacharyya. 2023. Multi-step prompting for few-shot emotion-grounded conversations. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 3886–3891.

Dayuan Fu, Biqing Qi, Yihuai Gao, Che Jiang, Guanting Dong, and Bowen Zhou. 2024. MSI-agent: Incorporating multi-scale insight into embodied agents for superior planning and decision-making. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 643–659, Miami, Florida, USA. Association for Computational Linguistics.

Xiaofeng Gao, Qiaozi Gao, Ran Gong, Kaixiang Lin, Govind Thattai, and Gaurav S Sukhatme. 2022. Dialfred: Dialogue-enabled agents for embodied instruction following. *arXiv preprint arXiv:2202.13330*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022b. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. 2020. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*.

Byeonghwi Kim, Jinyeon Kim, Yuyeong Kim, Cheolhong Min, and Jonghyun Choi. 2023. Context-aware planning and environment-aware memory for instruction following embodied agents. In *ICCV*.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. 2020. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4392–4412, Online. Association for Computational Linguistics.

Alice Kwak, Clayton Morrison, Derek Bambauer, and Mihai Surdeanu. 2024. Classify first, and then extract: Prompt chaining technique for information extraction. In *Proceedings of the Natural Legal Language Processing Workshop 2024*, pages 303–317, Miami, FL, USA. Association for Computational Linguistics.

Leonardo Lamanna, Luciano Serafini, Alessandro Saetti, Alfonso Gerevini, and Paolo Traverso. 2021. Online grounding of pddl domains by acting and sensing in unknown environments. *arXiv preprint arXiv:2112.10007*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*,

pages 7871–7880, Online. Association for Computational Linguistics.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. 2023a. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR.

Hao Li, Dingwen Zhang, Yalun Dai, Nian Liu, Lechao Cheng, Jingfeng Li, Jingdong Wang, and Junwei Han. 2024. Gp-nerf: Generalized perception nerf for context-aware 3d scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21708–21718.

Yiming Li, Zhiding Yu, Christopher Choy, Chaowei Xiao, Jose M Alvarez, Sanja Fidler, Chen Feng, and Anima Anandkumar. 2023b. Voxformer: Sparse voxel transformer for camera-based 3d semantic scene completion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9087–9098.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE.

Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. 2023. Text2motion: from natural language instructions to feasible plans. *Autonomous Robots*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023a. Llm+p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Zeyi Liu, Arpit Bahety, and Shuran Song. 2023b. Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*.

Subhankar Maity, Aniket Deroy, and Sudeshna Sarkar. 2024. A novel multi-stage prompting approach for language agnostic mcq generation using gpt. In *European Conference on Information Retrieval*, pages 268–277. Springer.

Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. 2019. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. 2021. Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*.

Aishwarya Padmakumar, Mert Inan, Spandana Gella, Patrick Lange, and Dilek Hakkani-Tur. 2023. Multimodal embodied plan prediction augmented with synthetic embodied dialogue. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6114–6131, Singapore. Association for Computational Linguistics.

Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. 2022. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2017–2025.

Alexander Pashevich, Cordelia Schmid, and Chen Sun. 2021. Episodic transformer for vision-and-language navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15942–15952.

Daniel Philipov, Vardhan Dongre, gokhan tur, and Dilek Hakkani Tur. 2024. Simulating user agents for embodied conversational AI. In *NeurIPS 2024 Workshop on Open-World Agents*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Gabriel Sarch, Sahil Somani, Raghav Kapoor, Michael J Tarr, and Katerina Fragkiadaki. 2024. Helper-x: A unified instructable embodied agent to tackle four interactive vision-language domains with memory-augmented language models. In *ICLR 2024 LLMAgents Workshop*.

Gabriel Sarch, Yue Wu, Michael Tarr, and Katerina Fragkiadaki. 2023. Open-ended instructable embodied agents with memory-augmented large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. 2022. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530.

Kunal Pratap Singh, Luca Weihs, Alvaro Herrasti, Jonghyun Choi, Aniruddha Kembhavi, and Roozbeh Mottaghi. 2022. Ask4help: Learning to leverage an expert for embodied tasks. In *Advances in Neural Information Processing Systems*, volume 35, pages 16221–16232. Curran Associates, Inc.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Shichao Sun, Ruifeng Yuan, Ziqiang Cao, Wenjie Li, and Pengfei Liu. 2024. Prompt chaining or stepwise prompt? refinement in text summarization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7551–7558, Bangkok, Thailand. Association for Computational Linguistics.

