Generating Executable Action Plans with Environmentally-Aware Language Models

Maitrey Gramopadhye[1] and Daniel Szafir[1]

***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], [8]. LLMs also demonstrate cross-domain and cross-modal generalizations, such as retrieving videos from text, visual question answering and task planning [9]. In particular, recent works have explored using LLMs to convert high-level natural language commands to actionable steps (e.g. "bring water" $\rightarrow$ "grab glass", "fill glass with water", "walk to table", "put glass on table") for intelligent agents [10], [11], [12], [13], [14], [15]. 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" a 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 it's 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 [15], [16], [17] or training models for downstream tasks using pretrained LLMs as frozen backbones [18], [19], [20], [21], [22], [23], [24], [25]. 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 [10], [11], [26] or constraining LLM output to a corpus of available action steps defined *a priori* that are known to be within a robot's capabilities [10], [11]. This line of research focuses on methods that can utilise 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 favourable outcome towards 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 [10]. From an *Example set* (see §IV) using the ActivityPrograms knowledge base collected by Puig et al., 2018 [27], we sample an example similar to the query task and environment and use it to design a prompt for a 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

[1]University of North Carolina at Chapel Hill, United States

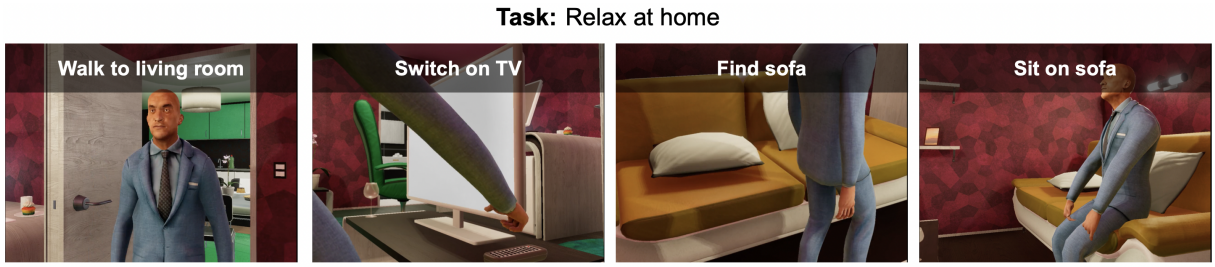

Fig. 1. Visualization of an example action plan being executed in VirtualHome. Within the virtual home environment a simulated humanoid agent carries out the robot task sequences generated by our environmentally-aware language model.

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 [27] (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 autonomously test generated action plans on VirtualHome. Overall, we found that our method increased action plan executability and correctness by 310% and 147% respectively over a state-of-the-art baseline.

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 [28], providing robotic agents with zero-shot action plans [10], and supplying high-level semantic knowledge about robot tasks [11]. 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 [29], [30], [31], with most approaches solving it by optimizing the generated plan, given environment constraints, [32], [33] and using symbolic planning [29], [31]. 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 [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49]. 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 [50], 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 [51], [52], [53] that enables their use in applications beyond NLP, such as retrieving visual features [54] and solving mathematical problems [55], [56] or as pretrained models for other modalities [57], [58]. In robotics, knowledge embedded in LLMs can be utilised 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 analysing it as series of lexical tokens to remove ambiguity and map language commands to admissible actions [59], [60], [61], [62]. 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 [63], [64], [65], [66], [67]. Techniques include prompting [10], [26] and constraining language model outputs to admissible actions [11], [12], [13], [14]. To also ground the output of language models in the environment of the agent, prior works have tried using LLMs as fixed backbones, [18], [20], [21], [22], [23], [24], [25], [68] fine-tuning or ranking model outputs through interactions with the environment [15], [16], [17]. 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.

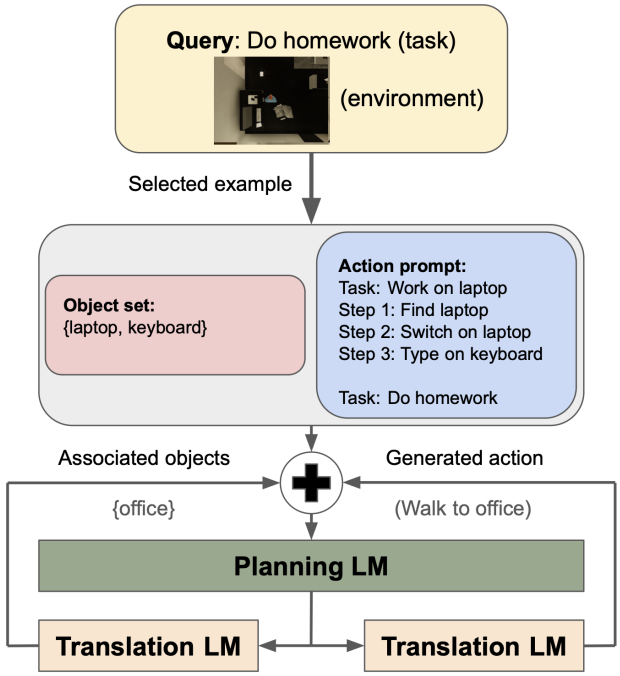

Fig. 2. An overview of our approach. We generate action plans by first selecting an example that has a similar task and environment to the query. We use this example to autoregressively prompt the Planning LM to generate an action plan and map the output to admissible actions and objects using the Translation LM.

