Planning with Large Language Models via Corrective Re-prompting

Shreyas Sundara Raman[*1], **Vanya Cohen**[2], **Eric Rosen**[1], **Ifrah Idrees**[1], **David Paulius**[1] **and Stefanie Tellex**[1]
[1]Brown University
[2]The University of Texas at Austin

Abstract

Extracting the common sense knowledge present in Large Language Models (LLMs) offers a path to designing intelligent, embodied agents. Related works have queried LLMs with a wide-range of contextual information, such as goals, sensor observations and scene descriptions, to generate high-level action plans for specific tasks; however these approaches often involve human intervention or additional machinery to enable sensor-motor interactions. In this work, we propose a prompting-based strategy for extracting executable plans from an LLM, which leverages a novel and readily-accessible source of information: *precondition errors*. Our approach assumes that actions are only afforded execution in certain contexts, i.e., implicit preconditions must be met for an action to execute (e.g., a door must be unlocked to open it), and that the embodied agent has the ability to determine if the action is/is not executable in the current context (e.g., detect if a precondition error is present). When an agent is unable to execute an action, our approach re-prompts the LLM with precondition error information to extract an executable *corrective action* to achieve the intended goal in the current context. We evaluate our approach in the VirtualHome simulation environment on 88 different tasks and 7 scenes. We evaluate different prompt templates and compare to methods that naively re-sample actions from the LLM. Our approach, using precondition errors, improves executability and semantic correctness of plans, while also reducing the number of re-prompts required when querying actions.

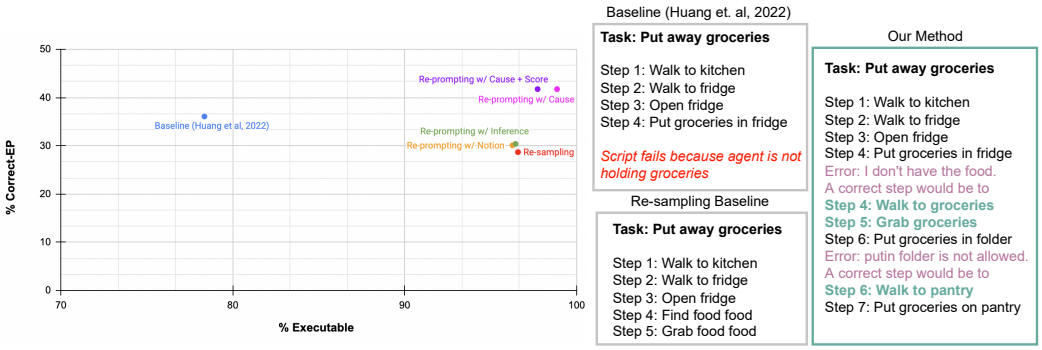


Figure 1: The percentage of executable plans v.s. semantic correctness, conditioned upon executability (**left**). A sample comparing plans generated by Huang et al. (2022) and re-sampling baselines with our method (**right)**

*Corresponding Author (Email: shreyas_sundara_raman@brown.edu)

Foundation Models for Decision Making Workshop at Neural Information Processing Systems, 2022

1 Introduction

Large Language Models (LLMs) excel at performing diverse natural language processing (NLP) tasks including translation, sentiment analysis, and arithmetic (Brown et al., 2020). LLMs learn a generative model of large web-sourced text datasets. This large scale pre-training encodes latent common-sense knowledge that can be utilized in downstream tasks (Bosselut et al., 2019). Our objective is to use LLMs to aid with intelligent decision-making for embodied agents that are performing long-horizon tasks (e.g., cleaning a room, prepping a meal, etc.).

Embodied agents exist in an environment with the purpose of accomplishing a goal. Extracting actionable knowledge from LLMs requires incorporating relevant context about the situation or embodiment of the agent. For example, related works have investigated zero-shot approaches to extract plans from LLMs using prompting strategies that include natural language descriptions of the agent's goal to generate action sequences (Huang et al., 2022). To provide context about the scene, other work has used visual-language models to determine what natural language actions are feasible based on sensor data (Ahn et al., 2022) or processed scene descriptions (e.g., common-sense reasoning for embodied agents in text-based video games (Yao et al., 2020; Singh et al., 2021)). In this work, we investigate a previously unexplored source of contextual information, *precondition errors* (which we subsequently define more concretely), to query LLMs with to extract action plans.

Preconditions come from the notion that actions are not executable in all circumstances, and that certain conditions must first be satisfied. In certain scenarios, these conditions are factorized and explicit to the agent (e.g., to follow a cooking recipe, the agent must access the requisite ingredients), while in other scenarios, there may be no easy or accessible way to explicitly decompose the preconditions, and so the agent only has the ability to determine whether an action is currently afforded or not (e.g., observing a cluttered scene and determining if a cup can currently be picked up or not). In the latter case, the agent only has the ability to report a precondition error stating that an action is currently infeasible, whereas in the former case the precondition error can contain more information about why that action is currently not feasible. To our knowledge, leveraging precondition errors (with or without explanatory reasons) to query LLMs for actions to accomplish a task has not previously been explored, and we suggest that it is a promising source of auxiliary information.

In this work, we propose a prompt-based strategy for querying plans from LLMs using precondition errors. Our approach takes a natural language description of a task and uses a LLM to produce a sequence of actions for the robot to execute. When the robot fails to execute a skill in the plan and reports a precondition error, we use a template-based prompt strategy to query new actions from the LLM to produce new plans for the embodied agent. In cases where the agent only has the ability to evaluate whether an action is afforded in the current situation, our approach uses a prompt format that incorporates a description of the failed action; in cases where the agent has structured information about the precondition error, our approach uses a prompt includes that explanatory information about why the action is currently not feasible. Therefore, our prompt-based strategy is adaptable and can be integrated into a wide-variety of approaches depending on what structured information and assumptions are made about the agent's knowledge regarding evaluating action affordance.

We compare our prompt-based approach that leverages precondition errors to a baseline strategy of naively re-sampling the LLM for new actions upon encountering an error, and also conduct ablations on different prompt-templates to evaluate the best way to leverage precondition errors to extract corrective actions for faulty plans. We evaluate our approach in the VirtualHome simulation environment (Puig et al., 2018) on a reserved set of 88 tasks, and measure the percentage of executable tasks and the percentage of correct plans produced by these methods as evaluated by human annotators. We show that LLMs are able to utilize error information to improve plan generation. Our re-sampling baseline outperforms current methods ((Huang et al., 2022)) by 16% in executability, on average; our prompt-based approach further outperforms the baseline by 2.27% in executability and 12.22% in human-evaluated correctness, and all re-prompting ablations similarly achieve high executability ($> 90\%$) and correctness ($20 - 21\%$) with variations within 1%.

Our contributions are as follows:

- We propose a method for querying plans from LLMs for embodied agents that leverages precondition errors to generate corrective actions via a re-prompting strategy. Our approach is adaptable to situations where agents are only able to evaluate whether an action is cur-

rently afforded or not, and to more structured settings where explicit knowledge about what preconditions are currently unmet are known.

- We evaluate our method for leveraging precondition errors against a baseline method for querying LLMs for corrective actions without precondition error information in the VirtualHome domain and demonstrate that our approach achieves near-perfect executability with minimal impact on the semantic correctness of plans.
- We compare different prompt templates for querying LLMs that assume different amounts of information regarding precondition error information (ranging from no information besides the skill is not executable to structured information about what precondition is not satisfied). This provides researchers an easy and flexible way to most effectively integrate our approach into their work depending on what information is available to them.

2 Background

2.1 In-Context Learning

Brown et al. (2020) introduces GPT-3, a 175 billion parameter language model capable of few-shot in-context learning of novel tasks. With in-context learning, the LLM is prompted with a sequence of tasks examples, as concatenated input-output pairs. To generate an output for a test example, the example is appended to the prompt. The language model is then queried for a completion of the prompt which forms the model output. This approach stands in contrast to tasks learning based on fine-tuning of pre-trained language representations (Radford et al., 2018; Devlin et al., 2018; Peters et al., 2018), and the zero-shot inference of GPT-2 (Radford et al., 2019). In-context learning offers a number of advantages over other learning paradigms; It enables task specific learning and updating of training datasets without fine-tuning or redeploying the LLM. In-context learning performs best when prompt examples are relevant to the test example, and we follow the method of Liu et al. (2021) and Huang et al. (2022) and retrieve prompt examples using a semantic similarity metric.

