Generating Executable Action Plans with Environmentally-Aware Language Models
Maitrey Gramopadhye and Daniel Szafir

Abstract—Large Language Models (LLMs) trained using massive text datasets have recently shown promise in generating action plans for robotic agents from high-level text queries. However, these models typically do not consider the robot’s environment, resulting in generated plans that may not actually be executable, due to ambiguities in the planned actions or environmental constraints. In this paper, we propose an approach to generate environmentally-aware action plans that agents are better able to execute. Our approach involves integrating environmental objects and object relations as additional inputs into LLM action plan generation to provide the system with an awareness of its surroundings, resulting in plans where each generated action is mapped to objects present in the scene. We also design a novel scoring function that, along with generating the action steps and associating them with objects, helps the system disambiguate among object instances and take into account their states. We evaluated our approach using the VirtualHome simulator and the ActivityPrograms knowledge base and found that action plans generated from our system had a 310% improvement in executability and a 147% improvement in correctness over prior work. The complete code and a demo of our method is publicly available at https://github.com/hri-ironlab/scene_aware_language_planner.

I. INTRODUCTION

Recent work in the natural language processing (NLP) and machine learning (ML) communities has made tremendous breakthroughs in several core aspects of computational linguistics and language modeling driven by advances in deep learning, data, hardware, and techniques. These advancements have led to the release of pretrained large (million and billion+ parameter) language models (LLMs) that have achieved the state-of-the-art across a variety of tasks such as text classification, generation, and summarization, question answering, and machine translation, that demonstrate some abilities to meaningfully understand the real world [1], [2], [3], [4], [5], [6], [7]. LLMs also demonstrate cross-domain and cross-modal generalizations, such as retrieving videos from text, visual question answering, and task planning [8].

In particular, recent works have explored using LLMs to convert high-level natural language commands to actionable steps (e.g. “bring water” → “grab glass”, “fill glass with water”, “walk to table”, “put glass on table”) for intelligent agents [9], [10], [11], [12], [13], [14]. Trained on diverse and extensive data, LLMs have the distinct ability to form action plans for varied high-level tasks.

While promising, the action steps generated by LLMs in prior work are not always executable by a robot platform. For instance, for a task “clean the room” an LLM might generate an output “call cleaning agency on phone” while being correct, this action plan might not be executable since the agent might not grasp the concept of “call” or have the “phone” object in its environment. This limitation arises because LLMs are trained solely on large text corpora and have essentially never had any interaction with an embodied environment. As a result, the action steps they generate lack context on the robot’s surroundings and capabilities.

To address this issue, prior works have explored grounding LLMs by fine-tuning models using human interactions [14], [15], [16] or training models for downstream tasks using pretrained LLMs as frozen backbones [17], [18], [19], [20], [21], [22], [23], [24]. However, these methods often require training on extensive annotated data, which can be expensive or infeasible to obtain, or can lead to loss of generalized knowledge from the LLM. Instead, recent research has investigated biasing LLM output without altering their weights by using prompt engineering [9], [10], [25] or constraining LLM output to a corpus of available action steps defined a priori that are known to be within a robot’s capabilities [9], [10]. This line of research focuses on methods that can utilize the capabilities of LLMs while preserving their generality and with substantially less additional annotated data.

While these systems effectively perform common sense grounding by extracting knowledge from an LLM, they employ a one-fits-all approach without considering the variations possible in the actionable environment. As a result, executing the action plans generated by these systems either requires approximations to the agent’s environment or time-consuming and costly pretraining to generate an affordance score to determine the probability that an action will succeed or produce a favorable outcome toward task completion, given the current agent and environment states. Additionally, since prior systems are environment agnostic, it is not possible to use them to generate executable action plans for tasks requiring object disambiguation. For example, to generate correct action plans for tasks that require interaction with multiple objects with the same name, the system needs to be able to distinguish among object instances.