III. APPROACH

In this section, we discuss our proposed method to generate directly executable action plans from high-level tasks (Figure 2 provides a visual overview). Motivated by Huang et al., 2022 [10], our approach uses two language models, a *planning* LM (LM_P) to generate the action plan and calculate a score for the similarity of an object with the other objects associated with the action plan; and a *translation* LM (LM_T) to calculate embeddings for objects and actions.

A. LLM Action Plan Prompt Generation

Large language models have the ability to learn from context during inference, i.e., when autoregressively sampled, LLMs can generate meaningful text to complete or extend a given textual prompt [1]. We leverage this capability in designing prompts for LLM sampling that generate action plans. Specifically, we select an example from an *Example set* of task and action plans synthesized from the ActivityPrograms dataset (see §IV) and construct a prompt for the LLM by prepending the example task and action plan to the current task.

We dynamically select the example during inference to design a prompt similar to the query. As in Huang et al., 2022 [10], we use the query task to select the example. However, one of our novel extensions is to also use the environment associated with the query to construct a prompt, keeping in mind the objects (and their states) the agent can currently interact with. For a query (Q) with task "Play video games", an example (Ex_1) with task "play board games" may be chosen considering just the task similarity. However, another example (Ex_2) with task "Use the computer" may be more relevant because the action plan for both Q and Ex_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 Ex_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 room" action plan, which uses a rag, and a "clean floor" plan, which uses a mop, may both have high task similarity to a "clean the house" query task. Considering the objects present in the environment (E) of the query (Q) (e.g., a rag is present, but not a mop) can help determine the better example.

We start by selecting $\{Q_e^i\}_{i=1}^{N_e}$ examples that have tasks $\{T_e^i\}_{i=1}^{N_e}$ 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:

$$S_M(T, T_e) = C(LM_T(T), LM_T(T_e))$$

We then compare the environments $\{E_e^i\}_{i=1}^{N_e}$ 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 (eg. grabbable, openable, movable, etc.) and the current states of the objects (eg. clean, closed, etc.). The edges in the graph represent the relations between objects (eg. inside, on, facing, close to, etc.). We calculate the environment similarity as the mean of the intersection over unions of the nodes and edges respectively:

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

Finally, from the selected N_e examples, we select one example Q_e^* that maximises the example score given by:

$$S_M(T_e^*, T) + W_s \cdot S_G(E_e^*, E)$$

where W_s is a hyperparameter. With the example task T_e^*, action plan A_e^*, and query task T, 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 actions 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 [10], 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 planner 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 maximises the action matching score given by:

$$P_{aM} = max(S_M(a_s, a_v); \forall a_v \in A_v)$$

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 a 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_o) as the mean of $-log(L_{ce})$ over all objects in Pr_o.

The object matching and relevance scores enable a 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 mean distance (d_o) of the object from the objects that the agent interacted with in the last generated step:

$$P_{oD} = exp(\frac{-d_o}{100})$$

This score prioritizes objects that are near the agent's location in the prior step. Thus, promoting shorter and 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 object name, this doesn't create an unfair bias for objects that have location information present. For each object, we also look for repetitions, i.e., instances where the current ($action$, $objects$) pair is found in the already generated action plan. For each such instance we incur a negative penalty to encourages the agent to interact with a different object with the same name. This negative score builds for every instance of repetition and can also reduce the overall score for the ($action$, $objects$) 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 ($action$, $objects$) pairs:

$$W_a \cdot P_a + W_{aM} \cdot P_{aM} + W_o \cdot P_o + W_{oM} \cdot P_{oM} + W_{oD} \cdot P_{oD}$$

Where W_a, W_{aM}, W_o, W_{oM} and 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_{oM}, P_o, P_{oD}), over the objects. We then select the highest ranking ($action$, $objects$) pair and append it to the action plan. We also append $action$ 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 ($action$, $objects$) pair is below a *cutoff* 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 [10], the state-of-the-art baseline for a

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_ids)` (e.g. `[PutBack] <glass> (2) <sink> (1)`).

For our experiments, we used the ActivityPrograms Knowledge Base released by Puig et al., 2018 [27]. 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 [27], 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_o). We then compared this graph with the graph of the environment formed after executing the ground truth action plan (E_G). We computed the set of nodes and edges that changed in the initial environment (E_{init}) after executing the output and ground truth action plans respectively: $Nodes_o$, $Edges_o$, $Nodes_G$, $Edges_G$. The final graph correctness was calculated as the mean of intersection over unions of the two sets of nodes and edges thus obtained:

$$Final\ correctness = \frac{1}{2} \cdot (IoU(Nodes_o, Nodes_G) + IoU(Edges_o, Edges_G))$$

Task: Open curtains

Walk to home office	[home_office 241]
Walk to curtain	[curtain 289]
Find curtain	[curtain 289]
Pull curtain	[curtain 289]
Open curtain	[curtain 289]
Walk to curtain	[curtain 290]
Find curtain	[curtain 290]
Pull curtain	[curtain 290]
Open curtain	[curtain 290]
Walk to home office	[home_office 241]
Walk to curtain	[curtain 291]
Find curtain	[curtain 291]
Pull curtain	[curtain 291]
Open curtain	[curtain 291]

