Long-Horizon Planning for Multi-Agent Robots in Partially Observable Environments

Siddharth Nayak [1] **Adelmo Morrison** [1] **Marina Ten Have** [1] **Vittal Thirumalai** [1] **Jackson Zhang** [1] **Darren Chen** [1]
Aditya Kapoor [2] **Eric Robinson** [3] **Karthik Gopalakrishnan** [4] **James Harrison** [5] **Anuj Mahajan**† [6]
Brian Ichter‡ [5] **Hamsa Balakrishnan** [1]

Abstract

Language Models (LMs) excel in understanding natural language which makes them a powerful tool for parsing human instructions into task plans for autonomous agents. Unlike traditional planning methods that rely on domain knowledge and handcrafted rules, LMs generalize from diverse data and adapt to various tasks with minimal tuning, acting as a compressed knowledge base. However, LMs in their standard form face challenges with long-horizon tasks, particularly in partially observable multi-agent settings. We propose an LM-based Long-Horizon Planner for Multi-Agent Robotics (LLaMAR), a cognitive architecture that employs a plan-act-correct-verify framework. It achieves state-of-the-art results in partially observable long-horizon planning tasks without relying on privileged information from oracles. Experiments show that LLaMAR achieves a 30% higher success rate compared to other state-of-the-art LM-based multi-agent planners in household tasks of varying complexity in the AI2-THOR environment.

1. Introduction

Developing embodied agents that assist humans poses a significant challenge, especially when multiple embodied agents are involved and communication happens using natural language. Recent works (Ichter et al., 2023; Huang et al., 2023b;c; Singh et al., 2023; Liang et al., 2022; Lin et al., 2023; Shah et al., 2022; Huang et al., 2023a; 2022) have shown that LMs can effectively use natural language instructions to generate task plans for robots. However, most studies focus on single-agent long-horizon task planning. To address this research gap, we propose a centralized LM-based cognitive framework in which decisions are made simultaneously for all agents based on their local observations. This framework is inspired from the centralized multi-agent system framework (CMAS) proposed in (Chen et al., 2023). Leveraging the ability of pre-trained LMs to generalize across diverse tasks, we aim to use them for long-horizon embodied multi-agent task planning.

Integrating an LM-based *plan-act-correct-verify* framework enables a robust and adaptive approach to multi-agent task planning in dynamic, partially observable environments that allows agents to: (1) plan subtasks required to complete the langugae instructed task, (2) select high-level actions for each agent to complete the proposed subtasks, (3) identify and correct high-level actions that fail in execution, and (4) verify subtask completion based on successfully executed high-level actions. Unlike existing methods, our approach uses real-time execution feedback, observations, and agent histories to iteratively refine action planning and execution. This allows agents to adjust strategies based on action execution insights, effectively handling failures without needing perfect environmental knowledge or oracle feedback. The correction and verification process in our cognitive architecture (Arora & Kambhampati, 2023; Sengar et al., 2022) is grounded in the environment's reality, which sets it apart from LM self-verification methods that reason on their perception of the environment reality (Valmeekam et al., 2023a). This framework enhances agents' ability to complete complex, long-horizon multi-agent tasks, yielding substantial improvement over state-of-the-art methods.

Similar to our approach, recent works (Kannan et al., 2023; Wang et al., 2024; Singh et al., 2024; Zhang et al., 2024; Yu et al., 2023; Mandi et al., 2023; Chen et al., 2023) utilize LMs for multi-agent planning, often adopting a hierarchical decision-making structure. The LMs are used for high-level planning to determine subtasks, sometimes in conjunction with planning domain definition language (PDDL) that together with the LM planner, functions as a feasibility solver. Specific actions are executed using low-level policies pretrained through reinforcement learning, behavior cloning, or

[1]Massachusetts Institute of Technology, Cambridge, USA [2]TCS, India [3]USAF-MIT AI Accelerator [4]Stanford, USA [5]Google, San Fransisco, USA [6]Apple, Cupertino, USA. Correspondence to: Siddharth Nayak <sidnayak@mit.edu>. †Work done outside Apple. ‡Now at Physical Intelligence.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

heuristic approaches. While these methods effectively use LMs as high-level planners, they assume perfect low-level primitive action policies and utilize privileged environmental information. By contrast, LLaMAR does not assume perfect knowledge of the environment, does not rely on oracle feedback, and does not assume perfect execution of low-level primitive policies bestowing itself as a good candidate for a real-world planner for robots.

The main contributions of this paper are:

LLaMAR: An LM-based iterative planning framework for long-horizon, multi-objective tasks in partially observable environments, with the following key features:

- It operates without prior knowledge of the environment, allowing agents to explore and make decisions based on new observations.
- It is capable of self-evaluation of outcomes without relying on oracles for feedback, enabling independent identification and correction of action failures.

2. Background

Problem Setting: We consider a multi-robot setting that perform a series of everyday tasks in a home-like environment that typically require long-horizon planning (100 low-level actions) to achieve the goal. Our objective is to compute plans for a robot team to execute high-level language instructions, $\mathcal{I}$. We formalize these tasks as partially observable Markov decision processes (POMDP) (Puterman, 1994; Kaelbling et al., 1998), denoted as $\langle N, \mathcal{I}, \{\mathcal{O}_i\}, \{\mathcal{A}_i\}, \mathcal{G}, T \rangle$. N is the number of agents and $\mathcal{I}$ is the high-level language instruction set. Here, $o \in \mathcal{O}$ denotes the observation set for all agents. Particularly, $o_i \in \mathcal{O}_i$ is the observation set of agent i, that captures incomplete environment state information. $a \in \mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \cdots \mathcal{A}_N$ represents the joint action space which comprises of different categories of high-level actions $\mathcal{A} = \mathcal{A}_{NAV} \cup \mathcal{A}_{INT} \cup \mathcal{A}_{EXP}$. $\mathcal{A}_{NAV}$, $\mathcal{A}_{INT}$ and $\mathcal{A}_{EXP}$ are the joint navigation actions (eg: `NavigateTo(location)`, joint interaction actions (eg: `PickUp(object)`) and joint exploration actions (refer C.2) respectively. Each high-level action is associated with a pre-trained low-level primitive policy. All agents execute actions *synchronously* at every high-level decision step. $\mathcal{G} = \{g_1, \cdots, g_k\}$ defines the subtasks that the agents need to accomplish to achieve the language instruction task. T is the length of the planning horizon.

Environment: We use AI2Thor (Kolve et al., 2017) to simulate real-world tasks due to its versatile interactions and photorealistic rendering. Our method, free from parametric training, can also be applied in environments like VirtualHome (Puig et al., 2018), Habitat (Puig et al., 2023), and ThreeDWorld (Gan et al., 2021), possibly extending beyond household domains (Baghel et al., 2021). We address a rearrangement task (Batra et al., 2020) with N collaborating agents that explore to gather task-relevant information. Unlike previous approaches (Zhang et al., 2024; Kannan et al., 2023; Wang et al., 2024), we do not rely on an oracle or preset condition checks for subtask validation. Appendix C has more details on observation and action spaces.

3. Approach

We describe our approach in this section. Figure 1 illustrates LLaMAR's architecture comprising four modules: *Planner*, *Actor*, *Corrector*, and *Verifier*, each an LM with a distinct role. Prior work (Prasad et al., 2023) shows that splitting roles across different LMs improves performance in sequential decision-making. Our initial experiments confirm that LMs tasked with reasoning about multiple inputs and providing long outputs perform poorly. We iterate through these four modules at every high-level decision step. The pseudocode for our approach is in Appendix 2. We define some key notation below:

- **Memory** $\mathcal{M}$: A textual description of the joint memory of all agents, summarizing past observations, high-level actions, plausible reasons for action failures, and specific subtasks that each agent is attempting to solve.
- **Open Subtasks** $\mathcal{G}_O \subset \mathcal{G}$: Feasible subtasks proposed by the *Planner* LM to achieve the environment task that are yet to be accomplished by the agents.
- **Completed Subtasks** $\mathcal{G}_S \subset \mathcal{G}$: Subtasks completed by the agents.
- **Corrective Actions** a_c: Corrective actions for each agent based on failure information from the previous step.

At the start of each episode, Memory $\mathcal{M}$, Open Subtasks $\mathcal{G}_O$, Completed Subtasks $\mathcal{G}_S$, Actions a, Corrective Actions a_c, and Failure Information $\mathcal{F}$ are initialized as empty sets.

Consider an example of a kitchen with groceries, a fridge, and a counter. Two agents are tasked with "Fetch the groceries and place them in the fridge". This example will help illustrate the utility of each module. All LMs receive a language task instruction $\mathcal{I}$, joint observations from all agents, and information about open and completed subtasks and memory unless stated otherwise. We next discuss the various components in our architecture in detail:

Planner Module The *Planner* LM module suggests feasible subtasks to ensure the completion of the environment task. The *Planner* suggests subtasks related to objects seen in the current observation or memory of all the agents. For the example considered, it decomposes the task into subtasks like "transport the tomato to the fridge" and "transport the lettuce to the fridge", which are added to $\mathcal{G}_O$.