We propose a novel method to address these issues while generating low-level action plans from high-level task specifications. Our approach is an extension to Huang et al., 2022 [9]. From an Example set (see §IV) using the ActivityPrograms knowledge base collected by Puig et al., 2018 [26], we sample an example similar to the query task and environment and use it to design a prompt for an LLM (details of which are given in §III-A). We then use the LLM to autoregressively generate candidates for each action step. To rank the generated candidates, we design multiple scores...
Task: Relax at home

Fig. 1. Visualization of an example action plan being executed in VirtualHome. Within the virtual home environment, a simulated humanoid agent carries out the robot task sequences generated by our environmentally-aware language model.

for the actions and their associated objects (see §III-B and §III-C). After the top candidate is selected, we append it to the action plan and repeat the process until the entire action plan is generated.

To evaluate our action plans, we use the recently released VirtualHome interface [26] (Figure 1 shows a visualization of an example action plan running in VirtualHome). We use several metrics (details in §IV-A), including executability, Longest Common Sub-sequence (LCS), and final graph correctness to condition the model output without any fine-tuning of the LLM or extra training to learn value functions for ranking LLM outputs.

II. RELATED WORK

Our work builds upon recent efforts in robotics to leverage the potential of LLMs. For instance, researchers are beginning to explore LLMs in the context of applying commonsense reasoning to natural language instructions [27], providing robotic agents with zero-shot action plans [9], and supplying high-level semantic knowledge about robot tasks [10]. Below, we review related research in task planning, LLMs, and action plan grounding.

A. Task Planning

The problem of task planning involves generating a series of steps to accomplish a goal in a constrained environment. Historically, this problem has been widely studied in robotics [28], [29], [30], with most approaches solving it by optimizing the generated plan, given environment constraints, [31], [32] and using symbolic planning [28], [30]. Recently, machine learning methods have been employed to relax the constraints on the environment and allow higher-level task specifications by leveraging techniques such as reinforcement learning or graph learning to learn task hierarchy [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47]. However, most of these methods require extensive training from demonstrations, or explicitly encoded environmental knowledge and may not generalize to unseen environments and tasks. The use of LLMs, which encapsulate generalized world knowledge, may help plan for novel tasks and new environments.

B. Large Language Models

Large language models (LLMs) are language models, usually inspired by the transformer architecture [48], tens of gigabytes in size, and trained on enormous amounts of unstructured text data. Recent advances in the field of natural language processing have shown that LLMs are useful for several downstream applications including interactive dialogue, essay generation, creating websites from text descriptions, automatic code completion, etc. [1], [2], [4], [3]. During their pretraining, LLMs can accumulate diverse and extensive knowledge [49], [50], [51] that enables their use in applications beyond NLP, such as retrieving visual features [52] and solving mathematical problems [53], [54] or as pretrained models for other modalities [55], [56]. In robotics, knowledge embedded in LLMs can be utilized to generate actionable plans for agents from high-level queries. However, in order for a plan to be executable by a robot, the outputs from the LLMs need to be grounded in the context of the robot’s environment and capabilities.

C. Grounding Natural Language in Action Plans

There has been considerable work towards grounding natural language in actionable steps. Prior research has focused on parsing natural language or analyzing it as a series of lexical tokens to remove ambiguity and map language commands to admissible actions [57], [58], [59], [60]. However, these methods usually require extensive, manually coded rules and thus fail to generalize to novel environments and tasks. More relevant to our approach, recent work has explored grounding language models using additional environment elements [61], [62], [63], [64], [65]. Techniques include prompting [9], [25] and constraining language model outputs to admissible actions [10], [11], [12], [13]. To also ground the output of language models in the environment of the agent, prior works have tried using LLMs as fixed backbones, [17], [19], [20], [21], [22], [23], [24], [66] fine-tuning or ranking model outputs through interactions with the environment [14], [15], [16]. Our work extends such approaches, where we use additional inputs from the environment (i.e., objects and their properties) to condition the model output without any fine-tuning of the LLM or extra training to learn value functions for ranking LLM outputs.
may be chosen considering just the task similarity. However, another example ($E_{x_2}$) with the task “Use the computer” may be more relevant because the action plans for both $Q$ and $E_{x_2}$ would have actions involving similar objects, such as “switch on computer”, “type on keyboard”, “push mouse”, etc., which may not be present in the action plan for $E_{x_1}$. Considering the environment in selecting the example may also help in disambiguating between examples with high task similarity but different objects. For example, a “clean the room” action plan, which uses a rag, and a “clean the floor” plan, which uses a mop, both may have high task similarity to a “clean the house” query task. Considering the objects present in the environment of the query (e.g., a rag is present, but not a mop) can help determine the better example.