Task: Clean mirror

Walk to bathroom	[bathroom 1]
Walk to mirror	[mirror 1002]
Find mirror	[mirror 1002]
Turn to mirror	[mirror 1002]
Look at mirror	[mirror 1002]
Wash mirror	[mirror 1002]

Task: Sleep

Walk to bedroom	[bedroom 220]
Walk to pillow	[pillow 239]
Find pillow	[pillow 239]
Grab pillow	[pillow 239]
Find bed	[bed 264]
Put pillow in bed	[pillow 240, bed 264]
Lie on bed	[bed 264]

Fig. 3. Example plans generated by our system. For each action step, matched environment objects with ids are identified in brackets. Our system can handle plans containing actions with multiple objects (e.g., pillow and bed) and can consider multiple objects of the same name (e.g., curtain).

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 a 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 [69] and SentenceTransformers [70] for our model choices. Our primary results used `GPT2-large` [3] as the Planning LM and `all-roberta-large-v1` [71] 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, unless mentioned otherwise. 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 [10] 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.

TABLE I

EVALUATION RESULTS OF GENERATED ACTION PLANS

Method	Steps	Executability	LCS	Correctness
Huang et al.	5.574	16.396%	8.795%	33.312%
Ours	8.380	**50.826%**	**10.941%**	**48.990%**

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

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)	8.39	**49.786%**	**11.140%**	**48.695%**

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) boosted results further as it made the resulting prompt more relevant to the current environment of the agent.

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

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%**

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], [72], 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 IV

ABLATIONS OF PLANNING LM

Planning LM	Steps	Executability	LCS	Correctness
GPT2	1.04	8.667%	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%	41.346%
OPT-350M	6.23	51.442%	13.72%	50.101%
OPT-1.3B	8.27	26.991%	7.932%	43.652%

3) Ablation over Translation LMs: We also explored using different size models of Sentence BERT and Sentence RoBERTa [2], [70], [71] 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.

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

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	8.49	40.279%	9.871%	48.383%
all-roberta-large-v1	8.39	49.786%	11.140%	48.695%

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 improvements. 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] "Natural language processing advancements by deep learning: A survey, author=Torfi, Amirsina and Shirvani, Rouzbeh A and Keneshloo, Yaser and Tavaf, Nader and Fox, Edward A," *arXiv preprint arXiv:2003.01200*, 2020.

[9] R. Bommasani, 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.

[10] 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.

[11] M. Ahn, 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.

[12] A. Suglia, 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

[13] A. Pashevich, C. Schmid, and C. Sun, "Episodic Transformer for Vision-and-Language Navigation," in *ICCV*, 2021.

[14] P. Sharma, A. Torralba, and J. Andreas, "Skill Induction and Planning with Latent Language," *ArXiv*, vol. abs/2110.01517, 2022.

[15] 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/2202.01771, 2022.

[16] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. E. Miller, 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.

[17] M. Reid, Y. Yamada, and S. S. Gu, "Can Wikipedia Help Offline Reinforcement Learning?" 2022.

[18] S. Nair, E. Mitchell, K. Chen, B. Ichter, S. Savarese, and C. Finn, "Learning Language-Conditioned Robot Behavior from Offline Data and Crowd-Sourced Annotation," in *CoRL*, 2021.

[19] C. Lynch and P. Sermanet, "Language conditioned imitation learning over unstructured data," *Robotics: Science and Systems*, 2021. [Online]. Available: https://arxiv.org/abs/2005.07648

[20] V. Blukis, C. Paxton, D. Fox, A. Garg, 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=NeGDZeyjcKa

[21] 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

[22] A. Akakzia, C. Colas, P.-Y. Oudeyer, M. CHETOUANI, 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=chPj_I5KMHG

[23] 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

[24] 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.

[25] P. C. Humphreys, D. Raposo, T. Pohlen, G. Thornton, R. Chhaparia, 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

[26] 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.

[27] 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.

[28] H. Chen, H. Tan, A. Kuntz, 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.

[29] 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

[30] E. D. Sacerdoti, "A Structure for Plans and Behavior," 1977.

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

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

[33] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning - extended abtract," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 6231–6235. [Online]. Available: https://doi.org/10.24963/ijcai.2019/869

[34] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, "Neural task programming: Learning to generalize across hierarchical tasks," *CoRR*, vol. abs/1710.01813, 2017. [Online]. Available: http://arxiv.org/abs/1710.01813

[35] D. Xu, R. Martín-Martín, D.-A. Huang, Y. Zhu, S. Savarese, and L. Fei-Fei, "Regression Planning Networks," in *Thirty-third Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[36] D. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," *CoRR*, vol. abs/1807.03480, 2018. [Online]. Available: http://arxiv.org/abs/1807.03480

[37] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/5c48ff18e0a47baaf81d8b8ea51eec92-Paper.pdf

[38] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=SygwwGbRW

[39] B. Ichter, P. Sermanet, and C. Lynch, "Broadly-Exploring, Local-Policy Trees for Long-Horizon Task Planning," in *CoRL*, 2021.

[40] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox, "A Joint Model of Language and Perception for Grounded Attribute Learning," in *ICML*, 2012.

