LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models

Chan Hee Song
The Ohio State University
song.1855@osu.edu

Jiaman Wu
The Ohio State University
wu.5686@osu.edu

Clayton Washington
The Ohio State University
washington.534@osu.edu

Brian M. Sadler
DEVCOM ARL
brian.m.sadler6.civ@army.mil

Wei-Lun Chao
The Ohio State University
chao.209@osu.edu

Yu Su
The Ohio State University
su.806@osu.edu

Abstract

This study focuses on using large language models (LLMs) as a planner for embodied agents that can follow natural language instructions to complete complex tasks in a visually-perceived environment. The high data cost and poor sample efficiency of existing methods hinders the development of versatile agents that are capable of many tasks and can learn new tasks quickly. In this work, we propose a novel method, LLM-Planner, that harnesses the power of large language models to do few-shot planning for embodied agents. We further propose a simple but effective way to enhance LLMs with physical grounding to generate and update plans that are grounded in the current environment. Experiments on the ALFRED dataset show that our method can achieve very competitive few-shot performance: Despite using less than 0.5% of paired training data, LLM-Planner achieves competitive performance with recent baselines that are trained using the full training data. Existing methods can barely complete any task successfully under the same few-shot setting. Our work opens the door for developing versatile and sample-efficient embodied agents that can quickly learn many tasks. [1]

1. Introduction

Building versatile embodied agents such as robots that can follow natural language commands to do different tasks as well as learn to do new tasks quickly has long been desired. However, contemporary language-driven agents still require a large number of labeled examples (pairs of language instructions and gold trajectories) to learn each task, which is highly costly and hinders the development of truly versatile agents [34, 29, 25, 8, 37, 17, 40, 27, 11, 2, 16]. Recently, an array of seminal work has shown the remarkable potential of large language models (LLMs) such as GPT-3 [4] as a few-shot planner for embodied AI agents [1, 13, 21, 35]. Agents equipped with LLM-based planners have started to show the ability to learn a new task with a few training examples.

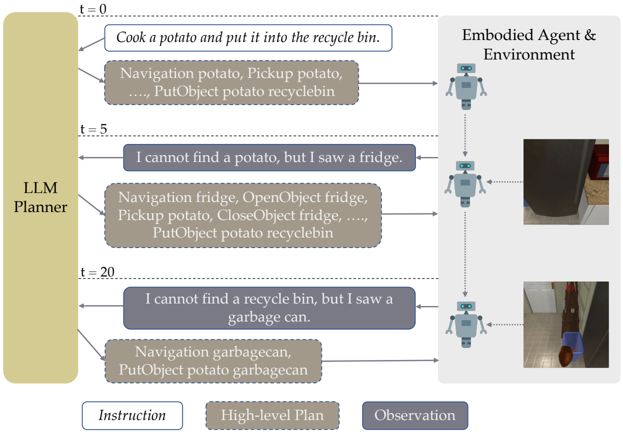

t = 0
LLM Planner
Cook a potato and put it into the recycle bin.
Navigation potato, Pickup potato, ..., PutObject potato recyclebin
Embodied Agent & Environment
t = 5
I cannot find a potato, but I saw a fridge.
Navigation fridge, OpenObject fridge, Pickup potato, CloseObject fridge, ..., PutObject potato recyclebin
t = 20
I cannot find a recycle bin, but I saw a garbage can.
Navigation garbagecan, PutObject potato garbagecan
Instruction
High-level Plan
Observation

Figure 1: An illustration of LLM-Planner for high-level planning. After receiving the natural language instruction ($t = 0$), LLM-Planner first generates a high-level plan by prompting a large language model (*e.g.*, GPT-3). When the embodied agent gets stuck during the execution of the current plan ($t = 5$ and 20), LLM-Planner re-plans based on observations from the environment to generate a more grounded plan, which may help the agent get unstuck. The commonsense knowledge in the LLM (*e.g.*, food is often stored in a fridge) allows it to produce plausible high-level plans and re-plan based on new environmental perception.

While showing great promises as proof of concepts, existing work still presents significant limitations that may prevent larger-scale applications beyond their limited eval-

[1] Website: https://dki-lab.github.io/LLM-Planner/

uation setting. As an example, SayCan [1], one of the pioneering work on using LLMs for embodied instruction following, is evaluated on two environments with only 15 object types. The agent is assumed to be able to enumerate all admissible skills (*i.e.*, [action, object] pairs) up front so it can use an LLM to rank the skills. This assumption could break easily in partially-observable environments when deploying an agent to new environments. The cost could also quickly pile up in more complex environments with more objects because the agent needs to call the LLM to evaluate every admissible skill at every step; efficiency deteriorates at the same time. Finally, most existing work [1, 35, 13, 24] uses LLMs to generate a single static plan from the language instruction and then executes on the entire plan. However, the optimal plan for the same language instruction is dependent on the environment; different environments may need different plans. There lacks a way to dynamically adjust the plan from LLMs based on environmental perception.

Building on existing work, we propose *LLM-Planner*, an LLM-based planner for embodied instruction following. An important design goal is to be able to handle a wide range of tasks in diverse, partially-observable environments, and can dynamically adjust the plan based on perceptions from the environment. Therefore, different from SayCan, we use LLMs to directly generate plans instead of ranking admissible skills, obviating the need to have sufficient knowledge about the environment *a priori* while also significantly reducing the number of calls to LLMs. Another unique strength of *LLM-Planner* is its ability to dynamically re-plan based on what the agent observes in the current environment, which produces more grounded plans.

More specifically, we adopt hierarchical planning models (e.g., [38, 33]), which consist of a *high-level planner* and a *low-level planner*. We use LLMs to generate high-level plans (HLPs), *i.e.*, a sequence of subgoals (*e.g.*, [Navigation potato, Pickup potato, Navigation microwave, ...]) that the agent needs to achieve, in the specified order, to accomplish the final goal specified by the language instruction. The low-level planner then maps each subgoal into a sequence of primitive actions for achieving that subgoal in the current environment and state. An important observation is that, *given a high-level plan, low-level planning becomes conditionally independent of the natural language instruction*. It becomes the classic object localization and navigation problem [6] (for navigation subgoals) or simply executing the specified interaction action with the right objects (for interaction subgoals). The low-level planner can be trained with data synthesized from the simulator (see, *e.g.*, [26, 3]).

Furthermore, we follow the in-context learning paradigm [4, 20] and only use a small number of paired examples. In addition, no parameter update is needed, which saves development time. For the example in Figure 1, at the beginning of an episode ($t = 0$), given a natural language instruction, we directly prompt the LLM to generate the HLP by giving it several exemplar pairs of (instruction, HLP) in its context. We also leverage established techniques such as dynamic in-context example retrieval [28, 31, 9, 19] and logit biases [10] to further improve the in-context learning performance.