We start by selecting $\{Q^i\}_{i=1}^{N_e}$ examples that have tasks $\{T^i\}_{i=1}^{N_e}$ similar to the task ($T$) of the query ($Q$). Here $N_e$ is a hyperparameter. We use the cosine similarity ($C$) of task embeddings to calculate task similarity given by:

$$S_M(T, T_e) = C(LM_T(T), LM_T(T_e)) \quad (1)$$

We then compare the environments $\{E^i\}_{i=1}^{N_e}$ of the selected examples with the environment ($E$) of the query ($Q$). An environment from a sample in our dataset is structured like a graph, with the graph nodes representing the available objects. The nodes also have information about object properties (e.g., grabbable, openable, movable, etc.) and the current states of the objects (e.g., clean, closed, etc.). The edges in the graph represent the relations between objects (e.g., inside, on, facing, close to, etc.). We calculate the environment similarity as the mean of the intersection over unions ($IoU$) of the nodes and edges respectively:

$$S_G(E, E_e) = \frac{1}{2} \cdot (IoU(\text{nodes}(E), \text{nodes}(E_e)) + IoU(\text{edges}(E), \text{edges}(E_e))) \quad (2)$$

Finally, from the selected $N_e$ examples, we select one example $E^*_e$ that maximizes the example score given by:

$$P_{eg} = S_M(T^*_e, T) + W_s \cdot S_G(E^*_e, E) \quad (3)$$

where $W_s$ is a hyperparameter. With the example task $T^*_e$, action plan $A^*_e$, and query task $T_e$, we form a prompt ($Pr_{a} = T^*_e + A^*_e + T$) for generating the action plan and a set of objects ($Pr_{o}$) associated with the action plan $A^*_e$. We use $Pr_{o}$ to calculate the similarity scores between any new objects and the objects already associated with the action plan (See §III-C).

### B. Action Step Generation

As in Huang et al., 2022 [9], we sample the $LM_P$ multiple times using prompt $Pr_{a}$ to get $k$ samples for each action step, and the LLM generation probability associated with each sample step ($P_{a}$). $P_{a}$ gives a score for how relevant the Planning LM thinks the sample is to the current action plan and prompt. However, since the output of the language model is unconstrained, it can include infeasible steps that the agent cannot actually execute. To make sure the actions generated are executable, we map each sample to its closest...
admissible action step that maximizes the action matching score given by:

\[ PaM = \max(S_M (a_s, a_v); \forall a_v \in A_v) \]  

Where \( S_M (a_s, a_v) = C(LM_T (a_s), LM_T (a_v)) \). \( A_v \) is the corpus of all admissible atomic action steps, constructed by matching every possible action with every known object.

C. Object Association

Each action step discussed so far is of the format [Action] <Object names>. For an agent to execute an action step, it needs to associate the object names in the step with objects in the environment. However, action plans generated without considering the environment often contain object names that cannot be directly mapped to objects present in the environment, making the plan not executable.

We propose a way to autonomously associate objects in the query environment with action steps during action plan generation, using only a list of the objects present in the environment and their locations, without the use of any hard-coded information from the environment. As a result, our action plans can be executed directly in any query environment. Additionally, since our action plans consider the agent's environment, we are able to generate action plans for tasks that were previously infeasible. For example, a task “set the table” has an action plan with steps “find a plate”, “put plate on table”, “grab cup”, “put cup on table”, etc. repeated multiple times, each time for different “plate” and “cup” objects. A system that does not have any scene information may generate an action plan that has these action steps repeated in the correct order, but map all “cup” objects to the same “cup” and all “plate” objects to the same “plate” during execution, creating an incorrect final result. Since our method can distinguish among the “cup” and “plate” objects in the scene, we can associate different objects of the same name with repeated action steps, leading to correct execution.

