CAPE: Corrective Actions from Precondition Errors using Large Language Models

Shreyas Sundara Raman[1*], Vanya Cohen[2], Ifrah Idrees[1], Eric Rosen[1], Ray Mooney[2], Stefanie Tellex[1], David Paulius[1]

***Abstract*— Extracting commonsense knowledge from a large language model (LLM) offers a path to designing intelligent robots. Existing approaches that leverage LLMs for planning are unable to recover when an action fails and often resort to retrying failed actions, without resolving the error's underlying cause. We propose a novel approach (CAPE) that attempts to propose corrective actions to resolve precondition errors during planning. CAPE improves the quality of generated plans by leveraging few-shot reasoning from action preconditions. Our approach enables embodied agents to execute more tasks than baseline methods while ensuring semantic correctness and minimizing re-prompting. In VirtualHome, CAPE generates executable plans while improving a human-annotated plan correctness metric from 28.89% to 49.63% over SayCan. Our improvements transfer to a Boston Dynamics Spot robot initialized with a set of skills (specified in language) and associated preconditions, where CAPE improves the correctness metric of the executed task plans by 76.49% compared to SayCan. Our approach enables the robot to follow natural language commands and robustly recover from failures, which baseline approaches largely cannot resolve or address inefficiently.**

I. Introduction

Generalized robots can assist humans by accomplishing a diverse set of goals in varying environments. Many such agents are equipped with a library of skills for primitive action execution. Here, natural language can enable more seamless human-robot interaction by leveraging these skill libraries [1]. Given a task description or command from a human, a robot must be able to autonomously propose a sequence of actions (from its skill repertoire) that realizes the given task. Critical to such an application is the agent's ability to ground skills specified in language to their environment and reason about state changes from skill execution or the relevance of proposed actions towards a task's objective. For instance, if a robot is commanded to "put away groceries", it must ground the concept of "groceries" to objects in its environment and decompose the task of "putting away" to meaningful constituent skills from its repertoire.

Thus, extracting actionable knowledge from LLMs requires context about the agent's embodiment and environment state. Related works that extract plans from LLMs using prompting strategies assume access to extra information such as: 1) predefined skills with preconditions [2] 2)visual-language models that determine affordance from observations like SayCan [2], 3) descriptions of the agent's goal [3, 4] or 4) descriptions of observation and action spaces for reasoning in text-based video games [5, 6]. These approaches do not efficiently nor explicitly resolve failure modes during planning: they either propose actions that are not afforded execution in the environment (i.e. violate preconditions, such as walking through a closed door), or resort to exploring the entirety of an agent's action library to identify affordable actions [2].

We use *precondition errors* to resolve action failure, which is motivated by the vast body of research on planning algorithms and definitions like PDDL [7]. In these settings, robots are equipped with a repertoire of skills, each requiring certain *preconditions* to be satisfied in order to afford their execution. We target the failure mode of executing skills without satisfying their preconditions in this setting. Using parametrized skills that are codified into natural language, we leverage a LLM to generate a sequence of actions for execution towards completing a task. When a robot or agent fails to execute an action due to precondition violation, we use a templated-prompting strategy called CAPE (*Corrective Actions from Precondition Errors*) to query the LLM for corrective actions (Figure 2). Our prompts either specify that the action failed or provide explanatory details about the cause of action failure, flexible to the extent of knowledge accessible to the robot about its skills or domain. This paper builds on our previous work [8] with more rigorous analysis, larger scale human evaluation, additional (more competitive) baselines and experiments both in simulation and real-world settings.

Our contributions are as follows: we introduce CAPE a novel approach for LLM planning that generates corrective actions to recover from failure, using prompts based on precondition errors and few-shot learning. We detail how our re-prompting strategy can be deployed on embodied systems with both large and small skill repertoires using different re-prompting methods. We also evaluate CAPE against several baselines [3, 2] and ablations to show our method achieves near-perfect plan executability and more semantically correct plans for various tasks executed on a Boston Dynamics Spot robot and a simulated agent in VirtualHome [9].

II. Background

In-Context Learning: Brown et al. [10] introduced GPT-3: a 175 billion parameter language model capable of few-shot learning for novel tasks, including Q&A, arithmetic, and comprehension, by prompting the LLM with in-context task examples used for structural and syntactic guidance. This

Project Website: `https://shreyas-s-raman.github.io/CAPE/`

*Corresponding Author (Email: shreyas_sundara_raman@brown.edu)
[1]Brown University, Providence, RI, USA.
[2]The University of Texas at Austin, Austin, TX, USA.

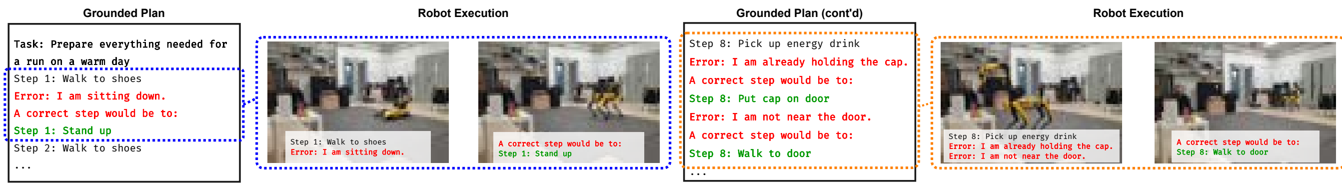

Fig. 1. Qualitative results of CAPE for robot execution of the task *"prepare for a run"*. We highlight 2 cases where re-prompting with precondition error information resolves action failures (*left:* resolving prerequisite for walking by standing; *right:* resolving one-armed manipulation constraint).

approach offers several advantages over task learning with fine-tuned pre-trained latent language representations [11, 12, 13] and zero-shot inference [14] due to sample efficiency and task generalization. In-context learning performs best when examples are relevant to the test task; we retrieve in-context examples based on their semantic similarity to a task [15, 3].

Open-Loop Plan Generation: CAPE extends the open-loop framework of Huang et al. [3], which generates a plan for a task zero-shot without feedback from the environment. Given a query task $\mathcal{Q}$ (i.e. the target task), first, a high-level example task $\mathcal{T}$ and its plan are chosen from a *demonstration set* as a contextual example of a free-form plan for the *Planning LLM*; note that $\mathcal{T}$ is selected to maximizes cosine similarity with the *query task* $\mathcal{Q}$. The *Planning LLM* auto-regressively generates actions for task $\mathcal{Q}$ in free-form language via in-context learning. The *Translation LLM* then utilizes a BERT-style LM (Sentence-BERT [16]) to embed the generated free-form actions (a_l) to the most semantically (i.e., cosine-similar) action in the agent's repertoire (a_e). Here, an admissible action refers to a language description of an action in the agent's skill repertoire. The chosen admissible action ($\hat{a}_e$) is then appended to the unfinished prompt to condition future auto-regressive step generation on admissible actions. We investigate how to improve planning in the closed-loop domain by leveraging precondition error feedback as an auxiliary modality of information.