While the HLPs generated by LLMs are already plausible at first glance, they still lack a fundamental aspect of embodied agents — *physical grounding*; *i.e.*, the generated HLP needs to be grounded to the environment the agent is in. Previous approaches [1, 35, 13] train a separate model that translates the LLM plans to the grounded admissible actions. However, this is possible under the assumption that the LLM plan can be matched to a reasonable admissible action. If the LLM plans are not contained in the list of admissible action, which is the case in the diverse environments, this creates an undetermined behavior for those agents. To overcome this problem, we propose a novel *grounded re-planning* algorithm to empower LLM-Planner with physical grounding. Specifically, as an agent is executing the initial HLP, whenever it has taken too many steps to reach the current subgoal or has made too many failed attempts, we dynamically prompt the LLM again to generate a new continuation of the partial HLP that has been completed at that point. For grounding, we add the list of objects perceived in the environment so far into the prompt as a simple but effective description of the current environment. Figure 1 demonstrates how our grounded re-planning algorithm can help the agent overcome a plan that is unattainable. For the example at $t = 5$, the agent is taking too long to find a potato. It re-prompts the LLM with the object fridge observed in the environment, and LLM-Planner generates a new HLP from scratch (because no subgoal has been completed so far) that directs the agent to look for a potato in the fridge. By introducing a way to incorporate feedback from the environment, we aim to create a *closed-loop* between the environment and the LLMs where LLMs can dynamically adapt the generated high-level plans to the environment.

While most existing work [1, 14, 13, 35, 24] is evaluated under a limited setting (*e.g.*, limited/known environments, short-horizon tasks, or simple environments with a small number of objects), we evaluate LLM-Planner on ALFRED [34], a large-scale dataset with diverse partially-observable environments and a wide variety of tasks and objects. We test our LLM-Planner by integrating it with the perception module and low-level planner from a strong baseline model, HLSM [3]. Using less than 0.5% of paired training data, LLM-Planner achieves competitive performance compared with HLSM and outperforms multiple other baselines, which are trained with the full training set. Under the same few-shot setting, existing methods can barely complete any task successfully. Our work opens a new door for developing versatile and extremely sample-efficient embodied agents by harnessing the power of large language models and grounding.

2. Related Work

2.1. Vision-and-language Navigation

In navigation-only VLN datasets such as R2R [2], models that generate the action sequence end-to-end with a Transformer model can already achieve a good performance [37, 27]. Recent work [17, 23, 25, 11] employs BERT and its variants [7, 22] to get better language understanding. These models jointly learn the linguistic and visual representations with cross-attention for grounding.

However, in more complex VLN, or embodied instruction following in datasets such as ALFRED [34], hierarchical planning models [3, 26, 18] that separate the high-level and low-level planning have proven to be most effective. These models use pretrained language models (*e.g*. BERT) to generate high-level plans and construct a semantic map to guide the agent to find the target objects specified in the high-level plan.

Recent work has shown that hierarchical planning models are advantageous in the low-data regime. $(SL)^3$ [33] uses 10% of ALFRED's training data to learn how to generate natural language subtasks and then match primitive actions to each subtask. We take this modular approach one step further and propose to use large language models (LLMs) under the few-shot setting. More discussion of $(SL)^3$ is in the supplementary materials.

2.2. Prompting for VLN

The use of LLMs for decision making has become an increasingly popular topic for research. Two major branches of LLM usages among existing works are 1) using the LLM as an auxiliary helper or 2) using the LLM as a planner. We categorize each work into these categories and outline the difference between those works and ours.

LLM as an Auxiliary Helper This branch of work uses LLM as an auxiliary helper to generate relevant information to help the main model. LM-Nav [32] prompts LLMs with raw navigation instructions and 3 in-context examples to generate a list of landmarks for a vision-language model to infer a joint probability distribution over landmarks and images. However, we show that LLM can be used for more than an auxiliary information generator and can be used to perform planning while being grounded to the environment.

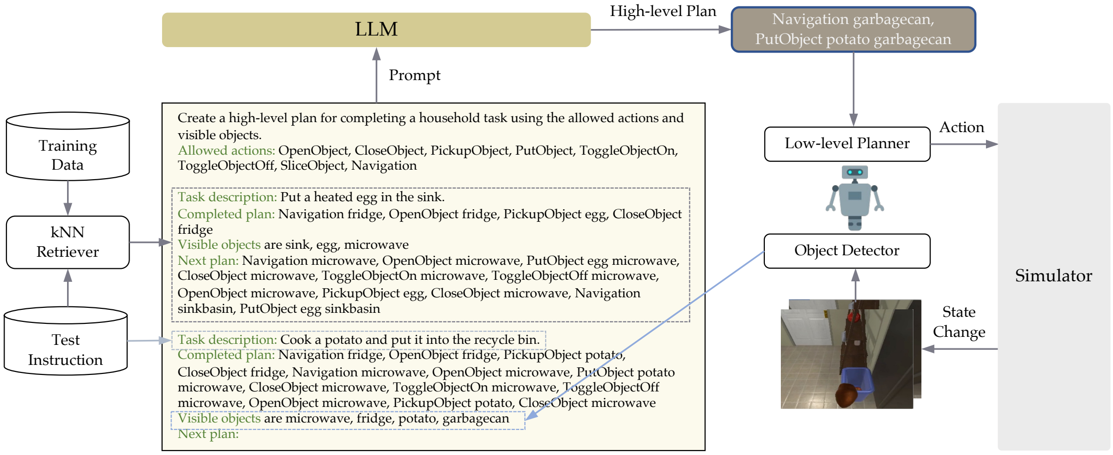
LLM as a Planner This branch of LLM usage focuses on the LLM's ability to generate a plan that is executable in the environment directly or indirectly by using a low-level planner. Several studies have explored the usage of LLM as a planner for embodied agents [1, 24, 13, 41, 12, 35]. Majority of the works assume the availability of admissible actions in the environment and formulate the approach based on that assumption. Some are due to the underlying evaluation setup [13, 35, 24], while others try to train a model to predict a list of admissible actions in the environment [1]. However, such an assumption leads to various implications on practicality: 1) This admissible action list may be hard or infeasible to obtain, especially in partially-observable environments, and 2) the length of the list grows combinatorially *w.r.t.* environment complexity (e.g., # of objects). In contrast, LLM-Planner is a *generative model*. It generates the high-level plan without assuming the knowledge of specifics of the current environment, and dynamically refines the plan based on new observations. To validate our claim, we implement one of the major works, SayCan [1] to our evaluation dataset (ALFRED) and compare the difference in section 5.