Actor Module The *Actor* LM predicts high-level actions to execute in the environment to progress the open subtasks and

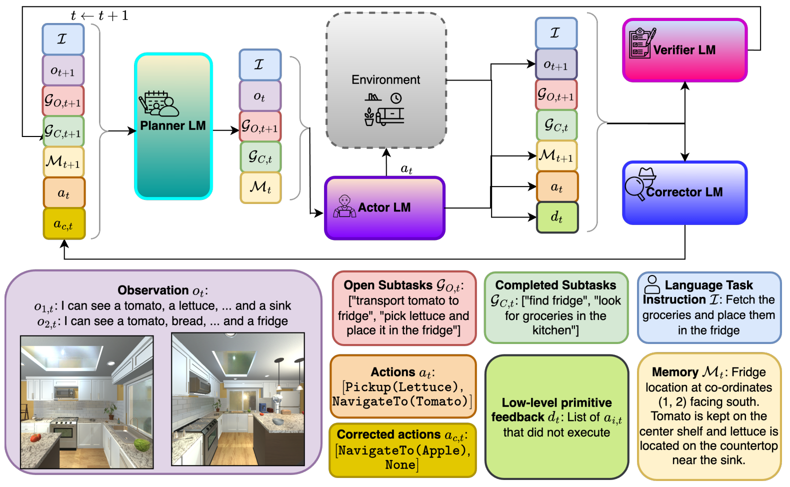

$t \leftarrow t+1$
$\mathcal{I}$
o_{t+1}
$\mathcal{G}_{O,t+1}$
$\mathcal{G}_{C,t+1}$
$\mathcal{M}_{t+1}$
a_t
$a_{c,t}$
Planner LM
$\mathcal{I}$
o_t
$\mathcal{G}_{O,t+1}$
$\mathcal{G}_{C,t}$
$\mathcal{M}_t$
Environment
a_t
Actor LM
$\mathcal{I}$
o_{t+1}
$\mathcal{G}_{O,t+1}$
$\mathcal{G}_{C,t}$
$\mathcal{M}_{t+1}$
a_t
d_t
Verifier LM
Corrector LM
Observation o_t:
$o_{1,t}$: I can see a tomato, a lettuce, ... and a sink
$o_{2,t}$: I can see a tomato, bread, ... and a fridge
Open Subtasks $\mathcal{G}_{O,t}$: ["transport tomato to fridge", "pick lettuce and place it in the fridge"]
Completed Subtasks $\mathcal{G}_{C,t}$: ["find fridge", "look for groceries in the kitchen"]
Language Task Instruction $\mathcal{I}$: Fetch the groceries and place them in the fridge
Actions a_t: [Pickup(Lettuce), NavigateTo(Tomato)]
Corrected actions $a_{c,t}$: [NavigateTo(Apple), None]
Low-level primitive feedback d_t: List of $a_{i,t}$ that did not execute
Memory $\mathcal{M}_t$: Fridge location at co-ordinates (1, 2) facing south. Tomato is kept on the center shelf and lettuce is located on the countertop near the sink.

Figure 1. An overview of LLaMAR's modular cognitive architecture. LLaMAR leverages LMs within four key modules: Planner, Actor, Corrector, and Verifier, each with specific roles. The Planner breaks down the high-level language instruction into feasible subtasks to achieve the environment goal. The Actor determines the high-level actions each agent should perform. These actions trigger low-level policies that generate and execute a sequence of primitive actions in sync across all agents. Based on execution feedback, the Corrector suggests corrections for high-level actions and the Verifier Module validates completion of subtasks.

update the memory. It additionally uses corrective actions suggested by the *Corrector* module in the previous time step. For instance, the *Actor* might suggest actions such as $a = [\texttt{Pickup(Tomato)}, \texttt{NavigateTo(Lettuce)}]$, updating memory with "A tomato is on the counter-top, Alice is picking up the tomato, and Bob is navigating to the lettuce".

Corrector Module The *Corrector* LM corrects high-level actions suggested by the *Actor* LM after failures in the previous step with reasons for failures and chosen corrections. execution[1] For example, it might suggest "Picking up the tomato failed because it is far away. Alice first needs to navigate closer to the tomato."; $a_c = [\texttt{NavigateTo(Tomato)}, \texttt{None}]$.

Verifier Module After executing high-level actions, the *Verifier* LM assesses whether these actions have completed any subtasks in the open subtask set. Successful subtasks are moved to the completed subtask set. Without it, the framework would need to rely on an oracle for subtask completion feedback. The *Verifier* LM additionally utilizes successful high-level actions execution information to predict subtask completion. For example, after transporting the lettuce to the fridge, the *Verifier* updates the completed subtasks with "transport lettuce to the fridge".

The natural language LM outputs are translated to executable actions by using the cosine-similarity method from (Huang et al., 2022). More details in Appendix C.3.

[1]We use the simulator just to verify the successful execution of the high-level action.

4. Experiments

We evaluate the performance of LLaMAR and benchmark other baseline methods on a set of planning tasks in AI2-THOR. The tasks vary in the ambiguity in the natural language instruction given as input to the agents. More details can be found in the Appendix **??**.

Metrics We use the Success Rate (SR), the fraction of episodes where all subtasks are completed, Transport Rate (TR), the fraction of the subtasks, Coverage (C), the fraction of successful interactions with target objects and the Average Steps (L) taken to finish the task. For all the metrics, we report the means along with the 95% confidence interval across all the tasks. Since SR is a binomial metric, we report the Clopper-Pearson Interval as the confidence interval. More details about the metrics can be found in Appendix D

Baselines For a fair comparison with our method, we make modifications to the baselines to make them compatible to the partially observable settings with limited reliance on the simulator. We compare different prompting methods in the reactive planning paradigm, specifically Act, ReAct (Yao et al., 2023) and Chain-of-Thought (CoT) (Wei et al., 2022). Along with this we also compare LLaMAR with SmartLLM (Kannan et al., 2023) and CoELA (Zhang et al., 2024). More details about implementations of baselines can be found in Appendix H. It should be noted that Act, Chain-of-Thought, ReAct, and SmartLLM are all CMAS frameworks whereas CoELA follows the DMAS framework (refer (Chen et al., 2023)).

Modules	Success Rate	Transport Rate	Coverage	Balance	Steps
Actor	0.33 (0.19, 0.49)	0.67 (0.59, 0.76)	0.91 (0.86,0.95)	0.59 (0.52,0.66)	24.92 (22.12,27.73)
Planner+ Actor+ Verifier	0.45 (0.29, 0.57)	0.78 (0.67, 0.84)	0.92 (0.84, 0.95)	69 (0.61, 0.75)	24.87 (20.48, 27.95)
Planner+ Actor+ Corrector	**0.67** (0.51, 0.80)	**0.91** (0.83, 0.96)	**0.97** (0.94, 0.99)	**0.84** (0.79, 0.89)	22.81 (19.95, 25.76)
LLaMAR	0.66 (0.50, 0.76)	**0.91** (0.81, 0.96)	**0.97** (0.93,0.99)	0.82 (0.75,0.87)	**21.87** (18.76, 26.43)

Table 1. Ablating different modules LLaMAR with GPT-4V as the underlying VLM, 2-agents scenarios.

5. Results and Discussion

Algorithm	Success Rate	Transport Rate	Coverage	Balance	Steps
Act	0.33 (0.19, 0.49)	0.67 (0.59, 0.76)	0.91 (0.86, 0.95)	0.59 (0.52, 0.66)	24.92 (22.12, 27.73)
ReAct	0.34 (0.20, 0.49)	0.72 (0.63, 0.80)	0.92 (0.86, 0.97)	0.67 (0.61, 0.73)	24.08 (21.27, 26.89)
CoT	0.14 (0.06, 0.28)	0.59 (0.51, 0.67)	0.87 (0.81, 0.92)	0.62 (0.56, 0.69)	28.4 (26.91, 29.97)
SmartLLM	0.11 (0.05, 0.23)	0.23 (0.13, 0.31)	0.91 (0.80, 0.96)	0.45 (0.37, 0.52)	29.87 (26.20, 30.00)
CoELA	0.25 (0.10, 0.36)	0.46 (0.35, 0.56)	0.76 (0.67, 0.85)	0.73 (0.67, 0.80)	28.93 (27.77, 30.00)
LLaMAR (~~vision~~)	0.51 (0.36, 0.66)	0.85 (0.80, 0.91)	0.95 (0.91, 0.98)	0.83 (0.78, 0.86)	25.80 (23.72, 27.88)
LLaMAR (~~exp~~)	0.62 (0.46, 0.76)	0.87 (0.80, 0.93)	0.95 (0.91, 0.98)	0.82 (0.77, 0.87)	23.44 (20.88, 26.00)
LLaMAR (exp)	**0.66** (0.50, 0.78)	**0.91** (0.81, 0.96)	**0.97** (0.93, 0.99)	**0.82** (0.75, 0.87)	**21.87** (18.76, 24.23)

Table 2. Comparison of evaluation metrics against baselines averaged across all tasks.

Baseline Comparisons: Table 2 compares our method, LLaMAR, with other baselines in a 2-agent scenario using GPT-4 as the underlying LM. Act and ReAct show similar performance, with Act struggles due to its lack of strategic planning and correction. ReAct performs slightly better by dynamically adjusting actions based on reasoning on immediate feedback. CoT's performance declines with longer planning horizons due to its inability to maintain coherence over extended planning sequences, consistent with findings in (Stechly et al., 2024), showing its effectiveness only with highly specific prompts. SmartLLM, operating in a *plan-and-execute* paradigm, generates impractical plans with issues like infinite loops and inability to handle low-level action failures, leading to lower success rates and poor transport metrics. It also tends to hallucinate objects. CoELA, using a decentralized multi-agent system (DMAS), performs poorly due to large input prompts and struggles to select the correct action from numerous choices. Its decentralized decision-making is less efficient than the centralized multi-agent system (CMAS) used by LLaMAR. Previous research (Chen et al., 2023) confirms CMAS frameworks are more effective than DMAS frameworks. LLaMAR when used solely with text inputs, exhibits a worse performance than with visual inputs. This is attributed to the agents' inability to reason about visual observations, which is particularly detrimental for the Corrector module. Overall, our method, LLaMAR, benefits from its modular cognitive architecture, which integrates planning, acting, correcting, and verifying through distinct LLM roles, resulting in superior performance across various evaluation metrics. By avoiding reliance on privileged information and incorporating a robust exploration strategy allowing it to scout for objects not initially visible, LLaMAR ensures higher success rates and balanced task execution among agents.

