CAPE: Corrective Actions from Precondition Errors using Large Language Models

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

Abstract— **Extracting knowledge and reasoning from large language models (LLMs) offers a path to designing intelligent robots. Common approaches that leverage LLMs for planning are unable to recover when actions fail and resort to retrying failed actions without resolving the underlying cause. We propose a novel approach (CAPE) that generates corrective actions to resolve precondition errors during planning. CAPE improves the quality of generated plans through few-shot reasoning on action preconditions. Our approach enables embodied agents to execute more tasks than baseline methods while maintaining semantic correctness and minimizing re-prompting. In VirtualHome, CAPE improves a human-annotated plan correctness metric from 28.89% to 49.63% over SayCan, whilst achieving competitive executability. Our improvements transfer to a Boston Dynamics Spot robot initialized with a set of skills (specified in language) and associated preconditions, where CAPE improves correctness by 76.49% with higher executability compared to SayCan. Our approach enables embodied agents to follow natural language commands and robustly recover from failures.**

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 a large language model (LLM) 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 executable actions [2].

We use *precondition errors* to resolve action failure, 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 to afford their execution. Our method targets the failure mode of executing skills without satisfying their preconditions in this setting. An LLM generates/translates natural language into parametrized skills, producing a sequence of actions for execution towards completing a task. When a robot or agent fails to execute an action due to precondition violations, CAPE (*Corrective Actions from Precondition Errors*) adopts a templated-prompting strategy to query the LLM for corrective actions (Figure 1). Our prompts either specify 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 environment. This paper builds on our previous work [8] with more rigorous analysis, scaled human evaluation, additional (more competitive) baselines, and experiments in both simulation and real-world settings.

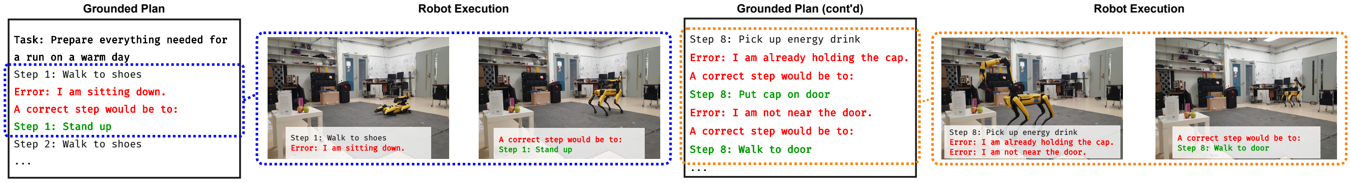
Our contributions are as follows: we introduce CAPE as a novel approach for LLM planning that can recover from failures by proposing corrective actions, using prompts based on precondition errors. We detail how different ablations of our re-prompting strategy can be deployed on embodied systems with large and small skill repertoires. We also evaluate against several baselines [3, 2] to show that CAPE achieves near-perfect 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 LLM capable of few-shot learning novel tasks (including Q&A, arithmetic, and comprehension) by prompting the LLM with in-context task examples used for structural and syntactic guidance. This approach offers much greater sample efficiency and task generalization over learning tasks with fine-tuned pre-trained latent language representations [11, 12, 13] and zero-shot inference [14]. In-

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.



```
Task: Prepare everything needed for
a run on a warm day
Step 1: Walk to shoes
Error: I am sitting down.
A correct step would be to:
Step 1: Stand up
Step 2: Walk to shoes
...
```

```
Step 1: Walk to shoes
Error: I am sitting down.
```

```
A correct step would be to:
Step 1: Stand up
```

```
Step 8: Pick up energy drink
Error: I am already holding the cap.
A correct step would be to:
Step 8: Put cap on door
Error: I am not near the door.
A correct step would be to:
Step 8: Walk to door
...
```

```
Step 8: Pick up energy drink
Error: I am already holding the cap.
Error: I am not near the door.
```

```
A correct step would be to:
Step 8: Walk to door
```

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).