Other work [41] that does not evaluate under that assumption uses LLM as a static generator for high-level plans. However, we take one step further and propose a LLM-Planner without the aforementioned assumptions. LLM-Planner is able to ground the LLM to the current environment by using a pre-trained vision model. Next, it can directly predict HLP without relying on a list of admissible actions in the current environment. Additionally, LLM-Planner can perform the aforementioned capabilities while re-planning during the task execution to dynamically adapt the high-level plans to the current environment. At last, LLM-Planner is evaluated on a diverse set of tasks in the ALFRED environment, testing the real-life applicability of our approach. With careful prompt design and other techniques for better in-context learning, we show that LLM-Planner can generate complete and high-quality high-level plans that are grounded in the current environment with a fraction of labeled data.

3. Preliminaries

Vision-and-Language Navigation. Embodied instruction following is often also referred as vision-and-language navigation (VLN), though it additionally involves interaction actions and usually features a much longer time horizon than typical VLN tasks (*e.g.*, Room2Room [2]). To be consistent with the literature, we will use these two terms interchangeably. We will primarily focus on the standard ALFRED [34] dataset, which is built on top of the AI2-Thor [15] simulator, but our method can easily generalize to other datasets and environment. We choose ALFRED mainly considering its diversity in task types (7 different task types) and long-horizon tasks (on average 50 actions per task).

The VLN task is defined as following: Given a language instruction I, an agent needs to predict and carry out a sequence of primitive actions in the environment E to accom-



Create a high-level plan for completing a household task using the allowed actions and visible objects.
Allowed actions: OpenObject, CloseObject, PickupObject, PutObject, ToggleObjectOn, ToggleObjectOff, SliceObject, Navigation

Task description: Put a heated egg in the sink.
Completed plan: Navigation fridge, OpenObject fridge, PickupObject egg, CloseObject fridge
Visible objects are sink, egg, microwave
Next plan: Navigation microwave, OpenObject microwave, PutObject egg microwave, CloseObject microwave, ToggleObjectOn microwave, ToggleObjectOff microwave, OpenObject microwave, PickupObject egg, CloseObject microwave, Navigation sinkbasin, PutObject egg sinkbasin

Task description: Cook a potato and put it into the recycle bin.
Completed plan: Navigation fridge, OpenObject fridge, PickupObject potato, CloseObject fridge, Navigation microwave, OpenObject microwave, PutObject potato microwave, CloseObject microwave, ToggleObjectOn microwave, ToggleObjectOff microwave, OpenObject microwave, PickupObject potato, CloseObject microwave
Visible objects are microwave, fridge, potato, garbagecan
Next plan:

Figure 2: Overview of LLM-Planner with prompt design and grounded re-planning.

plish the task. In datasets like ALFRED [34], the instruction I consists of a high-level goal I_H and (optionally) a list of step-by-step instructions I_L. A VLN task can thus be represented by a tuple (I, E, G), where G is the goal test. We consider hierarchical planning models [38] for VLN, which is explored to various extent in several recent studies [26, 3, 33, 36], but none of them considers the few-shot setting or LLMs for planning. In this formulation, planning is modeled in a hierarchical fashion. The high-level planner maps the instruction I into a high-level plan (HLP) $L_h = [g_0, g_1, \cdots, g_T]$, where each subgoal g_i is specified as (high-level action, object). We define a high-level action to be a collection of primitive actions that can complete a single goal-condition in ALFRED [34]. We take the the interaction actions directly from ALFRED and we only add the Navigation action. Therefore, the high-level action space consists of 1 navigation action (Navigation) and 7 interaction actions (PickupObject, PutObject, OpenObject, CloseObject, ToggleOnObject, ToggleOffObject, SliceObject). Similar actions are commonly used in other related work such as SayCan [1] and LM zero-shot planner [13].

The low-level planner maps each subgoal into a sequence of primitive actions $L_l = [a_0, a_1, \cdots, a_{T_i}]$. State-of-the-art VLN methods [26, 3] use a map-based low-level planner and a simple path-finding algorithm to find the target object in the current subgoal from the map. It is important to note that, once the high-level plan L_h is specified, the low-level planning becomes independent of the instruction I. More formally, $P(L_l|I, L_h, E) = P(L_l|L_h, E)$. All the components involved in the low-level planner are either deterministic or trained using synthetic data from the simulator. No paired data involving language instructions is needed.

In-Context Learning/Prompting. Recently, in-context learning (also known as prompting)[4] has drawn great attention with the rise of LLMs. By designing different prompts, LLMs can be adapted to different downstream tasks with a few examples as demonstration without updating any of the parameters. In this work, we explore in-context learning with LLMs for embodied agent planning.

True Few-Shot Setting. While only using a small number of training examples, many few-shot studies use a large validation set for prompt design and model selection [4]. Recent studies [28] have shown that such large validation sets are responsible for overestimation of the efficacy of language models because they create a strong bias for model selection and violate the intended few-shot setting. To avoid such bias, we adhere to the true few-shot setting [28] in which prompt design and model selection is conducted via cross-validation on the same small training set instead of using a separate validation set.

4. LLM-Planner

In this section, we describe our method, LLM-Planner, which leverages LLMs such as GPT-3 (TEXT-DAVINCI-003) to do few-shot grounded high-level planning for embodied agents.

4.1. Overview

LLMs such as GPT-3 are pre-trained to generate natural language. To adapt them as high-level planners, the first step is to design an appropriate prompt to guide them to generate high-level plans. We discuss our prompt design in Section 4.2. The choice of in-context examples is critical for the performance of LLMs, and recent works [28, 9] have shown that dynamically retrieving similar examples for each test example is beneficial. We adopt a k-nearest-neighbor (kNN) retriever to select the in-context examples (Section 4.3). We also use logit biases [10] to further constrain the output space of the LLM to the allowed set of actions and objects. With all the above designs, we have obtained the *static* version of LLM-Planner, which can already generate reasonable HLPs. In Section 4.4, we propose a novel grounded re-planning algorithm to enhance LLMs with the ability to ground to the current environment, which further improves the HLP quality. Finally, we discuss how to integrate LLM-Planner into existing embodied agents to empower them with few-shot planning capabilities in Section 4.5. An overview of LLM-Planner is shown in Figure 2.

4.2. Prompt Design

While GPT-3 is shown to be a powerful few-shot learner in a variety of tasks, its power can only be unleashed with carefully designed prompts that are tailored for the desired behavior. The final HLP quality can be sensitive to minor design choices in the prompt (*e.g.*, how the HLP is presented, or sometimes even the choice of punctuation). Therefore, we identify core components of the prompt and systemically compare different design choices under the true few-shot setting based on leave-one-out cross-validation (LOOCV). The evaluations for some of the key design choices are discussed in Section 5.5 and 5.6.