[41] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Perez, L. P. Kaelbling, and J. Tenenbaum, "Inventing relational state and action abstractions for effective and efficient bilevel planning," in *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2022.

[42] C. R. Garrett, C. Paxton, T. Lozano-Perez, L. P. Kaelbling, and D. Fox, "Online Replanning in Belief Space for Partially Observable Task and Motion Problems," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5678–5684, 2020.

[43] Y. Zhu, D. Gordon, E. Kolve, D. Fox, L. Fei-Fei, A. K. Gupta, R. Mottaghi, and A. Farhadi, "Visual Semantic Planning Using Deep Successor Representations," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 483–492, 2017.

[44] B. Wu, S. Nair, L. Fei-Fei, and C. Finn, "Example-Driven Model-Based Reinforcement Learning for Solving Long-Horizon Visuomotor Tasks," *ArXiv*, vol. abs/2109.10312, 2021.

[45] S. Nair and C. Finn, "Hierarchical Foresight: Self-Supervised Learning of Long-Horizon Tasks via Visual Subgoal Generation," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=H1gzR2VKDH

[46] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, "ReLMoGen: Leveraging Motion Generation in Reinforcement Learning for Mobile Manipulation," *arXiv preprint arXiv:2008.07792*, 2020.

[47] C. Li, F. Xia, R. Martin-Martin, and S. Savarese, "HRL4IN: Hierarchical Reinforcement Learning for Interactive Navigation with Mobile Manipulators," *arXiv preprint arXiv:1910.11432*, 2019.

[48] A. Pu, A. Katabarwa, B. Hiziroglu, and O. Dai, "[Re] Language as an Abstraction for Hierarchical Deep Reinforcement Learning," 2020, submitted to NeurIPS 2019 Reproducibility Challenge. [Online]. Available: https://openreview.net/forum?id=S1xcL6qfpr

[49] D. Shah, A. T. Toshev, S. Levine, and brian ichter, "Value Function Spaces: Skill-Centric State Abstractions for Long-Horizon Reasoning," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=vgqS1vkkCbE

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[51] J. Davison, J. Feldman, and A. Rush, "Commonsense knowledge mining from pretrained models," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 1173–1178. [Online]. Available: https://aclanthology.org/D19-1109

[52] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig, "How can we know what language models know?" *CoRR*, vol. abs/1911.12543, 2019. [Online]. Available: http://arxiv.org/abs/1911.12543

[53] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language Models as Knowledge Bases?" *ArXiv*, vol. abs/1909.01066, 2019.

[54] G. Ilharco, R. Zellers, A. Farhadi, and H. Hajishirzi, "Probing Text Models for Common Ground with Visual Representations," *ArXiv*, vol. abs/2005.00619, 2020.

[55] K. Cobbe, V. Kosaraju, M. Bavarian, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training Verifiers to Solve Math Word Problems," *ArXiv*, vol. abs/2110.14168, 2021.

[56] J. Shen, Y. Yin, L. Li, L. Shang, X. Jiang, M. Zhang, and Q. Liu, "Generate & rank: A multi-task framework for math word problems," *arXiv preprint arXiv:2109.03034*, 2021.

[57] K. Lu, A. Grover, P. Abbeel, and I. Mordatch, "Pretrained Transformers as Universal Computation Engines," *arXiv preprint arXiv:2103.05247*, 2021.

[58] M. Tsimpoukelli, J. Menick, S. Cabi, S. Eslami, O. Vinyals, and F. Hill, "Multimodal few-shot learning with frozen language models," *Proc. Neural Information Processing Systems*, 2021.

[59] Y. Artzi and L. Zettlemoyer, "Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions," *Transactions of the Association for Computational Linguistics*, vol. 1, pp. 49–62, 2013. [Online]. Available: https://aclanthology.org/Q13-1005

[60] D. Misra, K. Tao, P. Liang, and A. Saxena, "Environment-Driven Lexicon Induction for High-Level Instructions," in *ACL*, 2015.

[61] D. Misra, J. Sung, K. Lee, and A. Saxena, "Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions," *The International Journal of Robotics Research*, vol. 35, pp. 281 – 300, 2016.

[62] M. Tenorth, D. Nyga, and M. Beetz, "Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web," in *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, USA, May 3–8 2010, pp. 1486–1491. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5509955

[63] C. Sun, A. Myers, C. Vondrick, K. P. Murphy, and C. Schmid, "VideoBERT: A Joint Model for Video and Language Representation Learning," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7463–7472, 2019.

[64] L. H. Li, M. Yatskar, D. Yin, C.-J. Hsieh, and K.-W. Chang, "VisualBERT: A Simple and Performant Baseline for Vision and Language," in *Arxiv*, 2019.

[65] J. Lu, D. Batra, D. Parikh, and S. Lee, "ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks," in *NeurIPS*, 2019.

[66] R. Zellers, X. Lu, J. Hessel, Y. Yu, J. S. Park, J. Cao, A. Farhadi, and Y. Choi, "Merlot: Multimodal neural script knowledge models," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 23 634–23 651. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/c6d4eb15f1e84a36eff58eca3627c82e-Paper.pdf

[67] 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," *CoRR*, vol. abs/2103.00020, 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[68] C. Lynch and P. Sermanet, "Grounding Language in Play," *ArXiv*, vol. abs/2005.07648, 2020.

