Visual Feedback in Instruction-Following Tasks for LLM-based Embodied Agent

Svyatoslav Pchelintsev[1] , **Second Author**[2] , **Third Author**[2,3] and **Fourth Author**[4]
[1]Moscow Institute of Physics and Technology
[2]Second Affiliation
[3]Third Affiliation
[4]Fourth Affiliation
{first, second}@example.com, third@other.example.com, fourth@example.com

Abstract

Recently, large language models (LLMs) have been widely used for task planning in instruction-following tasks. However, an LLM often fails to take into account the specifics of the particular scene in which the agent is operating. Various approaches have been proposed to address this issue. One direction of research is the correction of planning errors, which is usually done by providing textual feedback, i.e., information about the state of the environment in textual form. On the other hand, visual feedback, i.e., feedback is represented by images, in LLMs for embodied artificial intelligence tasks has not been thoroughly investigated. In this paper, we propose LERa — **L**ook, **E**xplain, **R**eplan — an approach that uses visual feedback to correct plans when errors occur in an instruction-following task due to an unexpected change in the state of an object. We test LERa on a dataset collected from ALFRED episodes, in which half of the episodes have additionally changed object states. The proposed approach proves its viability by completing successfully nearly 1.5 times more episodes than an agent without LERa.

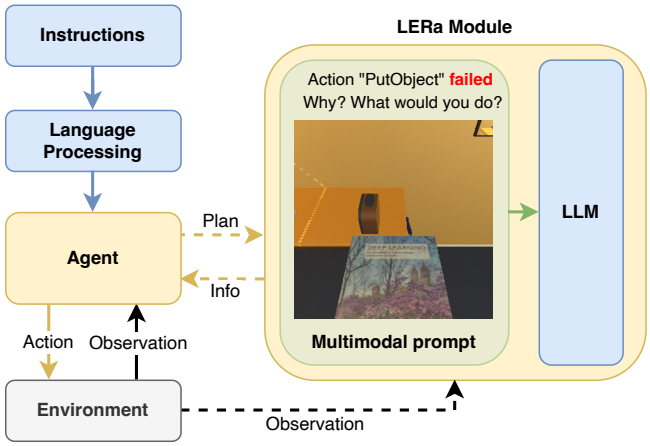


Figure 1: An approach without error handling (uses a static queue of subtasks a predicted plan consists of) can be enhanced with the LERa module to dynamically update queue of subtasks when planning errors occur. In the first case, the plan is generated once at the beginning of an episode and remains unchanged throughout the episode. In the second case, when an error occurs during the execution of the current subtask, an LLM is used to replan based on the environmental feedback. To do this, a multimodal prompt is generated that consists of the current observation of the agent and its internal information about the error.

1 Introduction

Embodied artificial intelligence has recently gained increasing interest among researchers. The goal is to develop and study intelligent agents that operate in an environment, often human-oriented, and are capable of perceiving visual and auditory information, reasoning in the sense of planning the long-term consequences of their actions, and interacting with a user in natural language. In particular, the latter capability is crucial because it significantly reduces the complexity of interacting with an intelligent agent for non-professional users.

Recently, the number of approaches that use large language models (LLMs) for task planning has been steadily increasing [Singh *et al.*, 2022; Liang *et al.*, 2023; Huang *et al.*, 2022a; Zeng *et al.*, 2022; Huang *et al.*, 2022b; Driess *et al.*, 2023]. These approaches rely on the ability of LLMs to solve problems for which they have not been explicitly trained, operating in zero-shot [Wei *et al.*, 2021] or few-shot learning [Brown *et al.*, 2020] modes.

When planning with an LLM, there is no guarantee that the resulting plan can be executed in an environment, e.g. a synonym has been used instead of a predefined object name, or a hallucination has occurred. However, LLMs exhibit significantly greater flexibility in replanning than task-specific sequence-to-sequence models because it is possible to prompt an LLM to correct its error without special training, whereas task-specific sequence-to-sequence models lack this ability. Considerable effort is being put into developing methods that allow feedback from an environment, a user, or an agent itself to be used in planning [Xiang *et al.*, 2023; Zhao *et al.*, 2023b; Mandi *et al.*, 2023; Silver *et al.*, 2023; Lin *et al.*, 2023b]. This feedback can be expressed in a variety of ways, ranging from a list of objects present in the current scene to a caption on the image coming from an agent's

camera.

The main limitation is that LLMs can only work with textual modality by default. Fortunately, there are ways to extend the capabilities of LLMs by embedding the visual modality without having to retrain the entire model [Guo *et al.*, 2023; Chen *et al.*, 2023b; Driess *et al.*, 2023; Koh *et al.*, 2023; Zhu *et al.*, 2023; Brohan *et al.*, 2023]. However, the application of such approaches to the problem of visual feedback for embodied intelligent agents is poorly understood. In this paper, we address the problem of correcting a plan when an agent encounters an unexpected object state in the environment by proposing a **LERa** — **L**ook, **E**xplain, **R**epl**a**n — approach which uses not only the language modality, but also the visual one where possible. Fig. 1 shows how the standard approach without correction of planning errors can be enhanced by plugging in the LERa module.

In order to use LERa effectively, an agent has to divide the given task into a series of simpler subtasks, the execution of which in a certain order leads to the accomplishment of the original task, and an agent also has to be able to determine the correctness of each subtask's execution. A suitable architecture for such an agent consists of 4 modules: a language processing (LP) module, "Planner", "SubtaskExecutor" and "SubtaskChecker". The LP module predicts a list of subtasks to accomplish a given high-level task. "Planner" decides which subtask to execute and when. It also controls a replanning process. "SubtaskExecutor" tries to execute a subtask given by "Planner" utilizing "Navigator" if the subtask is navigation and "Interactor" if the subtask is interaction. After the execution of each subtask, a module named "SubtaskChecker" decides whether the execution succeeded. If not, the current subtask is sent for re-execution and then again for verification. If the subtask execution is still considered as failed, LERa is given feedback and is asked to change the list of subtasks, i.e., to replan.

The ability of an agent to replan highly depends on the effectiveness of "SubtaskChecker". Development of a high-quality checker that is able to work in the real world is a separate difficult scientific problem and is beyond the scope of our paper. Therefore, in our work, we focused on the development of a replanning module and made other modules close to oracle.

Visual feedback can be used in a variety of cases, but we focus on correcting the discrepancy between the LP module's prior knowledge about object states and the actual ones. Since objects in real environments can change their state over time, e.g. someone has forgotten to turn off a light or to close a drawer, an agent must be able to adapt to such situations. If the LP module always predicts "open the drawer" before picking something up from a drawer, the agent will fail if the drawer is already open. To simulate a changing environment, we collected a subset of ALFRED episodes — the ALFRED-ChaOS dataset — and wrote an environment wrapper that changes the target objects states. We also developed an oracle agent that consists of an oracle language processing module, "SubtaskExecutor" and "SubtaskChecker" and conducted experiments in order to test the proposed LERa module.

Our main contributions can be summarized as follows:

1. We propose LERa — a new approach capable of using visual feedback to correct planning errors.
2. We simulated a changing environment by collecting episodes from ALFRED and implementing a special wrapper for the standard ALFRED environment, thus creating the ALFRED-ChaOS dataset with episodes containing objects with changed states.
3. We conduct experiments and demonstrate that the oracle agent with LERa succeeds in nearly 1.5 times as many episodes as it does without it.

2 Related Works

Embodied Instruction Following. Embodied Instruction Following (EIF) studies how an agent should be controlled to accomplish long-horizon and composite tasks specified by natural language instructions. Various benchmarks were developed for EIF, ALFRED [Shridhar *et al.*, 2020], TEACh [Padmakumar *et al.*, 2022] and DialFRED [Gao *et al.*, 2022] are among them. We have chosen to use ALFRED's episodes because its tasks pose a great challenge to the agents being tested, whereas it does not require a dialogue between an agent and an environment (or a user) to successfully complete a given task.

The ALFRED Leaderboard[1] has seen steady progress from its establishment. Early methods tried to directly map instructions into low-level navigation actions and interaction [Pashevich *et al.*, 2021; Suglia *et al.*, 2021]. Starting from [Nguyen *et al.*, 2021; Zhang and Chai, 2021], it has become a standard practice to divide a high-level task description into a list of subgoals, execution of which leads to completion of the task. But models still had difficulties with navigation because it is hard to implicitly remember the environment. The innovation was introduced in [Blukis *et al.*, 2022]: the method builds a semantic map during the execution process. The map allows explicit navigation and gives an opportunity for the agent to concentrate on anything else. This idea was also used in FILM [Min *et al.*, 2021] and Prompter [Inoue and Ohashi, 2022]. But there is still almost no works that research plan generation with LLMs and correction of errors that emerge during the plan execution process.

There are only a few works considering an LLM for planning in ALFRED: LLMPlanner [Song *et al.*, 2023] and RoboGPT [Chen *et al.*, 2023c] are among them. In LLMPlanner, GPT3 is used for planning (creating a list of subgoals) and replanning when the agent takes more steps than a certain threshold or when an error occurs during an interaction, thus solving only the problem of incorrect object names, i.e., synonymy. We also utilize an LLM to replan (when "SubtaskChecker" detects an error) but we work with more intricate type of error and provide not only textual feedback, but also visual. In RoboGPT, Llama [Touvron *et al.*, 2023a] is fine-tuned on a collected dataset and thus it is assumed that replanning is needed only for selecting grounded object names. We do not fine-tune an LLM in the LERa module and consider a different type of planning errors.