Our final optimal prompt is shown in Figure 2. The prompt begins with an intuitive explanation of the task and the list of allowable high-level actions. It is then followed by the in-context examples selected by the kNN retriever (Section 4.3). When we provide only the high-level goal instruction to GPT-3, we use the format “Task description: [high-level goal instruction].” When we include the step-by-step instructions, we include another line “Step-by-step instructions: [step-by-step instructions]” following the goal instruction. For dynamic grounded re-planning (Section 4.4), we add the subgoals that have been completed and the list of objects observed so far in the environment after the task description. Finally, we append the test example in the same format that ends with “Next plan:”.

4.3. In-context Example Retrieval

The in-context examples are an important source of task-specific information for the LLM. Different examples could provide different information for the current task. Intuitively, if the current task is to *“cook a potato,”* an in-context example that demonstrates the HLP for *“cooking an egg”* is likely more informative than one that demonstrates how to *“clean a plate.”* Specifically, we use a frozen BERT-base model [7] to evaluate the pairwise similarity between each training example and the current test example. The similarity of two examples is defined based on the Euclidean distance between the BERT embedding of their corresponding instruction. For each test example, we then retrieve the K most similar examples from the small set of paired training examples we have, where K is a hyperparameter that we tune under the true few-shot setting (Section 5.6).

Algorithm 1 Dynamic Grounded Re-planning with LLM-Planner

```
$I \leftarrow$ Instruction
$O \leftarrow$ Set of observed object
$G \leftarrow$ List of completed subgoals so far
$S \leftarrow$ LLM-Planner$(I, O, G)$                ▷ Full HLP
$t \leftarrow 0$                                      ▷ Time step
$k \leftarrow 0$                                      ▷ Subgoal index
$s \leftarrow S[k]$                                   ▷ First subgoal
$a_t \leftarrow$ Low-Level-Planner$(s)$               ▷ First action
while k <len(S) do
    execute $a_t$
    $O_t \leftarrow$ Object-Detector(current camera input)
    $O$.insert$(O_t)$
    if current subgoal $s$ fails or after $n$ time steps then
        $S \leftarrow$ LLM-Planner$(I, O, G)$        ▷ New HLP
        $k \leftarrow 0$
        $s \leftarrow S[k]$
    else if current subgoal $s$ is completed then
        $k \leftarrow k + 1$
        $s \leftarrow S[k]$                           ▷ Get next subgoal
    end if
    $t \leftarrow t + 1$
    $a_t \leftarrow$ Low-Level-Planner$(s)$
end while
```

4.4. Grounded Re-planning

Using LLM-Planner as a *static* high-level planner that only predicts an HLP at the beginning of a task already shows good data efficiency and accuracy. As discussed earlier, however, such static planning lacks grounding to the physical environment and can lead to incorrect objects and unattainable plans (Figure 1). When such issues happen, the agent cannot complete the current subgoal specified in the HLP, which will lead to one of two possible situations: 1) it fails to execute an action (*e.g.*, bumping into a wall or failing to interact with an object), or 2) it takes a long time and still has not completed the current subgoal (*e.g.*, wandering endlessly). Intuitively, knowing the objects in the current environment can be very helpful for addressing both of these issues. For example, knowing that there is a fridge, the LLM may produce an HLP that directs the agent to go to the fridge and try to find a potato in that, because it may have learned the commonsense knowledge that food is likely stored in a fridge during language model pre-training.

To this end, we present a simple but effective way to enhance LLMs with physical grounding by injecting a list of observed objects, which may be detected using the object detector of the embodied agent, from the environment into the prompt (Figure 2). We also add logit biases to these observed objects so LLM-Planner can prioritize producing a plan with those objects if they are relevant for the task.

Based on that, we propose a grounded re-planning algorithm (Algorithm 1) to dynamically update the HLP during the course of completing a task. This is in contrast with most existing work that adopts a similar hierarchical planning model (*e.g.*, [26]), which only predicts a fixed HLP up front and sticks to that no matter what happens during the execution. In our algorithm, re-planning will be triggered under either of two conditions: 1) the agent fails to execute an action, or 2) after a fixed number of time steps. A new continuation of the already-completed partial HLP will be generated by LLM-Planner based on the observed objects, and the agent will carry on with the new plan, which may help it get unstuck.

4.5. Integration with Existing VLN models

We now discuss how to integrate LLM-Planner with the existing models to empower them with the few-shot planning capability. LLM-Planner provides a fairly generic and flexible interface for integration. As shown in Algorithm 1, it only needs the embodied agent to provide an object list and has a low-level planner that can turn the predicted HLP into low-level actions. It has no assumption about the inner working of the agent. For evaluating the end-to-end task completion performance of LLM-Planner, we integrate it with a strong baseline method, HLSM [3], which satisfies such an interface.

5. Experiments

5.1. Dataset

We evaluate the efficacy of LLM-Planner in generating high-level plans using the ALFRED [34] benchmark, a vision-and-language navigation dataset that requires embodied agents to follow instructions and use visual input to complete tasks in a simulated, spatially continuous household environment. The dataset consists of 7 task types spanning across 207 unique environments, 115 different object types, and 4,703 tasks. The task ranges in difficulty from *moving a single object to a new location* to *placing a heated slice of an object into a receptacle*. Each task is accompanied by human-written annotations of a high-level goal and a series of more granular step-by-step instructions, created by human annotators as they watched expert demonstrations of the tasks. Due to the noise in the natural language instructions and the complexity of planning required to complete such long-horizon tasks, ALFRED is a challenging test of an embodied agent's ability to produce robust and accurate plans.

5.2. Metrics

We report two main metrics used by ALFRED and one metric created by us to calculate the high-level planning accuracy. Success rate (SR) is the percentage of tasks fully completed by the agent. A task is only considered complete when all the subgoals are completed. Goal-condition success rate (GC) is the percentage of completed goal-conditions. Goal-conditions are defined as state changes necessary to complete the task. For example, in the task *"Slice a heated bread,"* bread being sliced and bread being heated are both goal-conditions.

To directly evaluate high-level planning, we introduce a new metric named high-level planning accuracy (HLP ACC), *i.e.*, the accuracy of the predicted HLP compared to the ground-truth HLP. For the static planning setting, we compare the generated HLP with the ground-truth HLP and deems a plan as incorrect if it does not perfectly match the ground truth, and correct otherwise. For the dynamic planning setting, we report a range because we cannot fully separate LLM-Planner's performance with the low-level controller choice because we do not have access to an oracle low-level controller. The lower bound is the HLP accuracy of the full generated plan regardless of whether it was executed successfully by the low-level controller (*i.e.* same as evaluating static HLP). The upper bound is the HLP accuracy of the predicted HLP that was successfully executed in the environment by the low-level controller when a task has ended (*i.e.* a task success or a catastrophic failure).