Roles of different modules in LLaMAR: We conduct ablation studies to assess each module's effectiveness by comparing the framework's performance metrics with each module removed individually. The results are summarized in Table 1. Using only the *Actor* module corresponds to the "Act" baseline, which demonstrates its fundamental capabilities in isolation but shows limited effectiveness without planning and correction due to relatively lower SR and TR, and ensuring more effective task completion and even work distribution, as indicated by the increase in balance (B). Adding the Planner and Verifier module improves performance, benefiting from better task planning and validation, increasing the overall SR and TR. However, in scenarios where the suggested action fails, the actor suggests the same action in the next decision step since it is not able to reason on why the action failed until the end of the planning horizon. Incorporating the Corrector module with access to privileged information from an environment oracle significantly boosts performance, enhancing the SR, TR, and C, and reducing L by approximately two time steps on average, consistent with the findings in (Arora & Kambhampati, 2023). This highlights the Corrector module's importance in adjusting actions based on feedback, resulting in higher task success and more efficient task completion, albeit with reliance on oracle knowledge. Finally, the complete LLaMAR system, without privileged information, achieves SR, TR, and C values close to those of the oracle setup, with better L. This demonstrates the system's robustness and effectiveness in a realistic setting. The Corrector module plays a crucial role in enabling agents to learn from past failures and avoid repeating actions, preventing task failures due to timeout. Despite lacking oracle knowledge, LLaMAR performs nearly as well as the oracle-enhanced setup. These results highlight the importance of each module in our cognitive architecture. Removing any module diminishes effectiveness, highlighting their essential roles in achieving state-of-the-art results.

6. Conclusion

We address long-horizon planning in dynamic, partially observable multi-agent environments with LLaMAR, an LM-based planner using four specialized modules: *Planner*, *Actor*, *Corrector*, and *Verifier*. This framework iteratively refines action planning, adapts to failures, and verifies subtask completion using real-time observations and action feedback, without privileged information. We also introduce a heuristic-based exploration strategy to guide agents to semantically relevant regions. Empirical results show LLaMAR outperforms existing LM-based approaches, achieving a 30% higher success rate on AI2Thor's planning tasks.

Acknowledgements

We would like to thank Keerthana Gopalakrishnan, Sydney Dolan, Jasmine Aloor, and Victor Qin for helpful discussions and feedback. OpenAI credits for GPT-4 access was provided through OpenAI's Researcher Access Program. The research was sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

Arora, D. and Kambhampati, S. Learning and Leveraging Verifiers to Improve Planning Capabilities of Pre-trained Language Models. *arXiv preprint arXiv:2305.17077*, 2023.

Baghel, R., Kapoor, A., Bachiller, P., Jorvekar, R. R., Rodriguez-Criado, D., and Manso, L. J. A toolkit to generate social navigation datasets. In *Advances in Physical Agents II: Proceedings of the 21st International Workshop of Physical Agents (WAF 2020), November 19-20, 2020, Alcalá de Henares, Madrid, Spain*, pp. 180–193. Springer, 2021.

Barto, A. G. and Mahadevan, S. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13:41–77, 2003. URL https://api.semanticscholar.org/CorpusID:386824.

Batra, D., Chang, A. X., Chernova, S., Davison, A. J., Deng, J., Koltun, V., Levine, S., Malik, J., Mordatch, I., Mottaghi, R., Savva, M., and Su, H. Rearrangement: A Challenge for Embodied AI. *CoRR*, abs/2011.01975, 2020. URL https://arxiv.org/abs/2011.01975.

Brodeur, S., Perez, E., Anand, A., Golemo, F., Celotti, L., Strub, F., Rouat, J., Larochelle, H., and Courville, A. C. HoME: a Household Multimodal Environment. *CoRR*, abs/1711.11017, 2017. URL http://arxiv.org/abs/1711.11017.

Chen, Y., Arkin, J., Zhang, Y., Roy, N., and Fan, C. Scalable Multi-Robot Collaboration with Large Language Models: Centralized or Decentralized Systems? *arXiv preprint arXiv:2309.15943*, 2023.

Gan, C., Zhou, S., Schwartz, J., Alter, S., Bhandwaldar, A., Gutfreund, D., Yamins, D. L. K., DiCarlo, J. J., McDermott, J., Torralba, A., and Tenenbaum, J. B. The ThreeDWorld Transport Challenge: A Visually Guided Task-and-Motion Planning Benchmark for Physically Realistic Embodied AI, 2021.

Gramopadhye, M. and Szafir, D. Generating Executable Action Plans with Environmentally-Aware Language Models, 2023.

Hong, W., Wang, W., Lv, Q., Xu, J., Yu, W., Ji, J., Wang, Y., Wang, Z., Zhang, Y., Li, J., Xu, B., Dong, Y., Ding, M., and Tang, J. CogAgent: A Visual Language Model for GUI Agents, 2023a.

Hong, Y., Zhen, H., Chen, P., Zheng, S., Du, Y., Chen, Z., and Gan, C. 3D-LLM: Injecting the 3D World into Large Language Models, 2023b.

Huang, C., Mees, O., Zeng, A., and Burgard, W. Visual Language Maps for Robot Navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 2023a.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9118–9147. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/huang22a.html.

Huang, W., Xia, F., Shah, D., Driess, D., Zeng, A., Lu, Y., Florence, P., Mordatch, I., Levine, S., Hausman, K., and Ichter, B. Grounded Decoding: Guiding Text Generation with Grounded Models for Embodied Agents. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 59636–59661. Curran Associates, Inc., 2023b. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/bb3cfcb0284642a973dd631ec9184f2f-Paper-Conference.pdf.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Jackson, T., Brown, N., Luu, L., Levine, S., Hausman, K., and Ichter, B. Inner Monologue: Embodied Reasoning through Planning with Language Models. In Liu, K., Kulic, D., and Ichnowski, J. (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 14–18 Dec 2023c. URL https://proceedings.mlr.press/v205/huang23c.html.

Ichter, B., Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., Kalashnikov, D., Levine, S., Lu, Y., Parada, C., Rao, K., Sermanet, P., Toshev, A. T., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Yan, M., Brown, N., Ahn, M., Cortes, O., Sievers, N., Tan, C., Xu, S., Reyes, D., Rettinghouse, J., Quiambao, J., Pastor, P., Luu, L., Lee, K.-H., Kuang, Y., Jesmonth, S., Joshi, N. J., Jeffrey, K., Ruano, R. J., Hsu, J., Gopalakrishnan, K., David, B., Zeng, A., and Fu, C. K. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In Liu, K., Kulic, D., and Ichnowski, J. (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

Jain, U., Weihs, L., Kolve, E., Rastegari, M., Lazebnik, S., Farhadi, A., Schwing, A. G., and Kembhavi, A. Two Body Problem: Collaborative Visual Task Completion. *CoRR*, abs/1904.05879, 2019. URL http://arxiv.org/abs/1904.05879.

Jain, U., Weihs, L., Kolve, E., Farhadi, A., Lazebnik, S., Kembhavi, A., and Schwing, A. A cordial sync: Going beyond marginal policies for multi-agent embodied tasks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pp. 471–490. Springer, 2020.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(98)00023-X. URL https://www.sciencedirect.com/science/article/pii/S000437029800023X.

Kambhampati, S. Can large language models reason and plan? *Annals of the New York Academy of Sciences*, 1534 (1):15–18, 2024.

Kannan, S. S., Venkatesh, V. L., and Min, B.-C. SMART-LLM: Smart Multi-Agent Robot Task Planning using Large Language Models. *arXiv preprint arXiv:2309.10062*, 2023.

Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Gordon, D., Zhu, Y., Gupta, A., and Farhadi, A. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017.

Laurençon, H., Saulnier, L., Tronchon, L., Bekman, S., Singh, A., Lozhkov, A., Wang, T., Karamcheti, S., Rush, A. M., Kiela, D., Cord, M., and Sanh, V. OBELICS: An Open Web-Scale Filtered Dataset of Interleaved Image-Text Documents, 2023.

Laurençon, H., Tronchon, L., Cord, M., and Sanh, V. What matters when building vision-language models?, 2024.

Li, C., Zhang, R., Wong, J., Gokmen, C., Srivastava, S., Martín-Martín, R., Wang, C., Levine, G., Lingelbach, M., Sun, J., Anvari, M., Hwang, M., Sharma, M., Aydin, A., Bansal, D., Hunter, S., Kim, K.-Y., Lou, A., Matthews, C. R., Villa-Renteria, I., Tang, J. H., Tang, C., Xia, F., Savarese, S., Gweon, H., Liu, K., Wu, J., and Fei-Fei, L. BEHAVIOR-1K: A Benchmark for Embodied AI with 1,000 Everyday Activities and Realistic Simulation. In Liu, K., Kulic, D., and Ichnowski, J. (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 80–93. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/li23a.html.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. Code as Policies: Language Model Programs for Embodied Control. In *arXiv preprint arXiv:2209.07753*, 2022.