Affordance and Preconditions: Action preconditions and effects are commonly adopted in robot planning domains, such as those using PDDL [7] or STRIPS [17], where a set of predefined skills are accessible to robots. Structured affordance models factorize states into *preconditions*, which define affordance by independent state components that must be satisfied for execution. This can be formalized by the options framework [18], where options $\mathcal{O}(s)$ over the state space $\mathcal{S}$ form a set of temporally extended actions equivalent to those in an agent's skill repertoire. An initiation set of an option $\mathcal{I}(o)$ defines the states in which option execution is afforded (akin to preconditions), while a termination condition $\beta_o(s)$ describes the terminal state of the skill. If the current state fails to meet the initiation state of an option, a precondition error arises. Environment states in these domains can be factorized in a semantically meaningful manner to evaluate the validity of preconditions for a skill, thus enabling a skill's affordance to be measured. Learning and modeling preconditions have been largely studied in model-based approaches that leverage symbolic planning [19, 20]. Our work investigates how these preconditions can be leveraged to improve planning using LLMs.

III. Method

Given a task specified in natural language, we use LLMs to generate a plan. When an agent or robot fails skill execution, CAPE integrates precondition errors into a prompt that aims to repair plans.

A. Plan Generation via Re-prompting

In control theory, a closed-loop system relies on feedback from its outputs for adaptive control [21]. Similarly, CAPE leverages error feedback in a closed-loop planning setup, which allows it to correct a generated plan when any action proposed by the LLM is not afforded execution, by injecting precondition error information as *corrective prompts* (see Figure 3). Certain errors require more context about the agent's state, action history and environment. For instance, correcting an error in VirtualHome [9] such as <character> (1) does not have a free hand when executing "[GRAB] <obj> (1) [1]" requires knowledge of what objects the agent previously grabbed or is currently holding, as well as available adjacent objects on which to drop the held object and free the agent's hands. We construct corrective prompts composed of the following segments of information:

- **Contextual Information**: This includes relevant context and action history upon action failure. We supply the query task $\mathcal{Q}$ and the query steps up to the action that has failed for context.
- **Precondition Error Information**: We optionally include details on the violated precondition in the prompt, which is tailored based on the degree to which the agent can assess precondition violations.

In order for the Translation LLM to ground the natural utterance, we need to assume that the agent is equipped with a skill repertoire of actions that are admissible to the environment. Thus, preconditions only need to be defined for each general parametrized skill. It is important to note that the Planning LLM used by CAPE does not explicitly know about the agent's skills nor the preconditions for each skill during the re-prompting process. Instead, we utilize the preconditions (a set of logical propositions assessing a skill's affordance) defined for each parametrized skill in our skill repertoire to obtain precondition errors by comparing with the environment's current state. The environment state and precondition propositions are external to the LLM, but the error information produced by them can then be integrated into a corrective language prompt. As a result, there is a

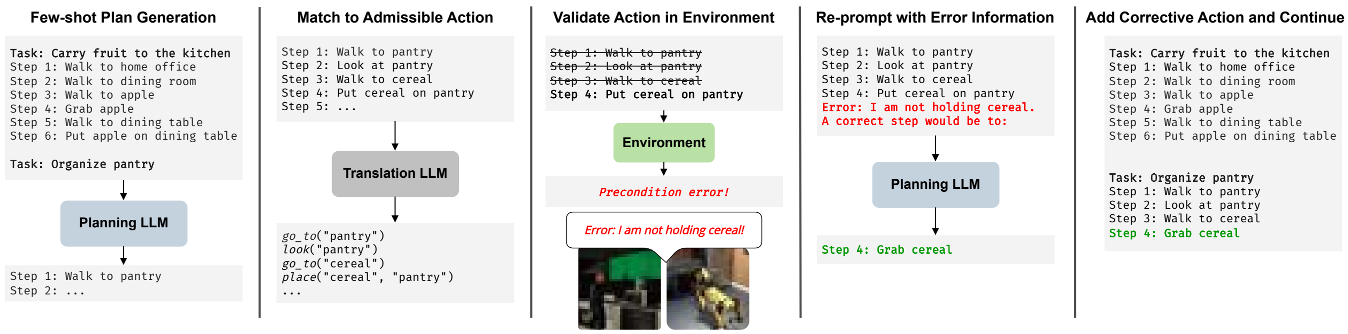

Fig. 2. Overview of CAPE: To generate an executable plan, we select an in-context example task (from a demonstration set) that is most semantically similar to the query task. The Planning LLM generates a natural language description for the next step in the plan. The Translation LLM [16] grounds this description to an admissible skill in the agent's repertoire. If this action violates preconditions for the proposed skill, the precondition error information is formatted into a *corrective prompt*, which along with the failed skill are provided to the LLM for corrective action proposal.

significant layer of abstraction, where the Planning LLM has to *infer* the cause of failures and environment mechanics based only on the context provided by the corrective prompt and the agent's own action history in order to propose an appropriate corrective action. The use of preconditions is typical in planning domains where the robot or agent has skills built on representations that define preconditions and effects, e.g., PDDL [7], STRIPS [17] or LTL [22]. Since preconditions are already defined in these representations, appropriate language feedback can be integrated into the precondition module with minimal extra effort.

Re-prompting Strategies: We re-prompt with varying degrees of precondition error detail in both zero-shot ($\mathcal{Z}$) and few-shot ($\mathcal{F}$) approaches, and denote either setting by P, where $P = \mathcal{Z} \vee \mathcal{F}$. Few-shot reprompting ($\mathcal{F}$) provides 3 incontext precondition errors and corrective actions, taken from the demonstration set that is separate from the query task, that are only injected when the LLM-Agent needs to propose corrective actions i.e. not for executable actions. Re-prompting strategies can be categorized as follows:

- **Re-prompting with Success Only ($\mathcal{Z}_S$):** solely informs the LLM that the action failed (i.e., "`Task Failed`").[1]
- **Re-prompting with Implicit Cause ($\mathcal{Z}_I$):** provides more detail to the LLM with a prompt template containing the name of the failed action and the object(s) the agent interacted with (i.e., "`I cannot <action> <object>`"). This requires the LLM to infer the cause of error when proposing corrective actions.
- **Re-prompting with Explicit Cause ($\mathcal{Z}_E$):** states the precondition violation that prevents action execution, in addition to feedback provided by $\mathcal{Z}_I$ (i.e., "`I cannot <action> <object> because <precondition violation>`").

P_E gives the most error feedback to the LLM. However, P_S and P_I only require a target object and skill associated with the failed action, which the LLM proposes. Likewise, a $\mathcal{P}_S$ prompt can work with visual-language model approaches like SayCan [2], whereas P_I and P_E can work with task and motion planning approaches [20]).

Scoring Grounded Actions: We use the scoring function $\mathcal{S}_w$ (Equation 1), a weighted combination of log probability and cosine similarity, which is thresholded to determine the feasibility of each proposed grounded step [3]. 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 parameterizes the pretrained Planning LLM and X_i is a generated step consisting of n tokens $(x_{i,1}, ..., x_{i,n})$. 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 Translation LLM embedding function, a is the predicted action, and a_e is the admissible action for which we estimate the distance with respect to:

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

where β is a weighting coefficient. $\mathcal{S}_w$ prioritizes the quality of natural language at the cost of semantic translation and often results in mistranslations, which are 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. Further, the mean log probability term is unbounded, which makes finding a score threshold more challenging. 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:

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

All re-prompting methods, by default, are reported using $\mathcal{S}_w$. We report results using $\mathcal{S}_g$ specifically for the re-prompting with explicit cause (P_E) method.