5.3. Implementation Details

We choose 100 examples for our LLM-Planner among 21,023 ALFRED training examples. We apply random stratified sampling to ensure we have a fair representation of all 7 task types in the 100-example set. For the kNN retriever, we use the pretrained BERT-base-uncased model from the Huggingface Transformers Library [39]. For the LLM, we use the public GPT-3 [4] API with 9 in-context examples chosen from the 100 training examples by the kNN retriever. We set the temperature to 0 and apply a logit bias of 0.1 to all allowable output tokens. The object list for grounded re-planning is retrieved from the object detector. Specifically, we use the pretrained object detector from HLSM's perception model. We only include objects with a label confidence more than 80% to reduce noise. It is worth noting that we can potentially use any object detector to obtain the object list, and we only use HLSM's perception model to save computation cost and time. To avoid violating our few-shot assumption, we use the pretrained navigation, perception, and depth model from HLSM which are trained using only synthesized trajectories from the simulator, without any paired training data involving natural language instructions or human annotations.

We compare with two main baseline models, HLSM [3] and FILM [26] They are also hierarchical planning models and achieve strong performance on the ALFRED leaderboard. We directly replace the trained high-level planner for both models with our LLM-Planner and did not modify any other parts. In addition, we re-train these models to compare with LLM-Planner under the same few-shot shot setting. We also compare with several other published baselines models that are trained with the full data. Additionally, we also implement SayCan[2] to ALFRED and compare under the same few-shot setting as LLM-Planner. Further implementation details can be found in the supplementary.

SayCan [1] is a ranking based high-level planner that requires a list of admissible actions and ranks them using the LLM. To make it possible for SayCan to work in the complex, partially-observable environments in ALFRED, we give it an *unfair competitive advantage*—it knows all the objects and affordances in the current environment *a priori* to compile the list of skills. We also equip SayCan with the same kNN retriever from LLM-Planner, which was not needed in their original paper because of the less diverse tasks. More details on the implementation is provided in the supplementary materials.

Other Baselines. For other baselines included in Table 1, we retrieve the results directly from the published version of the corresponding paper. If the ALFRED leaderboard entry is better than the numbers in the original paper, we report the higher.

5.4. Main Results

The main results are shown in Table 1. We first compare the performance of HLSM when using our LLM-Planner as the high-level planner compared with its native version, which is trained using the full training set of ALFRED. We find that LLM-Planner's few-shot performance is competitive to the original HLSM, and outperforms several recent baselines such as E.T., HiTUT, and M-TRACK, despite using less than 0.5% of paired training data. On the other hand, when trained using the same 100 examples (*i.e*., retraining HLSM's high-level planner), HLSM (and FILM as well) can barely complete any task successfully. Furthermore, the results show that SayCan still largely underperforms LLM-Planner despite the access to the full environment information. Another significant difference is *cost and efficiency*. Because of SayCan's ranking nature, it needs to call the LLM many more times than a generative model like LLM-Planner: *LLM-Planner calls GPT-3 avg. 7 times per task and SayCan calls it 22 times*, even with oracle knowledge of the current environment to shrink the skill list. Lastly, we see a considerable improvement from grounded re-planning over static planning, especially in the goal instruction only setting, where it improves 1.83% SR in the unseen test split. This confirms the effectiveness of the grounded re-planning. But we also note that there is still a large room for further improvement.

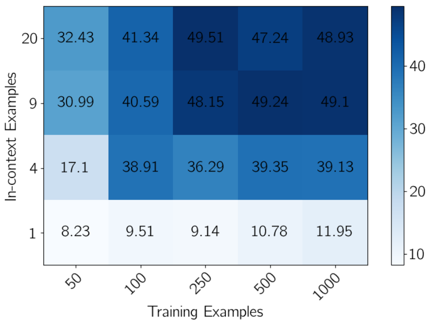


Figure 3: LOOCV HLP accuracy for varying number of in-context examples and training examples.

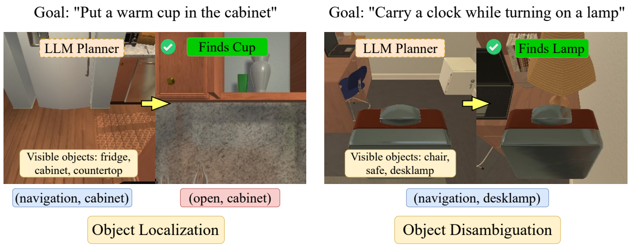

Goal: "Put a warm cup in the cabinet"
LLM Planner
Finds Cup
Visible objects: fridge, cabinet, countertop
(navigation, cabinet)
(open, cabinet)
Object Localization
Goal: "Carry a clock while turning on a lamp"
LLM Planner
Finds Lamp
Visible objects: chair, safe, desklamp
(navigation, desklamp)
Object Disambiguation

Figure 4: Case studies for LLM-Planner.

5.5. Ablation Studies

We conduct an ablation study on different components of LLM-Planner to validate their effectiveness. We follow the LOOCV process and use only the high-level planning accuracy to determine our choices. Results from this study are in Table 2. We first ablate the kNN retriever module, by replacing it with a retriever that randomly selects in-context examples from the 100 example set. Results in Table 2 show that this leads to a significant drop in performance, confirming the necessity of dynamic retrieval.

Furthermore, we find that enabling logit biases to favor objects that appear in the environment lead to a decent boost in the high-level planning accuracy. Having LLM-Planner favor objects that appear in the environment makes it more robust in the cases where the instruction is ambiguous or objects are referred with different names. For example, for an instruction *"Turn on the lamp,"* different types of lamps, *e.g*., table lamps or floor lamps, could be. By enabling logit biases to favor objects that appear in the environment (*e.g*., TableLamp), we can correctly guide LLM-Planner to output (TurnOnObject, TableLamp). Another example is when the instruction refers to RecycleBin but the object name used in the environment is GarbageCan. In this case, using logit biases can correctly guide LLM-Planner to output the relevant and correct objects.