[69] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "HuggingFace's

Transformers: State-of-the-art Natural Language Processing," *ArXiv*, vol. abs/1910.03771, 2019.
[70] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," *ArXiv*, vol. abs/1908.10084, 2019.
[71] 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*, vol. abs/1907.11692, 2019.
[72] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "OPT: Open Pre-trained Transformer Language Models," 2022.

APPENDIX

A. Pseudo-Code of the Algorithm

This section presents the pseudo-code for generating an action plan in Algorithm 1. A detailed discussion of the method and action step scores is given in §III.

Algorithm 1 Generating environment aware action plans

Legend
LM_P : Planning language model for text completion
LM_T : Translation language model for text embedding
$\{(T_e^i, A_e^i, E_e^i)\}_{i=1}^{N_e}$: Example set, where each sample consists of a task T_e, action plan A_e and environment E_e
A_v : The corpus of all available atomic actions
C : Cosine similarity function
$ExtractObjects$: Extract objects from an atomic action
$S_M(a_1, a_2)$: Similarity function for object or action embeddings = $C(LM_T(a_1), LM_T(a_2))$
$S_G(E_1, E_2)$: Similarity function for environment graphs
Input: Test task T and the query environment E
Output: Action plan comprising of executable action steps of type - ($action$, $objects$)

Get example (T_e^*, A_e^*, E_e^*) that maximises $S_M(T_e^*, T) + W_s \cdot S_G(E_e^*, E)$
Initialize prompt for actions (Pr_a) with $(T_e^* + A_e^* + T)$ and prompt for objects (Pr_o) with objects in E_e^*
for $step < max\_steps$ **do**
    Sample LM_P k times to obtain candidate actions
    for each $action$ **do**
        $objs$ = $ExtractObjects(action)$
        P_{aM} = $max(S_M(action, a_v); \forall a_v \in A_v)$
        P_a = generation probability returned by LM_P
        P_{oM} = $avg(max(S_M(\hat{o}, o_v); \forall o_v \in O); \forall \hat{o} \in objs)$
        P_o calculated from LM_P using prompt Pr_o
        d_o = mean distance of the object from the objects interacted with in the last step
        P_{oD} = $exp(-d_o/100)$
        Ranking score = $W_a \cdot P_a + W_{aM} \cdot P_{aM} + W_o \cdot P_o + W_{oM} \cdot P_{oM} + W_{oD} \cdot P_{oD}$
        Get ($action$, $objects$) pair with highest ranking and append it to output.
        Append $action$ to Pr_a and $objects$ to Pr_o
        Terminate action plan if ranking score $< cutoff$
    end for
end for

B. Admissible Atomic Action Steps

This section discusses the admissible atomic action steps that the agent can perform. This corpus was constructed by matching every possible action in the dataset with every known object in the VirtualHome simulator. Each of the possible 42 actions is in natural language and has an injective mapping to the format acceptable to the VirtualHome simulator, as given in Table VI. Table VII lists all of the 228 objects that can be present in a VirtualHome environment.

TABLE VI
ALL ADMISSIBLE ACTIONS

Natural Language	**VirtualHome acceptable format**
Sleep	[SLEEP]
Stand up	[STANDUP]
Wake up	[WAKEUP]
Close $< Obj >$	[CLOSE] $< Obj >$
Cut $< Obj >$	[CUT] $< Obj >$
Drink $< Obj >$	[DRINK] $< Obj >$
Drop $< Obj >$	[DROP] $< Obj >$
Eat $< Obj >$	[EAT] $< Obj >$
Find $< Obj >$	[FIND] $< Obj >$
Grab $< Obj >$	[GRAB] $< Obj >$
Greet $< Obj >$	[GREET] $< Obj >$
Lie on $< Obj >$	[LIE] $< Obj >$
Look at $< Obj >$	[LOOKAT] $< Obj >$
Move $< Obj >$	[MOVE] $< Obj >$
Open $< Obj >$	[OPEN] $< Obj >$
Plug in $< Obj >$	[PLUGIN] $< Obj >$
Plug out $< Obj >$	[PLUGOUT] $< Obj >$
Point at $< Obj >$	[POINTAT] $< Obj >$
Pull $< Obj >$	[PULL] $< Obj >$
Push $< Obj >$	[PUSH] $< Obj >$
Put back $< Obj >$	[PUTOBJBACK] $< Obj >$
Take off $< Obj >$	[PUTOFF] $< Obj >$
Put on $< Obj >$	[PUTON] $< Obj >$
Read $< Obj >$	[READ] $< Obj >$
Rinse $< Obj >$	[RINSE] $< Obj >$
Run to $< Obj >$	[RUN] $< Obj >$
Scrub $< Obj >$	[SCRUB] $< Obj >$
Sit on $< Obj >$	[SIT] $< Obj >$
Squeeze $< Obj >$	[SQUEEZE] $< Obj >$
Switch off $< Obj >$	[SWITCHOFF] $< Obj >$
Switch on $< Obj >$	[SWITCHON] $< Obj >$
Touch $< Obj >$	[TOUCH] $< Obj >$
Turn to $< Obj >$	[TURNTO] $< Obj >$
Type on $< Obj >$	[TYPE] $< Obj >$
Walk to $< Obj >$	[WALK] $< Obj >$
Wash $< Obj >$	[WASH] $< Obj >$
Watch $< Obj >$	[WATCH] $< Obj >$
Wipe $< Obj >$	[WIPE] $< Obj >$
Release $< Obj >$	[RELEASE] $< Obj >$
Pour $< Obj1 >$ into $< Obj2 >$	[POUR] $< Obj1 > < Obj2 >$
Put $< Obj1 >$ on $< Obj2 >$	[PUTBACK] $< Obj1 > < Obj2 >$
Put $< Obj1 >$ in $< Obj2 >$	[PUTIN] $< Obj1 > < Obj2 >$