Lin, K., Agia, C., Migimatsu, T., Pavone, M., and Bohg, J. Text2Motion: from natural language instructions to feasible plans. *Autonomous Robots*, 47(8):1345–1365, November 2023. ISSN 1573-7527. doi: 10.1007/s10514-023-10131-7. URL http://dx.doi.org/10.1007/s10514-023-10131-7.

Liu, H., Li, C., Li, Y., and Lee, Y. J. Improved Baselines with Visual Instruction Tuning, 2023a.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual Instruction Tuning. In *NeurIPS*, 2023b.

Mandi, Z., Jain, S., and Song, S. RoCo: Dialectic multi-robot collaboration with large language models. *arXiv preprint arXiv:2307.04738*, 2023.

Meyer, J., Praeuner, R., and Vanderbeek, L. Hierarchical Multi-Agent Reinforcement Learning. https://cse.unl.edu/~lksoh/Classes/CSCE475_875_Fall11/seminars/Seminar_JRL.pdf, 2020.

Misra, D. K., Bennett, A., Blukis, V., Niklasson, E., Shatkhin, M., and Artzi, Y. Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction. *CoRR*, abs/1809.00786, 2018. URL http://arxiv.org/abs/1809.00786.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-Efficient Hierarchical Reinforcement Learning. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/e6384711491713d29bc63fc5eeb5ba4f-Paper.pdf.

OpenAI, :, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. GPT-4 Technical Report, 2023.

Padmakumar, A., Thomason, J., Shrivastava, A., Lange, P., Narayan-Chen, A., Gella, S., Piramuthu, R., Tur, G., and Hakkani-Tur, D. TEACh: Task-driven Embodied Agents that Chat, 2021.

Pateria, S., Subagdja, B., Tan, A.-h., and Quek, C. Hierarchical Reinforcement Learning: A Comprehensive Survey. *ACM Comput. Surv.*, 54(5), jun 2021. ISSN 0360-0300. doi: 10.1145/3453160. URL https://doi.org/10.1145/3453160.

Prasad, A., Koller, A., Hartmann, M., Clark, P., Sabharwal, A., Bansal, M., and Khot, T. ADaPT: As-Needed Decomposition and Planning with Language Models, 2023.

Puig, X., Ra, K., Boben, M., Li, J., Wang, T., Fidler, S., and Torralba, A. VirtualHome: Simulating Household Activities via Programs, 2018.

Puig, X., Undersander, E., Szot, A., Cote, M. D., Partsey, R., Yang, J., Desai, R., Clegg, A. W., Hlavac, M., Min, T., Gervet, T., Vondruš, V., Berges, V.-P., Turner, J., Maksymets, O., Kira, Z., Kalakrishnan, M., Malik, J., Chaplot, D. S., Jain, U., Batra, D., Rai, A., and Mottaghi, R. Habitat 3.0: A Co-Habitat for Humans, Avatars and Robots, 2023.

Puterman, M. L. *Markov Decision Processes*. Wiley, 1994. ISBN 978-0471727828. URL http://books.google.com/books/about/Markov_decision_processes.html?id=Y-gmAQAAIAAJ.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning Transferable Visual Models From Natural Language Supervision. *CoRR*, abs/2103.00020, 2021. URL https://arxiv.org/abs/2103.00020.

Raman, S. S., Cohen, V., Rosen, E., Idrees, I., Paulius, D., and Tellex, S. Planning with Large Language Models via Corrective Re-prompting. January 2022. URL http://www.cs.utexas.edu/users/ai-labpub-view.php?PubID=127989.

Reimers, N. and Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, 2019.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., and Batra, D. Habitat: A Platform for Embodied AI Research. *CoRR*, abs/1904.01201, 2019. URL http://arxiv.org/abs/1904.01201.

Sengar, V., Kapoor, A., George, N., Vatsal, V., Gubbi, J., Pal, A., et al. Challenges in applying robotics to retail store management. *arXiv preprint arXiv:2208.09020*, 2022.

Shah, D., Osinski, B., Ichter, B., and Levine, S. LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=UW5A3SweAH.

Sharma, P., Torralba, A., and Andreas, J. Skill Induction and Planning with Latent Language. *CoRR*, abs/2110.01517, 2021. URL https://arxiv.org/abs/2110.01517.

Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., and Fox, D. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. *CoRR*, abs/1912.01734, 2019. URL http://arxiv.org/abs/1912.01734.

Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., Fox, D., Thomason, J., and Garg, A. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530, 2023. doi: 10.1109/ICRA48891.2023.10161317.

Singh, I., Traum, D., and Thomason, J. TwoStep: Multi-agent Task Planning using Classical Planners and Large Language Models. *arXiv preprint arXiv:2403.17246*, 2024.

Song, C. H., Wu, J., Washington, C., Sadler, B. M., Chao, W.-L., and Su, Y. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models, 2023.

Stechly, K., Valmeekam, K., and Kambhampati, S. Chain of Thoughtlessness: An Analysis of CoT in Planning. *arXiv preprint arXiv:2405.04776*, 2024.

Stooke, A., Lee, K., Abbeel, P., and Laskin, M. Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pp. 9870–9879. PMLR, 2021.

Sumers, T. R., Yao, S., Narasimhan, K., and Griffiths, T. L. Cognitive Architectures for Language Agents, 2023.

Tang, H., Hao, J., Lv, T., Chen, Y., Zhang, Z., Jia, H., Ren, C., Zheng, Y., Meng, Z., Fan, C., and Wang, L. Hierarchical Deep Multiagent Reinforcement Learning with Temporal Abstraction, 2019.

Valmeekam, K., Marquez, M., and Kambhampati, S. Can Large Language Models Really Improve by Self-critiquing Their Own Plans? *arXiv preprint arXiv:2310.08118*, 2023a.

Valmeekam, K., Marquez, M., Sreedharan, S., and Kambhampati, S. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005, 2023b.

Wang, J., He, G., and Kantaros, Y. Safe Task Planning for Language-Instructed Multi-Robot Systems using Conformal Prediction. *arXiv preprint arXiv:2402.15368*, 2024.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen, J.-R. A Survey on Large Language Model based Autonomous Agents, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E. H., Le, Q., and Zhou, D. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *CoRR*, abs/2201.11903, 2022. URL https://arxiv.org/abs/2201.11903.

Wohlke, J., Schmitt, F., and van Hoof, H. Hierarchies of Planning and Reinforcement Learning for Robot Navigation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2021. doi: 10.1109/icra48506.2021.9561151. URL http://dx.doi.org/10.1109/ICRA48506.2021.9561151.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., Zheng, R., Fan, X., Wang, X., Xiong, L., Zhou, Y., Wang, W., Jiang, C., Zou, Y., Liu, X., Yin, Z., Dou, S., Weng, R., Cheng, W., Zhang, Q., Qin, W., Zheng, Y., Qiu, X., Huang, X., and Gui, T. The Rise and Potential of Large Language Model Based Agents: A Survey, 2023.

Xia, F., Zamir, A. R., He, Z., Sax, A., Malik, J., and Savarese, S. Gibson Env: Real-World Perception for Embodied Agents. *CoRR*, abs/1808.10654, 2018. URL http://arxiv.org/abs/1808.10654.

Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., Yi, L., Chang, A. X., Guibas, L. J., and Su, H. SAPIEN: A SimulAted Part-based Interactive ENvironment. *CoRR*, abs/2003.08515, 2020. URL https://arxiv.org/abs/2003.08515.

Yang, J., Borovikov, I., and Zha, H. Hierarchical Cooperative Multi-Agent Reinforcement Learning with Skill Discovery. *CoRR*, abs/1912.03558, 2019. URL http://arxiv.org/abs/1912.03558.

Yang, R., Xu, H., Wu, Y., and Wang, X. Multi-task reinforcement learning with soft modularization. *Advances in Neural Information Processing Systems*, 33:4767–4777, 2020.

Yang, S., Nachum, O., Du, Y., Wei, J., Abbeel, P., and Schuurmans, D. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models, 2023.

Yu, B., Kasaei, H., and Cao, M. Co-NavGPT: Multi-Robot Cooperative Visual Semantic Navigation using Large Language Models. *ArXiv*, abs/2310.07937, 2023. URL https://api.semanticscholar.org/CorpusID:263909555.

Zhang, H., Du, W., Shan, J., Zhou, Q., Du, Y., Tenenbaum, J. B., Shu, T., and Gan, C. Building Cooperative Embodied Agents Modularly with Large Language Models. *ICLR*, 2024.

Zhu, Y., Gordon, D., Kolve, E., Fox, D., Fei-Fei, L., Gupta, A., Mottaghi, R., and Farhadi, A. Visual semantic planning using deep successor representations. In *Proceedings of the IEEE international conference on computer vision*, pp. 483–492, 2017.

A. Related Work

Reinforcement Learning (RL) for Long-Horizon Planning: While RL algorithms have shown promise in many applications, they still struggle with long-horizon tasks. Hierarchical reinforcement learning (HRL) has been used to address these challenges in both single-agent (Barto & Mahadevan, 2003; Nachum et al., 2018; Pateria et al., 2021; Wohlke et al., 2021) and multi-agent settings (Tang et al., 2019; Yang et al., 2019; Meyer et al., 2020). However, these approaches are typically applied to single-task, stationary environments, such as games, where agents solve for one goal in a fixed environment. Consequently, these methods do not generalize well across multiple environments or tasks. Multi-task RL has been explored as a potential solution, requiring sophisticated task planning to handle diverse objectives (Stooke et al., 2021; Yang et al., 2020). This often involves decomposing tasks into manageable subtasks, a process well-suited for hierarchical frameworks. However, subtasks are known apriori in multi-task RL formulations. Real-world long-horizon RL necessitates robust task planning, and LMs have emerged as a promising approach for this purpose.