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 plans for a task zero-shot without environment feedback. Given a query task $\mathcal{Q}$, first an example task $\mathcal{T}$ and its plan are chosen from a *demonstration set* as a contextual example for the *Planning LLM*; $\mathcal{T}$ is selected to maximize cosine similarity with the *query task* $\mathcal{Q}$. The *Planning LLM* auto-regressively generates actions for task $\mathcal{Q}$ in free-form language (a_l) via in-context learning. The *Translation LLM* then utilizes a BERT LM (Sentence-BERT [16]) to embed free-form actions (a_l) to the most semantically similar (i.e., cosine-similar) admissible action from the agent's repertoire (a_e). Here, an admissible action refers to the language description of the action. 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 plan in a closed-loop setting by leveraging precondition error feedback as an auxiliary mode 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 robots have access to a library of predefined skills. Affordance models factorize states into *preconditions*, where affordance is defined 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 fails skill execution, CAPE injects precondition errors into a prompt for plan repair.

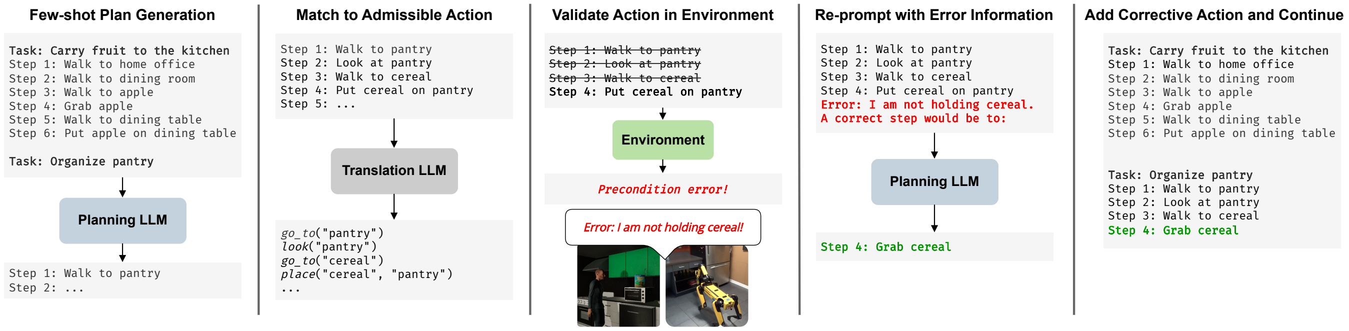
A. Plan Generation via Re-prompting

In control theory, a closed-loop system relies on feedback from its outputs for adaptive control [21]. Similarly, when the LLM proposes an action not afforded execution (output), CAPE closes the loop by injecting the corresponding precondition error (feedback) into a *corrective prompt* (see Figure 2), allowing the LLM to adaptively correct the generated plan. Certain errors require more context about the agent's state, action history and environment. For instance, correcting the VirtualHome [9] error `<character> (1) does not have a free hand when executing "[GRAB] <obj> (1) [1]"` requires knowledge of objects previously / currently grabbed by the agent and available adjacent receptacles on which held objects can be dropped, to free the agent's hands. Our corrective prompts are composed with the following segments of feedback:

- **Contextual Information**: includes relevant context and action history upon action failure, i.e., query task $\mathcal{Q}$ and query steps up to the failed action.
- **Precondition Error Information**: optionally includes details on violated preconditions, tailored to the degree to which the agent can assess precondition violations.

For the Translation LLM to ground language steps, we must assume the agent is equipped with a repertoire of skills admissible to the environment. Preconditions only need to be defined for each parametrized skill. More importantly, the Planning LLM used by CAPE has no explicit knowledge of the agent's skills nor their preconditions (i.e., logical propositions assessing a skill's affordance) during re-prompting. Instead, we evaluate the current environment state using preconditions (logical propositions) defined for each parametrized skill to identify precondition violations or errors. Although the environment state and preconditions are external to the LLM, linguistic error information can be integrated into the corrective prompt. As a result, the Planning LLM has to *infer* the cause of failures and environment mechanics over a wide layer of abstraction (based only on the context from corrective prompts and the agent's action history) before proposing an appropriate corrective action. CAPE can be applied to planning domains where skills are represented using preconditions and effects described using symbolic propositions (e.g., PDDL [7], STRIPS [17], or LTL [22]). Since preconditions are well-defined in these representations, minimal effort is required to integrate appropriate language feedback for precondition violations.