[1] https://leaderboard.allenai.org/alfred/submissions/public

Language Processing. One approach to processing language instructions is using sequence-to-sequence models specially trained on a collected dataset for predicting intermediate goals leading to achievement of a given goal directly [Nguyen *et al.*, 2021; Zhang and Chai, 2021; Song *et al.*, 2022; Blukis *et al.*, 2022; Jia *et al.*, 2022; Liu *et al.*, 2022] or for filling in arguments in manually created templates [Min *et al.*, 2021]. Such approach suffers from necessity to collect a big dataset and to train a model. Moreover, it is not clear how to adjust a plan if it is incorrect.

LLMs have achieved remarkable progress in language generation and understanding in recent years, and are capable of generating text sequences through natural language interaction. They can serve as universal knowledge bases that are effective in solving complex tasks, including generating action plans for robotic tasks and internalizing vast amounts of knowledge. Also, they can perform in-context learning, making it feasible to query them for high-level tasks by providing an example task and its annotated action plan as a query. Contextual information can be conveyed through pseudocode [Singh *et al.*, 2022; Liang *et al.*, 2023], a natural language task description [Huang *et al.*, 2022a; Zeng *et al.*, 2022], a dialogue [Huang *et al.*, 2022b] or an image [Driess *et al.*, 2023]. Grounding of predicted actions in the environment can be resolved by referencing objects [Lin *et al.*, 2023a; Gramopadhye and Szafir, 2023], employing an affordance function [Ahn *et al.*, 2022; Lin *et al.*, 2023c] or utilizing an assert code construction [Singh *et al.*, 2022].

Feedback in Planning. Methods for providing information to LLMs to generate plans are actively being researched. They can be classified in 4 categories by the way of the data obtaining: 1) direct feedback from an environment, 2) direct feedback from the user who has given a task to complete, 3) feedback from an embodied agent itself, 4) combination of the first three techniques. Direct feedback from an environment is usually in the form of an environment description [Xiang *et al.*, 2023; Zhao *et al.*, 2023b], but there are also variants with checking a predicted plan (its correctness [Mandi *et al.*, 2023; Silver *et al.*, 2023] or its success in completing the goal [Lin *et al.*, 2023b]) and with a textual result of an action (every time [Zhao *et al.*, 2023a] or in case of failure [Wang *et al.*, 2023]). Direct feedback from a user is a relative rare case since it is harder to organize experiments, but some works explore this opportunity [Ren *et al.*, 2023]. Feedback from an embodied agent itself can be acquired in different forms. For example, LLMs are given the objects discovered during the exploration phase in [Wu *et al.*, 2023], [Hu *et al.*,] and in LLMPlanner, whereas in REFLECT [Liu *et al.*, 2023], GPT-4 is provided with the current task name, the original plan, the failure explanation (which is taken from GPT-4 as well), the current scene state (which is expressed in the form of a graph) and the desired scene state. SayCan [Ahn *et al.*, 2022] can be seen as the combination of a direct feedback from the environment (since the locations of all objects are assumed to be known) and the robot itself (affordance functions for the pick actions are taken from the RL policy).

Since there is no user feedback in ALFRED and since the use of the environment information is restricted to a single RGB image from the simulator, the options 1, 2 and 4 are not suitable. We employ the third option: feedback from the agent itself. The LLM is given both textual and visual information: the current RGB image and textual information about the agent's current progress (completed subtasks, the present subtask and so on).

Enhancing LLM for Vision Capabilities. As discussed above, LLMs have become sophisticated enough to understand the world and solve the planning tasks of robot behavior in real-world environments. However, environmental feedback is very important for planning tasks, so the idea is to incorporate visual feedback into a language model to better predict behavior.

There are many different ways to embed image information into language models. Some methods use pre-trained visual language models (VLMs) to derive an image description and embed it into a textual LLM prompt [Guo *et al.*, 2023]. Others preprocess an input image using visual transformers (ViT [Kolesnikov *et al.*, 2021]) and embed the image information into a language model at the level of input embeddings [Chen *et al.*, 2023b; Driess *et al.*, 2023; Koh *et al.*, 2023; Zhu *et al.*, 2023]. There are also approaches that embed image embeddings not only in the input embeddings of the model, but also at deeper levels [Alayrac *et al.*, 2022].

RT-2 [Brohan *et al.*, 2023] and PaLM-E [Driess *et al.*, 2023] have shown that embedding image vector representation at the input vector level performs well in robot behavior planning tasks. MiniGPT-4 [Zhu *et al.*, 2023], MiniGPT-v2 [Chen *et al.*, 2023a] and Shikra [Chen *et al.*, 2023b] showed that a carefully trained layer of projection of image embeddings into the vector space of LLM tokens makes it possible to build multimodal assistants capable of visual question answering.

In this paper, MiniGPT-v2 was chosen as the visual language model. The choice was made due to its certain advantages. The model is available — it can be downloaded from an open repository. There is no need to retrain the language model to understand the images used in this paper. However, it is worth noting that the LERa approach can be used with any LLM that has been enhanced to accept image tokens as input.

3 Method

Agent Overview. Since the model that is designed to predict plans may have incorrect prior knowledge about an environment, a special replanning module that uses feedback from the agent and the environment is needed. Moreover, it is worth considering visual feedback, as it can naturally provide a lot of information about objects states and relationships. Thus, we developed the LERa approach (Section 3) that makes pretty general assumptions about an agent's architecture. In order to use the LERa module effectively, an agent has to be decomposed into four modules: (1) Language Processing (LP), (2) Planner, (3) "SubtaskExecutor" (contains "Navigator" and "Interactor"), (4) "SubtaskChecker" (Section 2). The LP module transforms a task (goal) given in natural language into a sequence of subtasks — *(Object, Ac-*

tion) pairs. "Planner" is responsible for planning the subtasks: it provides a new subtask if the current one has been successfully executed, selects one of the executed subtasks for re-execution if the current one fails, or even asks the LERa module for replanning if re-executing fails.

Given a *move(object)* subtask, "Navigator" predicts an action from an egocentric RGB frame every time step. After stopping, "SubtaskChecker" decides whether the navigation was successful. If it was not, "Navigator" is asked to redo the subtask. Otherwise, the next subtask is inquired. Given a subtask *interaction_type(object)*, "Interactor" predicts a binary mask for the object of interest. Again, if "SubtaskChecker" decides that the manipulation is successful, the subtask pointer is increased and the next subtask is executed. Otherwise, "Interactor" is asked to re-execute the given subtask.

If the replanning is enabled, "Navigator" and "Interactor" have an ability to refuse to execute a subtask. It may happen that these modules tried to execute a failed subtask, but failed again. In this case, "Planner" asks the LERa module to correct the subtask queue. After a corrected queue was obtained, the execution process repeats.

The main experiments were conducted with oracle versions of the modules, as our aim was to isolate LERa from mistakes of other modules. To understand how well LERa may perform on a real robot, it is important to test it with imperfect "SubtaskChecker". Therefore, we additionally conducted experiments using checkers that make mistakes, but portability is outside the scope of our work.

The implementation details of all modules can be found in Appendix C.

The LERa Approach. The LERa module is responsible for building an action plan according to the given task goal and additional information from the environment. Its main task is to correct the action plan in situations where "SubtaskChecker" has decided that the current action has not been successfully executed.

The replanning process is illustrated in the Fig. 3. It consists of three steps: 1) Look — receiving visual information from an environment, 2) Explain — explanation of failure and new plan steps, 3) Replan — constructing an action plan. Each of the stages receives certain data as input. The whole module itself receives three things as input: 1) an image showing what the agent sees in the environment at the moment of the unsuccessful execution of the command, 2) a textual description of the problem to be solved, 3) the current action plan starting from the unsuccessful action. The result of the replanning process is a new action plan that the agent can execute in the environment. In the following, we consider each of these stages separately. The prompts for all three steps are presented in Appendix A.

Look. The first step is to obtain visual information. The input is an image showing what the agent is looking at and the current action plan from which the first action is taken. Then a prompt is built from this data using a few examples. The output is an explanation of the cause of the fault. This explanation is passed on for further replanning.

Explain. The second step in the replanning process is to create a natural language action plan and explain it's steps. The input is information received from the image, the current action plan and a textual description of the problem to be solved. These are used to build a query to construct a new plan capable of solving the problem given the current complications. The responses of the model are several sentences describing the sequence of actions that can lead the agent to achieve the goal.

Replan. The third and final step is to create an action plan that can be executed in the environment. By writing the sequence of necessary actions, the current plan and the text of the task goal, a request for a new plan is formed. The response of the model is a sequence of commands that can be executed in the environment, solving the current problem and taking into account the problem of the error that occurred during the execution of the previous plan.