2.2 Open-Loop Plan Generation

The method for zero shot open-loop generation proposed by Huang et al. (2022) leverages the rich world knowledge internalized by LLMs to query an action plan to a task without feedback from the environment. Their method utilizes two frozen LLMs named the **Planning LLM** and the **Translation LLM** to auto-regressively generate action steps for a given task and perform in-context learning.

First, a high-level task $\mathcal{T}$ and its associated mid-level plan are chosen (from the *demonstration set*) as a contextual prompt of a free-form plan for the **Planning LLM**; note that $\mathcal{T}$ is chosen such that it maximizes cosine similarity with the *query task* $\mathcal{Q}$, i.e., the target task for which we would like to generate a plan. This step enables in-context learning for the high-level task $\mathcal{T}$ by using the similar query task $\mathcal{Q}$.

The **Translation LLM** (or Pre-trained Masked LLM) then utilizes a BERT-style LM (pre-trained with Sentence-BERT) to embed the generated free-form actions ($\hat{a}$) to an action admissible in the environment (a_e); the cosine similarity between the action's phrase ($\hat{a}$) embedding and the embeddings of each admissible action (a_e) is calculated to determine the admissible action with minimum semantic distance. The chosen admissible action (a_e) is then appended to the unfinished prompt, to condition auto-regressive step generation on admissible actions. A detailed review of this method can be found in Huang et al. (2022). Our approach extends this framework, as we investigate how to improve planning by leveraging precondition errors as an auxiliary information modality.

2.3 Preconditions and Affordances

Gibson originally introduced the notion of an affordance as the set of action possibilities latent to the environment and context the agent is in (Gibson, 1977). Many related works have investigated identifying robotic affordances in the context of robotic grasping (Yamanobe et al., 2017), path planning (Ardón et al., 2020), and generic skill execution (Ahn et al., 2022). While some affordance models only offer the ability to classify or generate states that afford skill execution, more structured models of affordance use a factorized model to decompose the initiation set of a skill into separate components, termed preconditions. Preconditions capture the notion that the affordance of

a skill is compromised of independent components, and can be used to generate cause explanations for why a skill is not currently afforded (i.e., because a certain condition is not satisfied). Learning and modeling preconditions have been largely studied in model-based approaches that leverage symbolic planning (Garrett et al., 2021; Konidaris et al., 2018), but in this work we investigate how precondition information can be leveraged to improve planning with LLMs.

3 Related Work

3.1 Large Language Models for Task Planning

Most closely related to our paper are Ahn et al. (2022) and Huang et al. (2022), which both aim to integrate LLMs into an open-loop planning pipeline for task execution. Huang et al. (2022) used a prompting strategy to derive step-by-step plans that achieve the goal presented in a prompt; however, as mentioned before, our work extends their approach by incorporating feedback from the environment as auxiliary input to improve the executability of a derived plan. Similarly, Ahn et al. (2022) introduced SayCan, an LLM-integrated pipeline that is capable of determining a sequence of actions to achieve specific goals grounded to "affordances" (pre-defined set of skills that the robot can perform, all manually demonstrated by expert). They used semantic similarity to determine the closest skill in the robot's skill library that would achieve the intended goal of an instructions extracted from the language prompt. However, as with Huang et al. (2022), they do not incorporate feedback for failed actions, as their focus is on finding the most affordable action based on a verbal command and the visual state of its environment.

3.2 Task and Motion Planning

Task and motion planning (TAMP) aims to decompose robot planning and execution processes in a hierarchical manner (Kaelbling & Lozano-Pérez, 2010; Garrett et al., 2021). This involves the integration of: 1) *task planning*, which aims to find a sequence of actions that realize state transitions needed to achieve a goal state corresponding to a high-level problem (Ghallab et al., 2016); and 2) motion planning, which aims to find physically consistent and collision-free trajectories that realizes the objectives of a task plan (Lozano-Pérez & Wesley, 1979; Dornhege et al., 2009). Rather than relying on explicitly defined structures or symbols as typically used in TAMP, LLMs can provide an agent or robot with an implicit representation of action and language, which allows it to interpret a task and identify key details (such as objects or actions) that are related to the problem at hand.

3.3 Semantic Parsing and Program Synthesis

The task of mapping natural language instructions to executable formal symbolic representations closely resembles tasks in the semantic parsing and program synthesis literature (Chen et al., 2021). Our method for plan generation resembles semantic parsing as a learned translation task (Wong & Mooney, 2006) and Guu et al. (2017) leverages weak supervision to learn to translate language to executable programs.

3.4 Common-Sense Knowledge in LLMs

Other work explores the degree to which large language models contain common-sense world knowledge. The Winograd Schema Challenge (Levesque et al., 2012) and larger WINOGRANDE benchmark (Sakaguchi et al., 2021) evaluate common-sense reasoning in word problems. The WinoGround dataset investigates common-sense reasoning in a related image caption disambiguation challenge (Thrush et al., 2022). LLMs have improved upon baseline methods for this task (Brown et al., 2020) indicating that language model scale contributes to common-sense reasoning performance. Our system leverages the finding that language models contain latent common-sense world knowledge to incorporate precondition error information to improve plan executability.

4 Method

In this section, we investigate our approaches for injecting corrective information to improve skill execution in embodied agents.

To arrive at an approach for closed-loop zero-shot planning using pre-condition error information, we build on existing methods for zero-shot open-loop planning proposed by Huang et al. (2022). We describe the adaptations to construct an inference-time procedure that corrects skill execution with closed-loop planning; we unravel two variations of our approach between *re-prompting* strategies (based on precondition error information) that encourage quality corrective actions and a *naive re-sampling* baseline for the closed-loop setting. These approaches address some of the shortcomings of open loop planning and offer improved results.

4.1 Plan Generation via Re-prompting

Our investigation initially presented a trade-off between open-loop planners suffering in executability and closed-loop planners integrating sensorimotor experiences being more expensive.

We propose an alternative middle-ground strategy: we use re-prompting to generate corrective actions for non-executable plans by embedding pre-condition error information about the cause of skill execution failure into *corrective prompts*. Previous works have shown that *pretrained* LLMs contain sufficient world knowledge to make goal-driven plans and particular language prompts improve an LLM's ability to outline logical reasoning (Ahn et al., 2022; Kojima et al., 2022), so we see language-based re-prompting as a potentially useful method to integrate environment error information to LLM planners.

In the context of control theory, closed-loop systems incorporate some form of feedback from their outputs for adaptive control (Golnaraghi & Kuo, 2017). In a similar vein, we leverage feedback (as errors) for evaluation and correction in our closed-loop planning setup. Specifically, we add a **Pre-condition Error Prompting Module** that transforms pre-condition errors into *corrective prompts*. We explore various re-prompting templates using different degrees of pre-condition error information and contextual error information – all following the visual model shown in Figure 2.

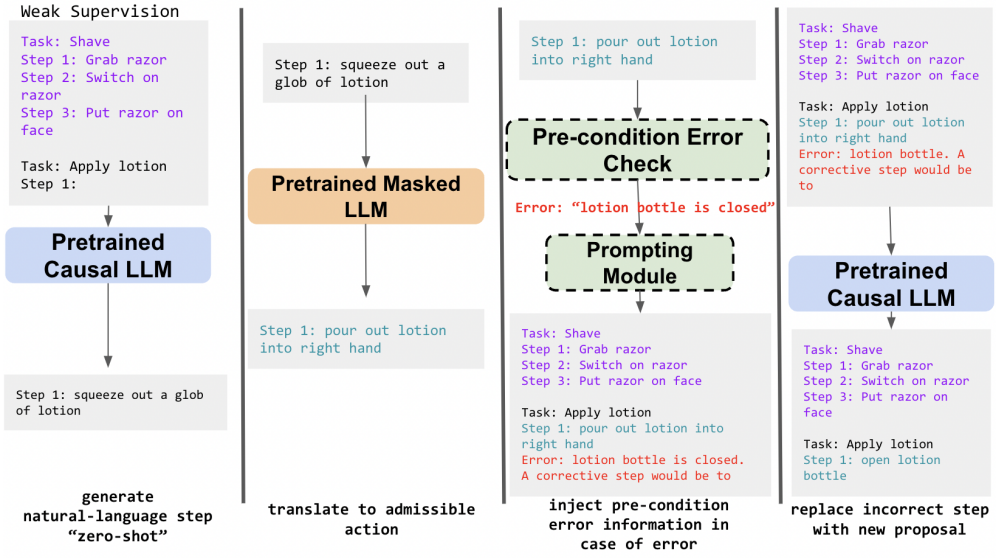


Figure 2: Our pipeline for plan generation with re-prompting. From left to right: (Column 1) A related task (with the highest cosine similarity to the target task) is chosen from the set of training tasks; a frozen autoregressive LLM (e.g., `davinci`, `curie`, `davinci-instruct`) generates a free-form language description of the current step; (Column 2) a frozen masked LLM (such as Roberta-Large) then maps the free-form description to an admissible action in the VH space; (Column 3) the admissible action is checked for violating pre-condition errors and (any potential error information) is crafted into a corrective prompt; (Column 4) the error-step, along with corrective prompt, are used to motivate the LLM to generate an executable corrective action.