Dietrich Trautmann. 2023. Large language model prompt chaining for long legal document classification. *arXiv preprint arXiv:2308.04138*.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. 2023a. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv: 2311.05997*.

Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023b. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–10.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Haopeng Zhang, Xiao Liu, and Jiawei Zhang. 2023. SummIt: Iterative text summarization via ChatGPT. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10644–10657, Singapore. Association for Computational Linguistics.

Yichi Zhang, Jianing Yang, Jiayi Pan, Shane Storks, Nikhil Devraj, Ziqiao Ma, Keunwoo Yu, Yuwei Bao, and Joyce Chai. 2022. DANLI: Deliberative agent for following natural language instructions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1280–1298, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Kaizhi Zheng, Xiaotong Chen, Odest Jenkins, and Xin Eric Wang. 2022a. VLMbench: A compositional benchmark for vision-and-language manipulation. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Kaizhi Zheng, Kaiwen Zhou, Jing Gu, Yue Fan, Jialu Wang, Zonglin Di, Xuehai He, and Xin Eric Wang. 2022b. Jarvis: A neuro-symbolic commonsense reasoning framework for conversational embodied agents. *arXiv preprint arXiv:2208.13266*.

## A TfD Dataset of TEACH

The TfD dataset in TEACH[14] is divided into three splits: train containing $1,482$ episodes, valid seen (i.e., episodes of the same room instances as the train split, but different object locations and initial properties) containing $181$ episodes and valid unseen containing $612$ episodes of new unseen rooms. In TEACH, the agent executes low-level navigational actions (Forward(), Backward(), Rotate Left(), Rotate Right(), Look Up(), Look Down(), Strafe Left(), Strafe Right())[15] and interactive actions (Pickup(X), Place(X), Open(X), Close(X), ToggleOn(X), ToggleOff(X), Slice(X), and Pour(X)), where X refers to the relative xy coordinate of the target object on the egocentric RGB frame. After each action, the agent obtains an egocentric RGB image. **Path Length Weighted (PLW)** is calculated as $P_m = m * L^*/max(L^*\hat{L})$, where $m$ is the evaluation metric (SR or GC), $\hat{L}$ is the number of actions the model took in the episode, and $L^*$ is the number of actions in the reference demonstration.

## B Planning Details

To create demonstrations for the planning prompt, we select $24$ input dialogues from training data (two for each task) and write their output plans, tasks and object locations. We note that few-shot prompting from a pool of only $24$ examples is used for the initial plan, whereas failure recovery employs zero-shot prompting. We use Sentence-BERT (Reimers and Gurevych, 2019) to select the most similar three examples from the created demonstrations to be included in the prompt. We show the planning prompt in Listing 1 and a sample of the examples used in few-shot planning in Listing 2. After generating the initial plan, if a subgoal includes an object that is not present in the predefined list of object categories (e.g., generating "Cupboard" when only "Cabinet" is available), the LLM is prompted with the generated object and the list of available object categories and is instructed to select the category that is most similar to the generated object. We note that we generate extra information in planning that we do not use (Listing 1). For example, we prompt the LLM to generate task_params, if exists, such as the number of objects required for the task (if the task is to clean 3 cups, task_params should include $N = 3$).

## C Execution Details

**Object Memory** The agent maintains a memory of the objects observed after each action (movement or interaction) along with their properties and locations. In particular, the agent keeps a dictionary of observed objects where the key is the object ID (e.g., Pot_1, Pot_2, etc.) and the value is the list of: (1) object properties (clean, sliced, open, etc), (2) xyz position, (3) parent objects if any (e.g., if a pan is in a sink it will have sink as its parent), and (4) child objects if any (the sink will have the pan as its child).