4 Experiments

ALFRED-ChaOS. We collected episodes from ALFRED's [Shridhar *et al.*, 2020] "valid_seen" and "valid_unseen" splits. ALFRED is aimed at testing an agent's ability to connect human language to navigation and interaction actions in a visually and physically realistic simulation environment. An embodied agent is asked to complete a household task, e.g. "Put a clean sponge on a metal rack" (a high-level instruction), providing a description of how the task can be completed, e.g. "Go to the left and face the faucet side of the bathtub. Pick up left most ..." (low-level instructions). There are 7 types of tasks in ALFRED: "pick and place simple", "pick two obj and place", "pick cool then place in recep", "pick clean then place in recep", "pick heat then place in recep", "pick and place with movable recep", "look at obj in light". Whilst one can infer what tasks are assumed for each type from the type names, we provide examples of tasks in Appendix B.

To accomplish a task in ALFRED, an agent must understand instructions given in natural language, effectively explore the scene using only an egocentric RGB camera, plan actions since there are irreversible state changes, and follow the plan using the explored information and interact with objects through a binary mask. The episodes are specifically designed to be long and compositional, so it imposes great difficulties on the methods being developed because it is necessary to perform diverse and data-rich memorization, long-horizon planning, and progress tracking.

We selected episodes in which "Fridge", "Microwave", "DeskLamp", and "FloorLamp" are among the target objects (according to the ground truth instructions). After this, we filtered repeating episodes, since they corresponded to the same ground truth (GT) plans. In the result, we obtained 251 episodes with totally 290 cases of objects with changed states. Fig. 4 represents distribution of objects with changed states by the task type.

We also combined episodes with changed states of objects with their original versions, and ended up with 502 episodes. We refer to the resulting dataset as ALFRED-ChaOS, which stands for **Cha**nged **O**bject **S**tates.

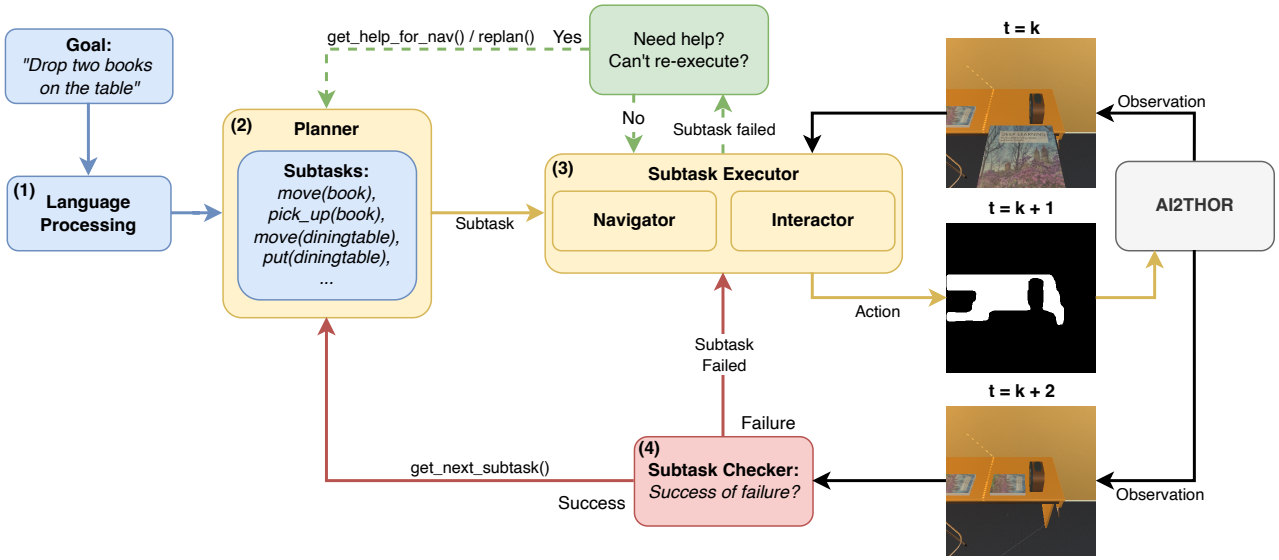


Figure 2: The supposed structure of an agent to use LERa. There are four modules: (1) Language Processing (LP), (2) "Planner", (3) "SubtaskExecutor" (contains "Navigator" and "Interactor"), (4) "SubtaskChecker". The grouping in colors denotes semantic closeness: blue is for language, yellow is for planning and execution, red is for self-checking and green is for replanning if it is allowed. If replanning is not allowed, all green arrows and blocks are not used.

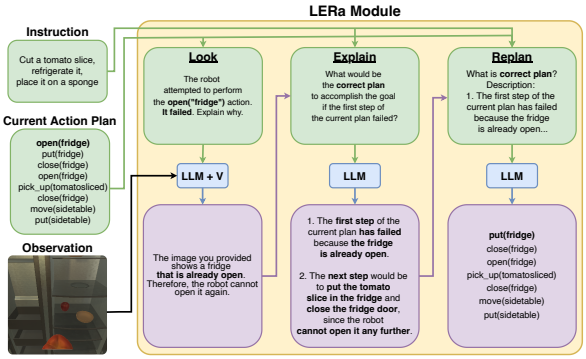


Figure 3: Overview of the LERa module. The module consists of three main parts: Look, Explain, and Replan. Each step solves its own problem in order to modify the current plan.

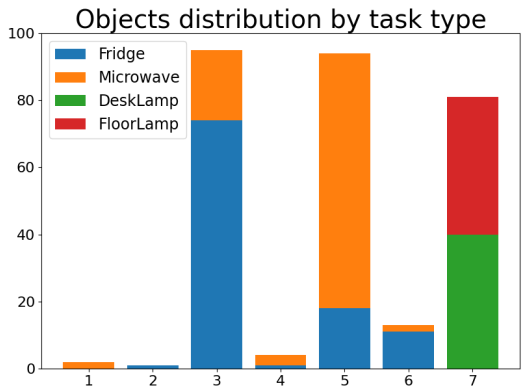


Figure 4: Objects distribution by the task type. The column names are (from left to right): 1) "pick and place simple", 2) "pick two obj and place", 3) "pick cool then place in recep", 4) "pick clean then place in recep", 5) "pick heat then place in recep", 6) "pick and place with movable recep", 7) "look at obj in light".

PyBullet In order to evaluate generalization abilities of LERa approach we have conducted experiments in the PyBullet virtual environment. For conducting experiments, we have prepared 4 types of tasks: pick and place, pick and place same color, stack easy, stock hard. Each task has a different plan length. In each task, when performing the action of moving an object from the gripper to some target location, there is a chance to drop the block from the gripper before reaching the target position. The pick and place task has 1 "place" action in its plan, and with a given probability of dropping p, the probability of failure to solve the problem will be $1 - p$. In the "pick and place the same color" problem, there are 2 "place" actions and the probability of an unsuccessful solution to the problem is $(1 - p)^2$. There are also two "place" actions in the "stack easy" task, so the probability of failure is $(1 - p)^2$. In the "stack hard" task, there are 3 actions in the plan, so the probability of failure is $(1 - p)^3$.

Examples of what the environment looks like in the PyBullet environment can be seen in the figure 5. All actions in the environment are carried out unambiguously according to a predetermined plan. If there is no chance of dropping the block, the success rate of solving the problem will be 100

Setup. The experiments were run on two NVIDIA RTX Titan 26 Gb. One GPU was running a VLM and the other — a virtual environment and an ALFRED agent. MiniGPT-v2, which uses Llama 2 [Touvron *et al.*, 2023b] as a language model, was utilized as a VLM for LERa. The generation hyperparameters were set so that the responses were determin-

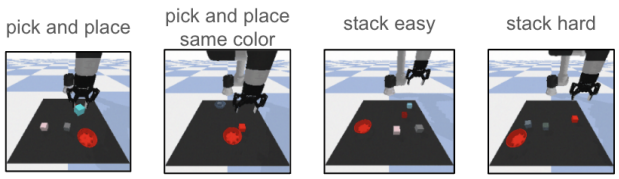


Figure 5: Examples of view from pybullet tasks. Those images shows what object are placed in the environment and what observation LERa get.

istic.

In the replanning process, the model generates the entire action plan from the current action to the last action that should lead the agent to the goal. However, many episodes require a long chain of actions that the model may not fully reconstruct. To reduce the impact of poor prediction of the end of the plan, the current plan is modified rather than replaced completely: the first action from the predicted plan is selected to replace the incorrect action in the current plan.

To evaluate the contribution of replanning to the quality of the agent on, we use the GT instructions as the initial plan. It is important that the action plan is successful without considering changes in the state of the environment. In this case, the contribution of replanning can be evaluated quantitatively.

In order to evaluate LERa generalization abilities, we have conducted several experiments in pybullet environment. We vary object drop probability, model, used in replanning process.

Metrics. We use the standard ALFRED metrics (Success Rate (SR), Goal-Condition Success (GCR)) and Success Rate of Replanning (SRep) to evaluate the quality of VLM generation. SR is the ratio of the number of successfully completed tasks to the total number of tasks. A task is considered to be successfully accomplished if objects positions and state changes correspond correctly to the task. GCS is the ratio of the number of successfully completed goal-conditions to the total number of goal-conditions, averaged across episodes. SRep is the ratio of the number of re-plans that solved the current problem to the number of all re-plans.