TABLE VII

ALL POSSIBLE OBJECTS

razor	conditioner	paper towel	homework
bowl	fork	toaster	food oatmeal
music stand	ironing board	light	tooth paste
dirt	toilet paper	chef knife	pen
button	hanger	food pizza	child
bedroom	shoes	juice	arms both
bookshelf	pantry	clothes jacket	picture
pot	sponge	mousepad	cup
mirror	printer	address book	kitchen
clothes socks	computer	cd player	knife
duster	food apple	trashcan	electric shaver
water glass	shower	alarm clock	laptop
sofa	remote control	comforter	cutting board
notes	table cloth	food turkey	notebook
nightstand	cleaning bottle	cloth napkin	document
detergent	fridge	centerpiece	food carrot
pasta	living room	glass	cookingpot
painting	bed	mop	oven
cellphone	sink	food sugar	feet both
desk	dishrack	hair	kitchen cabinet
fryingpan	bills	wall clock	dry pasta
home office	coffee maker	food snack	freezer
console	love seat	wine glass	textbook
paper	ground coffee	mug	dvd player
lighting	food dessert	spoon	food fish
cat	television	clothes skirt	chair
filing cabinet	faucet	receipt	crackers
milk	radio	keyboard	washing machine
brush	novel	keys	curtain
bathroom	garbage can	broom	groceries
toothbrush	teeth	food bread	pajamas
food vegetable	microwave	clothes hat	folder
magazine	food chicken	legs both	clothes pants
closet	wall	dresser	sauce pan
couch	spectacles	after shave	board game
face soap	dishwasher	water	man
drinking glass	placemat	cabinet	printing paper
stereo	slippers	coin	kitchen counter
shampoo	toy	table	face
purse	hands both	bookmark	lamp
food cheese	beer	soap	dish soap
kids bedroom	sheets	cupboard	food food
scissors	bathtub	mop bucket	drying rack
headset	mouse	toilet	clothes scarf
clothes shirt	clothes dress	towel	creditcard
carpet	rag	coffee cup	food egg
window	coffee table	floor	controller
coffee filter	iron	bag	woman
dog	book	drawing	facial cleanser
phone	newspaper	coffee pot	envelope
pillow	food noodles	floor lamp	telephone
entrance hall	food cereal	stove	vacuum cleaner
plate	coffee	fly	mail
napkin	lightswitch	shredder	blanket

dining room	bathroom counter
instrument piano	electrical outlet
toothbrush holder	cleaning solution
clothes underwear	laundry detergent
video game console	basket for clothes
video game controller	bathroom cabinet

C. Sample Action Plan

This section gives an example of an action plan from the ActivityPrograms knowledge base. The action plan consists of atomic action steps in the format described in §B. Since each natural language action can be mapped to a VirtualHome acceptable format, each action plan represented in natural language also has a unique action plan in VirtualHome acceptable format (see Table VIII).

TABLE VIII

SAMPLE ACTION PLAN FROM ACTIVITYPROGRAMS KNOWLEDGE BASE

Walk to kitchen	[WALK] <kitchen> (1)
Walk to dish soap	[WALK] <dish soap> (1)
Find dish soap	[FIND] <dish soap> (1)
Grab dish soap	[GRAB] <dish soap> (1)
Find plate	[FIND] <plate> (1)
Put dish soap on plate	[PUTBACK] <dish soap> (1) <plate> (1)
Find plate	[FIND] <plate> (1)
Grab plate	[GRAB] <plate> (1)
Wash plate	[WASH] <plate> (1)
Rinse plate	[RINSE] <plate> (1)
Find dishrack	[FIND] <dishrack> (1)
Put plate on dishrack	[PUTBACK] <plate> (1) <dishrack> (1)

D. Hyperparameter Tuning

We used the Validation set of size 25 to conduct a grid search for the hyperparameters given in Table IX.

TABLE IX

HYPERPARAMETERS

Hyperparameter name	**Values considered**
Weight for Object Matching Score (W_{oM})	{0, 0.3, 0.5, 0.7}
Weight for Object Relevance Score (W_o)	{0, 0.1, 0.25}
Weight for Object Disambiguation Score (W_{oD})	{0, 0.1, 0.3, 0.5}
Weight for Environment Similarity (W_s)	{0, 0.25, 0.5}
Temperature modulating LM_P sample probability (T)	{0, 0.3, 0.5}
Action step score cutoff	{0.8, 1.0, 1.2, 1.4, 1.6}

E. Qualitative Results for Action Plans

This section presents some more action plans generated by our method.