**Navigation** Before execution, the agent carries out an initial exploration to gather information about its surroundings. Sarch et al. (2023) achieves that by incrementally building a 2D occupancy map, randomly sampling locations from the map and then navigating to those locations until the environment is fully explored. We use a different approach where the agent goes to the center of the room floor, then the centers of the top-left, top-right, bottom-left and bottom-right quadrants of the room floor, making a full rotation at each of those points. The agent maintains a memory where it stores information about the objects observed at each point. During execution, when *Go_to(object)* subgoal is called, we calculate the shortest path from the agent's current position to the position of the target object. The agent navigates to the next point in the path by first orienting itself towards this point using Rotate Left() and Rotate Right() actions then executing the Forward() action. If navigation fails at any point, we allow the agent to renavigate one more time from where it failed to the target object. Navigation failures can arise from various factors. For example, if the agent is carrying an object, such as a pot, and encounters a large obstacle, such as a refrigerator, along its path, the carried object may collide with the obstacle, preventing successful navigation. In contrast, the same path may be traversable if the agent's hand is empty. Once the agent reaches the target object, positioning itself in close proximity and orienting towards it, it attempts to interact with the object. In the event of a failed interaction, heuristic-based adjustments are applied to refine the agent's posi-

[14] TEACH code is licensed under the MIT License, their images are licensed under Apache 2.0 and other data files are licensed under CDLA-Sharing 1.0 (see https://github.com/alexa/teach).

[15] Default distance of Forward() and Backward() is $0.25$ meters, angle change for Rotate Left() and Rotate Right() is $90^\circ$ and angle change for Look Up() and Look Down() is $30^\circ$.

tioning. These heuristics consist of a sequence of movement actions, with the interaction attempted again after each adjustment. The actions include:

1. Change the yaw rotation of the agent's body by executing Rotate Left()
2. Change the yaw rotation of the agent's body by executing Rotate Right()
3. Change the camera's pitch by executing Look Up()
4. Change the camera's pitch by executing Look Down()
5. Change the distance to the target object by moving closer with Forward() action
6. Change the distance to the target object by moving further with Backward() action

**Resources** All experiments were run using NVIDIA TITAN RTX 24GB GPUs.

## D Available Subgoals, Object Categories and Tasks

| | |
|---|---|
| Subgoals | Find(Object), Go_to(Object), Pick_up(Object), Place(Object,Receptacle), Open(Object), Close(Object), Toggle_on(Object), Toggle_off(Object), Slice(Object), Pour(Object,Receptacle), Fill_with_water(Object), Clean(Object), Empty(Object), Put_away(Object) |
| Tasks | Water plant, Boil potato, Make coffee, Make plate of toast, Clean all X, Put all X on Y, N slices of X in Y, Put all X in one Y, N cooked X slices in Y, Prepare breakfast, Prepare sandwich, Prepare salad |
| Object Categories | Cabinet, CounterTop, Sink, Towel, HandTowel, TowelHolder, SoapBar, ToiletPaper, ToiletPaperHanger, HandTowelHolder, SoapBottle, GarbageCan, Candle, ScrubBrush, Plunger, SinkBasin, Cloth, SprayBottle, Toilet, Faucet, ShowerHead, Box, Bed, Book, DeskLamp, BasketBall, Pen, Pillow, Pencil, CellPhone, KeyChain, Painting, CreditCard, AlarmClock, CD, Laptop, Drawer, SideTable, Chair, Blinds, Desk, Curtains, Dresser, Watch, Television, WateringCan, Newspaper, FloorLamp, RemoteControl, HousePlant, Statue, Ottoman, ArmChair, Sofa, DogBed, BaseballBat, TennisRacket, VacuumCleaner, Mug, ShelvingUnit, Shelf, StoveBurner, Apple, Lettuce, Bottle, Egg, Microwave, CoffeeMachine, Fork, Fridge, WineBottle, Spatula, Bread, Tomato, Pan, Cup, Pot, SaltShaker, Potato, PepperShaker, ButterKnife, StoveKnob, Toaster, DishSponge, Spoon, Plate, Knife, DiningTable, Bowl, LaundryHamper, Vase, Stool, CoffeeTable, Poster, Bathtub, TissueBox, Footstool, BathtubBasin, ShowerCurtain, TVStand, Boots, RoomDecor, PaperTowelRoll, Ladle, Kettle, Safe, GarbageBag, TeddyBear, TableTopDecor, Dumbbell, Desktop, AluminumFoil, Window, LightSwitch, AppleSliced, BreadSliced, LettuceSliced, PotatoSliced, TomatoSliced, Mirror, ShowerDoor, ShowerGlass, Floor |

Table 4: List of subgoals/actions the agent is allowed to execute in the environment, list of object categories available in AI2-THOR and list of tasks available in TEACH.