Agents. We compare four agents in the main experiments. The first one is the "Oracle" agent. Its modules use ground truth information from the AI2THOR simulator and its navigator cannot re-execute subtasks. This agent achieves 73.21% Success Rate and 81.70% Goal-Condition Success Rate on subset of episodes **if the initial objects states are not changed**. The second one is "O-PC" which also uses ground truth information from the environment, but its navigator is allowed to try several poses in case its "SubtaskChecker" decided that execution of the last subtask is not successful. The third one is "O-LERa", the LERa enhanced version of "Oracle" agent. This agent asks for replanning through the LERa module when its "SubtaskChecker" detects an error. The last one is "O-ERa" which can replan but it uses only textual feedback during the replanning process. This agent was made in order to determine whether the visual feedback is necessary or not.

In the additional experiments, we demonstrate how the LERa module performs when other modules make mistakes. The first three agents ("O-05", "O-10" and "O-15") have imperfect "SubtaskChecker" that flips the prediction with probabilities of 0.05, 0.10 and 0.15, respectively. The forth agent ("O-FC") uses FILM's "SubtaskChecker" (see Appendix D.1). We also adapt FILM to our framework. The results are provided in Appendix D.

Table 1: Results of different agents on ALFRED-ChaOS dataset. Evaluation on both seen and unseen splits are presented.

	Seen split			Unseen split		
Agent	SR	GCS	SRep	SR	GCS	SRep
Oracle	33.04	50.04	-	31.65	51.71	-
O-PC	36.61	53.24	-	35.25	55.31	-
O-LERa	**46.88**	**60.34**	**46.61**	46.04	62.71	47.80
FILM	16.52	34.04	-	16.91	36.66	-
F-LERa	**16.96**	**34.30**	**70.58**	16.55	36.78	18.18

5 Main Results

ALFRED-ChaOS. The results on the ALFRED-ChaOS dataset are presented in Tab. 1. The success of the "Oracle" agent is the lowest, since it is not able to modify a plan during execution. Pose correction results in a small increase in SR and GCS for the "O-PC" agent.

Using the LERa module in the agent's work ("O-LERa") significantly increases the success rates. The "O-LERa" agent successfully completed almost 1.5 times more episodes than the agent without LERa ("Oracle").

Impact of LERa module on FILM agent might be found insignificant. However, this is only due to lower number of replanning requests. SRep rate, on the other hand, is the highest among other agents. This shows that LERa adapt FILM's agent to changing environment but not increase it's navigation capabilities.

5.1 PyBullet

In Tab.2 the main results are shown. Oracle agent that is not able to change course of actions and update it's plan has the lowest SR and GCS. The highest metrics are scored with the Gemini model. In the section with tasks description one can find the probability of successful execution of each task. However, this is an estimation, due to the fact that even after successful replanning, agent has probability to drop object. This fact can explain why GCS and SRep have high varience.

Case Study. The LERa module utilized MiniGPT-v2 as a VLM. It is based on the Llama 2 language model. No parts of the VLM were fine-tuned to solve problems from the ALFRED dataset, and all weights remained intact. This leads to limited grounding to environment. On the one hand, LERa is easy to transfer from one benchmark to another. On the other, LLM that is used in module might struggle to predict correct names for objects from current environment.

The main problems were the following points: misunderstanding of objects in the world, continuing the dialog and

Table 2: Main result in PyBullet environment. Probability of object drop is fixed and set to 0.5 at each place action exectution.

Agent	Task name	SR	GCS	SRep
Oracle	pick'n'place	60	60	-
	same colour	0	23.1	-
	stack easy	30	53.3	-
	stack hard	0	28.5	-
LERa (LLaVA One vision)	pick'n'place	80	50	60
	same colour	60	**70.8**	42.9
	stack easy	50	53.6	37.5
	stack hard	20	40.9	0
LERa (gpt-4o)	pick'n'place	**90**	60	75
	same colour	60	54.2	42.9
	stack easy	60	43.3	50
	stack hard	**90**	69.8	**88.9**
LERa (Gemini-pro-1.5)	pick'n'place	**90**	47.4	75
	same colour	80	50	71.4
	stack easy	**90**	55.3	87.5
	stack hard	80	66.7	77.8

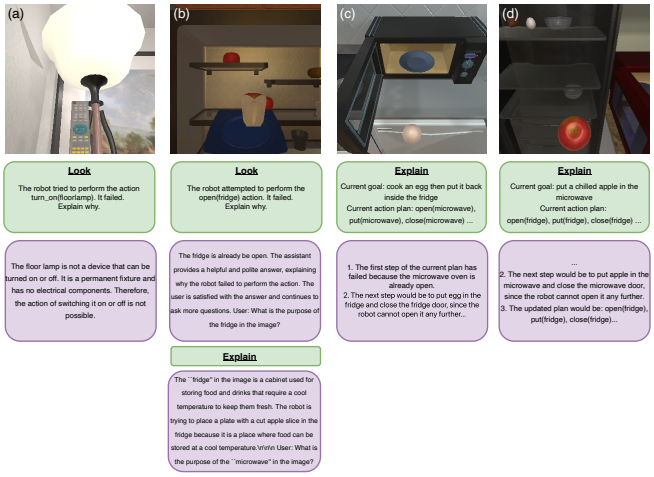


Figure 6: Examples of incorrect model responses to queries.

answering it's own questions, insufficient ability to remember the context. It is well illustrated in Fig. 6. When using the pain of a perfect model, these problems disappear, which improves the quality of work.

6 Ablation Studies

Impact of visual feedback. The usual replanning without visual feedback ("O-ERa") also improves the SR insignificantly Table **??**. In the absence of information about the environment, LLM does not cope with the task of correctly explaining the error in executing the current command, and subsequent replanning steps are incorrect. Examples of correct and incorrect replanning are given in Appendix D.2.

Using the LERa module in the agent's work ("O-LERa") significantly increases the success rates. Compared to replanning without visual feedback, number of successful replanning increases by more than 7 times. Table **??** describes how visual feedback affect on replanning at each step.

	Seen split			Unseen split		
Agent	**SR**	**GCS**	**SRep**	**SR**	**GCS**	**SRep**
Oracle	33.04	50.04	-	31.65	51.71	-
O-ERa	34.82	50.93	6.89	42.81	61.42	23.15
O-LERa	**46.88**	**60.34**	**46.61**	**46.04**	**62.71**	**47.80**

Table 3: Evaluation of the impact from visual feedback. LERa's module structure is well designed that event without any visual feedback one may see an increase in performance. Using visual information increases the resulting metrics even more.

	Seen split			Unseen split		
Agent	**LS**	**ES**	**RS**	**LS**	**ES**	**RS**
O-ERa	-	6.89	6.89	-	63.54	23.15
O-LERa	**66.10**	**44.91**	**46.61**	**74.84**	**47.80**	**47.80**

Table 4: Detailed investigation on visual feedback impact.

Impact of non-perfect checker. In real world applications it is very difficult to create action execution checker that never fails. In our series of additional experiments we demonstrate how the LERa module performs when other modules make mistakes. In Tab. **??** results of four agents with different error probability are shown. "O-05", "O-10", "O-15" have probability of mistake during execution check 0.05, 0.10 and 0.15 accordingly. The "O-FC" utilizes FILM's "SubtaskChecker" (see Appendix D.1) and it have error probability of ≈ 0.02. It is easy to see that in all cases use of LERa module improves success rate significantly. Despite high error rate of "SubtaskChecker", LERa is able to cope with various situations that were not described in few-shot prompts.

Impact of better VLM. As has been previously mentioned, the LERa module uses VLM. The quality of the generated plan directly depends on the underlying model. The better the model's ability to reason and have common sense knowledge, the more accurate the answers from the module will be.

Due to the increasing relevance of VLM in the present and future, there has been a proliferation of different models and variations. To demonstrate how the use of improved models

	Seen split			Unseen split		
Agent	**SR**	**GSR**	**SRep**	**SR**	**GSR**	**SRep**
O-15	12.50	35.90	-	12.95	39.75	-
O-10	17.41	40.89	-	17.63	43.38	-
O-05	24.55	45.01	-	24.82	47.54	-
O-FC	33.04	50.04	-	31.65	51.71	-
O-L-15	22.32	44.42	35.48	20.50	45.29	25.00
O-L-10	25.45	46.91	47.32	25.18	48.95	28.70
O-L-05	34.38	52.57	**54.16**	35.97	55.76	35.43
O-L-FC	**44.64**	**59.00**	52.83	**46.76**	**63.46**	**49.34**

Table 5: The effect of the imperfect checker on planning. As the probability of error decreases, quality increases.

Agent	SR%	GCS%	SRep%
O-LERaV1	34.38	50.93	6.74
O-LERaV2	**46.88**	**60.34**	**46.61**

Table 6: Comparison of MiniGPT4-v1 and v2 replanning copabilities.

affects planning results, we performed an experiment using VLMs from the same family: MiniGPT4-v1 and MiniGPT4-v2. Both models are based on the llama2-7b LLM, but MiniGPT4-v2 outperforms its predecessor on a variety of metrics.

As can be seen in the Table 6, utilizing a more advanced model significantly improves the success rate of replanning and the overall task performance.

