CAPE: Corrective Actions from Precondition Errors using Large Language Models

Shreyas Sundara Raman[1*], Vanya Cohen[2], David Paulius[1], Ifrah Idrees[1], Eric Rosen[1], Ray Mooney[2], Stefanie Tellex[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, reasoning about the state changes resulting from the skill's 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].

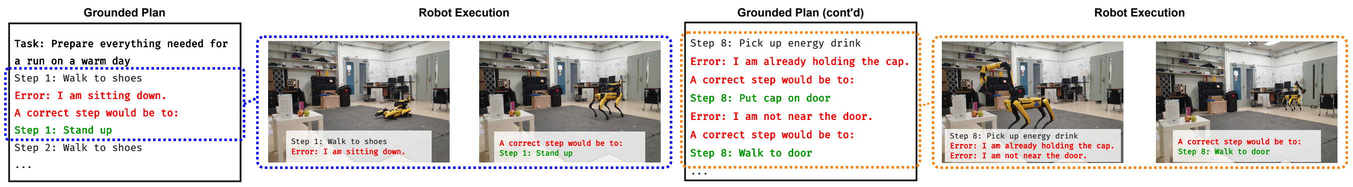
Following our previous work [7], we use *precondition errors* to resolve action failure, which is motivated by the vast body of research on planning algorithms and definitions like PDDL [8]. In these settings, robots are equipped with a repertoire of skills, each requiring certain *preconditions* to be satisfied in order to afford their execution. A key failure mode in this setting is executing skills without satisfying their preconditions. 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 [7] 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-sho learning of novel tasks, including Q&A, arithmetic, translation, and comprehension by prompting the LLM with

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).

in-context task examples used for structural and syntactic guidance. This 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 the 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 [8] 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

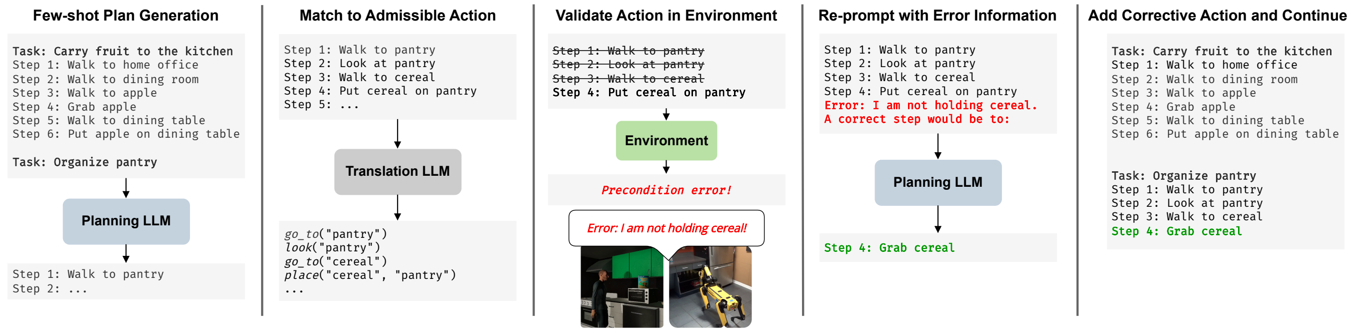
Given a task specified in natural language, we use LLMs for task planning. When an agent or robot fails skill execution, CAPE integrates precondition errors into a prompt that aims to repair plans.

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 object 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


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
Precondition error!
Error: I am not holding cereal!
Planning LLM

```
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
```

```
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 an executable plan, we select an in-context example task 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.

error information produced by them can then be integrated into a corrective language prompt. As a result, there is a 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 [8], 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 $P = \mathcal{Z} \vee \mathcal{F}$. This allows varying agent access to precondition error information. Re-prompting strategies can be categorized as follows:

- **Re-prompting with Success Only ($\mathcal{Z}_S$):** solely informs the LLM that the current 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]).

[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.

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:

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

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:

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

We report results using $\mathcal{S}_w$ with all re-prompting methods and using $\mathcal{S}_g$ with 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

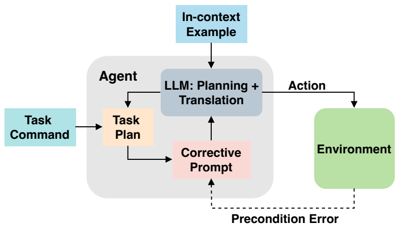


Fig. 3. CAPE uses a LLM to generate plans for tasks specified in natural language. When the agent fails to execute a step, we re-prompt the LLM with error information, utilizing the latent commonsense reasoning and few-shot learning capabilities of the LLMs to overcome execution errors.

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. Therea 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 sub-sample 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.

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

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

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

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. The zero-shot ablations of our method with varying specificity of error information outperform the SayCan and Huang et al. [3] baselines demonstrating 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.

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 [28], 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 [28] 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 [7], recent works have shown the efficacy of LLM-based autonomous agents in leveraging language feedback for reasoning about errors [29, 30, 31, 32]. Reflexion [31] 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 [29], Zhang et al. [30] 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 [29] 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. [30] 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 [32] 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 [33, 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 [34], and *motion planning*, which aims to find physically consistent and collision-free trajectories that realize the objectives of a task plan [35, 36]. 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 large language models contain commonsense world knowledge. The Winograd Schema Challenge [37] and WinoGrande benchmark [38] evaluate commonsense reasoning in word problems. The Winoground dataset [39] 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 language models 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. [30]) 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 [29]) 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 [32] have shown that grounding feedback from multiple modalities enables LLMs to reason about causes of skill failure. This approach reLeveraging a multi-modal approach 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 Low-level Control: Finally, CAPE does not deal with determining the successful execution of low-level skills, i.e., we instead abstract low-level control into a repertoire of high-level skills that we assume to be perfectly executed when planning at a high-level. Several works (such as SayCan [2], NLMap-SayCan [26], Huang et al. [3] and Inner Monologue [4]) make this assumption on high-level skills, but enabling failure detection and recovery at the lower control levels (like DoReMi [29]) would make CAPE a more robust failure recovery system.

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] 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

[8] 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.

[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] 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.

[29] 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.

[30] 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.

[31] 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.

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

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

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

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.