Appendices

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

- Appendix A: ALFRED Statistics.
- Appendix B: Additional Model Implementation Details.
- Appendix C: Comparison with $(SL)^3$ on ALFRED.
- Appendix D: Prompt design choices and prompt selection under true few-shot setting (cf. Section 4.2 in the main paper).
- Appendix E: Scaling Comparison with HLSM. (cf. Section 5 in the main paper).

A. ALFRED Statistics

ALFRED [7] is a large scale dataset containing 7 task types spanning across 207 unique environments, 115 different object types, and 4,703 tasks. It is one of the largest dataset in terms of number of unique environments, object types, and task number in the embodied instruction following task that involves object interactions and state changes.

B. Additional Model Implementation Details

B.1. HLSM

HLSM [2] 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.[1]

B.2. FILM

FILM [4] 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 [3] 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 re-trained 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.[2]

B.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 [3] 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

[1] https://github.com/valtsblukis/hlsm
[2] https://github.com/soyeonm/FILM
[3] 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 1: 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.

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. 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.

C. Comparison with (SL)^3 on ALFRED

(SL)^3 [6] 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 [5] 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 2: Scaling experiment of LLM-Planner and HLSM on valid unseen. Metric used is the task success rate.

D. Prompt Design Choices

In-context learning with GPT-3 could be sensitive to the prompt design. In Table 1, 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.

E. Scaling Comparison with HLSM

We show LLM-Planner's scaling experiments in comparison with the HLSM [2] in Table 2. 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.

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] 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. 1, 3

[3] 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. 1

[4] 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. 1

[5] 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. 2

[6] 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

[7] 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