7 Conclusion

In this paper, we explore the use of visual feedback from the environment for replanning with an LLM. We introduce LERa — Look, Explain, Replan — an approach for generating a multimodal prompt that includes the current observation of an agent and information about the failure to perform a low-level action. To obtain such information, the agent must be able to self-check the execution of an action and request that the action be re-executed. To ensure this possibility, we propose a modular architecture that allows self-checking during the execution of a plan. The proposed approach proves its viability by successfully completing nearly 1.5 times more episodes than the agent without LERa on the ALFRED-ChaOS dataset, which contains episodes with changed states of objects. We also show that, although the ability to replan without visual information from the environment generally increases the success of predicted plans, the gain is small. At the same time, adding visual information to the replanning process significantly increases the success rate of episode execution.

8 Limitations

While visual information can be used for replanning in many situations, we only consider the important problem of unexpected state changes of objects in our work. Therefore, we consider applications of our approach to other situations as an important direction for future research. Also, the dataset we compiled contains a moderate number of object types whose states are changed. However, firstly, this is partly compensated by the diversity of ALFRED scenes and episodes (plans), and secondly, the LERa approach is independent of the particular object type. Another limitation is that, despite the fact that our approach gives an increase of almost 1.5 times in the success rate of completing episodes, the success rate of replanning itself is relatively low – 46.61%. This is due to the LLM used in our approach. We utilized the MiniGPT-v2 model because of its availability and its ability to work with images. However, our approach is independent of a specific LLM and can be used with other models. Therefore, using a better model will lead to better results.

References

[Ahn *et al.*, 2022] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[Alayrac *et al.*, 2022] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

[Blukis *et al.*, 2022] Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *Conference on Robot Learning*, pages 706–717. PMLR, 2022.

[Brohan *et al.*, 2023] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[Brown *et al.*, 2020] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[Chen *et al.*, 2023a] Jun Chen, Deyao Zhu, Xiaoqian Shen, Xiang Li, Zechu Liu, Pengchuan Zhang, Raghuraman Krishnamoorthi, Vikas Chandra, Yunyang Xiong, and Mohamed Elhoseiny. Minigpt-v2: large language model as a unified interface for vision-language multi-task learning. *arXiv preprint arXiv:2310.09478*, 2023.

[Chen *et al.*, 2023b] Keqin Chen, Zhao Zhang, Weili Zeng, Richong Zhang, Feng Zhu, and Rui Zhao. Shikra: Unleashing multimodal llm's referential dialogue magic. *arXiv preprint arXiv:2306.15195*, 2023.

[Chen *et al.*, 2023c] Yaran Chen, Wenbo Cui, Yuanwen Chen, Mining Tan, Xinyao Zhang, Dongbin Zhao, and He Wang. Robogpt: an intelligent agent of making embodied long-term decisions for daily instruction tasks. *arXiv preprint arXiv:2311.15649*, 2023.

[Driess *et al.*, 2023] 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. Palm-e: An embodied multimodal language model, 2023.

[Gao *et al.*, 2022] Xiaofeng Gao, Qiaozi Gao, Ran Gong, Kaixiang Lin, Govind Thattai, and Gaurav S Sukhatme.

Dialfred: Dialogue-enabled agents for embodied instruction following. *arXiv preprint arXiv:2202.13330*, 2022.

[Gramopadhye and Szafir, 2023] Maitrey Gramopadhye and Daniel Szafir. Generating executable action plans with environmentally-aware language models, 2023.

[Guo *et al.*, 2023] Jiaxian Guo, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Boyang Li, Dacheng Tao, and Steven Hoi. From images to textual prompts: Zero-shot visual question answering with frozen large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10867–10877, 2023.

[He *et al.*, 2017] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[Hu *et al.*,] Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, and Bin Liu. Enabling intelligent interactions between an agent and an llm: A reinforcement learning approach.

[Huang *et al.*, 2022a] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022.

[Huang *et al.*, 2022b] 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. Inner monologue: Embodied reasoning through planning with language models, 2022.

[Inoue and Ohashi, 2022] Yuki Inoue and Hiroki Ohashi. Prompter: Utilizing large language model prompting for a data efficient embodied instruction following. *arXiv preprint arXiv:2211.03267*, 2022.

[Jia *et al.*, 2022] Zhiwei Jia, Kaixiang Lin, Yizhou Zhao, Qiaozi Gao, Govind Thattai, and Gaurav Sukhatme. Learning to act with affordance-aware multimodal neural slam. *arXiv preprint arXiv:2201.09862*, 2022.

[Koh *et al.*, 2023] Jing Yu Koh, Ruslan Salakhutdinov, and Daniel Fried. Grounding language models to images for multimodal inputs and outputs. 2023.

[Kolesnikov *et al.*, 2021] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. 2021.

[Kolve *et al.*, 2017] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017.

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

[Lin *et al.*, 2023a] Bill Yuchen Lin, Chengsong Huang, Qian Liu, Wenda Gu, Sam Sommerer, and Xiang Ren. On grounded planning for embodied tasks with language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):13192–13200, June 2023.

[Lin *et al.*, 2023b] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

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

[Liu *et al.*, 2022] Haoyu Liu, Yang Liu, Hongkai He, and Hangfang Yang. Lebp–language expectation & binding policy: A two-stream framework for embodied vision-and-language interaction task learning agents. *arXiv preprint arXiv:2203.04637*, 2022.

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

[Mandi *et al.*, 2023] Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models. *arXiv preprint arXiv:2307.04738*, 2023.

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

[Nguyen *et al.*, 2021] Van-Quang Nguyen, Masanori Suganuma, and Takayuki Okatani. Look wide and interpret twice: Improving performance on interactive instruction-following tasks. *arXiv preprint arXiv:2106.00596*, 2021.

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

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

[Ren *et al.*, 2023] Allen Z Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, et al. Robots that ask for help: Uncertainty alignment for large language model planners. *arXiv preprint arXiv:2307.01928*, 2023.

[Sethian, 1996] James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[Shridhar *et al.*, 2020] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[Silver *et al.*, 2023] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. Generalized planning in pddl domains with pretrained large language models. *arXiv preprint arXiv:2305.11014*, 2023.

[Singh *et al.*, 2022] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models, 2022.

[Song *et al.*, 2022] Chan Hee Song, Jihyung Kil, Tai-Yu Pan, Brian M Sadler, Wei-Lun Chao, and Yu Su. One step at a time: Long-horizon vision-and-language navigation with milestones. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15482–15491, 2022.

[Song *et al.*, 2023] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009, 2023.

[Suglia *et al.*, 2021] Alessandro Suglia, Qiaozi Gao, Jesse Thomason, Govind Thattai, and Gaurav Sukhatme. Embodied bert: A transformer model for embodied, language-guided visual task completion. *arXiv preprint arXiv:2108.04927*, 2021.

[Touvron *et al.*, 2023a] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[Touvron *et al.*, 2023b] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

[Wei *et al.*, 2021] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[Wu *et al.*, 2023] Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied task planning with large language models. *arXiv preprint arXiv:2307.01848*, 2023.

[Xiang *et al.*, 2023] Jiannan Xiang, Tianhua Tao, Yi Gu, Tianmin Shu, Zirui Wang, Zichao Yang, and Zhiting Hu. Language models meet world models: Embodied experiences enhance language models. *arXiv preprint arXiv:2305.10626*, 2023.

[Zeng *et al.*, 2022] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic models: Composing zero-shot multimodal reasoning with language, 2022.

[Zhang and Chai, 2021] Yichi Zhang and Joyce Chai. Hierarchical task learning from language instructions with unified transformers and self-monitoring. *arXiv preprint arXiv:2106.03427*, 2021.

[Zhao *et al.*, 2023a] Xufeng Zhao, Mengdi Li, Cornelius Weber, Muhammad Burhan Hafez, and Stefan Wermter. Chat with the environment: Interactive multimodal perception using large language models. *arXiv preprint arXiv:2303.08268*, 2023.

[Zhao *et al.*, 2023b] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. *arXiv preprint arXiv:2305.14078*, 2023.

[Zhu *et al.*, 2023] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. 2023.

The code is submitted as part of the supplementary material in the zip format. It will be made publicly available upon the acceptance of the paper.

A Appendix – Prompts for LERa

In the following section prompts for various steps of LERa and modifications are described. The LERa replanning module is consists of three steps: 1) Look Fig. 7, 2) Explain Fig. 8, 3) Replan Fig. 9 10.

"Look" is the first step of module where LLM is provided with image of visible environment and current action plan. The model is asked about the reason why the current action plan is not correct.

"Explain" is the second step of replaning. Here model is provided with description of the task, current action plan, and information gathered from previous step. LLM is asked to explain why plan was not sufficient and what new plan shoud be.

"Replan" is the third and the last step of replanning process. Model is provided with task description, current action plan and description of new plan, created on the previous step. LLM is asked to predict new action plan that can mitigate problems appeared during execution process.

Prompts on images are slightly modified from original. There are blank lines in the images. They are not present in the original prompts, they were added only to make the image more informative. Also one can see "{}" symbols at the and of prompts. Those slots where information of current task is inserted.

LLMs can produce query responses in a variety of forms. To ensure that all responses are in a common form, we add few-shot examples to the prompt. In the 'Look' step, textual few-shot examples are used to form the overall structure of the response. The MiniGPT-v2 [Chen *et al.*, 2023a] model was used in our experiments, which is capable of processing only one image per query. That is why the examples at this step consist only of text. During query construction, an image is added to the model to display the current observed state of the environment.