Task: Watch TV

Walk to living room	[dining_room 1]
Walk to remote control	[remote_control 1000]
Find remote control	[remote_control 1000]
Grab remote control	[remote_control 1000]
Walk to couch	[couch 192]
Sit on couch	[couch 192]
Touch remote control	[remote_control 1000]
Find television	[television 216]
Switch on television	[television 216]
Turn to television	[television 216]
Watch television	[television 216]

Task: Surf internet

Walk to home office	[home_office 246]
Walk to desk	[desk 282]
Find chair	[chair 283]
Sit on chair	[chair 283]
Find computer	[computer 321]
Switch on computer	[computer 321]
Turn to home office	[home_office 246]
Walk to computer	[computer 321]
Turn to computer	[computer 321]
Type on home office	[home_office 246]

Task: Take dishes out of dishwasher

Walk to kitchen	[oven 229]
Walk to dishwasher	[dishwasher 1000]
Find dishwasher	[dishwasher 1000]
Open dishwasher	[dishwasher 1000]

Task: Sort laundry

Wash clothes underwear	[clothes_underwear 2306]
Wash clothes socks	[clothes_socks 2029]

Task: Sleep

Walk to bedroom	[bedroom 220]
Walk to pillow	[pillow 239]
Find pillow	[pillow 239]
Grab pillow	[pillow 239]
Find bed	[bed 264]
Put pillow in bed	[pillow 240, bed 264]
Lie on bed	[bed 264]

Task: Hang keys

Walk to entrance hall	[dining_room 41]
Walk to purse	[purse 1001]
Find purse	[purse 1001]
Open purse	[purse 1001]
Find keys	[keys 1002]
Grab keys	[keys 1002]
Close purse	[purse 1001]

Task: Watch movie

Walk to living room	[dining_room 49]
Walk to remote control	[remote_control 1000]
Find remote control	[remote_control 1000]
Grab remote control	[remote_control 1000]
Walk to couch	[couch 230]
Sit on couch	[couch 230]
Touch remote control	[remote_control 1000]

Task: Vacuum

Walk to living room	[dining_room 41]
Walk to closet	[bathroom 1]
Open closet	[bathroom 1]
Find vacuum cleaner	[vacuum_cleaner 1001]
Grab vacuum cleaner	[vacuum_cleaner 1001]
Pull vacuum cleaner	[vacuum_cleaner 1001]
Plug in vacuum cleaner	[vacuum_cleaner 1001]
Switch on vacuum cleaner	[vacuum_cleaner 1001]
Pull vacuum cleaner	[vacuum_cleaner 2420]
Push vacuum cleaner	[vacuum_cleaner 1001]
Pull vacuum cleaner	[vacuum_cleaner 2461]
Push vacuum cleaner	[vacuum_cleaner 2420]
Pull vacuum cleaner	[vacuum_cleaner 2571]
Push vacuum cleaner	[vacuum_cleaner 2461]
Switch off vacuum cleaner	[vacuum_cleaner 1001]
Plug out vacuum cleaner	[vacuum_cleaner 1001]

Task: Eat

Walk to kitchen	[oven 82]
Walk to plate	[plate 1003]
Find plate	[plate 1003]
Grab plate	[plate 1003]
Find food food	[food_food 1000]
Grab food food	[food_food 1000]
Put food food on plate	[food_food 1000, plate 1003]
Walk to dining room	[dining_room 1]
Walk to table	[table 63]
Find table	[table 63]
Put plate on table	[plate 1003, table 63]
Find chair	[chair 60]
Sit on chair	[chair 60]
Eat plate	[plate 1003]
Eat food food	[food_food 1000]
Put food food on plate	[food_food 2021, plate 2166]

Task: Computer work

Walk to computer	[computer 379]
Turn to computer	[computer 379]
Look at computer	[computer 379]
Type on computer	[computer 379]
Switch off computer	[computer 379]

Task: Clean screen

Walk to home office	[home_office 241]
Walk to computer	[computer 276]
Find computer	[computer 276]
Turn to computer	[computer 276]
Look at computer	[computer 276]
Switch off computer	[computer 276]
Find brush	[brush 2108]
Turn to brush	[brush 2108]
Look at brush	[brush 2108]
Grab brush	[brush 2108]
Wipe computer	[computer 209]
Drop brush	[brush 2108]
Find napkin	[napkin 2060]
Grab napkin	[napkin 2060]
Wash napkin	[napkin 2060]
Wipe computer	[computer 276]
Drop napkin	[napkin 2060]
Find brush	[brush 2705]
Grab brush	[brush 2705]
Wipe brush	[brush 2108]

Task: Look at painting

Walk to living room	[dining_room 1]
Walk to painting	[drawing 139]
Turn to painting	[drawing 139]
Look at painting	[drawing 139]
Find painting	[drawing 139]
Turn to painting	[drawing 138]
Look at painting	[drawing 138]
Turn to painting	[drawing 140]
Look at painting	[drawing 140]
Turn to painting	[drawing 141]
Look at painting	[drawing 141]
Turn to painting	[drawing 142]
Look at painting	[drawing 142]
Turn to painting	[drawing 186]
Look at painting	[drawing 186]
Turn to painting	[drawing 283]
Look at painting	[drawing 283]
Turn to painting	[drawing 283]