Re-prompting Strategies: CAPE's re-prompting ablations provide varying degrees of precondition error detail in both


Few-shot Plan Generation
Match to Admissible Action
Validate Action in Environment
Re-prompt with Error Information
Add Corrective Action and Continue
Planning LLM
Translation LLM
Environment
Planning LLM
Error: I am not holding cereal!

```
Task: Carry fruit to the kitchen
Step 1: Walk to home office
Step 2: Walk to dining room
Step 3: Walk to apple
Step 4: Grab apple
Step 5: Walk to dining table
Step 6: Put apple on dining table

Task: Organize pantry
```

```
Step 1: Walk to pantry
Step 2: ...
```

```
Step 1: Walk to pantry
Step 2: Look at pantry
Step 3: Walk to cereal
Step 4: Put cereal on pantry
Step 5: ...
```

```
go_to("pantry")
look("pantry")
go_to("cereal")
place("cereal", "pantry")
...
```

~~Step 1: Walk to pantry~~
~~Step 2: Look at pantry~~
~~Step 3: Walk to cereal~~
Step 4: Put cereal on pantry

Precondition error!

```
Step 1: Walk to pantry
Step 2: Look at pantry
Step 3: Walk to cereal
Step 4: Put cereal on pantry
Error: I am not holding cereal.
A correct step would be to:
```

```
Step 4: Grab cereal
```

```
Task: Carry fruit to the kitchen
Step 1: Walk to home office
Step 2: Walk to dining room
Step 3: Walk to apple
Step 4: Grab apple
Step 5: Walk to dining table
Step 6: Put apple on dining table


Task: Organize pantry
Step 1: Walk to pantry
Step 2: Look at pantry
Step 3: Walk to cereal
Step 4: Grab cereal
```

Fig. 2. Overview of CAPE: To generate executable plans, 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. The Translation LLM [16] grounds this description to an admissible skill in the agent's repertoire. If violating preconditions prevent execution of the proposed skill, precondition error information is formatted into a *corrective prompt*, which is provided with the failed skill to the LLM for corrective action proposal.

zero-shot ($\mathcal{Z}$) and few-shot ($\mathcal{F}$) settings, denoted by $P = \mathcal{Z} \vee \mathcal{F}$. Few-shot ablations ($\mathcal{F}$) inject 3 in-context precondition error and corrective action pairs from the demonstration set. For all ablations, corrective prompts are only injected when the Planning LLM proposes corrective actions. There are 3 degrees of precondition error detail:

- **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 ($\mathcal{Z}_I$):** shares the failed action and object(s) the agent interacted with to the LLM in a prompt template (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 violated precondition for the non-afforded action, in addition to feedback provided by $\mathcal{Z}_I$ (i.e., "`I cannot <action> <object> because <unmet_precondition>`").

Scoring Grounded Actions: We define a scoring function $\mathcal{S}_w$ (Equation 1) as a weighted combination of log probability and cosine similarity, thresholded to determine the feasibility of each proposed grounded step [3]. Log probability is defined as $P_\theta(X_i) := \frac{1}{n_i}\sum_{j=1}^{n_i} \text{log} p_\theta(x_{i,j}|x_{i<j})$, 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's embedding function; a and a_e are the predicted and admissible action pair for which we estimate the distance. β is a weighting coefficient:

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

$\mathcal{S}_w$ prioritizes the quality of natural language at the cost of semantic translation. This often results in mistranslations 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 is also unbounded, which makes finding a score threshold more challenging. 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 $(0, 1)$ score that prioritizes both language generation and semantic translation objectives jointly

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

All CAPE ablations are reported only using $\mathcal{S}_w$. We additionally report $\mathcal{S}_g$ for explicit cause re-prompting (P_E).

B. Baseline: Plan Generation via Re-sampling

The closed-loop re-sampling method does not use error feedback when a grounded action is not executable. Instead, this approach iteratively evaluates the top k admissible actions proposed by the Planning LLM and grounded by the Translation LLM in reverse order of their S_w scores until an executable action is found. If none of the k re-sampled admissible actions are executable, plan generation terminates. Resampling assesses whether CAPE enables more efficient corrections due to the utility of re-prompts rather than benefiting from more attempts at proposing corrective actions.