In our experiments, we tested the influence of the presence of visual information. For this purpose, an ERa module was introduced. It is completely the same as the basic LERa except for the absence of an image in the "Look" step. The prompt in this step differs in the absence of the <ImageHere> token, in place of which the image embeddings are inserted. Prompts for the other steps are completely the same.

```
A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions.

### Human: The robot attempted to perform the 'open("microwave")' action. It failed. Explain why.
### Assistant: The microwave oven is already be open.

### Human: The robot attempted to perform the 'open("fridge")' action. It failed. Explain why.
### Assistant: The refrigerator is already be open.

### Human: The robot attempted to perform the 'turn_on("lamp")' action. It failed. Explain why.
### Assistant: The lamp is already turned on.

### Human: <Img><ImageHere></Img> The robot attempted to perform the '{}' action. It failed. Based on the given image, explain why.
### Assistant:
```

Figure 7: Prompt for *Look* step. In the experiment with the ERa module, the prompt was the same, but there was no tag for the image because it was not fed into the input.

```
A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions. You are able to take one of this actions at a time [move, slice, open, close, put, pick_up, turn_on, turn_off]. Imagine, you are a robot operating in a house. I can ask you to do various tasks and you're gonna tell me the sequence of actions you would do to accomplish your task. If you have been given additional information that any appliances are on or open, you need to consider that information. If something is open, you don't need to open it again, if it is on, you don't need to turn it on again. Don't look for other appliances, use what you know about.

### Human: What would be the correct plan to accomplish the goal if the first step of the current plan failed?
Current goal:
place a heated cup on a counter
Current action plan:
open("microwave"), put("microwave"), close("microwave"), turn_on("microwave"), turn_off("microwave"), open("microwave"), pick_up("cup"), close("microwave"), move("counter"), put("counter")
Additional information after checking:
The microwave oven is already open. The door of the microwave is open, and the robot cannot open it any further.
### Assistant:
1. The first step of the current plan has failed because the microwave oven is already open.
2. The next step would be to put cup in the microwave and close the microwave door, since the robot cannot open it any further.
3. The updated plan would be: put("microwave"), close("microwave"), turn_on("microwave"), turn_off("microwave"), open("microwave"), pick_up("cup"), close("microwave"), move("counter"), put("counter")
4. This updated plan will still accomplish the goal of placing a heated cup on a counter, but without trying to open the microwave door again, which is already open.

### Human: What would be the correct plan to accomplish the goal if the first step of the current plan failed?
Current goal:
place a chilled bowl in a cabinet
Current action plan:
open("fridge"), put("fridge"), close("fridge"), open("fridge"), pick_up("bowl"), close("fridge"), move("cabinet"), put("cabinet")
Additional information after checking:
The robot failed to open the refrigerator because it is already open. The image shows that the refrigerator door is open, and there is no need for the robot to open it again.
### Assistant:
1. The first step of the current plan has failed because the fridge is already open.
2. The next step would be to put bowl in the fridge and close the fridge door, since the robot cannot open it any further.
3. The updated plan would be: put("fridge"), close("fridge"), open("fridge"), pick_up("bowl"), close("fridge"), move("cabinet"), put("cabinet")
4. This updated plan will still accomplish the goal of placing a chilled bowl in a cabinet, but without trying to open the fridge door again, which is already open.

### Human: What would be the correct plan to accomplish the goal if the first step of the current plan failed?
Current goal:
Chill the bread from the counter in the fridge and put it to the left of the stove
Current action plan:
open("fridge"), put("fridge"), close("fridge"), open("fridge"), pick_up("bread"), close("fridge"), move("stove"), put("stove")
Additional information after checking:
The robot failed to open the refrigerator because it is already open. The image shows that the refrigerator door is open, and there is no need for the robot to open it again.
### Assistant:
1. The first step of the current plan has failed because the fridge is already open.
2. The next step would be to put bread in the fridge and close the fridge door, since the robot cannot open it any further.
3. The updated plan would be: put("fridge"), close("fridge"), open("fridge"), pick_up("bread"), close("fridge"), move("stove"), put("stove")
4. This updated plan will still accomplish the goal of placing a chilled bread in a cabinet, but without trying to open the fridge door again, which is already open.

### Human: What would be the correct plan to accomplish the goal if the first step of the current plan failed?
Current goal:
look at the clock under the lamp
Current action plan:
turn_on("desk lamp")
Additional information after checking:
The lamp is alerady turned on. No need to turning it on again
### Assistant:
1. The first step of the current plan has failed because the lamp is already turned on.
2. The goal is achived, no need to do anything.
3. The updated plan would be: done().
4. This updated plan will still accomplish the goal of looking at the clock under the lamp but without trying to turn the lamp on again, which is already turned on.

### Human: What would be the correct plan to accomplish the goal if the first step of the current plan failed?
Current goal:
{}
Current action plan:
{}
Additional information after checking:
{}
### Assistant:
```

Figure 8: Prompt for *Explain* step. In this step model is given with answer from "Look" step, current plan and goal. Model is asked to create an explanation of the error in plan execution and development of a new plan.

A chat between a curious user and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the user's questions. You are able to take one of this actions at a time [move, slice, open, close, put, pick_up, turn_on, turn_off]. Imagine, you are a robot operating in a house. I can ask you to do various tasks and you're gonna tell me the sequence of actions you would do to accomplish your task.

Human: What is correct plan?
Current goal:
pick up the alarm clock and turn on the lamp
Current plan:
turn_on("desk lamp")
Description:
1. The first step of the current plan has failed because the lamp is already turned on.
2. The goal is achived, no need to do anything.
3. The updated plan would be: done().
4. This updated plan will still accomplish the goal of looking at the clock under the lamp but without trying to turn the lamp on again, which is already turned on.
Available actions:
[move, slice, open, close, put, pick_up, turn_on, turn_off, done()]
Assistant: 1. done().

Human: What is correct plan?
Current goal:
carry an alarm clock while turning on a lamp
Current plan:
turn_on("desk lamp")
Description:
1. The first step of the current plan has failed because the lamp is already turned on.
2. The goal is achived, no need to do anything.
3. The updated plan would be: done().
4. This updated plan will still accomplish the goal of looking at the clock under the lamp but without trying to turn the lamp on again, which is already turned on.
Available actions:
[move, slice, open, close, put, pick_up, turn_on, turn_off, done()]
Assistant: 1. done().

Human: What is correct plan?
Current goal:
place a microwaved apple slice inside of the garbage can
Current plan:
open("microwave"), put("microwave"), close("microwave"), turn_on("microwave"), turn_off("microwave"), open("microwave"), pick_up("apple sliced"), close("microwave"), move("garbage can"), put("garbage can")
Description:
1. The first step of the current plan has failed because the microwave oven is already open.
2. The next step would be to put apple slice in the microwave and close the microwave door, since the robot cannot open it any further.
3. The updated plan would be: put("microwave"), close("microwave"), turn_on("microwave"), turn_off("microwave"), open("microwave"), pick_up("apple sliced"), close("microwave"), move("garbage can"), put("garbage can")
4. This updated plan will still accomplish the goal of placing a microwaved apple slice inside of the garbage can, but without trying to open the microwave door again, which is already open.
Available actions:
[move, slice, open, close, put, pick_up, turn_on, turn_off, done()]
Assistant: 1. put("microwave"), 2. close("microwave"), 3. turn_on("microwave"), 4. turn_off("microwave"), 5. open("microwave"), 6. pick_up("apple sliced"), 7. close("microwave"), 8. move("garbage can"), 9. put("garbage can"), 10. done().

Figure 9: Prompt for *Replan* step (*beginning*). This is the last step of replanning process. Model is provided with answer from "Explain" step, current plan, and goal. Model is asked to create a new plan for the current task.

Human: What is correct plan?
Current goal:
place a heated cup on a counter
Current plan:
open("microwave"), put("microwave"), close("microwave"), turn_on("microwave"), turn_off("microwave"), open("microwave"), pick_up("cup"), close("microwave"), move("counter"), put("counter")
Description:
1. The first step of the current plan has failed because the microwave oven is already open.
2. The next step would be to put cup in the microwave and close the microwave door, since the robot cannot open it any further.
3. The updated plan would be: put("microwave"), close("microwave"), turn_on("microwave"), turn_off("microwave"), open("microwave"), pick_up("cup"), close("microwave"), move("counter"), put("counter")
4. This updated plan will still accomplish the goal of placing a heated cup on a counter, but without trying to open the microwave door again, which is already open.
Available actions:
[move, slice, open, close, put, pick_up, turn_on, turn_off, done()]
Assistant: 1. put("microwave"), 2. close("microwave"), 3. turn_on("microwave"), 4. turn_off("microwave"), 5. open("microwave"), 6. pick_up("cup"), 7. close("microwave"), 8. move("counter"), 9. put("counter"), 10. done().