5.6. Fine-grained Analyses

Effect of Number of Examples. For the main experiments, we chose 100 as the number of training examples without

[2] https://github.com/google-research/google-research/tree/master/saycan

Model	Test Unseen		Test Seen		Valid Unseen			Valid Seen		
	SR	**GC**	**SR**	**GC**	**SR**	**GC**	**HLP ACC**	**SR**	**GC**	**HLP ACC**
Full-data setting: 21,023 (instruction, trajectory) pairs										
Goal instruction only										
HiTUT [40]	11.12	17.89	13.63	21.11	10.23	20.71	–	18.41	25.27	–
HLSM [3]	20.27	27.24	25.11	35.79	**18.28**	**31.24**	31.24 – **70.17**	29.63	38.74	38.74 – **77.64**
Step-by-step instructions										
E.T. [27]	8.57	18.56	**38.42**	**45.44**	7.32	20.87	–	**46.59**	**52.92**	–
HiTUT [40]	13.87	20.31	21.27	29.97	12.44	23.71	–	25.24	34.85	–
M-TRACK [36]	16.29	22.60	24.79	33.35	17.29	28.98	–	26.70	33.21	–
FILM [26]	27.80	**38.52**	28.83	39.55	–	–	54.93	–	–	60.86
LEBP [18]	**28.30**	36.79	28.97	36.33	–	–	–	–	–	–
Few-shot setting: 100 (instruction, high-level plan) pairs										
Goal instruction only										
LLM-Planner (Static) + HLSM	11.58	18.47	13.05	20.58	11.10	22.44	28.67	11.82	23.54	27.45
LLM-Planner + HLSM	13.41	22.89	15.33	24.57	12.92	25.35	33.81 – 55.85	13.53	28.28	35.08 – 54.33
Step-by-step instructions										
HLSM [3]	0.61	3.72	0.82	6.88	0.00	1.86	0.00	0.13	2.82	0.00
FILM [26]	0.20	6.71	0.00	4.23	0.00	9.65	0.00	0.00	13.19	0.00
SayCan [1]	-	-	-	-	9.88	22.54	37.57	12.30	24.52	35.15
LLM-Planner (Static) + HLSM	15.83	20.99	17.87	23.10	14.26	26.12	43.24	15.84	25.43	39.87
LLM-Planner + HLSM	**16.42**	**23.37**	**18.20**	**26.77**	**15.36**	**29.88**	46.59 – **68.31**	**16.45**	**30.11**	50.33 – **71.84**

Table 1: Main results on the ALFRED dataset. "(Static)" means the static planning setting, otherwise it is the default dynamic setting with grounded re-planning. Some methods support using only the goal instruction or additionally using the step-by-step instructions. We compare under both configurations. We could not evaluate SayCan on the test split because ALFRED prohibits using the test metadata, which is needed by SayCan for compiling the admissible actions.

	LOOCV HLP accuracy
Best Model	40.59
– **kNN Retriever**	17.48
– **Logit Biases**	38.10
– **Both**	13.43

Table 2: Ablation of LLM-Planner's components.

any cross-validation because it is our target number for the few-shot setting. We then use LOOCV to select the best number of in-context examples using the 100 sampled training examples. However, we are still curious about the effect of different choices, so we conduct this analysis after the main experiments to show the sensitivity to these hyperparameters. It is worth noting that the design choices for the main experiments are not informed by this analysis, to respect the true few-shot setting.

As shown in Figure 3, HLP accuracy generally improves with more training examples, though we start to get a diminishing return around 250 training examples. A decent improvement can be expected for the main experiments in Table 1 if we choose to use more training examples (*e.g.*, 250). Furthermore, we find that 9 is generally a good number for in-context examples. Although adding more in-context examples could still improve the performance slightly, it may not be meaningful enough to justify the additional cost. Not too surprisingly, more in-context examples is more beneficial when there is less training examples, because there are less useful examples to retrieve from.

Case Studies. In Figure 4, we show two examples where LLM-Planner helps with object localization and disambiguation through grounded re-planning. For the first case, even using only the high-level goal instruction, LLM-Planner correctly predicts that the cup is likely located in the cabinet after failing to find a cup but observing a cabinet in the environment. This shows LLM-Planner can achieve a similar effect to what the semantic map tries to achieve in FILM [26], *i.e.*, predicting plausible location for target objects. For the second case, we show that LLM-Planner can correctly ground the word *"lamp"* to the desklamp in the environment.

6. Conclusion

We demonstrate a novel high-level planner based on large language models for embodied agents that can be used in diverse, partially-observable, and complex environments. It can also dynamically re-plan based on environmental perception to produce more grounded plans. Our work can dramatically reduce the amount of human annotations needed for learning the instruction following task. Furthermore, it opens a new door for developing versatile and extremely sample-efficient embodied agents by harnessing the power of large language models and enhancing them with physical grounding. Promising future directions include exploring other LLMs such as Codex [5], better prompt design, and more advanced methods for grounding and dynamic re-planning.

Acknowledgement

The authors would like to thank the colleagues from the OSU NLP group for their thoughtful comments. This research was supported by ARL W911NF2220144.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022. 1, 2, 3, 4, 7, 8, 12

[2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sunderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, pages 3674–3683, 2018. 1, 3

[3] Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *Conference on Robot Learning*, pages 706–717. PMLR, 2022. 2, 3, 4, 6, 8, 12, 14

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. 1, 2, 4, 6

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Arun Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021. 9

[6] G.N. Desouza and A.C. Kak. Vision for mobile robot navigation: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):237–267, 2002. 2

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. 3, 5, 12

[8] Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. In *Neural Information Processing Systems (NeurIPS)*, 2018. 1

[9] Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. 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)*, pages 3816–3830, Online, Aug. 2021. Association for Computational Linguistics. 2, 4

[10] Bernal Jiménez Gutiérrez, Nikolas McNeal, Clay Washington, You Chen, Lang Li, Huan Sun, and Yu Su. Thinking about gpt-3 in-context learning for biomedical ie? think again. *arXiv preprint arXiv:2203.08410*, 2022. 2, 4

[11] Yicong Hong, Qi Wu, Yuankai Qi, Cristian Rodriguez-Opazo, and Stephen Gould. A recurrent vision-and-language bert for navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1643–1653, June 2021. 1, 3

[12] Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation. *arXiv preprint arXiv:2210.05714*, 2022. 3

[13] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022. 1, 2, 3, 4

[14] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*, 2022. 2

[15] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. 3

[16] Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-Across-Room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2020. 1

[17] Xiujun Li, Chunyuan Li, Qiaolin Xia, Yonatan Bisk, Asli Celikyilmaz, Jianfeng Gao, Noah A. Smith, and Yejin Choi. Robust navigation with language pretraining and stochastic sampling. 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)*, pages 1494–1499, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. 1, 3

[18] Hao Liu, Yang Liu, Hong He, and Hang Yang. Lebp - language expectation & binding policy: A two-stream framework for embodied vision-and-language interaction task learning agents. *ArXiv*, abs/2203.04637, 2022. 3, 8

[19] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114, Dublin, Ireland and Online, May 2022. Association for Computational Linguistics. 2

[20] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, sep 2022. 2

[21] Xiaotian Liu, Hector Palacios, and Christian Muise. A planning based neural-symbolic approach for embodied instruction following. *Interactions*, 9(8):17, 2022. 1

[22] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. 3

[23] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems*, 32, 2019. 3

[24] Yujie Lu, Weixi Feng, Wanrong Zhu, Wenda Xu, Xin Eric Wang, Miguel Eckstein, and William Yang Wang. Neuro-symbolic procedural planning with commonsense prompting, 2022. 2, 3