LMs for Embodied Single-Agent Planning: Recent studies have demonstrated the effectiveness of LMs in generating and executing plans in embodied single-agent environments (Yang et al., 2023; Wang et al., 2023; Xi et al., 2023; Sumers et al., 2023; Sharma et al., 2021; Raman et al., 2022; Gramopadhye & Szafir, 2023) and creating plans in single-agent embodied robotic environments (Kolve et al., 2017; Savva et al., 2019; Xia et al., 2018; Li et al., 2023; Padmakumar et al., 2021; Shridhar et al., 2019; Misra et al., 2018; Zhu et al., 2017; Brodeur et al., 2017; Xiang et al., 2020; Jain et al., 2019; 2020). Works like SayCan (Ichter et al., 2023) and Grounded Decoding (Huang et al., 2023b) use a combination of value functions and LLM predictions for long-horizon tasks. ProgPrompt (Singh et al., 2023) and Zero-Shot Language Planner (Huang et al., 2022) generate static plans executed in the environment, which may fail in partially observable and dynamic settings. To mitigate this, LLM-planner (Song et al., 2023) updates plans based on new observations, similar to our approach.

LMs for Multi-Agent Planning: Recent studies have demonstrated the effectiveness of LMs in generating and executing plans in embodied multi-agent environments. CoNavGPT (Yu et al., 2023) creates global plans for two robots in an embodied environment. RoCo (Mandi et al., 2023) and CoELA (Zhang et al., 2024) assign separate LMs to each agent for decentralized action prediction, allowing natural language communication between agents. However, RoCo and CoNavGPT require detailed environment information for planning, and CoELA's action space is filtered by an oracle. Relying on privileged information from an oracle is impractical in real-world applications. By contrast, our work focuses on free-form action generation and handles tasks with more ambiguous descriptions. Prior work (Chen et al., 2023) compare centralized (CMAS) and decentralized (DMAS) planning frameworks, showing that centralized planners perform better, though their experiments are in simple, known environments with limited number of agents. Two-Step (Singh et al., 2024) decomposes goals for main and helper agents, using PDDL planners for high-level actions. SmartLLM (Kannan et al., 2023) uses multiple LLM modules for subtask decomposition, multi-robot group formation and task allocation but assumes robots have complete knowledge of the environment, making plans prone to errors in unknown settings. Wang et al. (Wang et al., 2024) use LLMs with conformal prediction for safe multi-agent planning, but the action choices are limited to a small set of objects. Table 3 presents a comparison of the characteristics of different LM-based approaches to multi-agent planning with our work.

LMs can interpret high-level instructions and break them down into feasible subtasks, making them ideal for long-horizon, multi-task scenarios. Our work leverages LMs to enable long-horizon planning across a variety of tasks and environments, building on these advances to address the limitations of traditional RL and HRL methods. By integrating LMs into our planning framework, we enhance the ability to generalize across diverse tasks and scenarios, making significant strides toward practical, real-world applications of RL in dynamic, multi-agent settings.

Method	Dynamic Planning	Local Information	Failure Correction	Self Verification
Two-Step (Singh et al., 2024)	✗	✗	✗	✗
Smart LLM (Kannan et al., 2023)	✗	✗	✗	✗
Conformal Prediction LLM (Wang et al., 2024)	✓	✗	✗	✗
CoELA (Zhang et al., 2024)	✓	✓	✗	✗
LLaMAR (this paper)	✓	✓	✓	✓

Table 3. The proposed model, LLaMAR: 1) performs dynamic planning, avoiding the open-loop plan-and-execute paradigm; 2) operates without privileged simulator information (e.g., access to all objects in the environment); 3) re-plans when low-level actions fail, not assuming perfect execution; and 4) self-verifies subtask completion without relying on the simulator.

[1] re-planning is generally not required since the PDDL planner checks for all pre-conditions for the actions

B. Terminology

We differentiate between the terms subtasks and high-level actions in this section. In essence, multiple high-level actions are needed to be carried out in a sequence to complete a subtask. Multiple subtasks need to be accomplished in order to complete the high-level language instruction.

- Subtasks: A task is split up into multiple subtasks. For example, if a task is “Fetch all the groceries and put them in the fridge”, then the initial subtasks could include: “Locate the groceries”, “transport the groceries”, “Locate the fridge”. These subtasks could get updated with new observations. For example, while locating the groceries, the agents come across a tomato and a lettuce. Then the subtasks “transport the tomato to the fridge” and “transport the lettuce to the fridge” gets updated in the subtasks list. This basically splits up the high-level instruction $\mathcal{I}$ into multiple subtasks
- High-level actions: These are the set of actions required to complete the subtasks. For example, to complete the “transport the lettuce in the fridge”, we would require: the following set of actions:
  - Navigate to lettuce
  - Pickup lettuce
  - Navigate to the fridge
  - Open fridge
  - Put lettuce in the fridge
  - Close fridge

  Note that different agents have to complete different high-level actions that progress the subtasks efficiently whilst avoiding conflicts.
- Conflicts can arise in the following ways:
  - Same high-level actions: Agents performing the same action at the same time. For example, ”Open the fridge”.
  - Blocking: Agent 1 is blocking Agent 2 and not allowing it to complete its high-level action For example, Agent 1 is attempting to execute “PlaceObject(Tomato)” in front of the fridge to place the tomato in its hand in the fridge and Agent 2 is attempting to execute “OpenFreezer()” needs to interact with the fridge. Would require some form of conflict resolution in the state cell domain. Agent 1 should move away to allow fridge access to Agent 2. In LLaMAR, the *Corrector* module helps in figuring out these conflicts and suggest different corrective high-level actions.

C. Environment

The environment is based on the AI2Thor simulator with a multi-agent setup. All the experiments were performed in the single-room floor plans. When more than 3 agents are added to some of the floor plans (especially the kitchen floor plans), the environment gets crowded and does not allow for a lot of free space to navigate to different objects (the number of reachable paths reduces).

(a) Kitchen (b) Bedroom (c) LivingRoom (d) Bathroom

Figure 2. Photorealistic rendering of household scenarios in the AI2Thor simulator enables the usage of multiple autonomous robots to carry out daily tasks.

C.1. Observation Space

The observations for each robot include an image of size resolution $1000 \times 1000 \times 3$. The textual observation for each agent in the prompt is the list of objects visible in this image and the agents’ current location and rotation. The field of view is 90 degrees. The agents can interact with the objects only if it is within its visibility range of 1.5m.

C.2. Action Space

The actions space $\mathcal{A}$ consists of navigation actions $\mathcal{A}_{NAV}$, interaction actions $\mathcal{A}_{INT}$, exploration action $\mathcal{A}_{EXP}$.

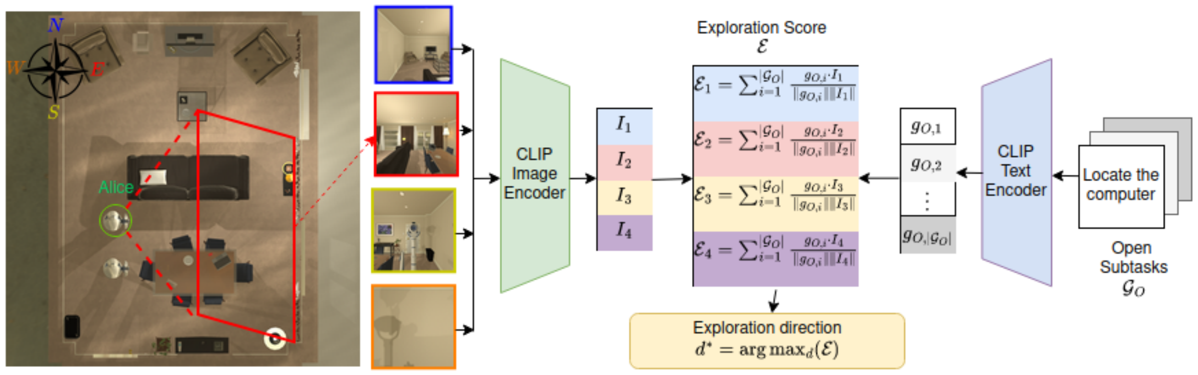
Navigation actions $\mathcal{A}_{NAV}$ consists of the following actions:

- `Move(<direction>)`: Moves the robot by 0.25m towards the specified direction where `<direction>` can be one of (`Ahead`, `Back`, `Right`, `Left`)
- `Rotate(<direction>)`: Rotates the robot by 90 degrees towards the specified direction where, `<direction>` can be one of (`Right`, `Left`)
- `LookUp(<angle>)` rotates the pitch of the robot camera upwards by the specified angle.
- `LookDown<angle>` rotates the pitch of the robot camera downwards by the specified angle.
- `NavigateTo(<object_id>)` makes the robot navigate to the specified object. The path is found using the A^*-shortest path algorithm. Note that the robot is only able to find a path to the specified object in the environment only if it has encountered that object previously during the episode. Otherwise, the `NavigateTo(.)` action will be unsuccessful and the agent will have to explore.