Certain errors require more information on the agent's action history or context about the agent's state or environment (e.g., correcting an error such as: `<character> (1) does not have a free hand when executing "[GRAB] <milk> (1) [1]"` might require knowledge of the objects the agent

has previously grabbed or is holding). Hence, we identified different categories of **contextual information** (environment information) and **corrective information** (re-prompting styles) whose cross-product forms a space of re-prompting strategies we explore. We define these categories as follows:

Contextual Information: the contextual/historical information prior to the corrective information. We investigate supplying the query task $\mathcal{Q}$, the query steps and the corrective information (e.g., The output of the prompting module in column 3 of Figure 2).

Pre-condition Error Information: the corrective message used to prompt suggestions for corrective actions, which are constructed as follows:

- Notion of Error: indicates to the agent that a step is not executable e.g. `'Task Failed'`. We call this approach Re-prompting with *Notion*.
- Inference of Error: provides the *object* interacted with and *atomic action* taken within the non-executable step, leaving the agent to *infer* the cause of error and correction e.g., `'I cannot <action> <object>'`. We call this approach Re-prompting with *Inference*.
- Cause of Error: provides the *object* interacted with, *action* taken and the object/action relation (ie the cause of the error) in the non-executable step e.g., `'I cannot <action> <object> because <reason>'`. We call this approach Re-prompting with *Cause*.

Whereas inference of error and notion of error do not require any additional information than the action information provided by the LLM, leveraging cause of error assumes the agent has a model of the preconditions for the action and the ability to explicitly evaluate what preconditions are not currently satisfied. Therefore, deciding which pre-condition error information in the corrective prompt depends on the modeling capabilities of the agent (i.e., re-prompting with inference can be integrated into visual-language model approaches like SayCan (Ahn et al., 2022), whereas re-prompting with inference can be integrated with task and motion planning approaches (Garrett et al., 2021)).

4.2 Scoring Function

When grounding free-form language to actions in the environment, Huang et al. (2022) used the mean log-probability to assess the quality of natural language generated by the LLM and the cosine similarity between embeddings to assess semantic translations.

Log-Probability: The log-probability is defined as $P_\theta(X_i) := \frac{1}{n_i}\sum_{j=1}^{n_i} \log p_\theta(x_{i,j}|x_{i<j})$, where θ parametrizes the pretrained Causal LLM and X_i is a sample generated step consisting of tokens $(x_{i,1}, x_{i,2}, x_{i,3}, ..., x_{i,n})$.

Cosine Similarity Cosine similarity is defined as $C(f(\hat{a}), f(a_e)) := \frac{f(\hat{a})\cdot f(a_e)}{||f(\hat{a})||||f(a_e)||}$, where f is the MaskedLLM embedding function, $\hat{a}$ is the predicted action and a_e is the admissible action for which we estimate the distance with respect to:

$$\mathcal{S}_w = \underset{a_e}{\text{argmax}}\,[\max_{\hat{a}} C(f(\hat{a}), f(a_e)) + \beta \cdot P_\theta(\hat{a})], \text{ where } \beta \text{ is a weighting coefficient.} \quad (1)$$

The original scoring function $\mathcal{S}_w$ (Equation 1) is based on a weighted combination of log-probability and cosine similarity, which is then thresholded to determine feasibility of each proposed grounded step. This scoring function prioritizes the quality of natural language at the cost of semantic translation or vice versa and results in mistranslations, which is prevalent when $C(f(\hat{a}), f(a_e))$ dominates the sum as $P_\theta(\hat{a})$ is close to 0 and β is low or when $P_\theta(\hat{a})$ dominates the sum as $C(f(\hat{a}), f(a_e))$ is close to 0 and β is large. The mean log-probability term is also unbounded, making it challenging to explore an appropriate score threshold. Hence, we propose a novel scoring function $\mathcal{S}_g$ (Equation 2) that considers the squared geometric mean of $C(f(\hat{a}), f(a_e))$ and $P_\theta(\hat{a})$, to produce a bounded non-negative $(0, 1)$ scoring function, which prioritizes both language generation and semantic translation objectives jointly, defined as:

$$\mathcal{S}_g = \underset{a_e}{\text{argmax}}\,[\max_{\hat{a}} \frac{C(f(\hat{a}), f(a_e)) + 1}{2} \cdot e^{P_\theta(\hat{a})}] \quad (2)$$

4.3 Plan Generation via Re-sampling

As a benchmark for closed-loop planning, we also implement a new baseline involving **re-sampling** the LLM. We refer to experiments following this new baseline as `Re-sampling <LM>`, where `<LM>` is replaced by a particular language model (e.g., GPT-3).

For the `Re-sampling <LM>`, when an action in the generated plan is not executable, the re-sampling closed-loop method does not use the error messages to generate corrective prompts. Instead, we iterate through and evaluate the top k admissible actions proposed by the MaskedLLM (in descending order of weighted sum of mean log-probability and cosine similarity – refer to Equation 3 in Huang et al. (2022)) until an executable action is found i.e., we re-sample from the set of k admissible actions. In the case where no executable actions are re-sampled from the top k proposals, plan generation is terminated.

The re-sampling method serves as a performance baseline in the closed-loop domain, in order to assess the net contribution that pre-condition error information and re-prompting templates have on generating more executable/correct action in fewer number of steps. That is, re-sampling aids in assessing whether re-prompting strategies perform well simply because they provide additional attempts for corrective action proposal or whether the pre-condition error information helps direct the corrective action.

5 Experiments

5.1 Environment

As described in Huang et al. (2022), generating plans for open-ended tasks with virtual embodied agents requires a virtual environment supporting diverse agent-environment interactions, a rich action space, and a method to generate grounded actions using LLM outputs. This is especially true in the closed-loop domain, where we leverage the pre-condition error information from the environment to extract descriptive corrections for skill execution. Similar to Huang et al. (2022), we use the VirtualHome environment (Puig et al., 2018) to simulate the execution of household tasks. The environment has 1120 different objects and 22 actions in 7 scenes with intricate preconditions for interactions, albeit within a household setting. The generated plans are captured by *programs* that contain a sequence of textual steps in the form:

`[ACTION] <object_1> (id_1) ... <object_n> (id_n),`

where `[ACTION]` refers to the action name and `<object_i>` and `(id_i)` refer to the i-th object argument name and identifier respectively. Each step contains one of the 22 atomic actions in VirtualHome, a list of objects or arguments defining the interactions, and a list of associated identifiers for each object instance as defined in the scene. When errors occur during program execution, the VirtualHome environment throws descriptive messages containing the step information above along with the cause of the error. An example program and associated error message are shown in Figure 5.1. A detailed description of all error types in VirtualHome can be found in Appendix A.1. We note that our method of incorporating precondition errors via cause of error only requires a natural language description of the precondition error, which is standard in other simulators (like AI2Thor (Kolve et al., 2017)), and therefore our approach is not specific to VirtualHome.