B. Baseline: Plan Generation via Re-sampling

When a plan action is not executable, the closed-loop re-sampling method does not use error feedback to generate corrective prompts. Instead the approach iteratively evaluates the top k admissible actions proposed by the Planning LLM and grounded by the Translation LLM in reverse order of the weighted sum of mean log probability and cosine similarity

[1] This is analogous to success detection used in Inner Monologue [4], which was used to determine whether to re-execute failed actions since low-level policy success is stochastic. However, our aim is to repair the high-level plans generated by the LLM with corrective actions that arise from a new distribution of actions using precondition feedback.

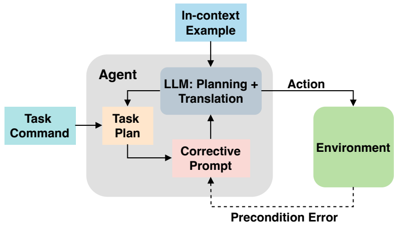

Fig. 3. CAPE uses a LLM to generate plans for tasks specified in natural language. When the agent fails to execute a step, we re-prompt the LLM with error information, utilizing the latent commonsense reasoning and few-shot learning capabilities of the LLMs to overcome execution errors.

until an executable action is found. If none of the k re-sampled admissible actions are executable, plan generation terminates. This ablation assesses whether CAPE's feedback allows for more efficient corrections due to the utility of precondition error information, rather than more attempts at proposing corrective actions.

C. Baseline: Plan Generation with SayCan

We compare to SayCan [2] as a baseline method. When generating every step, SayCan assigns a score for each action in the agent's repertoire and the action with the highest score is executed. This score is the product of the log probability and affordance for each action. This process is repeated until the termination skill (`done`) is assigned the highest score. There are two important adjustments in our SayCan implementation for experiments in VirtualHome [9]:

- As there are over 50K possible object-action pairs in VirtualHome, it is intractable to evaluate every admissible skill for every step during planning. Instead the LLM generates a *prototype* step. Using this we subsample the 500 most semantically similar object-action pairs (measured by cosine similarity) and at most 1000 object-action pairs containing the target object. This forms a subset of ≤ 1500 skills to iterate over and score. Subsampling semantically similar skills and matching to skills affecting the same objects ensures the $\leq$1500 subsampled skills also have the highest log probability according to the LLM. In most cases, nearly all the skills pertaining to a specific object are populated in the set of 1000, and additional semantically similar skills are added as part of the 500.
- A *perfect* affordance model is initially used, since heuristic based precondition checks in VirtualHome allow 0% affordance misclassification. However, as Ahn et al. [2] mentions a 16% of planning failure at minimum, where 35% of these failures originate from errors related to the affordance model, we also present a *noisy* ablation of SayCan with a 6% ($16\% \times 35\%$) random chance of misclassifying the oracle affordance, i.e., false when actually true or true when actually false.

Similar to CAPE, SayCan assumes that language descriptions of an agent's skills are known and available during planning. SayCan leverages a trained affordance model (value function) to evaluate the executability of skills and can easily be extended to check for or predict language-specified precondition violations, similar to those leveraged in our method.

IV. Evaluation

We test the hypothesis that corrective re-prompting can increase the executability of LLM models for interpreting language directed to robots while maintaining plan correctness. We focus on larger state-of-the-art LLMs, particularly those in OpenAI's `davinci-instruct` line, for their demonstrated capabilities in instruction-following and planning tasks [10, 23]. We evaluate eight approaches in a zero-shot setting: the three baselines – Huang et al. [3] (Section II), the closed-loop re-sampling (Section III-B), and SayCan [2] (Section III-C) – and CAPE with our proposed ablations (Section III-A). We refer to CAPE's zero-shot approaches as success only ($\mathcal{Z}_S$), implicit cause ($\mathcal{Z}_I$), explicit cause ($\mathcal{Z}_E$), and explicit cause with scoring function ($\mathcal{Z}_E + S_g$). We also evaluate CAPE with explicit cause re-prompting in a few-shot setting ($\mathcal{F}_E$), with and without S_g, where we present the LLM with three examples of precondition errors and corresponding corrective actions to infer the appropriate corrective action for the target task.

A. Experimental Setup

We evaluate CAPE across seven scenes in VirtualHome [9]) and with a Boston Dynamics Spot robot (see Figure 1) using the metrics discussed in the following section. Our objective is to show that corrective re-prompting resolves unmet preconditions during planning and execution by embodied agents and robots in a variety of settings; VirtualHome provides a large skill sets with many objects, while the robot environments focus on physical embodiment with fewer objects and skills. For VirtualHome, we evaluate plans generated for 100 household tasks (e.g., “make breakfast", “browse the Internet"). To show that our method can be extended to novel unstructured real-world environments, we compare plans generated by CAPE with those generated by the 3 baselines across 6 tasks for human-assistance and 2 scenes for each task.

B. Robot Demonstration

To demonstrate CAPE's capability on unstructured real-world tasks, we compare our re-prompting approaches against all 3 baselines on the Boston Dynamics Spot, a quadruped robot with a single 6-DOF arm. The demonstrations use two novel scenes (a lab environment and a kitchen) with structural variation in the maps and objects in the environment. On average 9 household objects (e.g., phone, bed, coffee, etc.), each with five state attributes (e.g, `location, grabbed, open, turned on`) are present in each scene. We evaluate performance on 6 tasks: 1) Prepare for a run on a warm day, 2) Put the phone on the nightstand, 3) Iron a shirt, 4) Put mail in storage, 5) Organize Pantry, and 6) Put away groceries. We assume the Spot robot has access to a set of 14 parametrized skills (e.g. *stand up*, *walk to*, *pick up*, *put*, *touch*, *look at*, *open* and *close*) and the initialization states (preconditions) needed for

TABLE I

PERFORMANCE OF BASELINES AND CAPE ACROSS 100 TEST-SET TASK TYPES AND 7 SCENES IN VIRTUALHOME [9] (700 TOTAL).

Method	%Correct↑	%Exec.↑	%Aff.↑	%GS↑	LCS↑	Fleiss' Kappa↑	Steps↓	Corrections↓
Baselines								
Huang et al. [3]	38.15	72.52	87.72	95.54	20.80	0.47	7.21	N/A
Re-sampling	38.89	76.43	75.24	95.65	23.45	0.45	6.87	7.67
SayCan [2] (Perfect)	28.89	**100.00**	**100.00**	94.17	22.98	0.33	7.56	N/A
SayCan [2] (Noisy)	22.59	97.33	99.89	94.68	19.43	0.46	**5.97**	N/A
CAPE: Zero-Shot ($\mathcal{Z}$)								
Success Only ($\mathcal{Z}_S$)	41.11	97.57	90.46	95.49	23.79	0.38	7.68	1.08
Implicit Cause ($\mathcal{Z}_I$)	42.22	97.86	90.05	95.64	23.20	**0.51**	7.48	0.93
Explicit Cause ($\mathcal{Z}_E$)	42.59	98.29	91.69	95.69	23.48	0.45	8.16	**0.72**
Explicit Cause ($\mathcal{Z}_E$ + $\mathcal{S}_g$)	48.52	**98.57**	91.28	96.23	23.30	0.35	8.81	1.31
CAPE: Few-Shot ($\mathcal{F}$)								
Explicit Cause ($\mathcal{F}_E$)	47.04	**98.57**	**92.29**	96.05	**24.20**	0.41	8.69	0.89
Explicit Cause ($\mathcal{F}_E$ + $\mathcal{S}_g$)	**49.63**	96.29	90.93	**96.29**	23.47	0.39	9.35	1.82

TABLE II

PERFORMANCE OF BASELINES AND CAPE ACROSS 6 TEST-SET TASKS AND 2 SCENES FOR HOUSEHOLD TASKS WITH ROBOT DEMO (12 TOTAL).