Interaction actions $\mathcal{A}_{INT}$ consists of the following actions:

- `Pickup(<object_id>)`: Picks up the object
- `Put(<receptacle_id>)`: Puts the object in the robots hand on the receptacle
- `Open(<object_id>)`: Opens the object
- `Close(<object_id>)`: Closes the open object
- `Slice(<object_id>)`: Slices the object
- `Clean(<object_id>)`: Cleans the object
- `ToggleOn(<object_id>)`: Toggles the object on
- `ToggleOff(<object_id>)`: Toggles the object off

Explore action : In unexplored environments, agents need to search for task-relevant objects. If agents cannot find the required objects, the language model can choose an 'exploration' action $a_{exp} \in \mathcal{A}_{EXP}$. We use a semantically-guided heuristic to determine the choice of region to be explored. The agent rotates to four cardinal directions $d \in North, South, East, West$, capturing image observations $o_{n,d}$. These images are processed through a pre-trained CLIP image encoder (Radford et al., 2021) to obtain embeddings I_d. The list of open subtasks $\mathcal{G}_O$ is processed through the corresponding CLIP text encoder to get text embeddings $g_{O,i}$. The exploration score $\mathcal{E}_d$ in direction d is defined as $\mathcal{E}_d = \sum_{i=1}^{|\mathcal{G}_O|} \frac{g_{O,i} \cdot I_d}{\|g_{O,i}\| \|I_d\|}$. The direction with the highest score $d^* = \arg\max_d \mathcal{E}_d$ is chosen. Summing the scores helps select the best direction to explore in expectation. The agent rotates towards d^* and moves $J = 2$ steps, repeating this process $K = 3$ times in one *explore* action. This approach ensures that images relevant to identifying potential subtasks are prioritized. For example, if $\mathcal{G}_O$ includes "locate a computer", it is more likely to find a computer on a table than on a sofa, resulting in a higher cosine similarity score between the subtask CLIP text embedding and table CLIP image embedding. The explore action is carried out by the heuristic mentioned in Algorithm 1. We use the `clip-vit-large-patch14-336` model for the CLIP weights which we download from https://huggingface.co/openai/clip-vit-large-patch14-336.



N
W
E
S
Alice
CLIP Image Encoder
I_1
I_2
I_3
I_4
Exploration Score $\mathcal{E}$
$\mathcal{E}_1 = \sum_{i=1}^{|\mathcal{G}_O|} \frac{g_{O,i} \cdot I_1}{\|g_{O,i}\|\|I_1\|}$
$\mathcal{E}_2 = \sum_{i=1}^{|\mathcal{G}_O|} \frac{g_{O,i} \cdot I_2}{\|g_{O,i}\|\|I_2\|}$
$\mathcal{E}_3 = \sum_{i=1}^{|\mathcal{G}_O|} \frac{g_{O,i} \cdot I_3}{\|g_{O,i}\|\|I_3\|}$
$\mathcal{E}_4 = \sum_{i=1}^{|\mathcal{G}_O|} \frac{g_{O,i} \cdot I_4}{\|g_{O,i}\|\|I_4\|}$
Exploration direction $d^* = \arg\max_d(\mathcal{E})$
$g_{O,1}$
$g_{O,2}$
$g_{O,|\mathcal{G}_O|}$
CLIP Text Encoder
Locate the computer
Open Subtasks $\mathcal{G}_O$

Figure 3. Choice of direction for the exploration heuristic: The agent (Alice) rotates towards 4 cardinal directions to get observations. The cosine similarity between the CLIP embeddings I_d for these 4 images are calculated with the CLIP embeddings for each subtask in the open subtasks set $\mathcal{G}_O$ to get the exploration score $\mathcal{E}_d$ for each direction. The direction with the highest $\mathcal{E}_d$ is chosen to explore and the agent moves $J = 2$ steps in that direction.

Algorithm 1 Exploration Heuristic

```
Input: Agent ID $n$, Environment $env$, number of exploration steps $K$, number of move steps $J$
0: $g_O = \text{CLIP}_{\text{text}}(\mathcal{G}_O)$
0: while $k < K$ do
0:     Exploration Score $\mathcal{E} \in \mathbb{R}^4 \leftarrow \mathbf{0}$
0:     for $d \in \{North, South, East, West\}$ do
0:         $o_{n,d} = env.step(\text{Rotate}(\text{Right}, \text{n}))$
0:         $I_d = \text{CLIP}_{\text{img}}(o_{n,d})$
0:         $\mathcal{E}_d = \frac{I_d \cdot g_O}{\|I_d\| \|g_O\|}$
0:     end for
0:     $d^* = \arg\max_d \mathcal{E}$
0:     while $j < J$ do
0:         $o_i = env.step(\text{Move}(d^*, n))$
0:         $j \leftarrow j + 1$
0:     end while
0:     $k \leftarrow k + 1$
0: end while=0
```

C.3. Admissible Action parsing with Semantic Translation

When LMs generate action plans, natural language outputs often fail to translate to executable high-level actions. This happens when the output does not match the predefined format or refers to unrecognized contextually similar objects. We use a cosine similarity method from (Huang et al., 2022), fine-tuning a pre-trained sentence-BERT (Reimers & Gurevych, 2019) to transform the free-form text into admissible high-level actions. Hyperparameters and additional details of the sentence transformer fine-tuning are provided in Table 4.

We finetuned a pre-trained BERT model to function as a semantic mapper between free-form natural language output and the robot's admissible actions in the environment. The pre-trained weights were obtained from https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. The model was trained on a dataset consisting of 2800 free-form input, valid action output pairs. It ran on one (1) Apple M1 core for a wall clock time of 5 minutes. Table 4 shows the hyper-parameters used for the pre-training of the BERT model.

Epochs	10
Max gradient norm	1
Learning rate	2×10^{-5}
Batch size	64
Encoding dimension	384
Optimizer	AdamW
Scheduler	Warm-up linear
Warm-up steps	45
Weight decay	0.01
Loss scale	20
Loss type	Multiple negatives ranking loss
Similarity function	Cosine similarity

Table 4. Hyper-parameters for the model fine-tuning including the loss.

D. Metrics

We evaluate the algorithms using the following metrics to compare their performances on the tasks:

- **Success Rate** (SR): The fraction of episodes in which all subtasks are completed. Success equals 1 if all subtasks are successfully executed in an episode, otherwise it is 0.
- **Transport Rate** (TR): The fraction of subtasks completed within an episode, which provides a finer granularity of task

Underlying LM	Success Rate	Transport Rate	Coverage	Balance	Steps
GPT-4	0.51 (0.36, 0.66)	0.85 (0.80, 0.91)	0.95 (0.91, 0.98)	0.83 (0.78, 0.86)	25.80 (23.72, 27.88)
LLaVA	0.54 (0.41, 0.65)	0.84 (0.71, 0.90)	0.91 (0.87, 0.98)	0.75 (0.64, 0.83)	26.21 (21.56, 28.97)
IDEFICS-2	0.57 (0.43, 0.67)	0.86 (0.74, 0.91)	0.94 (0.89, 0.98)	0.78 (0.65,0.84)	25.27 (20.14, 28.37)
CogVLM	0.61 (0.47, 0.68)	0.89 (0.73, 0.95)	0.95 (0.89, 0.99)	0.80 (0.73, 0.86)	23.21 (20.57, 26.82)
GPT-4V	**0.66** **(0.50, 0.76)**	**0.91** **(0.81, 0.96)**	**0.97** **(0.93,0.99)**	**0.82** **(0.75, 0.87)**	**21.87** **(18.76, 26.43)**

Table 5. Metrics by varying the underlying LLM/VLM in LLaMAR for the 2-agent scenario.

completion.
- **Coverage** (C): The fraction of successful interactions with target objects. It is useful to verify if the LMs can infer the objects to interact with, in scenarios where the tasks have objects that are specified implicitly.
- **Average steps** (L): The number of high-level actions taken by the team to complete the task, capped at $T = 30$ in our experiments. If the task is not completed within T steps, the episode is deemed a failure.
- **Balance** (B): The ratio between the minimum and maximum number of successful high-level actions executed by any agent that contributed towards making progress in a subtask necessary for the completion of the language instruction task. We only check for a subset of high-level actions that must be executed for accomplishing critical subtasks that leads to the successful completion of the language instruction task.If each agent i out of n agents completes s_i successful tasks, the balance is defined as: $B := \frac{\min\{s_1, \cdots, s_n\}}{\max\{s_1, \cdots, s_n\}+\epsilon}$. This measures how evenly the work is distributed among agents. A balance of zero indicates at least one agent performed no successful high-level actions, while a balance of one indicates all agents performed the same number of successful high-level actions. Here $\epsilon = 1e - 4$ is a small number to avoid division by zero.

E. Multi-agent Planning Tasks

We evaluate the performance of LLaMAR and benchmark other baseline methods on a set of tasks in AI2-THOR. These planning tasks are of varying difficulty levels determined by an increase in ambiguity in the language instructions. We include automatic checker modules to verify subtask completion and evaluate plan quality. Our planning problem set comprises of 45 tasks, each defined for five distinct floor plans, ensuring comprehensive testing and evaluation.

- **Explicit item type, quantity, and target location**: Agents are explicitly instructed to transport specific items to specific target locations. For example, `put bread, lettuce, and a tomato in the fridge` clearly defines the objects (tomato, lettuce, bread) and the target (fridge).
- **Explicit item type and target location but implicit item quantity**: The object type is explicitly described, but its quantity is not disclosed. For example, `Put all the apples in the fridge`. Agents must explore the environment to locate all specified items and also predict when to stop.
- **Explicit target location but implicit item types and quantity**: The target location is explicitly defined but the item types and their quantities are concealed. For example, `Put all groceries in the fridge`.
- **Implicit target location, item type and quantity**: Item types and their quantities along with the target location are implicitly defined. For example, `Clear the floor by placing the items at their appropriate positions`. The agent is expected to place items like pens, books, and laptops on the study table, and litter in the trash can..