We use the same test set as Huang et al. (2022) consisting of 88 realistic human tasks compiled from the *ActivityPrograms* knowledge base by Puig et al. (2018); the remaining 204 tasks in the knowledge base are used as a *demonstration set*, i.e., to prompt the LLM for plan generation. Our methods leverage the verb-object syntax embedded in environment error messages to generate corrective prompts and ground the actions generated by LLMs.

Due to the variation in the objects and preconditions across scenes, we find that a given query task ($\mathcal{Q}$) might require different re-prompts in different scenes to be executable (i.e., plans must be generated in an online fashion for each scene and query task). Hence, we generate a program for each combination of task and scene, and an average is taken across all 7 programs (corresponding to the 7 scenes provided in VirtualHome) generated for each of the 88 tasks when calculating our evaluation metrics.

```
[WALK] <dishwasher> (1)
[OPEN] <dishwasher> (1)
[GRAB] <bowl> (1)
[WALK] <sink> (1)
[PUTBACK] <bowl> (1) <sink> (1)
[GRAB] <plate> (1)
[WALK] <dining room> (1)
[PUTBACK] <plate> (1) <table> (1)

Script is not executable, since <character> (1) is not close to <table> (1) when
                executing "[PUTBACK] <plate> (1) <table> (1) [1]"
```

Figure 3: An example of a program in VirtualHome for the task "Empty dishwasher and fill dishwasher", and its associated error message due to the last step not currently being executable.

5.2 Evaluation Metrics

A plan that follows VirtualHome's program syntax would be largely executable in the environment, but may not be semantically correct nor transferable to other environments. Likewise, a plan that is not grounded in the environment might generate outputs with correct natural language but actions may not be executable in VirtualHome since free-form language has ambiguous formats and ignores pre-conditions (e.g., a door needs to be open before the agent can walk through it). Hence we need to consider both **executability** and **correctness** in our evaluation. We lift both of these metrics from Huang et al. (2022).

Incorporating re-prompts and re-samples would add multiple corrective steps to plans until they become executable; thus our methods would tend to increase the length and complexity of generated plans. Hence, we also track the **number of corrections** required to produce an executable and correct plan in order to compare the *effectiveness* of different re-prompting/re-sampling strategies.

Executability measures if the generated actions can be *correctly parsed* and if they *satisfy the pre-conditions and post-conditions* imposed by the environment; satisfying these conditions makes the program "executable" in VirtualHome. To be correctly parsed, each action must be syntactically correct and only contain actions/objects admissible by VirtualHome. To satisfy pre-conditions and post-conditions, each action must adhere to the conditions (of the objects in the interaction) defined by each scene graph in VirtualHome (e.g., the state of the fridge changes from "closed" to "open" after the agent opens it).

Correctness is a human annotated metric assesses whether the given steps could hypothetically accomplish the given task. Assessing the completion of tasks using natural language based skill-execution is difficult, particularly in a virtual environment; thus, we conduct human evaluations for all generated plans using 4 experimenters, asking participants to assess if the plan is "correct" or "incorrect" (a *boolean* metric). Given execution in the VirtualHome environment, only executable plans can be "correct." However the annotation task ignores this requirement and asks annotators to assess correctness independent of any specific environment or execution constraints. This metric attempts to determine the relevance of the plan to the task.

Correctness-Executable-Plans measures semantic correctness (as described above) but conditioned on plans being executable in the VirtualHome environment i.e., the plan up to the step where it remains executable.

The **Longest Common Sub-sequence (LCS)** between the generated program and the ground-truth program for a given task is also computed as a proxy for correctness, as measured by Puig et al. (2018). LCS only serves as a proxy since human-evaluated correctness is a more robust measure of plan semantics since it not hindered by the interactive richness of the environment and variability in approaches to complete a task that make it difficult to compare generated plans to a "golden standard" i.e., some fixed ground-truth program.

Scene Graph Similarity (GS) measures the degree of similarity between the scene graphs produced by executing the generated program ($\mathcal{G}_{gen}$) and the ground-truth human-written program ($\mathcal{G}_{gt}$) in VirtualHome. This is a normalized metric as it compares the final state of a scene graph with a fixed

size (i.e., over the union of all objects in $\mathcal{G}_{gen}$ and $\mathcal{G}_{gt}$), as opposed to optimizing a "template fit" to an arbitrary ground-truth plan like LCS does.

Steps and **Number of Corrections** measures the total number of steps taken in the environment and the number of corrective re-prompts or re-samples that are performed, upon encountering a precondition error, until an executable action is sampled/generated. While these metrics are incidental to the goal (i.e., minimizing the metrics does not necessarily imply improved performance), they assess the effective of the information in each contextual-prompt towards correcting skill execution.

Fleiss' Kappa measures inter-rater agreement between multiple raters on a categorical labeling task. The value ranges from 0 to 1. We report Fleiss' Kappa for the *% Correct-EP* human annotations.

Method	% Executable↑	% Correct↑	% Correct-EP↑	% GS↑
(Huang et al., 2022)	78.35(1.60)	**46.12**	36.08	99.67
Re-Sampling	96.59	28.69	28.69	99.51
Re-Prompting *w/ Notion*	96.27	30.11	30.11	99.46
Re-Prompting *w/ Inference*	96.45	30.40	30.40	99.48
Re-Prompting *w/ Cause*	**98.86**	41.76	**41.76**	**99.83**
Re-Prompting *w/ Cause* + $\mathcal{S}_g$	97.73	41.76	**41.76**	99.67

Table 1: The mean percentage of executable programs, correct programs, correct programs conditioned on them being executable, and scene graph similarity. Means and standard deviations are reported across 3 trials for the baseline method of Huang et al. (2022).

Method	LCS↑	Fleiss' Kappa↑	Steps	No. Corrections
(Huang et al., 2022)	**24.72 (0.28)**	0.5934(0.05)	6.70(0.23)	N/A
Re-Sampling	22.48	0.5510(0.05)	5.79	4.08
Re-Prompting *w/ Notion*	20.77	0.6760(0.04)	5.22	0.50
Re-Prompting *w/ Inference*	20.77	0.5658(0.05)	5.22	0.53
Re-Prompting *w/ Cause*	21.53	0.6485(0.04)	4.11	0.29
Re-Prompting *w/ Cause* + $\mathcal{S}_g$	20.21	**0.9416(0.05)**	6.11	0.90

Table 2: The mean Longest Common Sub-sequence (LCS), Fleiss' Kappa, number of steps, and number of corrections. Means and standard deviations are reported across 3 trials for the baseline method of Huang et al. (2022). Fleiss' Kappa is computed across four annotators. The proposed scoring function $\mathcal{S}_g$ (Section 4.2) was only applied to the best-performing re-prompting approach.

5.3 Evaluated Approaches

Brown et al. (2020) has shown that novel Codex models (with thresholded outputs) outperform `GPT-Neo` and `GPT-J` with $> 2\times$ strict accuracy on code generation benchmarks. Similarly, Douglas Summers-Stay & Voss (2021) show that the `davinci-instruct` line outperforms its non-instruct counterpart across all prompt variations on the McCarthy logical reasoning and question-answer benchmark. We also performed ablations on smaller models (such as `babbage` or `ada`) but only chose to report the performance with the best-performing model. Thus, we choose to focus on larger state-of-the-art LLMs (`davinci-instruct`) for its demonstrated capabilities. We evaluate 6 different approaches: Huang et al. (2022) (Section 2.2), Re-sampling (Section 4.3), Re-prompting *w/ Notion*, Re-prompting *w/ Inference*, Re-prompting *w/ Cause* (Section 4.1) and Re-prompting *w/ Cause* + $\mathcal{S}_g$ (Section 4.2) using this model.