C. Baseline: Plan Generation with SayCan

For every step generated, SayCan [2] assigns a score to each action in the agent's repertoire and the highest scoring action is executed. This score is the product of the LLM's log probability and affordance for each action. This process is repeated until the termination skill (`done`) is assigned the highest score. We make two important adjustments in our SayCan implementation for VirtualHome [9]:

- It is intractable to evaluate all possible skills in VirtualHome (over 50K admissible object-action pairs) for every step, so we use the Planning LLM to generate a *prototype* step with which we sub-sample the 500 most semantically similar object-action pairs (by cosine similarity) and at most 1000 object-action pairs containing objects in the prototype step, forming a subset of $\leq$ 1500 skills to be scored. This ensures sub-sampled skills have the highest log probability according to the LLM and are contextually relevant to the task.
- We evaluate a *perfect* SayCan, with 0% affordance model misclassification, using VirtualHome's oracle

[1] This is analogous to Inner Monologue's [4] "success detection", which determines whether to re-execute failed actions due to stochasticity of low-level policies. However, our aim is to repair high-level plans with corrective actions that arise from a new distribution of actions given precondition feedback.

precondition checks on the environment state. However, as Ahn et al. [2] cites a minimum planning failure of 16% with 35% of these originate from affordance model errors, we also present a *noisy* ablation of SayCan with a 6% (16% × 35%) random chance of misclassifying the oracle affordance, i.e., false when actually true or true when actually false.

Similar to CAPE, SayCan assumes access to an agent's skill repertoire with language descriptions. SayCan leverages a trained affordance model (value function) to evaluate the executability of skills and can easily be extended to identify or predict language-specified precondition violations.

IV. Evaluation

We test the hypothesis that corrective re-prompting increases executability of plans generated by LLM models while maintaining plan correctness. We focus on OpenAI's `davinci-instruct` model line for their demonstrated capabilities in instruction-following and planning tasks [10, 23]. We evaluate eight approaches in a zero-shot setting: three baselines – Huang et al. [3] (Section II), the closed-loop re-sampling (Section III-B), and SayCan [2] (Section III-C) – and our proposed ablations of CAPE (Section III-A).

A. Experimental Setup

We evaluate CAPE across 7 scenes in VirtualHome [9] and 2 scenes with a Boston Dynamics Spot robot (Figure 1). The contrast in environments aims to demonstrate that corrective re-prompting can resolve unmet preconditions for embodied agents and robots across a variety of tasks; VirtualHome provides a large skillset with many objects, while the robot environments focus on physical embodiment with fewer objects and skills. In VirtualHome, we evaluate plans for 100 household tasks (e.g., *"Make breakfast"*, *"Browse the Internet"*). To show that our method extends to unseen unstructured real-world environments, we compare plans generated by CAPE with those generated by the 3 baselines across 6 human-assistance tasks and 2 scenes for each task.

B. Robot Demonstration

To demonstrate CAPE's performance on unstructured real-world tasks, we compare our re-prompting ablations against all 3 baselines with the Boston Dynamics Spot (a quadruped robot with a single 6-DOF arm) across 2 scenes (a lab environment and a kitchen) with structural variation in the maps and object inter-placement. On average, 9 household objects (e.g., phone, bed, coffee, etc.) each with five state attributes (e.g, *at location*, *grabbed*, *opened*, *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 robot has access to a set of 14 parametrized skills (including *stand up*, *walk to*, *pick up*, *put*, *touch*, *look at*, *open* and *close*). We first build a semantic map from images and waypoints recorded in each scene; vision-language models (VLM) (CLIP [24] and CLIPSeg [25]) are used to ground admissible skills to spatial targets for navigation or grasping in the physical environment, similar to NLMap-SayCan [26]. The robot's embodiment (single arm), a limited skill repertoire and extensibility to new unstructured environments make this a challenging setting for task completion. Figure 1 shows Spot successfully completing the task *"Prepare for a run on a warm day."* Re-prompting resolves precondition violations caused by the robot's initial state and single-arm embodiment. Demonstrations for additional tasks and scenes are in our supplementary video.