F. Additional Experiments

F.1. Varying underlying LM

To understand the impact of the underlying LM's quality on decision-making, we experiment with different LMs. Specifically, we utilize both the language-only and vision-language models of GPT-4 (OpenAI et al., 2023), IDEFICS-2 (Laurençon et al., 2023; 2024), LLaVA (Liu et al., 2023b;a), and CoGVLM (Hong et al., 2023a). Among these, GPT-4, when used solely with

# agents	Success Rate	Transport Rate	Coverage	Balance	Steps
2	0.62 (0.46, 0.76)	0.87 (0.80, 0.93)	0.95 (0.91, 0.98)	0.82 (0.77,0.87)	23.44 (20.88, 26.00)
3	0.70 (0.55, 0.82)	0.91 (0.85, 0.95)	0.98 (0.95, 0.99)	0.66 (0.61, 0.71)	21.30 (18.60, 23.99)
4	0.68 (0.52, 0.79)	0.90 (0.84, 0.94)	0.99 (0.95, 0.99)	0.62 (0.57, 0.68)	22.83 (19.63, 25.69)
5	0.62 (0.46, 0.75)	0.90 (0.85, 0.94)	0.99 (0.97,1.00)	0.54 (0.48, 0.59)	22.91 (20.26, 25.57)

Table 6. LLaMAR with more agents

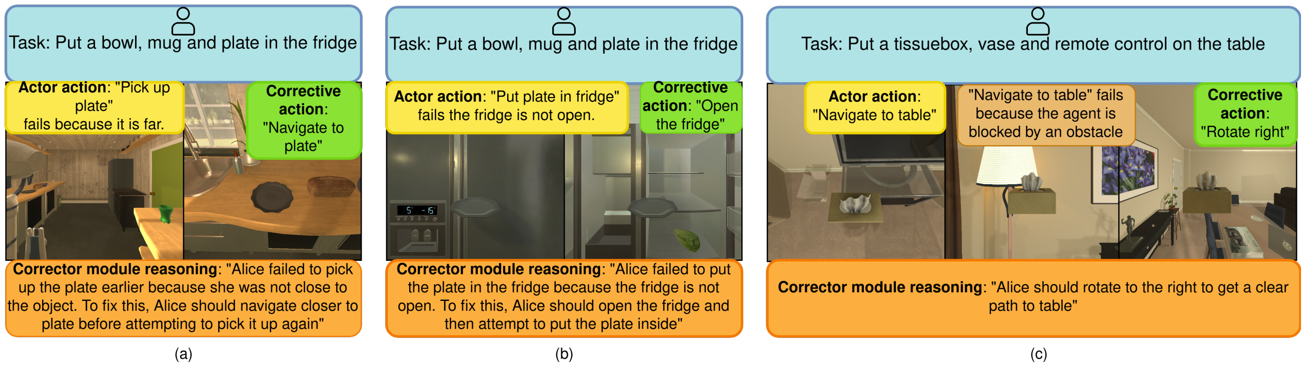

Task: Put a bowl, mug and plate in the fridge
Actor action: "Pick up plate" fails because it is far.
Corrective action: "Navigate to plate"
Corrector module reasoning: "Alice failed to pick up the plate earlier because she was not close to the object. To fix this, Alice should navigate closer to plate before attempting to pick it up again"
(a)
Task: Put a bowl, mug and plate in the fridge
Actor action: "Put plate in fridge" fails the fridge is not open.
Corrective action: "Open the fridge"
Corrector module reasoning: "Alice failed to put the plate in the fridge because the fridge is not open. To fix this, Alice should open the fridge and then attempt to put the plate inside"
(b)
Task: Put a tissuebox, vase and remote control on the table
Actor action: "Navigate to table"
"Navigate to table" fails because the agent is blocked by an obstacle
Corrective action: "Rotate right"
Corrector module reasoning: "Alice should rotate to the right to get a clear path to table"
(c)

Figure 4. A few examples of the *Corrector* module mitigate failures in predicted actions by the *Actor* module. (a) the Corrector suggests getting closer to the agent before attempting to pick it up, (b) the Corrector recommends opening the fridge because the previous action of placing the plate failed, (c) the Corrector advises rotating right so that it can access the table to place the tissue box on it when the low-level navigation policy failed to find a path to the table

text inputs, exhibits the poorest performance. This is attributed to the agents' inability to reason about visual observations, which is particularly detrimental for the *Corrector* module. Substituting GPT-4V with other vision-language models results in a decline in performance (refer Table 5) and hence we use GPT-4V as the underlying VLM for the experiments while comparing to the baselines.

F.2. Increasing the number of agents

Increasing the number of agents in the environment shows clear trends in our method's performance metrics (refer Table 6). With 2 agents, we establish a solid baseline for success rate (SR) and transport rate (TR). Adding a third agent improves both SR and TR, indicating enhanced task completion and transportation efficiency. Coverage (C) also increases, suggesting better exploration and interaction with objects. However, balance (B), which measures the even distribution of tasks among agents, decreases with more agents. This drop highlights the challenge of ensuring equal contributions from all agents in a larger multi-agent system. While SR remains high, the balance metric drops significantly from 2 to 5 agents, indicating some agents do more work than others. The number of steps (L) taken to complete tasks generally decreases with more agents, showing improved efficiency but is less pronounced from 4 to 5 agents, suggesting diminishing returns on efficiency with each additional agent. In summary, adding more agents improves task performance and efficiency but introduces challenges in maintaining balanced contributions. Addressing this imbalance is crucial for refining multi-agent planning algorithms.

F.3. Correcting Failures

: In numerous instances, the actions proposed by the *Actor* module, such as `pick up <object>`, are unsuccessful due to the agent's insufficient proximity to the target object. In such situations, the *Corrector* module uses visual feedback to learn from these failures and recommends appropriate corrective actions, such as `navigate to <object>` to facilitate closer proximity. Figure 4 shows examples where the *Corrector* module interprets low-level action failures and suggests remedies, highlighting its importance.

G. Pseudocode for LLaMAR

Algorithm 2 LLaMAR

Input: N agents, Task instruction $\mathcal{I}$, Environment env
Initialize: Memory $\mathcal{M} \leftarrow \emptyset$; Open Subtasks $\mathcal{G}_O \leftarrow \emptyset$;
Completed Subtasks $\mathcal{G}_C \leftarrow \emptyset$; Actions $a \leftarrow \emptyset$;
Corrective Actions $a_c \leftarrow \emptyset$
Actions Executed $d \leftarrow \emptyset$

```
0: $o = (o_1, \cdots, o_N) = env.reset()$
0: while $t < T$ do
0:     $\mathcal{G}_O \leftarrow$ Planner$(\mathcal{I}, o, \mathcal{G}_O, \mathcal{G}_C, \mathcal{M})$
0:     $a, \mathcal{M} \leftarrow$ Actor$(\mathcal{I}, o, a_c, \mathcal{G}_O, \mathcal{G}_C, \mathcal{M})$
0:     $o = (o_1, \cdots, o_N), d = (d_1, \cdots, d_N) = env.step(a)$
0:     $a_c \leftarrow$ Corrector$(\mathcal{I}, o, a, d, \mathcal{G}_O, \mathcal{G}_C, \mathcal{M})$
0:     $\mathcal{G}_C \leftarrow$ Verifier$(\mathcal{I}, o, a, d, \mathcal{G}_O, \mathcal{G}_C, \mathcal{M})$
0:     if $\mathcal{G}_O = \emptyset$ then
0:         break
0:     end if
0:     $t \leftarrow t + 1$
0: end while=0
```

H. Baselines

While there are a lot of impressive LLM-based multi-agent planners as mentioned in Table 3, they vary in the assumptions about access to information about the environment. We were not able to find the official codebase for the Safe Multi-Agent Planning with Conformal Prediction (Wang et al., 2024) and TwoStep (Singh et al., 2024).

- **Act**: We query the LLM with the task and the observations to suggest a high-level action.
- **Chain-of-Thought** (Wei et al., 2022): We modify the Act prompt with a chain-of-thought style addendum to let the LM reason about the possible implications while selecting a high-level action.
- **ReAct** (Yao et al., 2023): We use a ReAct-style prompting to let the LMs reason after suggesting high-level actions and possibly suggest ways to correct for any failures.
- **SmartLLM** (Kannan et al., 2023): We modify the official codebase to only include information from the local observations of the agents instead of assuming full observability.
- **CoELA** (Zhang et al., 2024): We modify the list of available high-level actions to include all possible valid combinations of actions with interactable objects in the agent's local observation. As the scene becomes more cluttered, this list and the prompt becomes combinatorially longer. In the original implementation, the list of available actions is filtered based on the feasibility of the actions as suggested by a conditional checker.

We describe the prompts used for our model as well as every baseline. Note that we show the prompt for the 2-agent case, but it is easily modified to generalize to the n-agent case. The italics and bolding added for emphasis.

H.1. LLaMAR

We describe the prompts used for each of the modules used in LLaMAR:

Prompt for Planner Module in LLaMAR

You are an excellent planner who is tasked with helping 2 embodied robots named Alice and Bob carry out a task. Both robots have a partially observable view of the environment. Hence they have to explore around in the environment to do the task.