Method	%Correct↑	%Exec.↑	%Aff.↑	%GS↑	LCS↑	Fleiss' Kappa↑	Steps↓	Corrections↓
Baselines								
Huang et al. [3]	16.67	41.64	56.46	66.03	26.77	0.28	**2.40**	N/A
Re-sampling	13.33	75.00	47.98	67.33	32.92	**0.71**	4.60	13.19
SayCan [2] (Perfect)	28.33	83.33	**83.33**	68.02	41.13	0.26	6.80	N/A
SayCan [2] (Noisy)	16.67	66.67	79.13	67.54	38.36	0.22	6.80	N/A
CAPE: Zero-Shot ($\mathcal{Z}$)								
Success Only ($\mathcal{Z}_S$)	18.33	75.00	43.05	66.02	32.45	0.28	3.04	2.25
Implicit Cause ($\mathcal{Z}_I$)	20.00	75.00	52.37	66.25	32.44	0.32	3.14	1.83
Explicit Cause ($\mathcal{Z}_E$)	31.67	**100.00**	79.69	69.18	48.12	0.11	6.30	1.91
Explicit Cause ($\mathcal{Z}_E + \mathcal{S}_g$)	23.33	**100.00**	79.04	69.85	46.68	0.12	6.30	**1.73**
CAPE: Few-Shot ($\mathcal{F}$)								
Explicit Cause ($\mathcal{F}_E$)	45.00	**100.00**	81.36	**77.91**	65.07	0.23	11.70	2.91
Explicit Cause ($\mathcal{F}_E + \mathcal{S}_g$)	**50.00**	**100.00**	80.70	77.40	**69.77**	0.12	11.30	2.90

their execution. The robot first builds a semantic map from images taken and waypoints set across the scene; visual-language models (VLM) like (CLIP [24] and CLIPSeg [25]) are then used to ground admissible skills to spatial points for navigation or grasping in the physical environment, similar to approaches like NLMap-SayCan [26]. The robot's embodiment (a single arm), a limited skill repertoire and extensibility to novel unstructured environments make this a challenging setting for task completion. Figure 1 highlights how corrective prompting enables successful completion of the task *"prepare for a run on a warm day"*. Re-prompting enables the Spot to resolve precondition failures caused by the robot's initial state and due to its single-arm embodiment. We provide demonstrations for additional tasks and scenes in our supplementary video.

C. Human Evaluation

As in Huang et al. [3], we use human evaluation to determine the correctness of generated plans through the crowd-sourcing platform Prolific.[2] 50% of the total tasks across all baselines and ablations were supplied to annotators. For each task, five annotators evaluate the grounded plan in English to determine whether it accomplishes the given task objective. Each plan is generated in a randomly selected environment.

[2]Prolific – `https://www.prolific.co`

D. Evaluation Metrics

We adopt the % Executability and % Correctness metrics from Huang et al. [3]. **% Executability** measures if *all* grounded actions satisfy preconditions imposed by the environment i.e. if the *entire* plan can be executed by the agent as afforded to its environment and state. **% Affordability** measures the average percentage of all plan steps that are executable, after skipping non-executable steps, in cases where the entire plan is not afforded execution (i.e. partial executability).

% Correct is a human-annotated assessment of semantic correctness and relevance of a grounded plan to the target task. Assessing "quality" of natural language-based plans is difficult and potentially ambiguous using only executability i.e. an fully executable plan need not realize the task objective; thus, we conduct human evaluations where participants assign a binary score reflecting whether a plan is *correct* or *incorrect*. For a fairer representation of correctness, we account for executability constraints (i.e., precondition errors) by presenting human evaluators the plans up to the step where they remain executable by the agent for all methods (including baselines). Additionally, we report **Fleiss' Kappa** for *% Correct* inter-annotator agreement among participants in a categorical labeling task for our human annotations. This ranges from 0 to 1. Higher values indicate a stronger agreement between annotators [27].

Longest Common Subsequence (LCS) measures raw string overlap between generated grounded programs and the ground-truth programs as proposed by Puig et al. [9]. LCS serves as a proxy for correctness as human evaluations more robustly measure plan semantics, i.e., human evaluations are not constrained by the richness of interactions in the embodied environment and variability of approaches to complete a task. We also report the average number of **Steps** and **Corrections** across tasks, which assess the total number of steps and corrective re-prompts/re-samples, respectively, needed to generate a plan. While these metrics are incidental to the goal (i.e. minimizing these metrics does not necessarily correlate to improved performance), they assess the relative efficiency of each prompting/sampling ablation towards correcting skill execution. Finally, **Scene-Graph Similarity (%GS)** reflects the percentage of state-object attributes that match between the final states resulting from execution of the generated grounded program ($\mathcal{G}_{gen}$) and the ground-truth human-written program ($\mathcal{G}_{gt}$). The number of matching attributes are normalized over the union of objects in both $\mathcal{G}_{gen}$ and $\mathcal{G}_{gt}$. This metric is invariant to differences in length and ordering of steps between generated and ground-truth plans, compared to a string-matching metric like LCS.

V. Discussion

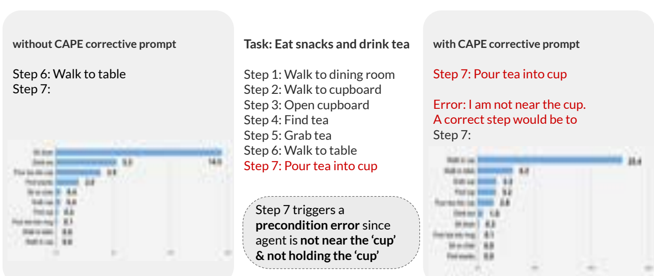

Fig. 4. A qualitative example highlighting the impact of CAPE's corrective prompt on the Planning LLM's assigned probability distribution. CAPE's corrective prompt shifts the distributions towards actions that resolve preconditions and achieve the task objective

In VirtualHome [9], CAPE generates plans that outperform competing methods (Table I). Our method **CAPE: Few-Shot with Explicit Cause** ($\mathcal{F}_E$ + $\mathcal{S}_g$) attains the highest combined performance for plan % *Correct* (49.63%) and *Executability* (96.29%). For % Correct, our method improves on SayCan (Perfect) by 71.80% (absolute improvement of 20.74%) while maintaining comparable executability and percentage of afforded steps, even though SayCan operates in an oracle setting with 0% affordance misclassification. For all methods in Virtual Home experiments, the Fleiss' Kappa indicates moderate inter-annotator agreement for the % Correct metric. Our zero-shot ablations with varying specificity of error information outperform the SayCan and Huang et al. [3] baselines as well as other baselines (Singh et al. [28]) that report 90% executability and 72% graph similarity on 77 Virtual Home tasks using 3 in-context examples with `davinci-codex` model. This demonstrates the effectiveness of our method even without few-shot learning. The results also show that increasing the specificity of error information improves the performance of CAPE. Our method's plans are also higher quality while requiring fewer *Corrections* than the Re-Sampling baseline, which indicates the added utility of corrective actions from precondition error information. Our method also outperforms SayCan across nearly all metrics, even though SayCan implicitly assumes additional environment feedback in the form of a trained affordance model. Furthermore, our method significantly reduces time complexity over SayCan, $O(n)$ compared with $O(|s|^n)$ respectively, where s is the skill repertoire and n the number of plan steps, since SayCan iterates the entire skill space before generating every step.