6 Results

The results of our experiments for all the evaluated approaches can be found in Tables 1 and 2. Re-Prompting *w/ Cause* performs the best in terms of percent executable (98.86%), whereas the original Huang et al. (2022) approach performs best in terms of percent of programs that are correct (46.12%). However, whilst Huang et al. (2022) performs the worst accounting for the percent of programs that are correct conditioned on them being executable, Re-Prompting *w/ Cause* performs the

best (41.76%). Therefore, re-prompting with causes demonstrably improves the model's capability to produce more executable plans that are correct. In addition, while Re-Prompting *w/ Inference* and Re-prompting *w/ Notion* have a slightly lower percentage of executable programs (96.45% and 96.27%) compared to the re-sampling baselines (96.59%), they both have a higher generated percent of correct programs and a higher percent of correct programs conditioned on them being executable (30.40% and 30.11% vs. 28.69%). Therefore, while supplying detailed precondition errors to the LLM helps the most in terms of improving executability and correctness (as seen with Re-prompting *w Cause*), re-prompting while asking for the inference of the error or simply providing a notion of error gives improvements over resampling. We do note however that Huang et al. (2022) does perform the best in terms of LCS (24.72%), but this is only a proxy for correctness. In addition, Huang et al. (2022) and the re-sampling strategy generate the longer plans (6.7 and 5.79 respectively), and re-sampling requires sampling many more corrective actions (4.08) as compared to re-prompting *w/ Notion*, *w/ Inference*, and *w/ Cause* (0.50, 0.53 and 0.29 respectively). Incorporating the scoring function slightly reduces executability (98.86% to 97.73%) and maintains semantic correctness (at 41.76%) with respect to Re-prompting *w/ Cause*, though the inter-annotator agreement (from 0.6485 to 0.9416) increases substantially. This implies generated plans are correct with much greater consistency when using the new scoring function. These results emphasize the usefulness of constructing contextualized query prompts that leverage precondition information as an additional modality for improving embodied decision-making with LLMs. A more detailed analysis on the distribution of errors encountered by the models can be found in Appendix A.3.

7 Conclusion

Our work investigates re-prompting strategies to improve the executability and correctness of LLM-generated plans. Our approach supplies contextual information in the form of precondition errors to improve the generation of plans. Our results demonstrate that LLMs that can leverage precondition errors with error-cause-information to produce more semantically correct and executable plans. We also show that leveraging affordance information (i.e., determining that an action is not currently feasible to executable but without explaining why) for prompt-based strategies is more effective than naively re-sampling actions from the LLM. Finally, our scoring function is shown to improve the consistency of correct plans with higher inter-rater agreements

8 Future Work

We propose 3 areas for future work to explore: providing more contextual information about the scene and agent's skill repertoire within the LLM prompt, thresholding sampled actions based on their executability in the environment and removing assumptions on the availability of strictly templated prompts and a related example for in-context learning.

Acknowledgments and Disclosure of Funding

Part of this research was conducted using computational resources and services at the Center for Computation and Visualization, Brown University.

The authors would also like to acknowledge Wenlong Huang for providing us with access to the codebase from their previous work (Huang et al., 2022) for baseline comparison.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do As I Can and Not As I Say: Grounding Language in Robotic Affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

Paola Ardón, Èric Pairet, Katrin S Lohan, Subramanian Ramamoorthy, and Ronald Petrick. Affordances in robotic tasks–a survey. *arXiv preprint arXiv:2004.07400*, 2020.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. COMET: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4762–4779, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1470. URL https://aclanthology.org/P19-1470.

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, et al. Evaluating Large Language Models Trained on Code. 2021. doi: 10.48550/ARXIV.2107.03374. URL https://arxiv.org/abs/2107.03374.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *2009 IEEE International Workshop on Safety, Security & Rescue Robotics (SSRR 2009)*, pp. 1–6. IEEE, 2009.

Claire Bonial Douglas Summers-Stay and Clare Voss. What Can a Generative Language Model Answer About a Passage? 2021. URL https://aclanthology.org/2021.mrqa-1.7.pdf.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293, 2021.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. doi: 10.1017/CBO9781139583923.

James J Gibson. The theory of affordances. *Hilldale, USA*, 1(2):67–82, 1977.

Farid Golnaraghi and Benjamin C Kuo. *Automatic Control Systems*. McGraw-Hill Education, 2017.

Kelvin Guu, Panupong Pasupat, Evan Zheran Liu, and Percy Liang. From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood. 2017. doi: 10.48550/ARXIV.1704.07926. URL https://arxiv.org/abs/1704.07926.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. *arXiv preprint arXiv:2201.07207*, 2022.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large Language Models are Zero-Shot Reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

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 preprint arXiv:1712.05474*, 2017.

George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What Makes Good In-Context Examples for GPT-3? *arXiv preprint arXiv:2101.06804*, 2021.

Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. VirtualHome: Simulating Household Activities via Programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8494–8502, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Ishika Singh, Gargi Singh, and Ashutosh Modi. Pre-trained Language Models as Prior Knowledge for Playing Text-based Games, 2021.

Tristan Thrush, Ryan Jiang, Max Bartolo, Amanpreet Singh, Adina Williams, Douwe Kiela, and Candace Ross. Winoground: Probing vision and language models for visio-linguistic compositionality. 2022. doi: 10.48550/ARXIV.2204.03162. URL `https://arxiv.org/abs/2204.03162`.

Yuk Wah Wong and Raymond Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pp. 439–446, 2006.

Natsuki Yamanobe, Weiwei Wan, Ixchel G Ramirez-Alpizar, Damien Petit, Tokuo Tsuji, Shuichi Akizuki, Manabu Hashimoto, Kazuyuki Nagata, and Kensuke Harada. A brief review of affordance in robotic manipulation research. *Advanced Robotics*, 31(19-20):1086–1101, 2017.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep CALM and Explore: Language Models for Action Generation in Text-based Games. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

A Appendix

A.1 Error Types in VirtualHome

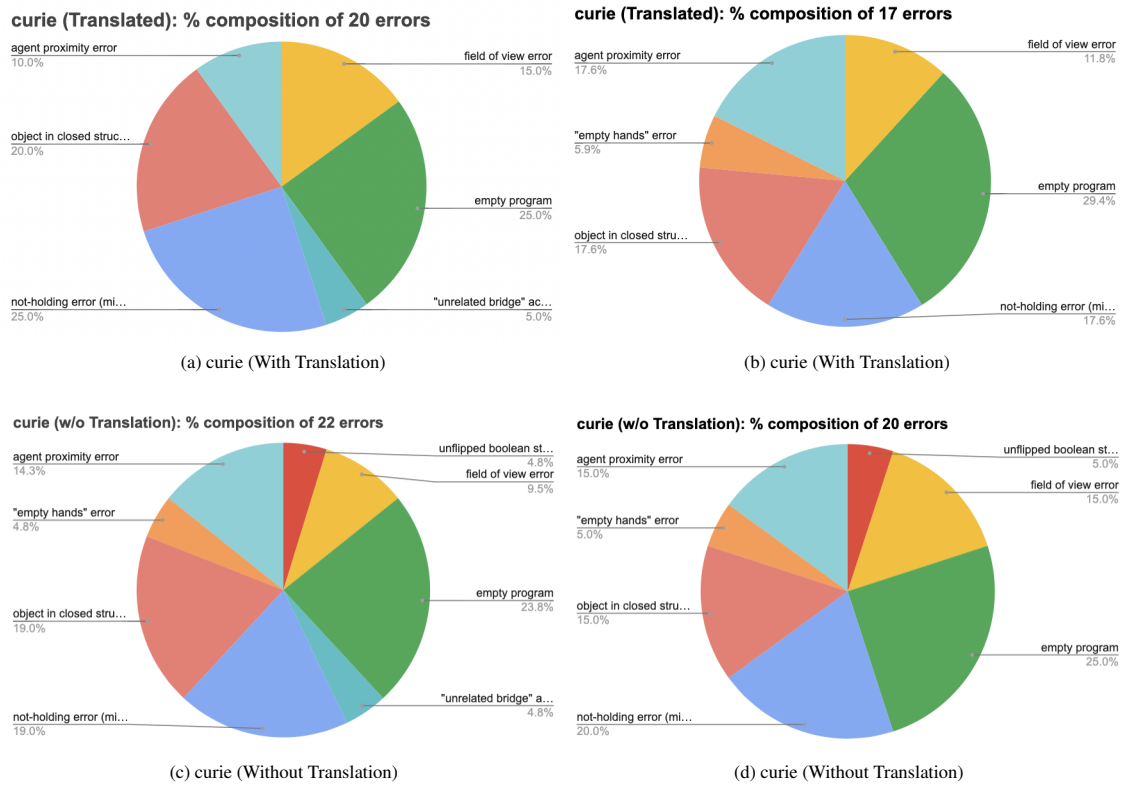
Error ID	Error Type	Error Template (from VirtualHome)	Explanation
1	Unflipped Boolean state	`<object> (ID) is not {object state e.g., open, on, closed, off} when executing "[action] <object> (ID) [1]"`	The action applied to an object with a Boolean state does not change its state (e.g., action `open` is applied to an already opened door).
2	Field of view	`<character> (ID) does not face <object> (ID) when executing "[action] <object> (ID) [1]"`	The agent does not face an object when performing an action on the object (e.g., one needs to `[TURNTO]` a television in order to `[WATCH]` television).
3	Empty program	`empty program`	The LLM has not been able to generate a plan – containing at least one step – for the given task.
4	Absent from room	`char room <character_room> (ID) is not node room <target_room> (ID) when executing "[action] <object> (ID) [1]"`	The target object does not exist in any room within the specific virtual scene.
5	Missing object	`<character> (ID) is not holding <object> (ID) when executing "[action] <object> (ID) [1]"`	The agent is not holding the target object in the current step, which prevents it from completing the action.
6	Enclosed object	`<object> (ID) is inside other closed thing when executing "[action] <object> (ID) [1]"`	The target object is contained within a closed structure, and the action does not liberate the object for use.
7	Invalid action	`<object> (ID) does not have {property} when executing "[action] <object> (ID) [1]"`	The agent is attempting to execute an action on a target object that is not afforded to it (e.g., the agent cannot execute `[PULL]` on the ceiling).
8	Over-occupied agent	`<character> (ID) does not have a free hand when executing "[action] <object> (ID) [1]"`	The agent's hands are occupied or are already holding/interacting with objects, leaving no room to interact with the target object in the current step.
9	Agent proximity	`<character> (ID) is not close to <object> (ID) when executing "[action] <object> (ID) [1]"`	The agent is not close enough to the target object, which prevents it from completing the action.
10	Other precondition	-	The agent attempts to execute an action whose pre-conditions are not satisfied.