## E Examples of Scene Representations

| Task | Scene Representation |
|---|---|
| Put all watches on one sidetable | (Watch_1 in SideTable_1), (Watch_2 in SideTable_1), (SideTable_1 contains Watch_1, Watch_2, KeyChain_1 and Box_1), (agent holding nothing) |
| Make coffee | (Mug_1 is filled with water), (Mug_1 is dirty), (Sink_1 contains Cup_1, WineBottle_1, Fork_1, Spoon_1 and WineBottle_2), (CoffeeMachine_1 is toggled on), (CoffeeMachine_1 in CounterTop_1), (agent holding Mug_1), |
| 1 cooked slices of potato in a bowl | (StoveBurner_1 contains Pan_1), (StoveBurner_2 contains Pan_2), (CounterTop_1 contains Apple_1, SaltShaker_1, SoapBottle_1, Knife_1 and Microwave_1), (Fridge_1 is closed), (Knife_1 in CounterTop_1), (Bowl_1 is not filled with water), (Bow_1 is clean), (Bowl_1 in DiningTable_1), (agent holding nothing) |

Table 5: Examples of scene representations taken at random points during task execution.

## F Per-task Performance

| Task | SR | GC |
|---|---|---|
| Put All X In One Y | 40.00 (25.32) | 50.00 (31.67) |
| Put All X On Any Y | 72.73 (47.54) | 82.20 (51.74) |
| Make Coffee | 66.67 (44.37) | 69.44 (45.33) |
| Boil a Potato | 20.00 (2.72) | 20.00 (2.72) |
| Water Plant | 72.73 (42.59) | 72.73 (42.59) |
| Clean All X | 23.81 (9.39) | 27.38 (11.27) |
| N Slices Of X In Y | 33.33 (20.43) | 49.07 (28.87) |
| N Cooked Slices Of X In Y | 20.00 (8.00) | 46.67 (24.33) |
| Make Plate of Toast | 60.00 (23.15) | 78.33 (33.54) |
| Make Breakfast | 0.00 (0.00) | 32.39 (17.26) |
| Make Salad | 5.88 (0.77) | 33.27 (15.56) |
| Make Sandwich | 0.00 (0.00) | 29.32 (19.49) |

Table 6: Results of CMFR-GPT-4o on each task in the seen data split.

## G Used Prompts

We include, in Listings 1, 2, 3, 4, 5, 6 and 7, the various prompts we use in this work.