We present the results of the robot demonstration in Table-II. Our method **CAPE: Few-Shot with Explicit Cause** ($\mathcal{F}_E$ + $\mathcal{S}_g$) attains the highest *Executability* (100%) due to re-prompting with precondition errors. Our method also shows improvement in combined performance for plan % *Correct* (50%) and *Executability* (96.29%). For % Correct, our method improves upon SayCan (Perfect) by 76.49% (absolute improvement of 21.67%) while attaining comparable percentage of afforded steps, even though SayCan operates in an oracle setting and is guaranteed to produce executable skills. SayCan usually fails because the affordance function "funnels" (severely limits) the available actions, sometimes leading plans into local optima i.e. afforded actions with highest log-probability do not resolve precondition errors that are critical to task completion and afforded actions that do resolve these precondition do not have sufficient log-probability. For all methods, the Fleiss' Kappa indicates modest inter-annotator agreement between annotators for the % Correct metric, except for Re-Sampling where annotators unanimously agree that the generated plans do not successfully complete the task.

upto 2x more resolved/unresolved | upto 3x more resolved/unresolved | upto 5x more resolved/unresolved | upto 10x more resolved/unresolved | upto 20x more resolved/unresolved | >20x more resolved/unresolved

	D1: Difficulty 1 [7]	D2: Difficulty 2 [7]	D3: Difficulty 3 [4]	D4: Difficulty 4 [4]	Total
Resampling	13% (1)	32% (5)	53% (3)	2% (2)	202
	14% (1)	23% (6)	61% (4)	1% (2)	5,168
Success Only	15% (2)	19% (3)	64% (3)	2% (3)	191
	49% (1)	5% (3)	43% (4)	2% (3)	553
Implicit Cause	14% (3)	19% (3)	65% (3)	1% (1)	217
	22% (1)	7% (2)	68% (2)	1% (2)	436
Explicit Cause	16% (1)	28% (4)	45% (2)	11% (2)	214
	0% (0)	7% (2)	92% (4)	2% (3)	292
Explicit Cause + Sg	7% (2)	24% (5)	62% (2)	7% (3)	356
	3% (3)	4% (1)	91% (3)	2% (3)	558
Few-Shot Explicit Cause	11% (1)	18% (3)	63% (4)	9% (3)	396
	3% (1)	3% (1)	71% (2)	23% (2)	225
Few-Shot Explicit Cause + Sg	10% (4)	17% (5)	70% (4)	3% (3)	474
	15% (2)	2% (2)	72% (2)	11% (3)	797

Fig. 5. The distribution of precondition errors that are resolved (top) and unresolved (bottom) for all reprompting methods, across 4 difficulties. Values in bracket show the no. of error types for a given difficulty

Finally, more explicit CAPE ablations resolve a larger proportion of more difficult precondition errors. Figure 5 shows the distribution of 22 VirtualHome precondition error types across 4 difficulty levels for all CAPE and resampling ablations. Difficulty levels include errors that require: no corrections (D1: e.g., “opening an open door”), one-step corrections (D2), multi-step corrections (D3), and long-term planning with ambiguous resolution (D4: e.g., “too many objects on the table”). More difficult errors require broader historical/environment context to resolve. A precondition error in step i is ‘resolved’ only if the plan progresses to step $\geq i+1$ before the next error. There are 5 observations: (1) majority of resolved/unresolved errors in all ablations fall under D3; (2) $\mathcal{F}_E$ is the only ablation with more resolved (396) than unresolved (225) errors and an average of $4\times$ more resolutions across difficulties; (3) re-sampling has $25\times$ more unresolved errors with a minimum of $20\times$ more non-resolutions across difficulties; (4) increased error specificity can more readily resolve D1–3 errors, with sharpest increase for D2 errors; (5) whilst $\mathcal{S}_g$ disproportionately increases total number of unresolved errors, diluting the ratio of resolved errors, $\mathcal{S}_g$ also maintains the *proportion* of unresolved errors in each difficulty and broadens the diversity of resolved errors compared to unresolved errors.

As shown in Figure 4, CAPE’s corrective prompts shift the Planning LLM’s assigned probability distribution towards natural-language tokens (that ground to actions) that resolve precondition errors. Not only is the assigned probability distribution shifted, but higher relative weightage is assigned to actions that resolve precondition errors compared to those that do not – due to CAPE’s corrective prompts.

VI. Related Work

Large Language Models for Task Planning: Works that are significantly related to our paper are Huang et al. [3], SayCan [2], and Gramopadhye and Szafir [29], which integrate LLMs into an open-loop planning pipeline. Huang et al. [3] use a prompting strategy to derive step-by-step plans that achieve the goal presented in a prompt. Our work extends their approach by incorporating feedback from the environment as an auxiliary input to improve the executability of derived plans.Gramopadhye and Szafir [29] also improves upon the Huang et al. [3] by providing environmental context to the LLM to generate contextually suitable plans.

Ahn et al. [2] introduces SayCan, a LLM-integrated pipeline that proposes a sequence of actions to achieve specific goals grounded to affordance with a predefined set of robot-executable skills (all demonstrated by an expert) using semantic similarity from language prompt. However, these works only implicitly incorporate "feedback" by selecting actions that are visually afforded in the current state. They do not address action failure or failure recovery.

Visual & Language Feedback for Planning: Following our prior work [8], recent works have shown the efficacy of LLM-based autonomous agents in leveraging language feedback for reasoning about errors [30, 31, 32, 33]. Reflexion [32] converts scalar feedback (from heuristic-based evaluators) into structured linguistic feedback with long-term memory to improve decision making via trial-and-error; in contrast, CAPE does not enable multiple trials nor access retrospective feedback to re-plan from initialization. CAPE only utilizes the agent’s current action history and does not assume access to long-term feedback over multiple episodes. Other works such as DoReMi [30], Zhang et al. [31] also assume access to a set of primitive skills but combine VLMs and LLMs to detect action failures by monitoring properties associated with constraints (either from planning domains or proposed by LLM) for the skills being executed. DoReMi [30] focuses on low-level failure recovery and assumes the LLM has direct access to additional information (e.g., the entire skill repertoire, skills’ constraints, task instructions) whilst CAPE provides implicit feedback to the LLM for specific skill preconditions. Zhang et al. [31] also use VLMs to verify action affordances based on preconditions extracted from PDDL and track updated environment state after skill execution, which is provided to the LLM during next step generation. Environment state information is stored external to the agent in CAPE: the LLM used by CAPE does not directly have access to the underlying state and only receives implicit feedback in the form of re-prompts with which the LLM has to infer the current state and propose an appropriate next step. Additionally, both methods assume VLMs have access to the global visual state during skill execution in order to detect failures, which may not translate naturally to the environments and embodiment types we study, i.e., simulated and real-world agents that have partial observability and use egocentric image feedback. REFLECT [33] utilizes multi-modal feedback to extract a hierarchy of events and visually informed scene graphs, which are then used to explain failures during planning. However, assessing object states from visual and auditory feedback requires pre-defining audio labels and object state labels for visual/audio grounding, also requiring a non-trivial amount of extra effort in addition to pre-defining all skills.