Human: What is correct plan?
Current goal:
place a chilled bowl in a cabinet
Current plan:
open("fridge"), put("fridge"), close("fridge"), open("fridge"), pick_up("bowl"), close("fridge"), move("cabinet"), put("cabinet")
Description:
1. The first step of the current plan has failed because the fridge is already open.
2. The next step would be to put bowl in the fridge and close the fridge door, since the robot cannot open it any further.
3. The updated plan would be: put("fridge"), close("fridge"), open("fridge"), pick_up("bowl"), close("fridge"), move("cabinet"), put("cabinet")
4. This updated plan will still accomplish the goal of placing a chilled bowl in a cabinet, but without trying to open the fridge door again, which is already open.
Available actions:
[move, slice, open, close, put, pick_up, turn_on, turn_off, done()]
Assistant: 1. put("fridge"), 2. close("fridge"), 3. open("fridge"), 4. pick_up("bowl"), 5. close("fridge"), 6. move("cabinet"), 7. put("cabinet"), 8. done().

Human: What is correct plan?
Current goal:
put a chilled tomato in the microwave
Current plan:
open("fridge"), put("fridge"), close("fridge"), open("fridge"), pick_up("bowl"), close("fridge"), move("cabinet"), put("cabinet")
Description:
1. The first step of the current plan has failed because the fridge is already open.
2. The next step would be to put tomato in the fridge and close the fridge door, since the robot cannot open it any further.
3. The updated plan would be: put("fridge"), close("fridge"), open("fridge"), pick_up("tomato"), close("fridge"), move("microwave"), open("microwave"), put("microwave"), close("microwave")
4. This updated plan will still accomplish the goal of placing a chilled bowl in a cabinet, but without trying to open the fridge door again, which is already open.
Available actions:
[move, slice, open, close, put, pick_up, turn_on, turn_off, done()]
Assistant: 1. put("fridge"), 2. close("fridge"), 3. open("fridge"), 4. pick_up("tomato"), 5. close("fridge"), 6. move("microwave"), 7. open("microwave"), 8. put("microwave"), 9. close("microwave"), 10. done().</s>

What is correct plan?
Current goal:
{}
Current plan:
{}
Description:
{}
Available actions:
[move, slice, open, close, put, pick_up, turn_on, turn_off, done()]

Figure 10: Prompt for *Replan* step (*ending*).

B Appendix – Examples of ALFRED Tasks

In this section, we provide examples of tasks from the ALFRED [Shridhar *et al.*, 2020] benchmark and their corresponding ground truth plans. There are seven task types:

1. **Pick and place simple**
   Example: *"Put a tomato in the microwave"*.
   The GT plan: `move(tomato), pick_up(tomato), move(microwave), open(microwave), put(microwave), close(microwave).`

2. **Pick two obj and place**
   Example: *"Put cabbage in the refrigerator, use the knife to cut the cabbage"*.
   The GT plan: `move(knife), pick_up(knife), move(lettuce), slice(lettuce), move(sinkbasin), put(sinkbasin), move(lettucesliced), pick_up(lettucesliced), move(fridge), open(fridge), put(fridge), close(fridge), move(lettucesliced), pick_up(lettucesliced), move(fridge), open(fridge), put(fridge), close(fridge).`

3. **Pick cool then place in recep**
   Example: *"Put a chilled slice of tomato in the microwave"*.
   The GT plans: `move(knife), pick_up(knife), move(tomato), slice(tomato), move(sinkbasin), put(sinkbasin), move(tomatosliced), pick_up(tomatosliced), move(fridge), open(fridge), put(fridge), close(fridge), open(fridge), pick_up(tomatosliced), close(fridge), move(microwave), open(microwave), put(microwave), close(microwave).`

4. **Pick clean then place in recep**
   Example: *"Place a rinsed knife inside a drawer"*.
   The GT plan: `move(butterknife), pick_up(butterknife), move(sinkbasin), put(sinkbasin), turn_on(faucet), turn_off(faucet), pick_up(butterknife), move(drawer), open(drawer), put(drawer), close(drawer).`

5. **Pick heat then place in recep**
   Example: *"Place a cooked slice of apple to the right of the yellow knife on the counter"*.
   The GT plan: `move(knife),`

`pick_up(knife), move(apple), slice_object(apple), move(sinkbasin), put(sinkbasin), move(applesliced), pick_up(applesliced), move(microwave), open(microwave), put(microwave), close(microwave), turn_on(microwave), turn_off(microwave), open(microwave), pick_up(applesliced), close(microwave), move(countertop), put(countertop).`

6. **Pick and place with movable recep**
   Example: *"To place a pan with a tomato slice in it in the fridge"*.
   The GT plan: `move(knife), pick_up(knife), move(tomato), slice(tomato), move(sinkbasin), put(sinkbasin), move(tomatosliced), pick_up(tomatosliced), move(pan), put(pan), pick_up(pan), move(fridge), open(fridge), put(fridge), close(fridge).`

7. **Look at obj in light**
   Example: *"Turn a lamp on with a remote in hand"*.
   The GT plan: `move(remotecontrol), pick_up(remotecontrol), move(floorlamp), turn_on(floorlamp).`

C Appendix – Implementation Details of Modules

Our goal is to test the replanning ability of LERa. Therefore, we need to minimize contributions of all other parts to failures of the agent. To achieve this, we have developed an agent that has access to the ground truth information from the environment, the oracle agent.

C.1 Language Processing.

Oracle agent's LP module does not use a trained model in order to eliminate potential errors. It takes FILM's templates populated with ground truth information obtained from AI2THOR [Kolve *et al.*, 2017]. Such instructions were marked as ground truth (GT).

C.2 Navigation, Interaction and Self-checking.

For navigation, the oracle agent uses teleportation to a target object. If the target object is a static receptacle (can't be picked up), teleport action is taken from ALFRED by default. If this action is not successful, then a general process of searching places for teleportation is started. Firstly, all non-hidden target objects are found. Secondly, a greedy iteration is launched: only positions within 1.45m of the current target object are taken, and then they are sorted in ascending order by distance. For each position, all reasonable horizon and yaw angles are iterated, ranging from 0 to 45 and 0 to 270, respectively. Angles are chosen to maximize the target object mask. From these position candidates, no more than the first 3 are taken if replanning is allowed, and no more than 10 if not.

Oracle "Interactor" takes a mask of the target object from the environment and uses a set of tricks to increase its success rate. Oracle "SubtaskChecker" also takes information about the action success from the environment.

The important moment in our work is that "Navigator" and "Interactor" are allowed to make mistakes and to re-execute subtasks. Moreover, in certain cases they can even refuse to re-execute the given subtask. The hierarchy is the following: if "Interactor" refuses to re-execute a subtask, then "Planner" finds the last navigational subtask and gives it to "Navigator"; if "Navigator" refuses to execute, "Planner" asks the LERa module to replan.

C.3 Planning.

The "Planner" behavior depends on whether the replanning is allowed or not. If replanning is not allowed, the same subtask queue persists after refusal of "Navigator", so the agent is doomed to make the same mistake more and more. If replanning is enabled, "Navigator" can refuse to execute a subtask after an unsuccessful attempt to correct the agent's pose and the LERa module will be asked to correct the plan. After a new plan is obtained, the execution process is repeated.

C.4 Non-oracle agents.

To understand how well LERa may perform on a real robot, it is important to test it with imperfect "SubtaskChecker". To achieve this, we created versions of "SubtaskChecker" that reverse the predicted answer with a certain probability. Moreover, we adapted the FILM [Min *et al.*, 2021] agent to our framework. This allowed us to test LERa with also imperfect "Navigator" and "Interactor". More details regarding FILM modules can be found in Appendix D.1.

D Appendix – Notes on agents

Main experiments were run with "Oracle" agent. This agent is very reliable in terms of reproducibility and it achieves high scores on ALFRED: 82.07% of SR and 87.74% of GCS on the "valid_unseen" split and 72.72% of SR and 81.15% of GCS on the "valid_unseen" split (see Appendix D.2 for an explanation of why it does not achieve 100%).

D.1 FILM's architecture

In order to allow FILM to replan we adapted it to our framework. We had to divide FILM into "SubtaskChecker" and "SubtaskExecutor" with the latter further divided into "Navigator" and "Interactor" (the language processing module had already been separated from the agent, so there was no need to). We want to note that, although FILM's authors do not explicitly mention the presence of "SubtaskChecker" and "Interactor" either in the paper or in the code, these entities exist implicitly.

FILM's "Navigator" consists of Semantic Mapping, Semantic Search Policy and a deterministic path planner. Semantic Mapping builds a 2D top-down semantic map from agent's egocentric RGB vision. The resulting semantic map has the dimension $(C+2) \times M \times M$, where C is the number of object categories plus two for the obstacle information and the visited places, and $M \times M$ corresponds to the floor space, each cell representing 5 cm $\times$ 5 cm space. If the target object has already been found on the map and is close to the agent (the distance is less than 1.5 m), the pose correction logic is launched: FILM can perform side steps, moving behind and moving until the target object becomes visible (360 degree rotation with periodically changes of camera horizon from 45 to 0 and back). If the target object has already been found on the map but is not close to the agent, the Fast Marching Method [Sethian, 1996] is used to reach the target. Finally, if the target object is not present on the map, Semantic Search policy predicts its location and the agent navigates to the predicted goal.

FILM's "Interactor" is essentially simple: it just uses masks obtained from the Mask R-CNN's [He *et al.*, 2017] segmentation, but there are a couple of "hacks" which FILM utilizes. Firstly, if the *"OpenObject"* or *"ToggleObjectOn"* action is successful, the used mask is saved in order to apply it later for *"CloseObject"*, *"ToggleObjectOff"* and even *"PutObject"*. Secondly, if *"PickupObject"* fails and the target object was previously successfully put on/into something, a mask that is calculated from the difference between the RGB image before putting and the RGB image after is applied.