Listing 1: Prompt for generating initial plan.

```
You are a household robot. You are given a dialogue snippet that contains information about the task
    you should execute. Your job is to generate the following in JSON format:
(1) 'task': the required task from this list of tasks
    - Water plant
    - Boil potato
    - Make coffee
    - Make plate of toast
    - Clean N Object
    - Put N Object on any Receptacle
    - N slices of Object in Receptacle
    - Put N Object in one Receptacle
    - N cooked Object slices in Receptacle
    - Prepare breakfast
    - Prepare sandwich
    - Prepare salad

(2) 'task_params': task parameters (if exists) which includes number of required objects 'N', 'Object
    ' type and 'Receptacle' type

(3) 'objects of interest' for this task

(4) 'object locations' if mentioned in the dialogue

(5) 'subgoals' which are the steps required to execute the task described in the dialogue. You should
     only generate subgoals from the following list:
    - Find(Object)
    - Go_to(Object)
    - Pick_up(Object)
    - Place(Object,Receptacle)
    - Open(Object)
    - Close(Object)
    - Toggle_on(Object)
    - Toggle_off(Object)
    - Slice(Object)
    - Pour(Object,Receptacle)
    - Fill_with_water(Object)
    - Clean(Object)
    - Empty(Object)
    - Put_away(Object)

** Any Object or Receptacle generated in the subgoals or objects of interest SHOULD be chosen from
    the following list:
Cabinet, CounterTop, Sink, Towel, HandTowel, TowelHolder, SoapBar, ToiletPaper, ToiletPaperHanger,
    HandTowelHolder, SoapBottle, GarbageCan, Candle, ScrubBrush, Plunger, SinkBasin, Cloth,
    SprayBottle, Toilet, Faucet, ShowerHead, Box, Bed, Book, DeskLamp, BasketBall, Pen, Pillow,
    Pencil, CellPhone, KeyChain, Painting, CreditCard, AlarmClock, CD, Laptop, Drawer, SideTable,
    Chair, Blinds, Desk, Curtains, Dresser, Watch, Television, WateringCan, Newspaper, FloorLamp,
    RemoteControl, HousePlant, Statue, Ottoman, ArmChair, Sofa, DogBed, BaseballBat, TennisRacket,
    VacuumCleaner, Mug, ShelvingUnit, Shelf, StoveBurner, Apple, Lettuce, Bottle, Egg, Microwave,
    CoffeeMachine, Fork, Fridge, WineBottle, Spatula, Bread, Tomato, Pan, Cup, Pot, SaltShaker,
    Potato, PepperShaker, ButterKnife, StoveKnob, Toaster, DishSponge, Spoon, Plate, Knife,
    DiningTable, Bowl, LaundryHamper, Vase, Stool, CoffeeTable, Poster, Bathtub, TissueBox,
    Footstool, BathtubBasin, ShowerCurtain, TVStand, Boots, RoomDecor, PaperTowelRoll, Ladle, Kettle,
     Safe, GarbageBag, TeddyBear, TableTopDecor, Dumbbell, Desktop, AluminumFoil, Window,
    LightSwitch, AppleSliced, BreadSliced, LettuceSliced, PotatoSliced, TomatoSliced, Mirror,
    ShowerDoor, ShowerGlass, Floor

Your output SHOULD strictly be in JSON format.

Here are some examples to show you what is required:
{RETRIEVED_EXAMPLES}

Now this is the example you should solve:
Dialogue: {INPUT_DIALOGUE}

Output:
```

Listing 2: Examples from the RETRIEVED_EXAMPLES in the planning prompt

```
                                                    Example 1
Dialogue:
<Driver> What is my first task today?
<Commander> Hi
<Commander> We are
<Commander> We are serving 1 slice of lettuce in a bowl
<Driver> Can you help me find the lettuce?
<Commander> The bowl is on the top shelf directly above the sink
<Commander> The lettuce is there as well
<Commander> sorry the lettuce is on the table that has the toaster
<Commander> on your right
<Commander> Perfect!
<Commander> We a knife
<Driver> Where is the knife?
<Commander> The knife is right on the sink
<Commander> Awesome
<Driver> Got it.
<Commander> Now to cut the lettuce
<Commander> Is the bowl clean?
<Commander> If it is place a slice of lettuce in the bowl
<Driver> Okay. It is done. What else?
<Commander> Perfect we're done
<Commander> Thank you so much!
<Commander> It was a pleasure doing the task with you
<Driver> Thank you.

Output:
{
  "task": "N slices of Object in Receptacle",
  "task_params": {
    "N": 1,
    "Object": "Lettuce",
    "Receptacle": "Bowl"
  },
  "objects of interest": [
    "Lettuce",
    "Bowl",
    "Knife"
  ],
  "object locations": [
    "(Bowl_1,on,Shelf_1)",
    "(Shelf_1,above,Sink_1)",
    "(Lettuce_1,on,Table_1)",
    "(Toaster_1,on,Table_1)",
    "(Knife_1,on,Sink_1)"
  ],
  "subgoals": [
    "Find(Bowl_1)",
    "Pick_up(Bowl_1)",
    "Place(Bowl_1,Table_1)",
    "Find(Lettuce_1)",
    "Pick_up(Lettuce_1)",
    "Place(Lettuce_1,Table_1)",
    "Find(Knife_1)",
    "Pick_up(Knife_1)",
    "Slice(Lettuce_1)",
    "Place(Knife_1,Table_1)",
    "Pick_up(LettuceSliced_1)",
    "Place(LettuceSliced_1,Bowl_1)"
  ]
}

                                                    Example 2
Dialogue:
<Driver> what can i do today
<Commander> boil the potato by cooking it in water
<Driver> where can i find the potato please
```

```
<Commander> let's find it
<Commander> have you looked in the fridge
<Driver>
<Commander> it's in the fridge
<Commander> are you done
<Driver> not yet
<Commander> ok waiting
<Commander> waiting
<Driver> done next?

Output:
{
  "task": "Boil potato",
  "task_params": {
    "N": 1,
    "Object": "",
    "Receptacle": ""
  },
  "objects of interest": [
    "Potato",
    "Pot",
    "StoveBurner",
    "Sink",
    "Fridge"
  ],
  "object locations": [
    "(Potato_1,inside,Fridge_1)"
  ],
  "subgoals": [
    "Find(Pot_1)",
    "Pick_up(Pot_1)",
    "Fill_with_water(Pot_1)",
    "Pick_up(Pot_1)",
    "Find(StoveBurner_1)",
    "Place(Pot_1,StoveBurner_1)",
    "Find(Fridge_1)",
    "Open(Fridge_1)",
    "Find(Potato_1)",
    "Pick_up(Potato_1)",
    "Place(Potato_1,Pot_1)",
    "Toggle_on(StoveBurner_1)"
  ]
}
```