Since the admissible action steps for our agent all follow the same schema, we first parse the step to extract the object names (e.g., “put cup on table” has the object names “cup” and “table”). For each extracted object name \( \hat{o} \), we then assign an object from the environment. To do so, we first use a Translation LM \( LM_T \) to find \( o' \), the closest available object name to \( \hat{o} \) that is present in the agent’s environment, chosen by maximizing an object matching score given by:

\[ P_{oM} = \max(S_M (\hat{o}, o); \forall o \in O) \]  

where \( O \) is the corpus of all object names present in the agent’s current environment. We also calculate an object relevance score that denotes the similarity of \( o' \) with the object names in the action plan of the example \( A^* \) and those associated with the previous steps of the generated action plan. For each object name \( o^* \) in \( Pr_o \), we construct the text “\( o^* \) and \( o' \) are related”. We forward the text through the Planning LM \( LM_P \) to calculate the associated cross-entropy loss \( L_{ce} \). We compute the object relevance score \( P_{oD} \) as the mean of \(-log(L_{ce})\) over all objects in \( Pr_o \).

The object matching and relevance scores enable consideration of environment objects for action plan generation, but will be the same for all objects in the scene with the same name. To disambiguate among such objects, we calculate an object disambiguation score \( P_{oD} \) as being inversely related to the mean distance \( (d_o) \) of the object from the objects that the agent interacted with in the last generated step:

\[ P_{oD} = \exp\left(\frac{-d_o}{100}\right) \]  

This score prioritizes objects that are near the agent’s location in the prior step. Thus, promoting more efficient routes for the agent to complete an action step. For some object names, the location is absent in the dataset, and object disambiguation is impossible. In such cases, we set \( P_{oD} = 0 \). However since disambiguation occurs among objects with the same name, this doesn’t create an unfair bias for objects that have location information. For each object, we also look for repetitions, i.e., instances where the current \((\text{action, objects})\) pair is found in the already generated action plan. For each such instance, we incur a negative penalty to encourage the agent to interact with a different object with the same name. This negative score builds for every instance of repetition and also reduce the overall score for the \((\text{action, objects})\) pair, deterring our action plans from loops of multiple actions (e.g., “walk to table”, “walk to chair” repeated over and over) that prior works suffer from. We assign the object with the greatest \( P_{oD} \) to \( \hat{o} \).

D. Ranking and Termination

We take a weighted sum of the scores described above to rank all \((\text{action, objects})\) pairs:

\[ P_{pair} = W_a \cdot P_a + W_{oM} \cdot P_{aM} + W_o \cdot P_o + W_{oD} \cdot P_{oD} \]  

Where \( W_a, W_{oM}, W_o, W_{oD} \) are hyperparameter weights for each score. If an action step has multiple objects, we calculate a mean for each of the 3 object scores \( (P_{aM}, P_{oM}, P_{oD}) \), over the objects. We then select the highest ranking \((\text{action, objects})\) pair and append it to the action plan. We also append the selected action to the prompt for action generation \( Pr_a \) and objects to the set for object relevance score \( Pr_o \). If the score for the highest ranking \((\text{action, objects})\) pair is below a cutoff score hyperparameter, we terminate the action plan.

The resulting action plans generated by the language model are in natural language. To execute and evaluate plans, we used the VirtualHome simulation platform (see §IV), which required parsing this plan to create an action plan matching the VirtualHome agent schema. This parsing was performed using a predefined mapping since all natural language action steps follow a fixed pattern.