[25] Arjun Majumdar, Ayush Shrivastava, Stefan Lee, Peter Anderson, Devi Parikh, and Dhruv Batra. Improving vision-and-language navigation with image-text pairs from the web. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI*, pages 259–274, 2020. 1, 3

[26] So Yeon Min, Devendra Singh Chaplot, Pradeep Kumar Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. FILM: Following instructions in language with modular methods. In *International Conference on Learning Representations*, 2022. 2, 3, 4, 6, 8, 12

[27] Alexander Pashevich, Cordelia Schmid, and Chen Sun. Episodic Transformer for Vision-and-Language Navigation. In *ICCV*, 2021. 1, 3, 8

[28] Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. 2, 4

[29] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018. 1

[30] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. 13

[31] Timo Schick and Hinrich Schütze. It's not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352, 2021. 2

[32] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Robotic navigation with large pre-trained models of language, vision, and action. In *6th Annual Conference on Robot Learning*, 2022. 3

[33] Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1713–1726, Dublin, Ireland, May 2022. Association for Computational Linguistics. 2, 3, 4, 13

[34] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1, 2, 3, 4, 6

[35] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. 2022. 1, 2, 3

[36] Chan Hee Song, Jihyung Kil, Tai-Yu Pan, Brian M. Sadler, Wei-Lun Chao, and Yu Su. One step at a time: Long-horizon

vision-and-language navigation with milestones. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15482–15491, June 2022. 4, 8

[37] Alessandro Suglia, Qiaozi Gao, Jesse Thomason, Govind Thattai, and Gaurav Sukhatme. Embodied bert: A transformer model for embodied, language-guided visual task completion, 2021. 1, 3

[38] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999. 2, 4

[39] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. 6

[40] Yichi Zhang and Joyce Chai. Hierarchical task learning from language instructions with unified transformers and self-monitoring. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4202–4213, Online, Aug. 2021. Association for Computational Linguistics. 1, 8

[41] Kai Zheng, KAI-QING Zhou, Jing Gu, Yue Fan, Jialu Wang, Zonglin Li, Xuehai He, and Xin Eric Wang. Jarvis: A neurosymbolic commonsense reasoning framework for conversational embodied agents. *ArXiv*, abs/2208.13266, 2022. 3

Appendices

In this supplementary material, we present additional details and clarifications that are omitted in the main text due to space constraints.

- Appendix A: Additional Model Implementation Details
- Appendix B: Comparison with (SL)^3 on ALFRED.
- Appendix C: Prompt design choices and prompt selection under true few-shot setting (cf. section 4.2 in the main paper).
- Appendix D: Additional fine-grained analyses (cf. section 5 in the main paper).

A. Additional Model Implementation Details

A.1. HLSM

HLSM [3] consists of three components: a semantic voxel map, a high-level planner, and a low-level planner. First, a 3D semantic voxel map is constructed by applying semantic segmentation and depth estimation to the visual inputs, which stores the agent's and the objects' real-time locations. Next, the high-level planner takes the language instructions, the semantic map encoding, and the previous subgoal history to predict the next subgoal. Lastly, the low-level planner is a mixture of deterministic algorithms and learned components (*e.g.*, learning a yaw and pitch angle to face the object). HLSM first processes the sensory image input to create/update a map, which is used as an input to the high-level planner along with the language instructions to predict the next subgoal. Finally, the low-level planner maps the subgoal into a sequence of primitive actions.

To adapt HLSM to the few-shot setting, we need to re-train the components of the model that need paired trajectory-instruction data for training. For HLSM, paired data was only used for training the high-level controller. Therefore, we re-train the high-level controller with the same 100 training examples we use for LLM-Planner. Specifically, we use the same set of hyperparameters as HLSM. While the original HLSM focuses on the goal instruction only setting, we found that the step-by-step instructions are essential for the few-shot setting, so we concatenate goal instruction with step-by-step instructions for re-training HLSM's high-level planner. We leave the other components intact, which are downloaded from the official codebase.[3]

A.2. FILM

FILM [26] consists of four components: a semantic map, a semantic search policy, a template-based high-level planner, and a low-level planner. At the beginning of each task, five separate BERT-based classifiers [7] are used to predict five parameters (task type, target objects, receptacles, parent objects, and whether slicing is needed), each of which takes the goal and optionally the step-by-step instructions as input to predict the respective parameter. FILM then generates the high-level plan by choosing a pre-defined template based on the predicted task type and filling the other parameters into the template. In addition, the semantic map is updated at each time step with the sensory image inputs. At every 25 steps, the semantic search policy predicts the coordinates of the target object on the semantic map, which are then used by a deterministic low-level planner to decide on a low-level plan to navigate from the current location to the target object's location.

Only the BERT-based classifiers need the language-related data for training. Therefore, to adapt FILM to the few-shot setting, the five BERT-based classifiers are retrained with the same 100 training examples used by the LLM-Planner. Similar to HLSM, we concatenate the goal and the step-by-step instructions as input to the BERT-based classifiers. We use default hyperparameters for BERT models that are found in the paper. We use the predictions from these models to generate the high-level plans with the same pre-defined templates in FILM. We leave other components intact, which are downloaded from the official codebase.[4]

A.3. SayCan

SayCan [1] consists of 3 components: an LLM ranker, set of skills, and a value function. We use the LLM ranker adapted from SayCan's codebase [5] with the same settings (*e.g.* temperature and log probability) and use GPT-3 (text-davinci-003) as the choice of LLM. First, SayCan generates a list of skills and their affordance score in the current environment using a pre-trained value function. Then, it prompts the LLM with natural language description of each skill and generates a probability that represents how relevant it is to the task success. Finally, SayCan combines the skill's LLM probability and the affordance score to choose which skill to execute.

To adapt SayCan to ALFRED, we need to define a *skill* in the ALFRED environment. From SayCan, a *skill* is defined as *"atomic" behaviors that are capable of low-level visuomotor control*. Each skill can perform a short task, such as picking up a particular object. This is identical to our definition of high-level plan in §3, therefore we treat each skill as analogous to the (high-level action, object) pair. This formulation allows us to use the same low-level controller we used for LLM-Planner. Furthermore, the value function is an another important concept for the SayCan.

[3] https://github.com/valtsblukis/hlsm

[4] https://github.com/soyeonm/FILM

[5] https://github.com/google-research/google-research/tree/master/saycan

Options	Task Introduction	Goal Instruction	Step-by-step Instructions	Plan List	Object List	Retrieval Message
Default	Create a high-level plan for completing a household task using the allowed actions and visible objects. Allowed actions are [action list]	Task description: [goal instruction]	Step-by-step instructions: [instructions]	(Completed, Next) plan: [subgoals]	Visible objects are [objects]	Next plan:
Punctuation	(”PickupObject”) (PickupObject) **PickupObject**			(”PickupObject”, ”Apple”) (PickupObject, Apple) **PickupObject, Apple**		
Naturalization	**PickupObject** Pickup Pick up			**PickupObject** Pickup Pick up		
Delimiter			**Pick up, go to** Pick up. Go to. Pick up \n Go to	**Pickup, Navigate** Pickup. Navigate Pickup \n Navigate	**Apple, orange** Apple. orange Apple \n Orange	

Table 3: For each element in our prompt design, we list the default phrasing. For the representation of actions, objects, and lists, we additionally experiment with different choices of punctuation, naturalization, and the delimiter between elements in a list. We select the optimal prompt design using LOOCV on the 100 training examples. The chosen options are highlighted in bold.