Example 3

```
Dialogue:
<Driver> how can i help today?
<Commander> can you make a plate of toast? one slice
<Driver> sure, where can i find the bread?
<Commander> is in the top cupboard to the left above microwave
<Driver> i also need a plate
<Driver> where can i find one?
<Commander> plate is on the chair
<Commander> behind island
<Driver> all done
<Commander> Thank you

Output:
{
  "task": "Make plate of toast",
  "task_params": {
    "N": 1,
    "Object": "",
    "Receptacle": ""
  },
  "objects of interest": [
    "Bread",
    "Plate",
    "Toaster",
    "Knife"
```

```
  ],
  "object locations": [
    "(Bread_1,in,Cabinet_1)",
    "(Cabinet_1,above,Microwave_1)",
    "(Plate_1,on,Chair_1)"
  ],
  "subgoals": [
    "Find(Cabinet_1)",
    "Open(Cabinet_1)",
    "Find(Bread_1)",
    "Pick_up(Bread_1)",
    "Find(CounterTop_1)",
    "Place(Bread_1,CounterTop_1)",
    "Find(Knife_1)",
    "Pick_up(Knife_1)",
    "Slice(Bread_1)",
    "Put_away(Knife_1)",
    "Pick_up(BreadSliced_1)",
    "Find(Toaster_1)",
    "Place(BreadSliced_1,Toaster_1)",
    "Toggle_on(Toaster_1)",
    "Toggle_off(Toaster_1)",
    "Find(Plate_1)",
    "Clean(Plate_1)",
    "Place(Plate_1,CounterTop_1)",
    "Pick_up(BreadSliced_1)",
    "Place(BreadSliced_1,Plate_1)"
  ]
}
```

Listing 3: Prompt for subgoal importance (stage 1) in failure recovery

```
You are a robot trying to execute a plan of actions to perform a task in an environment, and you are
    failing to execute one of the steps.
You are given the task, your plan of actions and the step you are failing to execute. You are also
    given information from your environment about the locations and properties of the objects you
    are interacting with to achieve the task and about what you (the agent) are currently holding in
     hand.
You should determine whether the step you are failing at is important for the task, or you can ignore
     it and move on to the next step. You also need to justify your answer.
Your answer SHOULD be a JSON answering whether this step is impotant and the justification for that.

Task: {TASK}

Plan:
{EXECUTION_HISTORY}

Failing step:
{FAILING_SUBGOAL}

Information from environment:
{SCENE_REPRESENTATION}

Is {FAILING_SUBGOAL} important to achieve the task of: {TASK}? and why?

Answer:

"""
```

Listing 4: Prompt for preconditions check (stage 2) in failure recovery

```
You are a robot trying to execute a plan of actions to perform a task in an environment, and you are
    failing to execute one of the actions.
You are given the task, your plan of actions, the action you are failing to execute, and why this
    action is important for task success. You are also given information from your environment about
     the locations and properties of the objects you are interacting with to achieve the task and
    about what you (the agent) are currently holding in hand.
Your task is to reason about the environment and output a JSON of two keys (1) "prior required
    actions": indicating whether there are prior actions required to execute the failing action
    successfully and (2) "actions": which is a list of those required prior actions (if the answer
    to the previous question is yes).
You should ONLY generate actions from the following list:
    - Find(Object)
    - Go_to(Object)
    - Pick_up(Object)
    - Place(Object,Receptacle)
    - Open(Object)
    - Close(Object)
    - Toggle_on(Object)
    - Toggle_off(Object)
    - Slice(Object)
    - Pour(Object,Receptacle)
    - Fill_with_water(Object)
    - Clean(Object)
    - Empty(Object)
    - Put_away(Object)

Task: {TASK}

Plan:
{EXECUTION_HISTORY}

Failing step:
{FAILING_SUBGOAL}

Step {FAILING_SUBGOAL} is important to achieve the task of {TASK} because: {
    JUSTIFICATION_FROM_STAGE_1}

Information from environment:
{SCENE_REPRESENTATION}

Let's think step by step.

"""
```