IV. EVALUATION

We evaluated our approach in generating environmentally-aware action plans against plans generated using the method in Huang et al., 2022 [9], the state-of-the-art baseline for an
LLM action plan generation system, which does not consider environment information. We executed all plans using VirtualHome, a multi-agent simulation platform. VirtualHome provides diverse and customizable household environments that support a wide array of possible interactions in the form of atomic action steps. An atomic action step is specified by [Action] <Objects> (object form of atomic action steps. An atomic action step is specified by matching environment objects with ids are identified in brackets. Our system then compared this graph with the graph of the environment formed after executing the ground truth action plan (E_G).

For our experiments, we used the ActivityPrograms Knowledge Base released by Puig et al., 2018 [26]. This dataset contains 292 unique high-level household tasks, with 1374 unique action plans and 6201 unique environments in total extracted from VirtualHome, and task and action plan samples manually annotated by Amazon Mechanical Turk workers. Each data point consists of a high-level task, a graphical representation of the agent’s environment, and an action plan consisting of atomic actions. Out of the 292 tasks, 285 tasks have action plans and environments that execute without error in the VirtualHome interface. We performed our experiments on these 285 tasks, randomly split into 3 sets: an Example set of 160 tasks, a Validation set of 25 tasks, and a Test set of the remaining 100 tasks.

A. Metrics

We evaluate our action plans across three metrics: executability, longest common sequence, and correctness.

**Executability** measures whether the generated actions follow a logical order and satisfy the constraints of the environment (e.g., action preconditions are met by preceding actions, action plan objects are present in the environment, and object states support planned actions). We executed the action plan step-by-step on the VirtualHome interface and calculated the number of steps that executed without the action plan failing. We report the percentage of steps that executed as the action plan executability score.

Following Puig et al., 2018 [26], we computed **Longest Common Sub-sequence (LCS)** as the length of the longest common sub-sequence of steps between a generated action plan and a “ground truth” action plan from the ActivityPrograms dataset written by human annotators; divided by the length of the longer action plan. We required that the arrangement of the steps in the sub-sequence remains the same, but allowed gaps between them. LCS provides a metric to understand the number of correct actions being generated and also their short-term order, while penalizing longer action plans that have irrelevant or repeated actions. However, it does not judge the correctness of the action plan as a whole because it does not consider the position of the longest common sequence in the action plan. Also, as LCS only compares the natural language action plans, it does not offer a way to judge whether the action plan is able to disambiguate among objects of the same name.

We propose **Final Graph Correctness** as a new metric to evaluate the final correctness of the environment graph after execution of the action plans. We executed each generated action plan in VirtualHome up to the last executable step and extracted the graph for the resulting environment (E_G). We then compared this graph with the graph of the environment formed after executing the ground truth action plan (E_G).

**B. Experimental Setup**

We evaluated and ablated over the Test set (100 tasks) and used the Example set (160 tasks) for prompt engineering. The Validation set (25 tasks) was used for hyperparameter search. We ran all our experiments on Google Colab using an NVIDIA A100-SXM4-40GB GPU.

In action plans generated by the baseline, objects in action steps were not originally associated with the objects in the environment. For each object in each action step, we randomly selected an object of the same name if it is available, and assigned it to the action step.

We used open-source resources from Hugging Face Transformers [67] and SentenceTransformers [68] for our models. Our primary results used GPT2-large [3] as the Planning LM and all-roberta-large-v1 [69] as the Translation LM.

V. RESULTS

In this section we present our results. The step count is measured as the number of steps and all other metrics reported are a percentage of 100. Some action plans generated by our method are illuminated in Figure 3.

**A. Environment Aware Action Plans**

We computed the action plans for each of the samples in our Test set, using our method and the baseline [9] that doesn’t use any scene information. As each task can have multiple action plans and associated environments, we conducted 5 runs every time we generated an action plan; in each run, we randomly selected an action plan and environment for each task. We report the average results over all the runs.
Table I reports the mean step count, executability, LCS, and final graph correctness for both methods. We observed that on average our method produced longer action plans, which allowed the agent to complete more complicated and longer-horizon tasks. Our method, using the information from the scene, generated action plans that were not only more executable but also led to final environment graphs that were closer to the desired results. As seen from the average LCS, using the environment information to select the example and inform action step generation also led to action plans that were closer to the ground truth action plans. However, we found that the extra computation associated with the agent’s environment slowed down our method as compared to the baseline. On average, computing a step with our method took 1.435 seconds, as compared to the 0.838 seconds of the baseline.