C. Human Evaluation

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

D. Evaluation Metrics

We adopt % Executability and % Correctness from Huang et al. [3]. **% Executability** measures if *all* grounded actions satisfy preconditions imposed by the environment, i.e., if the *entire* plan is afforded execution in the agent's environment and state **% Affordability** measures the average percentage of all plan steps in order that are executable, skipping steps that are 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 the quality of plans in natural language using only executability is difficult and ambiguous; thus, we conduct human evaluations where participants assign binary scores reflecting whether a plan is *correct* (1) or *incorrect* (0). For a fairer representation of correctness, we consider executability constraints (i.e., precondition errors) and present all plans to human evaluators up to the step where they remain executable by the agent. We also report **Fleiss' Kappa** for inter-annotator agreement among human annotators of our *% Correct* categorical labeling task. The score ranges from 0 to 1, where higher values indicate a stronger agreement between annotators [27]. **Longest Common Subsequence (LCS)** measures raw string overlap between generated grounded programs and ground-truth programs as proposed by Puig et al. [9]. LCS serves as a less robust proxy for plan semantics as human evaluations (correctness) 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** (number of corrective re-prompts or re-samples) needed across tasks to generate a plan. Whilst these metrics are incidental to the goal (i.e., minimizing them does not necessarily correlate to improved performance), they still assess the relative efficiency of each method towards correcting skill execution. Finally, **Scene Graph Similarity (%GS)** reflects the percentage of state-object attributes that overlap between the final states resulting from execution

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

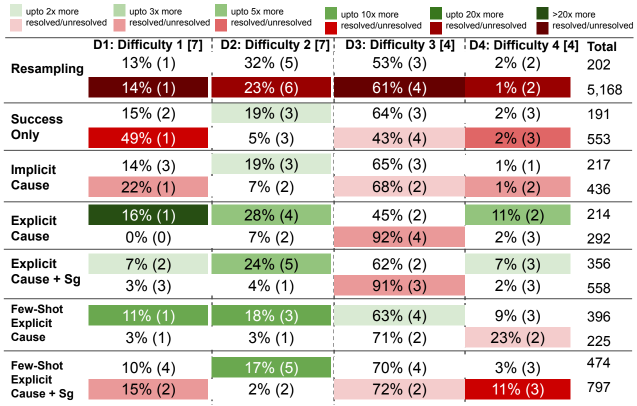
of the generated grounded program ($\mathcal{G}_{gen}$) and the ground-truth human-written program ($\mathcal{G}_{gt}$). The number of matching attributes are normalized over the union of objects in both $\mathcal{G}_{gen}$ and $\mathcal{G}_{gt}$. This metric is invariant to differences in length and ordering of steps between generated and ground-truth plans, compared to a string-matching metric like LCS.

V. DISCUSSION

CAPE ablations outperform baselines on most metrics in VirtualHome [9] environment (see Table I). **CAPE: Few-Shot with Explicit Cause** ($\mathcal{F}_E$ + $\mathcal{S}_g$) attains the highest combined performance for *% Correct* (49.63%) and *Executability* (96.29%). This ablation improves % Correct over SayCan (Perfect) by 71.80% (absolute improvement of 20.74%) while maintaining comparable executability and percentage of afforded steps. The Fleiss' Kappa across all ablations indicates moderate inter-annotator agreement for the % Correct metric. Our zero-shot ablations with varying specificity of error information outperform SayCan and Huang et al. [3] baselines as well as other baselines [28] that report 90% executability and 72% graph similarity using 3 in-context examples with `davinci-codex` model. Thus, CAPE's re-prompting is effective even without few-shot learning; increasing specificity of errors also improves performance. Our method generates higher quality plans while requiring up to 10× fewer *corrections* than the re-sampling baseline, which reflects the utility of corrective prompts. Our method also outperforms SayCan across nearly all metrics, though SayCan implicitly provides environment feedback through the affordance model. Furthermore, our method significantly reduces time complexity below 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 to generate every step.

CAPE ablations also outperform baselines on robot demonstration (see Table II). **CAPE: Few-Shot with Explicit Cause** ($\mathcal{F}_E$ + $\mathcal{S}_g$) attains the highest combined performance for *% Correct* (50%) and *Executability* (100%). This ablation improves % Correct over 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. SayCan usually fails because the affordance function severely constrains available actions, sometimes leading to local optima, i.e., afforded actions with highest log probability do not resolve precondition errors critical to task completion, while afforded actions that do resolve these preconditions do not have sufficient log probability. The Fleiss' Kappa across all ablations for robot demonstration tasks indicate modest inter-annotator agreement between annotators for the % Correct metric, except for re-sampling where annotators unanimously agreed that generated plans do not successfully complete the task.

Finally, Figure 3 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



	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. 3. The distribution of precondition errors that are resolved (top) and unresolved (bottom) for all re-prompting methods on the VirtualHome environment, across 4 difficulty levels (D1–4). Values in square brackets show the number of error types for a given difficulty. Color and intensity correspond to ratio of errors resolved (green) vs. unresolved (red) errors.