Listing 5: Prompt for the workaround (stage 3) in failure recovery

```
You are a robot trying to execute a plan of actions to perform a task in an environment, and you are
    failing to execute one of the actions.
You are given the task, your plan of actions, the action you are failing to execute, and why this
    action is important for task success. You are also given information from your environment about
     the locations and properties of the objects you are interacting with to achieve the task and
    about what you (the agent) are currently holding in hand.
The action you are failing at is impossible to execute and therefore you should think of a workaround
     (i.e., an alternative sequence of actions to achieve the target of the failing action).
Your task is to reason about the environment and output a JSON with a key 'solution' and its value is
     an array of the actions in your aorkaround solution.
You should ONLY generate actions from the following list:
    - Find(Object)
    - Go_to(Object)
    - Pick_up(Object)
    - Place(Object,Receptacle)
    - Open(Object)
    - Close(Object)
    - Toggle_on(Object)
    - Toggle_off(Object)
    - Slice(Object)
    - Pour(Object,Receptacle)
    - Fill_with_water(Object)
    - Clean(Object)
    - Empty(Object)
    - Put_away(Object)

Task: {TASK}

Plan:
{EXECUTION_HISTORY}

Failing step:
{FAILING_SUBGOAL}

Step {FAILING_SUBGOAL} is important to achieve the task of {TASK} because: {
    JUSTIFICATION_FROM_STAGE_1}

Information from environment:
{SCENE_REPRESENTATION}

Can you think of a workaround to {FAILING_SUBGOAL} that achieves the same target?

Let's think step by step.

"""
```

Listing 6: Prompt for post execution stage (stage 4) in failure recovery

```
You are a robot trying to a task in an environment. You generated a plan of actions and finished
    executing it successfully, but still failed at the task.
You are given the task, your successfully executed actions and information from your environment
    about the locations and properties of the objects you are interacting with to achieve the task
    and about what you (the agent) are currently holding in hand.
You should reason about the current state of the environment to identify why you failed at the task
    and suggest the corrective/missing actions required to succeed at the task.
Your output should be a JSON with a key 'solution' and its value is an array of the actions to
    succeed at the task.
You should ONLY generate actions from the following list:
    - Find(Object)
    - Go_to(Object)
    - Pick_up(Object)
    - Place(Object,Receptacle)
    - Open(Object)
    - Close(Object)
    - Toggle_on(Object)
    - Toggle_off(Object)
    - Slice(Object)
    - Pour(Object,Receptacle)
    - Fill_with_water(Object)
    - Clean(Object)
    - Empty(Object)
    - Put_away(Object)

Task: {TASK}

Plan:
{EXECUTION_HISTORY}

Information from environment:
{SCENE_REPRESENTATION}

Let's think step by step.
"""
```

Listing 7: Prompt for object search where {GOAL} refers to the search task (e.g., to find a potato) , {OBJECT_LOCATIONS} is generated by the planner and {RETRIEVED_EXAMPLES} is a fixed set of four demonstrations