These results are promising as our method, which implicitly considers object states and properties in our example selection module, could be readily integrated with a robot perception stack during real deployments.

B. Ablations

1) Ablation of Implementation Choices: We ablated the scores we propose and show results over the Test set in Table II. Compared to the baseline, we saw our biggest jump in executability and final correctness by incorporating object matching with the action steps and using the object scores ($P_{OM}$, $P_o$) to inform action step ranking. The additional scores also encouraged longer and more complicated action plans. Adding in the disambiguation score ($P_{oD}$) enabled the agent to distinguish between objects with the same name and allowed for more accurate action plans. It also discouraged redundant or repeated actions, thus resulting in shorter and more executable plans. Finally, since we used the objects in the environment to inform action step selection, also including this information in example selection ($S_G^e$) boosted results further as it made the resulting prompt more relevant to the current environment of the agent.

| TABLE II | ABLATIONS OF IMPLEMENTATION CHOICES |
|------------------|------------------|------------------|------------------|------------------|
| Method          | Steps          | Executability | LCS             | Correctness     |
| Baseline        | 5.57           | 16.396%       | 8.795%          | 33.312%         |
| + Object scores ($P_{OM}$, $P_o$) | 10.87 | 43.892% | 8.749% | 47.715% |
| + Object disamb. ($P_{oD}$) | 7.92 | 46.353% | 9.390% | 48.056% |
| + Env. similarity ($S_G^e$) | 8.39 | 49.786% | 11.140% | 48.695% |

To further evaluate the usefulness of object disambiguation, we separately evaluated the performance of the 23 action plans which had an (action, objects) pair repeated in the action plan when object disambiguation was omitted. The results for this ablation are shown in Table III.

2) Ablation over Planning LMs: We ablated over different sized Planning LMs (ranging from 117M parameters to 1.5B parameters in size) from two families of models [3], [70], while fixing all-roberta-large-v1 as the Translation LM (Table IV). We observed that within a family of models, the medium-sized models (GPT2-large, OPT-350M) gave the best results. We found that the small models (GPT2, OPT-125M) resulted in shorter action plans that were unable to capture the details required in action steps and ended up generating high-level instructions which were vague and not executable by the agent (e.g. a task “work in office” generated an action plan - “Step 1: Go to office”, “Step 2: work”). On the other hand, the large models (GPT2-xl, OPT-1.3B) often generated complicated samples that couldn’t effectively be mapped to any available actions and thus resulted in action plans that were not relevant to the query task (e.g., a task “Shampoo hair” generated a sample “grab a shampoo bottle and get in the shower” which couldn’t be mapped to any atomic action step).

| TABLE III | ACTION PLANS WITH MULTIPLE OBJECTS OF THE SAME NAME |
|------------------|------------------|------------------|------------------|------------------|
| Method          | Steps          | Executability | LCS             | Correctness     |
| w/o Object disamb. | 19.44 | 42.578% | 7.686% | 59.718% |
| w/ Object disamb. | 8.96 | 58.570% | 12.006% | 61.462% |

3) Ablation over Translation LMs: We also explored using different size models of Sentence BERT and Sentence RoBERTA [2], [68], [69] for the Translation LM, fixing GPT2-large as the Planning LM (Table V). We found that larger Translation LMs (stsb-bert-large, stsb-roberta-large, all-roberta-large-v1) created better performing and slightly shorter action plans compared to smaller model variants (stsb-bert-base, stsb-roberta-base). We speculate that larger models may create more meaningful embeddings for actions and objects and thus better guide the Planning LM to correct actions; however, they are harsher towards Planning LM outputs that don’t effectively match any atomic action step and thus caused the action plans to terminate faster.