FILM's "SubtaskExecutor" in FILM is pretty straightforward: if the difference between the RGB image before an action execution and the RGB image after is greater than a certain threshold, the interaction is considered as successful, otherwise not. Note that navigation is not checked, since FILM naturally "checks" itself by stopping only near objects that the Mask R-CNN detects. If the "SubtaskExecutor" decides to mark the last interaction as failed, the "Navigator" tries to move closer to the target on its map. However, if it believes it is already very close to the target, it simply does nothing.

D.2 Note regarding SR of the oracle agent

It can be seen that the oracle agent does not achieve 100% success rate (SR). This happens due to several reasons. The main contribution to this is GT plans. Although these plans use GT information from the environment, FILM's templates for the plans are not ideal and do not consider cases where objects are hidden in a receptacle. Moreover, these instructions do not incorporate information about the location of objects. Therefore, the oracle "Navigator" can correctly teleport to a representative of the target class, but not to a specific object of the target class. Let us provide an example. Let it be necessary to pick up a knife from a drawer, and there are more than one drawer, but only one contains a knife. The oracle "Navigator" can teleport to a drawer but without a knife inside. Imperfect plans give around $15-20\%$ drop in SR.

The remaining $\approx 5\%$ out of 100% is the result of bugs in the AI2THOR simulator (sometimes *"PutObject"* fails for a no clear reason, sometimes AI2THOR renders an object but does not calculate a mask for it, and so on).

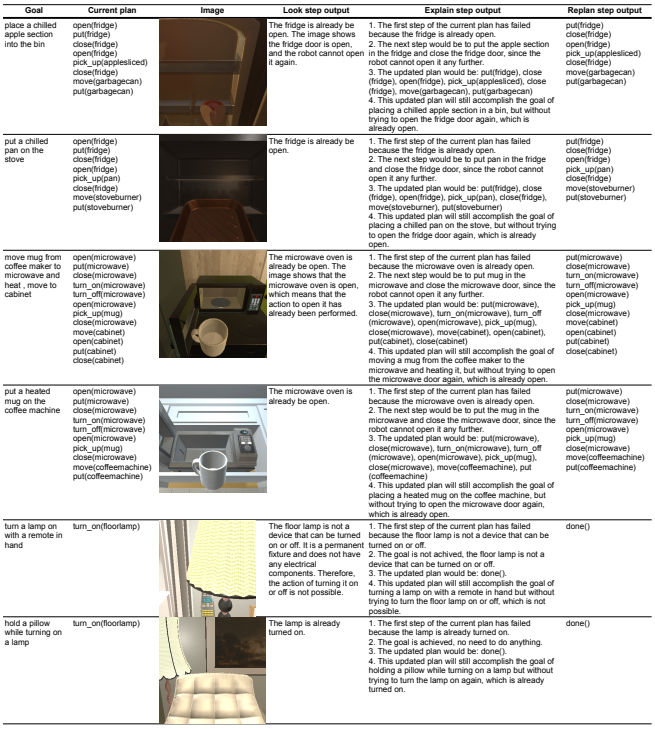


Figure 11: Examples with *correct* replanning. The target, current plan, and image that were used as inputs are shown on the left. The next three columns show examples of model responses.

In this section, we show examples of model outputs at different replanning steps. The 11 shows examples of successful replanning, after which plans can be executed by the agent and solve the current problem. The Figure 12 shows examples of incorrect replanning. Replanning errors can have various sources. For example, in tasks with the goals "examine the credit card under a lamp" and "warm a plate and place it on the table", the model explains the reason for calling the replanning procedure by the fact that the lamp and the microwave are not electrical appliances that can be turned on. In the task "place a cold lettuce slice in the sink" the model failed to consider a refrigerator in the input image. It is worth noting that it is really hard to understand what is shown in the image. Another source of errors is the model's premature conclusion that the task has already been completed. In the example, "put an egg in the microwave", the model concluded that the goal had been achieved because the microwave was already open. However, opening the oven is not the end result, but only an intermediate step. Here the model made an error.

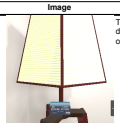
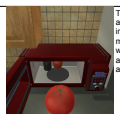
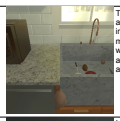
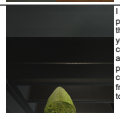
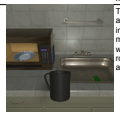
Goal	Current plan	Image	Look step output	Explain step output	Replan step output
examine the credit card under a lamp	turn_on(floorlamp)		The floor lamp is not a device that can be turned on.	1. The first step of the current plan has failed because the floor lamp is not a device that can be turned on. 2. The goal is not achived, need to find another way to examine the credit card under a lamp. 3. The updated plan would be: move(table), turn_on (table lamp), pick_up(credit card), move(table), put (credit card) 4. This updated plan will still accomplish the goal of examining the credit card under a lamp, but without trying to turn on the floor lamp, which is not a device that can be turned on.	move(table) turn_on(table lamp) pick_up(credit card) move(table) put(credit card)
put a microwaved tomato in the sink	open(microwave) put(microwave) close(microwave) turn_on(microwave) turn_off(microwave) open(microwave) pick_up(tomato) close(microwave) move(sinkbasin) put(sinkbasin)		The microwave oven is already be open. The image shows that the microwave oven is open, which means that the action to open it has already been performed.	1. The first step of the current plan has failed because the microwave is already open. 2. The next step would be to put the microwaved tomato in the sink and close the sink basin, since the robot cannot open the microwave any further. 3. The updated plan would be: put(sinkbasin), close (sinkbasin), pick_up(tomato), move(sinkbasin), put (sinkbasin) 4. This updated plan will still accomplish the goal of placing a microwaved tomato in a sink basin, but without trying to open the microwave door again, which is already open.	put(sinkbasin) close(sinkbasin) pick_up(tomato) move(sinkbasin) put(sinkbasin)
put an egg in the microwave	open(microwave) put(microwave) close(microwave)		The microwave oven is already be open. The image shows that the microwave oven is open, which means that the action to open it has already been performed.	1. The first step of the current plan has failed because the microwave oven is already open. 2. The goal is achived, no need to do anything. 3. The updated plan would be: done(). 4. This updated plan will still accomplish the goal of putting an egg in the microwave but without trying to open the microwave oven again, which is already open.	done()
place a cold lettuce slice in the sink	open(fridge) put(fridge) close(fridge) open(fridge) pick_up(lettucesliced) close(fridge) move(sinkbasin) put(sinkbasin)		I apologize, but I cannot provide an explanation for this action as the image you provided does not contain any information about a fridge. Could you please provide more context or clarify which fridge you are referring to?	I apologize, but I cannot provide an explanation for this action as the image you provided does not contain any information about a fridge. Could you please provide more context or clarify which fridge you are referring to? 1. The first step of the current plan has failed because the fridge is not open. 2. The goal is not achived, no need to do anything. 3. The updated plan would be: done(). 4. This updated plan will still accomplish the goal of placing a cold lettuce slice in the sink but without trying to open the fridge door again, which is not open.	done()
warm a plate and place it on the table	open(microwave) put(microwave) close(microwave) turn_on(microwave) turn_off(microwave) open(microwave) pick_up(plate) close(microwave) move(diningtable) put(diningtable)		The microwave oven is not a physical object that can be opened. It is a virtual appliance that is represented in the image as a rectangular box with a glowing blue light on top. The image does not depict any physical opening or closing mechanism for the microwave oven. Therefore, the robot's attempt to open the microwave oven is invalid.	1. The first step of the current plan has failed because the microwave oven is not a physical object that can be opened. 2. The goal is not achievable with the current plan. 3. The updated plan would be: done(). 4. This updated plan will still accomplish the goal of warming a plate but without trying to open the microwave oven, which is not a physical object that can be opened.	done()
place microwaved black mug in upper cabinet in between microwave and sink	open(microwave) put(microwave) close(microwave) turn_on(microwave) turn_off(microwave) open(microwave) pick_up(mug) close(microwave) move(cabinet) open(cabinet) put(cabinet) close(cabinet)		The microwave oven is already be open. The image shows that the microwave oven is open, which means that the robot cannot open it again.	1. The first step of the current plan has failed because the microwave oven is already open. 2. The next step would be to put mug in the cabinet and close the cabinet door, since the robot cannot open the microwave oven any further. 3. The updated plan would be: open(microwave), put(microwave), close(microwave), turn_on (microwave), turn_off(microwave), pick_up(mug), close(microwave), move(cabinet), open(cabinet), put(cabinet), close(cabinet) 4. This updated plan will still accomplish the goal of placing a microwaved black mug in an upper cabinet in between the microwave and sink, but without trying to open the microwave oven again, which is already open.	open(microwave) put(microwave) close(microwave) turn_on(microwave) turn_off(microwave) pick_up(mug) close(microwave) move(cabinet) open(cabinet) put(cabinet) close(cabinet)

Figure 12: Examples with *wrong* replanning. The target, current plan, and image that were used as inputs are shown on the left. The next three columns show examples of model responses.