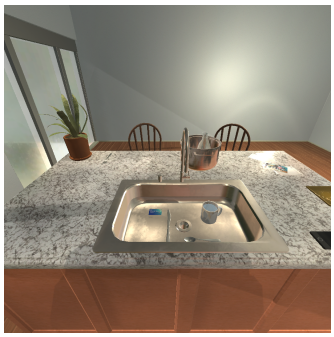
```
You are a robot trying to find an object in a room. Given the goal object you are trying to find and
    information about some object locations, predict the steps required to locate your object.
For example, if a potato is inside a fridge, you need to Open(Fridge) to find the potato.
Your output should be a JSON that consists of of an array of one or more of the following actions:
    - Find(Object)
    - Go_to(Object)
    - Pick_up(Object)
    - Place(Object,Receptacle)
    - Open(Object)
    - Close(Object)
    - Toggle_on(Object)
    - Toggle_off(Object)
    - Slice(Object)
    - Pour(Object,Receptacle)
    - Fill_with_water(Object)
    - Clean(Object)
    - Empty(Object)
    - Put_away(Object)

{RETRIEVED_EXAMPLES}

Goal: {GOAL}

object locations: {OBJECT_LOCATIONS}

Output:
```

## H Examples of Failures due to Navigation and Agent Positioning

(a) Subgoal: Open(Microwave_1). The agent is failing to open the microwave at the top, and 'looking up' did not fix the problem.

(b) Subgoal: Pick_up(Knife_1). The agent fails to pick up the knife in the sink.

(c) Subgoal: Pick_up(RemoteControl_1). The agent fails to pick up RemoteControl_1 which is not visible in the current view. This means that there was a problem in navigating/orienting the agent to the correct direction.



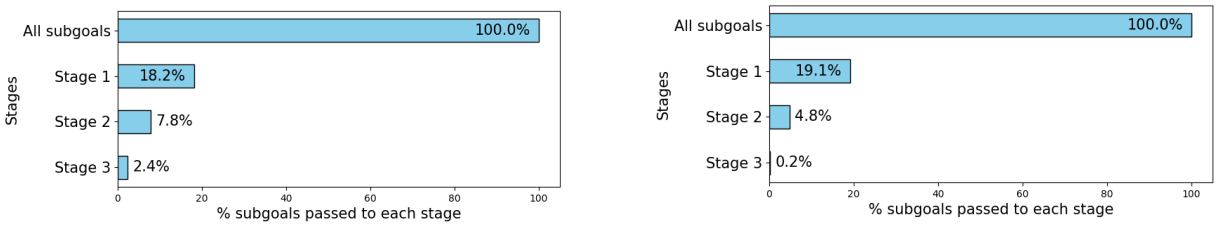
(d) Subgoal: Toggle_on(Sink_1). The agent fails to toggle on the sink although it is standing in front of it.

(e) Subgoal: Place(Pot_1,Sink_1). The agent fails to place the pot it is holding in the sink behind. The pot obstructs the sink making it invisible. We tried to move the pot up and down but it did not work.

(f) Subgoal: Pour(Mug_1,Sink_1). The agent fails to pour the mug it is holding in the sink.

Figure 3: Examples of RGB images from the agent's view when it is failing to execute a subgoal. The failures are due to the position of the agent relative to the target object.

# I Subgoals that reaches each stage of CMFR in Llama3.1 and Qwen2.5




(a) Llama-3.1

(b) Qwen2.5

Figure 4: Subgoals that reach each stage of CMFR, during execution, from 'All subgoals' generated in the initial plan. Used CMFR models are Llama-3.1 and Qwen2.5.

# J Examples of Preconditions and Possible Recoveries from TEACH

| Action | Preconditions |
|---|---|
| Pick_up(Object) | 1- if object is not pickupable, skip<br>2- if agent is holding an object in hand, put away then pick up the new object<br>3- if object is inside a closed receptacle, open receptacle |
| Place(Object, Receptacle) | 1- if agent is not holding the object, pick it up first<br>2- if receptacle is full, empty before placing<br>3- if receptacle is closed, open it |
| Slice(Object) | 1- if object is not sliceable, skip<br>2- if agent is not holding a knife, find a knife and pick it up first |
| Open(Receptacle) | 1- if receptacle is toggled on, toggle off first |
| Pour(Object,Receptacle) | 1- if object is not filled with liquid, skip<br>2- if object is not in hand, pick it up first<br>3- if agent is far from receptacle, go to receptacle |
| Clean(Object) | 1- if object is already clean, skip<br>2- if object is not in hand, pick it up first<br>3- if there is no space in sink, empty it |
| Fill_with_water(Object) | 1- if object cannot be filled with water or is already filled with water skip<br>2- if object is not in hand, pick it up first<br>3- if there is no space in sink, empty it |

Table 7: Examples of preconditions and their possible recoveries for executing the actions in the left column.