Table 3: Description of error types observed in the VirtualHome environment.

To arrive at an effective method for prompt engineering and re-sampling, we looked for patterns in executability errors generated by VirtualHome. This could be incorporated into any environment with an embodied agent that can explicitly identify what preconditions are currently unmet.

Qualitative analysis show us these errors neatly organize into 10 broad *error types*, each with a uniquely structured error message and cause for the error. These error types are largely structured around the types of pre-condition failures in the environment as listed in Table 3.

A.2 Pre-condition Error Distributions using different seeds



(a) curie (With Translation)

(b) curie (With Translation)

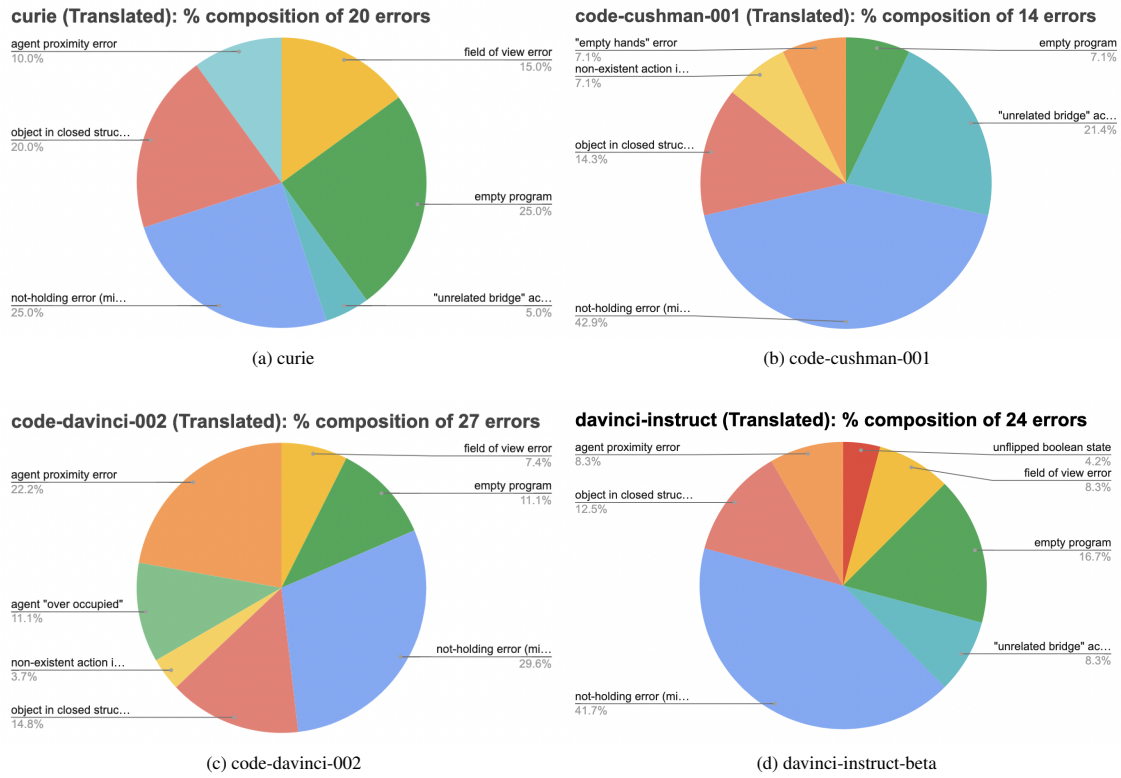
(c) curie (Without Translation)

(d) curie (Without Translation)

Figure 4: Decomposition of pre-condition errors into error types for the same LLM (`curie`) run with different random seeds with the open-loop method

To assess the variability introduced by random seeds (different starting states in a scene graph) in the natural language generated by LLMs, we decomposed the pre-condition errors produced by running the same LLM (`curie`) using the open-loop method in Figure 4.

Both with and without translations, the types pre-condition errors remains consistent. The distribution across error types seems to vary less drastically (by 2.2% on average) without translation than with translation (by 5% on average); though the distribution remains consistent when considering the 3 most prominent error types.



(a) curie

(b) code-cushman-001

(c) code-davinci-002

(d) davinci-instruct-beta

Figure 5: Decomposition of pre-condition errors into error types for the best performing and state-of-the-art LLMs with the open-loop method

A.3 Causes for execution errors with open-loop planning

Figure 7 highlights the percentage composition of errors of the 2 LLMs with highest executability, i.e., `curie` and `code-cushman-001` (Figures 5a and 5b respectively) and 2 novel state-of-the-art LLMs, i.e., `code-davinci-002` and `davinci-instruct-beta` (Figures 5c and 5d respectively).

From the figures, "*missing object*" in dark blue (see definition in Table 3) is the most prominent error type across the 4 LLMs – accounting for 34.8% of errors on average – followed by either "*agent proximity*" errors, "*empty program*" errors or "*other (precondition)*" errors. Note the large "*empty program*" error presence for `curie` with Translation and the prominent "*other (precondition)*" error presence for `code-cushman-001` with Translation. In fact, for each of the 4 LLMs shown above, just the "*missing object*", "*agent proximity*" and "*empty program*" error types account for at least 50% of errors produced by the LLM and these error-types could be made executable using *single step* corrections with appropriate prompting.

Figure 6 shows the union of all tasks that fail between the `curie` and `code-cushman-001` LLMs, with and without the Translation LLM, visualizing the cause of the execution error using the color key to the right; `curie` and `code-cushman-001` are the language models with highest translated executability.

The symmetrical color pattern along pairs of columns (for both `curie` and `code-cushman-001`) indicates that incorporating the Translation LLM does not *modify* or *alter* the distribution of errors for non-executable tasks i.e., the Translation LLM seems to be a purely *subtractive* component that eliminates certain types of errors. This is further corroborated by qualitative observations that the *sequence of actions* taken and plan syntax remain identical with and without Translation LLM, with the exception of grounding atomic actions and object tokens to be more executable.