Task: Put clothes away

Find clothes dress	[clothes_dress 2134]
Turn to clothes dress	[clothes_dress 2134]
Point at clothes dress	[clothes_dress 2134]
Wash clothes dress	[clothes_dress 2134]
Turn to clothes dress	[clothes_dress 2202]
Look at clothes dress	[clothes_dress 2134]
Wash clothes dress	[clothes_dress 2202]
Look at clothes dress	[clothes_dress 2202]
Wash clothes dress	[clothes_dress 2229]
Look at clothes dress	[clothes_dress 2229]
Wash clothes dress	[clothes_dress 2467]
Look at clothes dress	[clothes_dress 2467]
Wash clothes dress	[clothes_dress 2592]
Look at clothes dress	[clothes_dress 2592]

Task: Start computer

Walk to home office	[home_office 179]
Walk to chair	[chair 227]
Pull chair	[chair 227]
Sit on chair	[chair 227]
Find computer	[computer 264]
Switch on computer	[computer 264]
Turn to computer	[computer 264]
Look at computer	[computer 264]
Turn to home office	[home_office 179]
Walk to chair	[chair 133]
Walk to chair	[chair 134]
Pull chair	[chair 134]
Sit on chair	[chair 134]
Find computer	[computer 320]
Switch on computer	[computer 320]
Turn to computer	[computer 320]
Look at computer	[computer 320]
Turn to home office	[home_office 179]

Task: Compute

Walk to home office	[home_office 1]
Walk to chair	[chair 17]
Find chair	[chair 17]
Sit on chair	[chair 17]
Find computer	[computer 98]
Switch on computer	[computer 98]
Find notebook	[laptop 2077]
Grab notebook	[laptop 2077]

Task: Straighten pictures on wall

Walk to living room	[dining_room 41]
Walk to wall	[wall 68]
Find picture	[picture 2114]
Turn to picture	[picture 2114]
Look at picture	[picture 2114]
Grab picture	[picture 2114]
Walk to wall	[wall 2]
Put picture on wall	[picture 2114, wall 2]
Find picture	[picture 2331]
Turn to picture	[picture 2331]
Look at picture	[picture 2331]
Grab picture	[picture 2331]
Walk to wall	[wall 3]
Put picture on wall	[picture 2331, wall 3]
Find picture	[picture 2402]
Grab picture	[picture 2402]
Walk to wall	[wall 4]
Put picture on wall	[picture 2402, wall 4]
Find picture	[picture 2417]
Grab picture	[picture 2417]

Task: Work

Walk to desk	[desk 104]
Walk to chair	[chair 103]
Find chair	[chair 103]
Sit on chair	[chair 103]
Find laptop	[laptop 2120]
Open laptop	[laptop 2120]

Task: Put away keys

Walk to entrance hall	[dining_room 1]
Walk to purse	[purse 2067]
Find purse	[purse 2067]
Open purse	[purse 2067]
Find keys	[keys 1000]
Grab keys	[keys 1000]
Close purse	[purse 2067]

Task: Spread table with appropriate supplies

Walk to table	[table 123]
Put food food in table	[food_food 2006, table 123]
Put food food in bowl	[food_food 2006, bowl 2284]

Task: Send email

Walk to chair	[chair 103]
Sit on chair	[chair 103]
Find laptop	[laptop 2053]
Switch on laptop	[laptop 2053]
Turn to laptop	[laptop 2053]
Look at laptop	[laptop 2053]
Find mail	[mail 1000]
Grab mail	[mail 1000]
Read mail	[mail 1000]
Find keyboard	[keyboard 168]
Type on keyboard	[keyboard 168]
Switch off laptop	[laptop 2053]

Task: Put in chair

Walk to dining room	dining_room (100]
Walk to chair	chair (126]
Put chair in table	chair 126, table 128]
Sit on chair	chair (126]
Put chair on table	chair 126, table 128]
Sit on chair	chair (127]
Put back chair	chair (127]
Sit on chair	chair (124]
Put back chair	chair (124]
Sit on chair	chair (125]
Put back chair	chair (125]
Sit on chair	chair (17]
Put back chair	chair (17]
Sit on chair	chair (240]
Put back chair	chair (240]
Sit on chair	chair (240]
Put back chair	chair (126]
Sit on chair	chair (126]
Put back chair	chair (126]
Sit on chair	chair (2002]

Task: Open curtains

Walk to home office	[home_office 241]
Walk to curtain	[curtain 289]
Find curtain	[curtain 289]
Pull curtain	[curtain 289]
Open curtain	[curtain 289]
Walk to curtain	[curtain 290]
Find curtain	[curtain 290]
Pull curtain	[curtain 290]
Open curtain	[curtain 290]
Walk to home office	[home_office 241]
Walk to curtain	[curtain 291]
Find curtain	[curtain 291]
Pull curtain	[curtain 291]
Open curtain	[curtain 291]
Walk to curtain	[curtain 22]
Find curtain	[curtain 22]
Pull curtain	[curtain 22]
Open curtain	[curtain 22]
Walk to curtain	[curtain 206]
Find curtain	[curtain 206]