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”). A precondition error in step i is ‘resolved’ only if the plan progresses to step $\geq i+1$ before the next error. There are 4 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) increased error specificity can more readily resolve D1–3 errors, with the sharpest increase for D2 errors; (4) whilst $\mathcal{S}_g$ disproportionately increases total number of unresolved errors, diluting the ratio of resolved errors, $\mathcal{S}_g$ does not change the *distribution* of unresolved errors across difficulties and broadens the diversity of resolved errors compared to unresolved errors.

VI. Related Work

LLMs for Task Planning: Several works use LLMs for planning [3, 2, 29, 30]. Huang et al. [3] uses LLMs to generate step-by-step plans for tasks described in language. We additionally incorporate precondition error feedback from the environment as auxiliary input. Similarly, Song et al. [29] and Gramopadhye and Szafir [30] provide information about the agent’s environment and action set to help LLMs generate contextualized plans. However, CAPE assumes less information with similar output: the Planning LLM is not given access to information about what objects lie in the environment nor the agent’s high-level action set. SayCan [2] uses an LLM to score a predefined set of robot skills (demonstrated by an expert), implicitly incorporating feedback through affordance detection to propose action sequences.

Visual & Language Feedback for Planning: Following our prior work [8], several works use language feedback to reason about errors [31, 32, 33, 34]. Reflexion [33] converts scalar feedback (from heuristic evaluators) into structured linguistic feedback with long-term memory to improve decision making via trial-and-error. CAPE does not assume access to long-term feedback over multiple episodes. Other works like DoReMi [31] and Zhang et al. [32] assume access to primitive skills and detect action failures by monitoring properties associated with action constraints (either from planning domains like PDDL or from the LLM) using VLMs and LLMs. CAPE only provides implicit feedback in the form of corrective prompts with which the LLM has to infer the current state and propose an appropriate next step. REFLECT [34] uses multi-modal feedback to extract a hierarchy of events and visually informed scene graphs, which are used to explain failures during planning. However, assessing object states from visual and auditory feedback requires predefined audio and object state labels for visual/audio grounding, which requires a non-trivial extra effort in addition to predefining all skills.

Task and Motion Planning (TAMP): TAMP [35, 20] hierarchically decouples robot planning and execution into *task planning*, which aims to find a sequence of actions that realize state transitions and goal state corresponding to a high-level problem [36], and *motion planning*, which aims to find collision-free trajectories that realize objectives of a task plan [37, 38]. LLMs provide agents or robots with implicit representations of actions in language, without having to rely on explicitly defined structures or symbols that are typically used in TAMP, to identify key details (actions or objects) underpinning the problem at hand.

VII. Conclusion

We introduce CAPE, a re-prompting strategy for LLM-based planners, that injects precondition errors parsed from environment feedback as contextual information. CAPE substantially improves executability and correctness of generated plans, enabling agents to resolve action failures. Our experiments show that corrective prompting generates more semantically correct plans with fewer precondition errors than those generated by baseline methods (Huang et al. [3], SayCan [2], and re-sampling). CAPE also overcomes the computational intractability of SayCan in environments with a large number of admissible skills, as CAPE explores a narrower subset of skills and uses far fewer interjections.

However, we acknowledge important limitations. First, CAPE carries a strong assumption that skill preconditions can be predetermined. We will incorporate methods that propose action preconditions using the innate reasoning capabilities of LLMs [31, 32]. Second, CAPE abstracts low-level control into a repertoire of high-level skills that we assume have perfect low-level execution, such that only logical precondition errors (the focus of our work) can disrupt plan execution. We plan to use multi-modal feedback, which would allow CAPE to reason over a wider range of error types that cannot be described in language and enable recovery for low-level control as in related work [31, 34].

Acknowledgements

This work is supported by ONR grants N00014-21-1-2584 and N00014-22-1-2592, NSF award CNS-2038897 and Echo Labs, and is based on work supported by the Defense Advanced Research Projects Agency (DARPA) under contract HR001122C0007.

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] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, “LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] Z. Liu, A. Bahety, and S. Song, “REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction,” in *7th Annual Conference on Robot Learning*, 2023.

[35] L. P. Kaelbling and T. Lozano-Pérez, “Hierarchical planning in the now,” in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[36] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.

[37] 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.

[38] 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.