Additionally, of the top 5 "most difficult" tasks (i.e., tasks for which all 4 LLMs produced non-executable programs), all 4 language models encountered errors of identical type, viz. "*object en-*

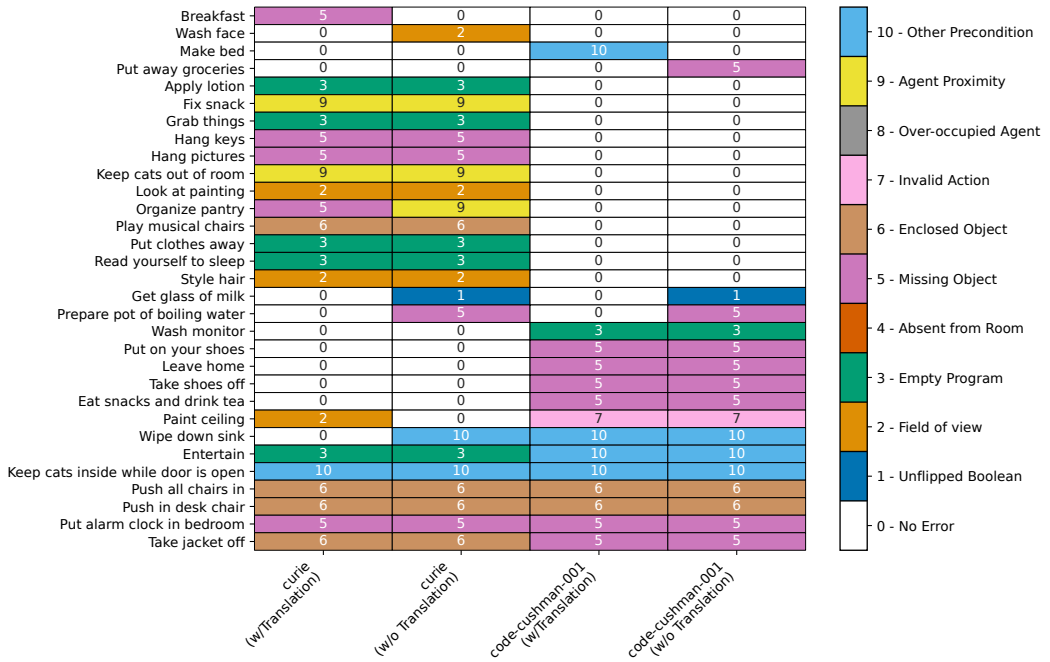


Figure 6: Matrix showing tasks that failed using `curie` and `code-cushman-001` LLMs along with causes for those errors (based on Table 3).

closed" and "*other (precondition)*" errors. For example, the models failed with the task "Keep cats inside while door is open" because the `door` was closed.

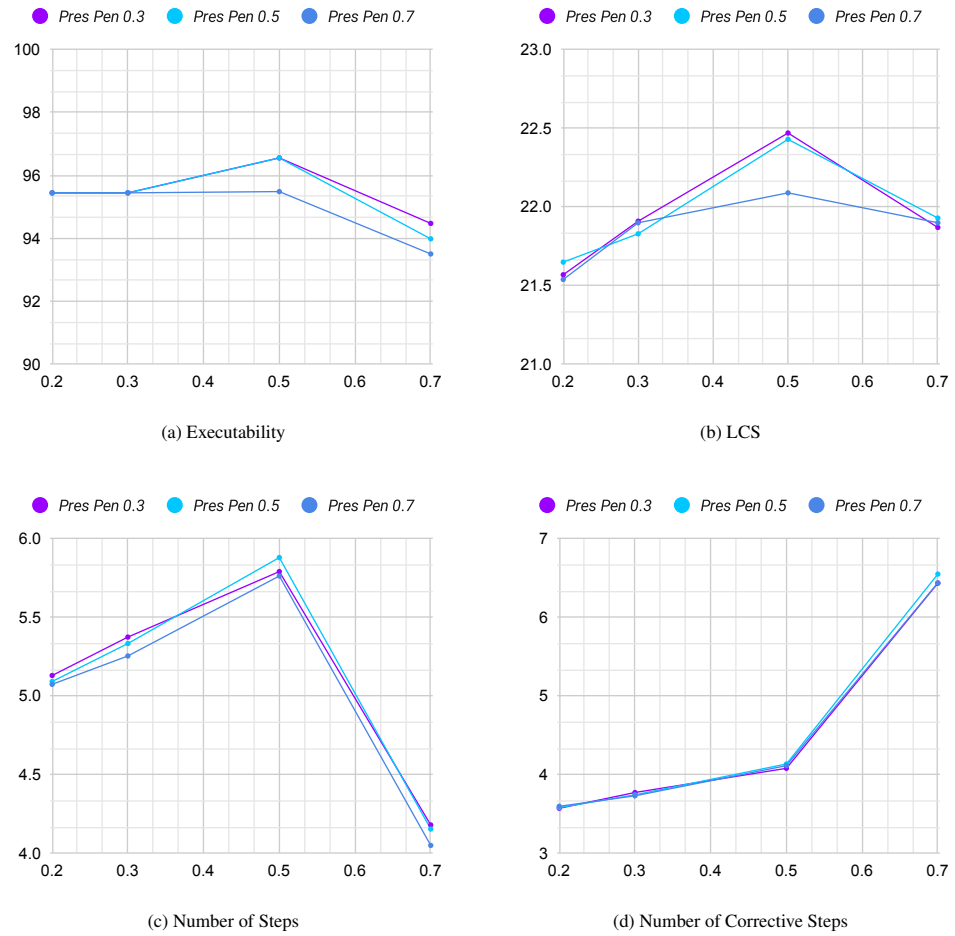
Hence, we find certain tasks are *intrinsically* more complex, increasing the number and difficulty of failed pre-requisites to target with re-prompting and corrective actions. There are several qualitative observations that further validate this finding:

- Tasks such as "Take jacket off" or "Entertain" have *implicit* assumptions (e.g., jacket is *already* on) or are extremely vague, respectively, which could impede a language model's ability to generate sensible actions without observing the current state.
- Certain tasks require initial states or objects that *are not present* in the scene (e.g., tasks like "push all chairs in" usually require *no actions* since all chairs begin tucked under a table). Therefore, generating viable plans for such tasks requires knowledge of the initial state.
- LLMs using *pure language* (i.e., without environment feedback) are too optimistic, as LLMs often *infer object locations* from the task/prompt and assume objects are *available with no obstructions*. Therefore, plans generated by LLMs without feedback are usually only executable in the theoretical regime of "describe *a* plan for `task`", as opposed to "within constraints of an embodied environment".

The observations above tell us that neither seed variations nor the addition of Translation LLM influence error distributions, and that certain tasks are *intrinsically* harder to correct for given the assumptions made by LLMs. Figures 7 and 6 also highlight that each LLM produces a unique pattern of error types. Hence, we determine that error distributions are unique artefacts of LLMs themselves, and some of the most *common* error types have the potential to be corrected with re-prompting.

A.4 Hyper-parameter Search for Re-sampling Basline

Our approach (Section 4.1 and Section 4.3) encourages the LLM to generate "corrective actions" upon receiving pre-condition error information from the environment, thereby making it a closed-loop planning system. We hypothesized that the inclusion of additional re-prompts and corrective actions would increase plan length (during re-sampling) and the frequency of tokens used (during both



(a) Executability

(b) LCS

(c) Number of Steps

(d) Number of Corrective Steps

Figure 7: Graphs showing hyper-parameter search results across several metrics and penalty values.

re-prompting and re-sampling). Thus we performed a hyper-parameter search on the re-sampling baseline, defined in Section 4.3, to assess how the LLM sensitivity to hyper-parameters and optimize parameter selection for our re-prompting approaches in the closed-loop domain.

Our hyper-parameter search explores different temperatures and presence penalties in the ranges shown in Table 4 below. These parameters influence how out of distribution a proposed action is and penalize the repetition of old topics, respectively, making them the strongest drivers of LLM performance.