Task and Motion Planning: In task and motion planning (TAMP), robot planning and execution processes are decoupled in a hierarchical manner [34, 20]. This involves the integration of *task planning*, which aims to find a sequence of actions that realize state transitions and goal state corresponding to a high-level problem [35], and *motion planning*, which aims to find physically consistent and collision-free trajectories that realize the objectives of a task plan [36, 37]. Instead of 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, allowing it to interpret a task and identify key details (such as objects or actions) that are related to the problem at hand.

Commonsense Knowledge in LLMs: Other works explore the degree to which LLMs contain commonsense world knowledge. The Winograd Schema Challenge [38] and WinoGrande benchmark [39] evaluate commonsense reasoning in word problems. The Winoground dataset [40] investigates commonsense reasoning in a related image caption disambiguation challenge. LLMs have improved upon baseline methods for this task [10] indicating that language model scale contributes to commonsense reasoning performance. Our system supports the finding that LLMs contain latent commonsense world knowledge sufficient to improve plan executability given precondition errors.

VII. Conclusion

We propose CAPE, a re-prompting strategy for LLM-based planners, which injects contextual information in the form of precondition errors, parsed from environment feedback, which substantially improves the executability and correctness of LLM-generated plans and enables agents to resolve action failure. Our experiments in VirtualHome [9] and on the robot demonstration show that corrective prompting results in more semantically correct plans with fewer precondition errors than those generated by baseline LLM-planning frameworks (Huang et al. [3] and SayCan [2]) and re-sampling. CAPE overcomes the computational intractability of applying SayCan to environments with large numbers of agent skills. CAPE enables more executable and correct plans in less time, while exploring a narrower subset of the skills and using far fewer interjections.

A. Limitations

CAPE achieves strong competitive performance over baseline methods by leveraging a minimal but efficient architecture while only receiving implicit uni-modal (linguistic) feedback from the environment. However, we acknowledge several limitations of CAPE:

Relaxing precondition assumption: CAPE can be more flexible by restricting the assumption that precondition propositions with language feedback are known. Incorporating methods to automatically ground preconditions to binary questions (like Zhang et al. [31]) could allow CAPE to automatically detect or predict the cause of skill failures using additional prompts; furthermore, utilizing LLMs to generate preconditions for future actions (e.g., deriving grounded constraints using methods like the constraint generation module in DoReMi [30]) could allow CAPE to scale efficiently to larger action spaces and define parametrized dependencies or constraints for skills that are not manually defined.

Open-Query Error Handling: Methods like REFLECT [33] have shown that grounding feedback from multiple modalities enables LLMs to reason about causes of skill failure. This approach leverages a multi-modal which could allow CAPE to verify action affordances and generate prompts in an open-query style for a wider range of error types than the ones specified by the precondition definition. Multi-modal feedback can even be used upon successful skill execution to allow CAPE to update an internal structured representation of the current environment state, which can be used to determine the affordance of future actions without having to encode all environment state transitions.

Correcting Perception & Low-level Control: To control the influence of low-level skill (perception, joint manipulation, end effector) errors, CAPE abstracts low-level control into a repertoire of high-level skills that we assume execute perfectly for the purpose of high-level planning. The same abstraction is applied to baselines as well, such that only logical pre-condition errors (the focus of our work) can disrupt plan execution. Several works (SayCan [2], NLMap-SayCan [26], Huang et al. [3] and Inner Monologue [4]) make similar assumptions on high-level skills, though integrating failure detection and recovery for low-level control (DoReMi [30]) could enable CAPE to more robustly recover from additional failure-types.

Acknowledgements

This work is supported by ONR under grant award numbers N00014-21-1-2584 and N00014-22-1-2592, NSF under award number CNS-2038897, and with support from Echo Labs. Additionally, this material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001122C0007. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

References

[1] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, "Robots That Use Language ," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 25–55, 2020.

[2] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances," in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, 14–18 Dec 2022, pp. 287–318.

[3] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.

[4] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, T. Jackson, N. Brown, L. Luu, S. Levine, K. Hausman, and b. ichter, "Inner Monologue: Embodied Reasoning through Planning with Language Models," in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, 14–18 Dec 2022, pp. 1769–1782.

[5] S. Yao, R. Rao, M. Hausknecht, and K. Narasimhan, "Keep CALM and explore: Language models for action generation in text-based games," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Nov. 2020, pp. 8736–8754.

[6] I. Singh, G. Singh, and A. Modi, "Pre-trained Language Models as Prior Knowledge for Playing Text-based Games," in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1729–1731.

[7] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL – The Planning Domain Definition Language," CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Tech. Rep., 1998.

[8] S. S. Raman, V. Cohen, E. Rosen, I. Idrees, D. Paulius, and S. Tellex, "Planning With Large Language Models Via Corrective Re-Prompting," in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. [Online]. Available: https://openreview.net/forum?id=cMDMRBe1TKs

[9] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba, "VirtualHome: Simulating Household Activities via Programs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8494–8502.

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language Models are Few-Shot Learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[11] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[13] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 2227–2237.

[14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[15] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen, "What Makes Good In-Context Examples for GPT-3?" in *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. Association for Computational Linguistics, May 2022, pp. 100–114.

[16] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," *arXiv preprint arXiv:1907.11692*, 2019.

[17] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[18] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[19] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[20] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 265–293, 2021.

[21] F. Golnaraghi and B. C. Kuo, *Automatic Control Systems*. McGraw-Hill Education, 2017.

[22] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, 1977, pp. 46–57.

[23] D. Summers-Stay, C. Bonial, and C. Voss, "What can a generative language model answer about a passage?" in *Proceedings of the 3rd Workshop on Machine Reading for Question Answering*. Association for Computational Linguistics, Nov. 2021, pp. 73–81.

[24] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning Transferable Visual Models From Natural Language Supervision," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8748–8763.

[25] T. Lüddecke and A. Ecker, "Image segmentation using text and image prompts," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 7086–7096.

[26] B. Chen, F. Xia, B. Ichter, K. Rao, K. Gopalakrishnan, M. S. Ryoo, A. Stone, and D. Kappler, "Open-vocabulary Queryable Scene Representations for Real World Planning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 11 509–11 522.

[27] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[28] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 11 523–11 530.

[29] M. Gramopadhye and D. Szafir, "Generating Executable Action Plans with Environmentally-Aware Language Models," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[30] Y. Guo, Y.-J. Wang, L. Zha, Z. Jiang, and J. Chen, "DoReMi: Grounding Language Model by Detecting and Recovering from Plan-Execution Misalignment," *arXiv preprint arXiv:2307.00329*, 2023.
[31] X. Zhang, Y. Ding, S. Amiri, H. Yang, A. Kaminski, C. Esselink, and S. Zhang, "Grounding Classical Task Planners via Vision-Language Models," *ICRA 2023 Workshop on Robot Execution Failures and Failure Management Strategies*, 2023.
[32] N. Shinn, F. Cassano, B. Labash, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *arXiv preprint arXiv:2303.11366*, vol. 14, 2023.
[33] Z. Liu, A. Bahety, and S. Song, "REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction," in *7th Annual Conference on Robot Learning*, 2023.
[34] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.
[35] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.
[36] T. Lozano-Pérez and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.
[37] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel, "Integrating symbolic and geometric planning for mobile manipulation," in *2009 IEEE International Workshop on Safety, Security & Rescue Robotics (SSRR 2009)*. IEEE, 2009, pp. 1–6.
[38] H. J. Levesque, E. Davis, and L. Morgenstern, "The Winograd Schema Challenge," in *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, ser. KR'12. AAAI Press, 2012, pp. 552—561.
[39] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "WinoGrande: An Adversarial Winograd Schema Challenge at Scale," *Commun. ACM*, vol. 64, no. 9, p. 99–106, Sep 2021.
[40] T. Thrush, R. Jiang, M. Bartolo, A. Singh, A. Williams, D. Kiela, and C. Ross, "Winoground: Probing Vision and Language Models for Visio-Linguistic Compositionality," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 5238–5248.