| TABLE IV | ABLATIONS OF PLANNING LM |
|------------------|------------------|------------------|------------------|
| Planning LM | Steps | Executability | LCS | Correctness |
| GPT2          | 1.04 | 8.866% | 0.188% | 33.441% |
| GPT2-large     | 8.39 | 49.786% | 11.140% | 48.695% |
| GPT2-xl        | 9.58 | 32.018% | 8.641% | 47.132% |
| OPT-125M       | 4.52 | 29.183% | 10.681% | 46.010% |
| OPT-350M       | 6.23 | 51.442% | 13.72% | 50.101% |
| OPT-1.3B       | 8.27 | 26.991% | 7.932% | 43.652% |

| TABLE V | ABLATIONS OF TRANSLATION LM |
|------------------|------------------|------------------|
| Translation LM | Steps | Executability | LCS | Correctness |
| stsb-bert-base | 9.76 | 40.555% | 10.892% | 47.393% |
| stsb-bert-large | 9.07 | 47.412% | 12.602% | 49.711% |
| stsb-roberta-base | 9.53 | 38.479% | 9.306% | 47.386% |
| stsb-roberta-large | 8.40 | 40.279% | 9.871% | 48.383% |
| all-roberta-large-v1 | 8.39 | 49.786% | 11.140% | 48.695% |

3573
VI. FUTURE WORK & CONCLUSIONS

In this paper, we propose a method to condition large language models on the information contained in an agent’s environment to generate environmentally-aware action plans from high-level tasks. We propose multiple scores to rank the outputs of LLMs and ground them in the agent’s surroundings. We discuss the performance of our generated action plans for complex and diverse tasks on the VirtualHome interface. While our results demonstrate improved performance in terms of plan executability and correctness over the state-of-the-art baseline, there are several areas for further improvement. For example, our approach makes implicit use of various object properties and states when selecting examples for prompt generation, but cannot make use of this information directly during plan generation. Future research might explore how to further improve plan executability by addressing this limitation. In addition, future work is needed to validate our approach in a real-world robot deployment, beyond the VirtualHome simulator, where object information can be derived from a robot perception and affordance reasoning stack. We hope this work spurs further investigations into how robotics may leverage LLMs in dynamic environments.

REFERENCES

[1] 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.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in NAACL, 2019.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners,” 2019.

[4] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu et al., “Exploring the limits of transfer learning with a unified text-to-text transformer.” J. Mach. Learn. Res., vol. 21, no. 140, pp. 1–67, 2020.

[5] B. Z. Li, M. Nye, and J. Andreas, “Implicit Representations of Meaning in Neural Language Models,” in ACL, 2021.

[6] A. Roberts, C. Raffel, and N. M. Shazeer, “How Much Knowledge Can You Pack into the Parameters of a Language Model?” ArXiv, vol. abs/2002.08910, 2020.

[7] D. W. Otter, J. R. Medina, and J. K. Kalita, “A survey of the usages of deep learning for natural language processing,” IEEE transactions on neural networks and learning systems, vol. 32, no. 2, pp. 604–624, 2020.

[8] R. Bonnmasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill et al., “On the opportunities and risks of foundation models,” ArXiv preprint arXiv:2108.07258, 2021.

[9] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, “Language models as zero-shot planners: Extracting actionable knowledge for embodied agents,” ArXiv preprint arXiv:2201.07207, 2022.

[10] M. Aha, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog et al., “Do as I Can, Not as I Say: Grounding language in robotic affordances,” ArXiv preprint arXiv:2204.01691, 2022.

[11] A. Suglja, Q. Gao, J. Thomason, G. Thattai, and G. Sukhatme, “Embodied BERT: A Transformer Model for Embodied, Language-guided Visual Task Completion,” arXiv, 2021. [Online]. Available: https:// arxiv.org/abs/2108.04927.