Table 4: Hyper-parameters and their corresponding values used in our search process

Hyper-parameter	**Search Values**
Temperature	0.2, 0.3, 0.5, 0.7
Presence Penalty	0.3, 0.5, 0.7

In Table 5 below, the columns represent presence penalties (PresPen) and the rows represent temperature (Temp). Each table entry includes a set of metrics in top-to-bottom order: executability, LCS, correctness, number of steps and the number of corrections (as described in Section 5.2)

Presence Penalty does not seem to influence executability at lower temperatures; for higher temperatures, executability decreases with increased presence penalties. No clear relationship exists between LCS and presence penalty. The number of corrective steps seems to be independent of

	PresPen@0.3	PresPen@0.5	PresPen@0.7
Temp@0.2	95.45%	95.45%	95.45%
	21.57%	21.65%	21.54%
	-	-	-
	5.131	5.093	5.075
	3.575	3.584	3.601
Temp@0.3	95.45%	95.45%	95.45%
	21.91%	21.83%	21.90%
	-	-	-
	5.375	5.334	5.255
	3.776	3.746	3.734
Temp@0.5	**96.59%**	**96.59%**	95.49%
	22.47%	22.43%	22.08%
	-	-	-
	5.791	5.880	5.763
	4.083	4.138	4.117
Temp@0.7	94.48%	93.99%	93.51%
	21.87%	21.93%	21.90%
	-	-	-
	4.182	4.156	4.052
	6.435	6.550	6.438

Table 5: Hyper-parameter search performance (executability, LCS, correctness, no. steps and no. corrections) over different temperatures (Temp) and presence penalties (PresPen) using the `instruct-davinci` LLM

presence penalty as well.No other significant observations can be made as to the independent influence of presence penalty

Temperature and executability seem to follow a quadratic relationship i.e., there is an optimal temperature value around which executability is lower, for all presence penalties. A similar quadratic correlation is also seen between LCS and temperature, as well as between the number of steps and temperature, peaking at approx. Temp = 0.5 in both cases. More evidently, the number of corrective steps seem to monotonically increase with temperature

Maximum executability and LCS are observed using presence penalty 0.3 and temperature of 0.5. We use these hyper-parameters for our re-sampling and re-prompting methods in Sections 4.3 and 4.1

A.5 Samples of Qualitative Examples

The following examples highlight 10 sample tasks comparing the Huang et al. (2022) baseline with the re-sampling baseline and our Re-prompting *w/Cause* method.

We find qualitative improvements in the planner's corrective capabilities (correcting pre-condition errors due to re-prompting) as well as the degree of relevant specificity and detail in plans (due to optimized hyperparameter selection in the closed loop domain)

Task	**Baseline Method** (Huang et al. (2022))	**Re-Sampling**	**Re-Prompting**
Get glass of milk	Step 1: Walk to dining room Step 2: Walk to kitchen counter Step 3: Find kitchen counter Step 4: Open stove Step 5: Find milk Step 6: Grab milk Step 7: Close stove Step 8: Walk to fridge Step 9: Find fridge Step 10: Open fridge	Step 1: Walk to dining room Step 2: Walk to kitchen counter Step 3: Find kitchen counter Step 4: Find fridge Step 5: Open fridge Step 6: Find milk Step 7: Grab milk	Step 1: Walk to dining room Step 2: Walk to kitchen counter Step 3: Find kitchen counter Step 4: Find fridge Step 5: Open fridge Step 6: Find milk Step 7: Grab milk
Prepare pot of boiling water	Step 1: Find stove Step 2: Find pot Step 3: Turn to stove	Step 1: Find stove Step 2: Find pot Step 3: Turn to stove	Step 1: Find stove Step 2: Find pot Step 3: Turn to stove Step 4: Switch on stove Step 5: Put pot on stove Error: I don't have the pot. A correct step would be to Step 5: Find pot
Look at mirror	Step 1: Walk to bedroom Step 2: Walk to mirror Step 3: Find mirror Step 4: Turn to mirror Step 5: Look at mirror	Step 1: Walk to bedroom Step 2: Walk to mirror Step 3: Find mirror Step 4: Turn to mirror Step 5: Look at mirror	Step 1: Walk to bedroom Step 2: Walk to mirror Step 3: Find mirror Step 4: Turn to mirror Step 5: Look at mirror Step 6: Find face Step 7: Turn to face Step 8: Look at face Step 9: Find hair Step 10: Turn to hair Step 11: Look at hair Step 12: Find clothes dress Step 13: Turn to clothes dress Step 14: Look at clothes dress

Task	**Baseline Method** (Huang et al. (2022))	**Re-Sampling**	**Re-Prompting**
Listen to Music	Step 1: Walk to home office Step 2: Walk to remote control Step 3: Find remote control Step 4: Grab remote control Step 5: Walk to couch Step 6: Sit on couch *Error: agent is sitting so cannot execute Step 8 until the agent stands up* Step 7: Touch remote control Step 8: Find television Step 9: Switch on television Step 10: Turn to television Step 11: Watch television Step 12: Switch on stereo Step 13: Turn to stereo Step 14: Switch on cd player Step 15: Turn to cd player	Step 1: Walk to home office Step 2: Walk to remote control Step 3: Find remote control Step 4: Grab remote control Step 5: Walk to couch Step 6: Sit on couch Step 7: Touch remote control Step 8: Find music stand	Step 1: Walk to home office Step 2: Walk to remote control Step 3: Find remote control Step 4: Grab remote control Step 5: Walk to couch Step 6: Sit on couch Step 7: Touch remote control Step 8: Find television Error: I am not near the television. A correct step would be to Step 8: Walk to tv Error: I am sitting. A correct step would be to Step 8: Stand up Step 9: Walk to stereo Step 10: Switch on stereo Step 11: Turn to stereo Step 12: Find cd Step 13: Grab cd
Hang pictures	Step 1: Walk to home office Step 2: Walk to table Step 3: Find keys Step 4: Grab keys Step 5: Walk to wall Step 6: Find hanger Step 7: Put keys on hanger Step 8: Walk to livingroom Step 9: Find picture *Error: agent has not picked up picture, so cannot execute Step 10* Step 10: Put picture on hanger	Step 1: Walk to livingroom Step 2: Walk to table Step 3: Find picture Step 4: Grab picture Step 5: Walk to wall Step 6: Find hanger Step 7: Put picture on hanger	Step 1: Walk to livingroom Step 2: Walk to table Step 3: Find picture Step 4: Grab picture Step 5: Walk to wall Step 6: Find hanger Step 7: Put picture on hanger
Eat snacks and drink tea	Step 1: Walk to livingroom Step 2: Sit on couch Step 3: Drink tea Step 4: Eat food snack Step 5: Drink juice	Step 1: Walk to livingroom Step 2: Walk to couch Step 3: Walk to kitchen Step 4: Walk to cupboard Step 5: Open cupboard Step 6: Find tea Step 7: Grab tea Step 8: Walk to kettle Step 9: Switch on kettle Step 10: Pour water into cup Step 11: Pour tea into cup Step 12: Drink tea Step 13: Drink water	Step 1: Walk to dining room Step 2: Walk to cupboard Step 3: Open cupboard Step 4: Find tea Step 5: Grab tea Step 6: Walk to table Step 7: Pour tea into cup
Receive credit card	Step 1: Walk to home office	Step 1: Walk to home office Step 2: Find chair Step 3: Sit on chair	Step 1: Walk to home office Step 2: Walk to cabinet Step 3: Find cabinet Step 4: Open cabinet Step 5: Find creditcard Step 6: Grab creditcard

Task	**Baseline Method** (Huang et al. (2022))	**Re-Sampling**	**Re-Prompting**
Empty dishwasher and fill dishwasher	Step 1: Walk to dishwasher Step 2: Open dishwasher	Step 1: Walk to dishwasher Step 2: Open dishwasher Step 3: Walk to sink Step 4: Find faucet Step 5: Switch on faucet Step 6: Find bowl Step 7: Grab bowl Step 8: Rinse bowl	Step 1: Walk to dishwasher Step 2: Open dishwasher Step 3: Walk to sink Step 4: Find faucet Step 5: Switch on faucet Step 6: Find bowl Step 7: Grab bowl Step 8: Rinse bowl Step 9: Switch off faucet Error: I am not near the faucet. A correct step would be to Step 9: Walk to faucet
Take shoes off	Step 1: Walk to home office Step 2: Walk to shoes Step 3: Find shoes	Step 1: Walk to home office Step 2: Walk to shoes Step 3: Find shoes	Step 1: Walk to home office Step 2: Walk to shoes Step 3: Find shoes Step 4: Take off shoes Step 5: Walk to dresser Step 6: Open dresser Step 7: Grab shoes Step 8: Put shoes in dresser Error: putin dresser is not allowed. A correct step would be to Step 8: Put shoes in closet