Task Type	HLP Accuracy	
	Valid Unseen	Valid Seen
Pick & Place	51	46
Stack & Place	38	25
Place Two	39	45
Examine	44.4	49
Heat & Place	36	48
Cool & Place	43	46
Clean & Place	48.8	32

Table 4: Static LLM-Planner's high-level planning accuracy breakdown by task type.

The value function predicts how likely an individual skill is to be executable in the current environment. However, due to the resource constraint we were not able to generate the data and train a policy for the value function. On the other hand, we decided to give SayCan an unfair advantage: we use the ground truth object information to construct an oracle value function. Additionally, instead of iterating through a list of all possible (high-level action, object), we shrink the size of the skill to contain only the object type available in the current environment. As we described in §5.3, this gives SayCan an *unfair competitive advantage* by giving it the oracle knowledge of all objects and affordances in the current environment *a priori* to compiling the list of skills. Even though SayCan can shrink the skill space with the extra knowledge, SayCan's ranking nature calls LLM significantly more times than a generative model like LLM-Planner. In fact, LLM-Planner calls GPT-3 avg. 7 times per task and SayCan calls it 22 times even with the oracle knowledge of the current environment to shrink the skill list.

B. Comparison with (SL)^3 on ALFRED

(SL)^3 [33] is a recent hierarchical planning model that is also evaluated on the ALFRED benchmark. It randomly samples 10% of ALFRED's training data for training. The high-level planner is based on a pre-trained T5-small [30] model, which is fine-tuned to generate high-level plans from the goal instruction. The low-level planner is another fine-tuned T5-small model, which is tasked of generating a low-level plan for each subgoal in the high-level plan. Both goal and step-by-step instructions are needed for training, but only goal instructions are needed at inference time.

We could not compare (SL)^3 under the same few-shot setting as LLM-Planner because its code was not publicly available at the time of submission. However, we would like to highlight that our method achieves comparable performance on the validation set despite using only less than $1/20$ of training data than (SL)^3 (0.5% vs. 10% of ALFRED's training data).

Training Size	50	100	500	1k	10k	Full (21k)
LLM-Planner	10.06	15.36	16.59	16.46	16.83	17.80
HLSM	0.00	0.00	0.37	1.59	9.51	18.28

Table 5: Scaling experiment of LLM-Planner and HLSM on valid unseen. Metric used is the task success rate.

C. Prompt Design Choices

In-context learning with GPT-3 could be sensitive to the prompt design. In Table 3, we show different prompt design choices we have experimented for LLM-Planner. We structure our prompt into six consecutive parts: task introduction, goal instruction, step-by-step instruction, plan list, object list, and retrieval message. For each part, we have a default phrase and a list of additional options to try on top of the default phrasing signified as []. All the options listed only modify the phrase that goes in []. First, we try adding punctuation marks around actions and object. Next, we naturalize each action name as a plain English text. Lastly, we experiment with finding the optimal delimiter between action list and step-by-step instruction list. We compared comma, period, and newline inserted between the sentences. The best prompt was chosen from the LOOCV accuracy for high-level plans and is bolded.

D. Additional Fine-Grained Analyses

D.1. HLP Accuracy by Task Type

We show LLM-Planner's high-level planning (HLP) accuracy breakdown by task type in Table 4. Because it is difficulty to determine a single value for the HLP accuracy for dynamic LLM-Planner, here we focus on the static version, but the HLP accuracy of the dynamic version generally correlates well with that of the static version. From the results, we observe that the results do not depend much on the difficulty of the task. For example, the task *"Stack & Place"* is often considered as the most difficult task based on the success rate of state-of-the-art models, but LLM-Planner's HLP accuracy is similar to those of easier tasks such as *"Place two"*. We find that LLM-Planner is not overly sensitive to the complexity of tasks. This suggests that it could generalize well to different types of tasks with only a few in-context examples.

D.2. End-to-End Performance by Task Type

We show the end-to-end performance breakdown by task type of dynamic LLM-Planner + HLSM in Figure 5. As a reference, we also compare with HLSM and FILM trained with the full training set of ALFRED. Keep in mind that this is not apples-to-apples comparison because LLM-Planner is under the few-shot setting. Despite that, we can see that LLM-Planner + HLSM achieves comparable performance with HLSM, and the distribution of the two are similar. This is likely due to the shared low-level planner and object detector, which introduce a similar error profile. This again shows that our few-shot high-level planner is as good as HLSM's high-level planner that is trained with the full training set. On the other hand, it also shows that there is still a large room to improve by using better low-level planners and object detectors. For example, even though our HLP accuracy for *"Heat & Place"* is 36% as shown in Table 4, we could only get 1.8% success rate due to the object detector from HLSM often failing to detect the *"microwave"*. If we use FILM's low-level planner and object detector, we may be able to achieve much better performance on this task.

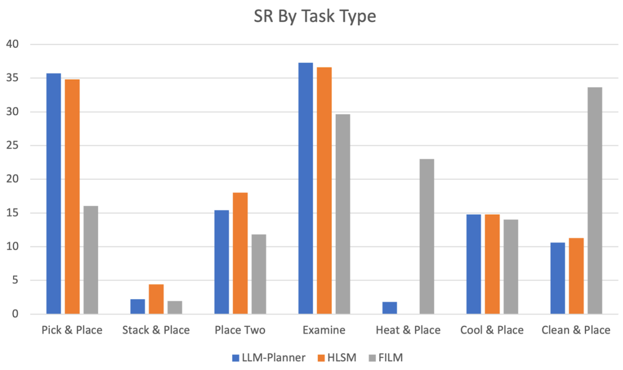


Figure 5: Success rate by task type on ALFRED valid unseen split.

D.3. Scaling Comparison with HLSM

We show LLM-Planner's scaling experiments in comparison with the HLSM [3] in Table 5. We can see that LLM-Planner significantly outperforms HLSM on almost all data size except for the full data setting. This result shows that LLM-Planner is more data-efficient across the board compared to the existing methods. Even with the full data setting, LLM-Planner only falls behind 0.48 SR compared to the HLSM. Our work can dramatically reduce the amount of human annotations needed for learning the task while maintaining a similar performance.