APPENDIX

A. Error Types in VirtualHome

To assess the viability of engineered corrective prompts, we qualitatively analyze and categorize the 22 precondition error types in Virtual Home across 4 categories of increasing difficulty. Since 'corrective prompts' are to generate corrective actions that resolve preconditions, it becomes vital to assess the nature of precondition errors and how they might be corrected for. The errors types and their difficulty classifications are listed in Tables III IV and V below.

B. Causes for execution errors with open-loop planning

Figure 6 highlights the percentage composition of the 192 errors observed when running the Huang et al. [3] baseline (on 700 task scene combinations) using the `davinci-instruct` LLM line.

The "agent is not holding X" error is the most prominent precondition error type, accounting for a majority (51%) of errors. This is followed by the "empty program" (14.6%) and "agent is sitting" (10.9%) precondition errors. The significant presence of "empty program" errors indicates that a rescoring of generated actions (using scoring function $\mathcal{S}_{\}}$ with the Planning LLM) may be needed in addition to corrective prompting. Of the 9 error types observed, the majority lie between difficulty 2 and 3, indicating that correcting most precondition errors for the Huang et al. [3] baseline requires just 1 or up to a few $(2-3)$ corrective actions.

Our qualitative observations find that certain tasks are *intrinsically* more complex, increasing the difficulty of precondition errors and the number of preconditions to satisfy via corrective prompting. For example, the baseline failed on the task "Keep cats inside while door is open" because the `door` was closed in the initial state. There are some qualitative observations that validate this:

- Tasks such as "Entertain" are generally vague, which could impede the LM's ability to generate sensible actions without accessing feedback about the environment's current state.
- Certain tasks enforce implications on the environment's initial state (e.g,. tasks like "push all chairs in" require no actions or precondition resolution since all chairs begin tucked under a table). Therefore, viable plans requires feedback about the initial environment state.
- The Planning LLM generating steps without environment feedback is usually very optimistic, inferring object locations from the task/prompt and assuming objects are readily available. Therefore, without environment feedback, generated plans are not aligned with the constraints of the embodied environment.

C. Hyper-parameter search for LLMs in closed-loop domain

Our approaches (Sections III-A and III-B) form a closed-loop system by providing a Planning LLMs with precondition error information (using environment feedback) to generate corrective actions. It is clear that including additional re-prompts and corrective actions would increase plan length and the frequency of tokens used, however the optimal LLM hyper-parameters for the closed-loop domain are unclear. Thus we perform a hyper-parameter sweep to optimize the Planning LLM's parameter selection, using the re-sampling baseline (see Section III-B) as a proxy for all closed-loop approaches i.e. re-sampling, SayCan and CAPE re-prompting.

Our hyper-parameter sweep is performed across different temperatures and presence penalties in the ranges shown in

Precondition Error	Difficulty Level	Description
X is not movable	1	The `move` action does not apply to object `X`. Difficulty 1 since unmovable objects are governed by environment dynamics (not a planning failure) so the plan can proceed without needing to resolve any preconditions
X cannot be opened	1	The `open` action does not apply to object `X`. Difficulty 1 since unopenable objects are governed by environment dynamics (not a planning failure) so the plan can proceed without needing to resolve any preconditions
X is not cuttable	1	The `cut` action does not apply to object `X`. Difficulty 1 since cuttable objects are governed by environment dynamics (not a planning failure) so the plan can proceed without needing to resolve any preconditions
X is not a receptacle	1	The `put down` action cannot be done onto object `X` since it is not a receptacle. Difficulty 1 since receptacle attribute is governed by environment dynamics (not a planning failure) so the plan can proceed without needing to resolve any preconditions
X is not lookable	1	The `look` action does not apply to object `X`. Difficulty 1 since lookable attribute is governed by environment dynamics (not a planning failure) so the plan can proceed without needing to resolve any preconditions
agent is already sitting	1	The `sit` action cannot be performed since the agent is already sitting down. Difficulty 1 because the preconditions for the action are already satisfied and no further preconditions need to be resolved
X is sitting	1	The failed action has a precondition requiring the agent to be standing, which is not satisfied. Difficulty 1 because the `stand up` only needs to be executed, which itself has no preconditions i.e. `stand up` is universally afforded execution
X is not lying or sitting	2	The failed action has a precondition requiring the agent to be sitting or lying down, which is not satisfied. Difficulty 2 because either the `sit` or `lie down` actions need to be executed, which themselves at least 1 precondition e.g. `sit` requires the agent to be near a sittable object and not already sitting/lying down
X is turned off / closed twice	2	The `turn on` or `open` action cannot be performed since object `X` is already turned on (or open). Difficulty 2 because preconditions for the action are already satisfied but the 'corrective action' (`turn off` or `close`) is not universally executable
X is turned on / opened twice	2	The `turn off` or `close` action cannot be performed since object `X` is already turned off (or closed). Difficulty 2 because preconditions for the action are already satisfied but the 'corrective action' (`turn on` or `open`) is not universally executable

TABLE III

DESCRIPTION OF ERROR TYPES OBSERVED IN THE VIRTUALHOME ENVIRONMENT.

Precondition Error	Difficulty Level	Description
X is not closed	2	The failed action has a precondition requiring object `X` to be closed, which is not satisfied. Difficulty 2 because a single action `close X` needs to be executed to resolve the precondition error, which has at least 1 precondition e.g. agent needs to be near object `X` object `X` must be open
X is not open (or not openable)	2	The failed action has a precondition requiring object `X` to be open, which is not satisfied. Difficulty 2 because a single action `open X` needs to be executed to resolve the precondition error, which has at least 1 precondition e.g. agent needs to be near object `X` and object `X` must be closed
agent is not facing X	2	The failed action has a precondition requiring agent to turn towards object `X`, which is not satisfied. Difficulty 2 because a single action `turn to X` needs to be executed to resolve the precondition error, which has at least 1 preconditions e.g. agent needs to be near object `X`
X is not grabbed	3	The failed action has a precondition requiring agent to be holding object `X`, which is not satisfied. Difficulty 3 because a multiple actions need to potentially be executed to resolve the precondition error (`walk to X`, `put down Y` - object being held currently, `grab X`) i.e. the agent needs to be near object `X` and have at least 1 free hand
agent is not holding X	3	The failed action has a precondition requiring agent to be holding object `X`, which is not satisfied. Difficulty 3 because a multiple actions need to potentially be executed to resolve the precondition error (`walk to X`, `put down Y` - object being held currently, `grab X`) i.e. the agent needs to be near object `X` and have at least 1 free hand
agent is not holding anything	3	The failed action has a precondition requiring agent to be holding a (specific) object, but the agent is not holding anything. Difficulty 3 because precondition resolution requires multiple corrective actions e.g. inferring what object to grab, `walk to` object, `grab` object
agent is not close to X	3	The failed action has a precondition requiring the agent to be near object `X`, which is not satisfied. Difficulty 3 because precondition resolution requires multiple corrective actions e.g. `walk to X`, `open door` in case door in path is closed
X is inside another closed object	4	The failed action has a precondition requiring object `X` to be accessible (removed) from within another closed object, which is not satisfied. Difficulty 4 because precondition resolution requires long-term planning with corrective actions that themselves have at least 1 precondition e.g. inferring the closed object, `open Y` the object that is closed, `grab X` to remove from within object `Y`

TABLE IV

DESCRIPTION OF ERROR TYPES OBSERVED IN THE VIRTUALHOME ENVIRONMENT.