[12] A. Pashevich, C. Schmid, and C. Sun, “Episodic Transformer for Vision-and-Language Navigation,” in ICCV, 2021.

[13] P. Sharma, A. Torralba, and J. Andreas, “Skill Induction and Planning with Latent Language,” ArXiv, vol. abs/2110.01517, 2022.

[14] S. Li, X. Puig, Y. Du, C. J. Wang, E. Akyürek, A. Torralba, J. Andreas, and I. Mordatch, “Pre-Trained Language Models for Interactive Decision-Making,” ArXiv, vol. abs/2203.01771, 2022.

[15] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Wang, S. Agarwal, R. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. E. Mitchell, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. J. Lowe, “Training language models to follow instructions with human feedback,” ArXiv, vol. abs/2203.02155, 2022.

[16] M. Reid, Y. Yamada, and S. S. Gu, “Can Wikipedia Help Offline Reinforcement Learning?” 2022.

[17] S. Nair, E. Mitchell, K. Chen, B. Ichtier, S. Savarese, and C. Finn, “Learning-Language-Conditioned Robot Behavior from Offline Data and Crowd-Sourced Annotation,” in CoRL, 2021.

[18] C. Lynch and P. Sermentan, “Language conditioned imitation learning under unstructured data,” Robotics: Science and Systems, 2021. [Online]. Available: https://arxiv.org/abs/2005.07648.

[19] V. Bhalis, C. Paxton, D. A. Garip, and Y. Artzi, “A Persistent Spatial Semantic Representation for High-level Natural Language Instruction Execution,” in 5th Annual Conference on Robot Learning, 2021. [Online]. Available: https://openreview.net/forum?id=NeGdZEyeJKa.

[20] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta, “R3m: A universal visual representation for robot manipulation,” 2022. [Online]. Available: https://arxiv.org/abs/2203.12601.

[21] A. Akakzia, C. Colas, P.-Y. Oudeyer, M. CHETOUMANI, and O. Sigaud, “Grounding Language to Autonomously-Acquired Skills via Goal Generation,” in International Conference on Learning Representations, 2021. [Online]. Available: https://openreview.net/forum?id=chP_dJ5KMHG.

[22] R. Zellers, A. Holtzman, M. Peters, R. Mottaghi, A. Kembhavi, A. Farhadi, and Y. Choi, “Piglet: Language grounding through neuro-symbolic interaction in a 3D world,” in Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). Online: Association for Computational Linguistics, Aug. 2021. pp. 2040–2050. [Online]. Available: https://aclanthology.org/2021.acl-long.159.

[23] F. Hill, S. Mokra, N. Wong, and T. Harley, “Human Instruction-Following with Deep Reinforcement Learning via Transfer-Learning from Text,” ArXiv, vol. abs/2005.09382, 2020.

[24] P. C. Humphreys, D. Raposo, T. Pohlen, G. Thornton, R. Chhipara, A. Muldal, J. Abramson, P. Georgiev, A. Santoro, and T. Lillicrap, “A data-driven approach for learning to control computers,” in Proceedings of the 39th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 9466–9482. [Online]. Available: https://proceedings.mlr.press/v162/humphreys22a.html.

[25] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou, “Chain of Thought Prompting Elicits Reasoning in Large Language Models,” ArXiv, vol. abs/2201.11903, 2022.

[26] 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.

[27] H. Chen, H. Tan, A. Kunz, M. Bansal, and R. Alterovitz, “Enabling robots to understand incomplete natural language instructions using commonsense reasoning,” in IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 1963–1969.

[28] 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, pp. 189–208, 1971. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370271900105.

[29] E. D. Sacerdoti, “A Structure for Plans and Behavior,” 1977.

[30] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila, “SHOP: Simple Hierarchical Ordered Planner,” in IJCAI, 1999.

[31] M. Toussaint, “Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning,” in IJCAI, 2015.

[32] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, “Differentiable physics and stable modes for tool-use and manipulation planning - extended abstract,” in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-