Table VI below. These parameters, respectively, influence how out of distribution a proposed action might be and penalize the repetition of previously generated actions (topics), making them strong levers for Planning LLM performance in the closed-loop domain.

The columns in Table VI represent presence penalties

Precondition Error	Difficulty Level	Description
too many things on X	4	The failed action has a precondition requiring the receptacle X to be empty or below maximum capacity, which is not satisfied. Difficulty 4 because precondition resolution requires contextualization and long-term planning with corrective actions that themselves have at least 1 precondition e.g. inferring objects consuming capacity on object X, grabbing said objects, walking to another empty receptacle, placing said objects on receptacles
agent does not have a free hand	4	The failed action has a precondition requiring 1 of the agent's hand to be free i.e. not holding or grabbing an object, which is not satisfied. Difficulty 4 because precondition resolution requires contextualization and long-term planning with corrective actions that themselves have at least 1 precondition e.g. inferring the objects in the agent's hands, walking to another empty receptacle, placing objects on empty receptacles.
door between room X and room Y is closed	4	The failed action has a precondition requiring the door connecting rooms X and Y to be open, which is not satisfied. Difficulty 4 because precondition resolution requires contextualization with corrective actions that themselves have at least 1 precondition e.g. inferring which door instance is closed, walking to door, opening door

TABLE V

DESCRIPTION OF ERROR TYPES OBSERVED IN THE VIRTUALHOME ENVIRONMENT.

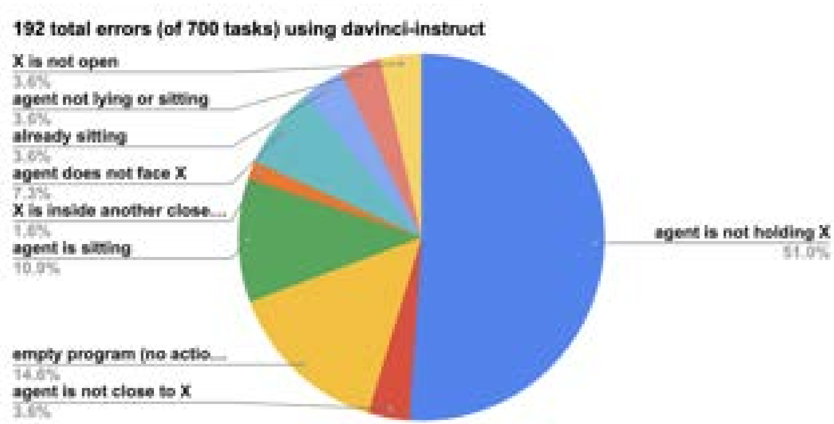

Fig. 6. Decomposition of precondition errors into error types when running Huang et al. [3] baseline on `davinci-instruct` LLM

TABLE VI

THE RANGE OF HYPER-PARAMETER (TEMPERATURE AND PRESENCE PENALTY) VALUES EXPLORED AS PART OF PARAMETER SWEEP

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

(PresPen) and the rows represent temperature (Temp). Each table element evaluates a set of metrics from top to bottom for the particular temperature-presence penalty combination: executability, % executed, graph similarity, LCS-EP, number of steps and the number of corrections (as from Section IV-D).

Presence Penalty does not seem to influence executability at higher temperatures; for lower temperatures, executability decreases with increased presence penalties. The number of corrective steps and total steps appear to be independent of presence penalty as well.

Temperature and executability seem to have an 'optimal temperature' that maximizes executability, around which (for higher or lower temperatue) executability is lower, for all presence penalties tested. A similar correlation is seen

	PresPen@0.3	PresPen@0.5	PresPen@0.7
Temp@0.3	75.03%	74.43%	74.15%
	73.81%	73.24%	72.96%
	22.35	22.42	22.44
	6.37	6.33	6.37
	7.60	7.59	7.62
Temp@0.5	75.72%	76.43%	**77.15%**
	74.44%	75.24%	**75.86%**
	23.28	**23.45**	23.38
	6.90	6.87	6.93
	7.87	7.67	7.79
Temp@0.7	75.23%	75.29%	75.15%
	74.04%	74.20%	73.99%
	23.53	22.69	22.68
	7.23	7.21	7.18
	8.831	8.78	8.66

TABLE VII

HYPER-PARAMETER SEARCH PERFORMANCE (EXECUTABILITY, % AFFORDED, LCS, NO. STEPS AND NO. CORRECTIONS) OVER DIFFERENT TEMPERATURES (TEMP) AND PRESENCE PENALTIES (PRESPEN) USING THE `instruct-davinci` LLM.

between LCS and temperature as well as affordance and temperature with peak LCS/executability at a temperature value of 0.5. Additionally, the number of corrective steps and total steps seem to both monotonically increase with temperature.

Overall, as per Table VII, the maximum executability and % affordance is observed using presence penalty and temperature of 0.7 and 0.5. Maximum LCS, however, is observed using presence penalty 0.5 and temperature of 0.5. Since affordance filtering (SayCan baseline) and corrective prompts would naturally induce higher executability and affordance, we deem LCS a more challenging metric for improved performance. Thus we use 0.5 presence penalty and temperate for our zero-shot re-prompting, few-shot re-prompting, re-sampling baseline and SayCan baseline.

D. *Qualitative Examples of Generated Plans*

We present the generated plans for 10 sample tasks comparing the Huang et al. [3] baseline with the re-sampling baseline, CAPE's zero-shot and few-shot ablations.

We find qualitative improvements in the planner's corrective capabilities (in resolving precondition errors) as well as improved specificity and alignment with the task's objective.

Task	**Baseline Method** [3]	**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
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

TABLE VIII

QUALITATIVE EXAMPLES OF GENERATED PLANS FOR THREE CASES: 1) THE BASELINE METHOD [3], 2) OUR RE-SAMPLING METHOD, AND 3) CAPE'S REPROMPTING ABLATIONS

Task	**Baseline Method** Huang et al. [3]	**Re-Sampling**	**Re-Prompting**
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 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 Error: I am not near the cup. A correct step would be to Step 7: Walk to cup Step 8: Find cup Step 9: Grab cup Step 10: Pour tea into cup Step 11: Drink tea ...
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

TABLE IX

QUALITATIVE EXAMPLES OF GENERATED PLANS FOR THREE CASES: 1) THE BASELINE METHOD [3], 2) OUR RE-SAMPLING METHOD, AND 3) CAPE'S REPROMPTING ABLATIONS

Task	Baseline Method [3]	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

TABLE X

QUALITATIVE EXAMPLES OF GENERATED PLANS FOR THREE CASES: 1) THE [3] BASELINE, 2) OUR RE-SAMPLING BASELINE, AND 3) CAPE'S REPROMPTING ABLATION.