You will get a description of the task robots are supposed to do. You will get an image of the environment from Alice's perspective and Bob's perspective as the observation input. To help you with detecting objects in the image, you will also get a list objects each agent is able to see in the environment. Here the objects are named as "<object_name>_<object_id>".
So, **along with the image inputs you will get the following information:**

### INPUT FORMAT ###
{**Task**: description of the task the robots are supposed to do,
Alice's observation: list of objects the Alice is observing,
Bob's observation: list of objects the Bob is observing,
Robots' open subtasks: list of subtasks the robots are supposed to carry out to finish the task. If no plan has been already created, this will be None.
Robots' completed subtasks: list of subtasks the robots have already completed. If no subtasks have been completed, this will be None.
Robots' combined memory: description of robots' combined memory}

Reason over the robots' task, image inputs, observations, open subtasks, completed subtasks and memory, and then output the following:
* **Reason**: The reason for why new subtasks need to be added.
* **Subtasks**: A list of open subtasks the robots are supposed to take to complete the task. Remember, as you get new information about the environment, you can modify this list. You can keep the same plan if you think it is still valid. Do not include the subtasks that have already been completed.
The "Plan" should be in a list format where the subtask are listed sequentially.
For example:
[*"locate the apple", "transport the apple to the fridge", "transport the book to the table"*]
[*"locate the cup", "go to cup", "clean cup"*]
When possible do not perform additional steps when one is sufficient (e.g. CleanObject is sufficient to clean an object, no other actions need to be done) Your output should be in the form of a python dictionary as shown below.

Example output:
{**"reason"**: *"Since the subtask list is empty, the robots need to transport the apple to the fridge and transport the book to the table."*,
"plan": *["transport the apple to the fridge", "transport the book to the table"]*}

Ensure that the subtasks are not generic statements like "do the task". They should be specific to the task at hand.
Do not assign subtasks to any particular robot. Try not to modify the subtasks that already exist in the open subtasks list. Rather add new subtasks to the list.

* NOTE: DO NOT OUTPUT ANYTHING EXTRA OTHER THAN WHAT HAS BEEN SPECIFIED
Let's work this out in a step by step way to be sure we have the right answer.

Prompt for Verifier Module in LLaMAR

You are an excellent task verifier who is tasked with helping 2 embodied robots named Alice and Bob carry out a task. Both robots have a partially observable view of the environment. Hence they have to explore around in the environment to do the task.

You will get a description of the task robots are supposed to do. You will get an image of the environment from Alice's perspective and Bob's perspective as the observation input. To help you with detecting objects in the image, you will also get a list objects each agent is able to see in the environment. Here the objects are named as "<object_name>_<object_id>".
So, **along with the image inputs you will get the following information**:

### INPUT FORMAT ###
{**Task**: description of the task the robots are supposed to do,
Alice's observation: list of objects the Alice is observing,
Alice's state: description of Alice's state,
Alice's previous action: the action Alice took in the previous step and if it was successful,
Bob's observation: list of objects the Bob is observing,
Bob's state: description of Bob's state, Bob's previous action: the action Bob took in the previous step,
Robots' open subtasks: list of open subtasks the robots in the previous step. If no plan has been already created, this will be None.
Robots' completed subtasks: list of subtasks the robots have already completed. If no subtasks have been completed, this will be None.
Robots' combined memory: description of robots' combined memory}

Reason over the robots' task, image inputs, observations, previous actions, open subtasks, completed subtasks and memory, and then **output the following**:
* **Reason**: The reason for why you think a particular subtask should be moved from the open subtasks list to the completed subtasks list.
* **Completed Subtasks**: The list of subtasks that have been completed by the robots. Note that you can add subtasks to this list only if they have been successfully completed and were in the open subtask list. If no subtasks have been completed at the current step, return an empty list.
The "Completed Subtasks" should be in a list format where the completed subtasks are listed.
For example: [*"locate the apple", "transport the apple to the fridge", "transport the book to the table"*]

Your output should be in the form of a python dictionary as shown below.

Example output:
{
"reason": *"Alice placed the apple in the fridge in the previous step and was successful and Bob picked up the the book from the table. Hence Alice has completed the subtask of transporting the apple to the fridge, Bob has picked up the book, but Bob has still not completed the subtask of transporting the book to the table"*,
"completed subtasks": [*"picked up book from the table", "transport the apple to the fridge"*]
}

* NOTE: DO NOT OUTPUT ANYTHING EXTRA OTHER THAN WHAT HAS BEEN SPECIFIED
When you output the completed subtasks, make sure to not forget to include the previous ones in addition to the new ones.
Let's work this out in a step by step way to be sure we have the right answer.

Prompt for the Actor Module in LLaMAR

You are an excellent planner and robot controller who is tasked with helping 2 embodied robots named Alice, and Bob carry out a task. All 2 robots have a partially observable view of the environment. Hence they have to explore around in the environment to do the task.

They can perform the following actions:
[*"navigate to object <object_id>", "rotate in <rotation> direction", "pick up object <object_id>", "put object on <receptacle_id>", "open object <object_id>", "close object <object_id>", "slice object <object_id>", "toggle object <object_id> on", "toggle object <object_id> off", "clean object <object_id>", "look up by angle <angle>", "look down by angle <angle>", "move in <translation> direction", "stay idle", "Done"*]

Here *"Done"* is used when all the robots have completed the main task. Only use it when you think all the subtasks are complete.
"stay idle" is used when you want the robot to stay idle for a one-time step. This could be used to wait for the other robot to complete its subtask. Use it only when you think it is necessary.
Here <*rotation*> can be one of [*"Right", "Left"*].
Here <*angle*> is the angle in degrees and can only be one of $[30, 60, 90, 120, 150, 180]$.
Here <*translation*> can be one of [*"Ahead", "Back", "Left", "Right"*].

So, **along with the image inputs you will get the following information**:

### INPUT FORMAT ###
{**Task**: description of the task the robots are supposed to do,
Alice's observation: list of objects the Alice is observing,
Alice's state: description of Alice's state,
Alice's previous action: description of what Alice did in the previous time step and whether it was successful,
Alice's previous failures: if Alice's few previous actions failed,
description of what failed,, Bob's observation: list of objects the Bob is observing,
Bob's state: description of Bob's state,
Bob's previous action: description of what Bob did in the previous time step and whether it was successful,
Bob's previous failures: if Bob's few previous actions failed, description of what failed,
Robots' open subtasks: list of subtasks supposed to carry out to finish the task. If no plan has been already created, this will be None.
Robots' completed subtasks: list of subtasks the robots have already completed. If no subtasks have been completed, this will be None.
Robots' subtask: description of the subtasks the robots were trying to complete in the previous step,
Robots' combined memory: description of robot's combined memory}

### OUTPUT FORMAT ###
First of all you are supposed to reason over the image inputs, the robots' observations, previous actions, previous failures, previous memory, subtasks and the available actions the robots can perform, and think step by step and then **output the following things**:

*** Failure reason**: If any robot's previous action failed, use the previous history, your current knowledge of the room (i.e. what things are where), and your understanding of causality to think and rationalize about why the previous action failed. Output the reason for failure and how to fix this in the next timestep. If the previous action was successful, output "None".
Common failure reasons to lookout for include: one agent blocking another so must move out of the way, agent can't see an object or its destination and must explore (such as move, rotate, or look in a different direction) to find it, agent doing extraneous actions (such as drying objects when cleaning), etc. If the previous action was successful, output "None".

*** Memory**: Whatever important information about the scene you think you should remember for the future as a memory. Remember that this memory will be used in future steps to carry out the task. So, you should not include information that is not relevant to the task. You can also include information that is already present in its memory if you think it might be useful in the future.

(CONTINUED) Prompt for the Actor Module in LLaMAR

* **Reason**: The reasoning for what each robot is supposed to do next

* **Subtask**: The subtask each robot should currently try to solve, choose this from the list of open subtasks.

* **Action**: The actions the robots are supposed to take just in the next step such that they make progress towards completing the task. Make sure that these suggested actions make these robots more efficient in completing the task as compared to only one agent solving the task.
Your output should just be in the form of a python dictionary as shown below.

Examples of output:
Example 1:
{ **"failure reason"**: *"Bob failed to put the mug in the cabinet earlier because Alice was blocking it when she was putting the knife. To fix this, Alice should close the cabinet and move away , Charlie should move away to a different open area than Alice to avoid congestion, and Bob should wait until the next timestep until Alice can move aside."*,
"memory": *"Alice finished putting the knife in the cabinet when Alice was at co-ordinates (1, .5) and was facing north. Bob wanted to put the mug in the cabinet when Bob was at co-ordinates (1, 0.25) and was facing north."*,
"reason": *"Alice can close the cabinet door and then later back out in order help Bob with completing the task. Bob can be idle until the next timestep when Alice moves aside, by then Bob can navigate to the cabinet."*,
"subtask": *"Alice is currently closing the cabinet door, Bob is currently waiting to get to navigate to the cabinet"*,
"Alice's action" : *"close the Cabinet_1"*,
"Bob's action" : *"stay idle"*
}

Example 2: {
"failure reason": *"Bob failed to clean the cup earlier because Bob had not navigated to it, Bob assumed the cup to be in the sink which was erroneous. To fix this, Bob should navigate to the cup and in the next step clean cup."*,
"memory": *"Alice finished navigating to the dish when Alice was at co-ordinates (-.5, .5) and was facing east. Bob was not able to clean the cup in the cabinet when Bob was at co-ordinates (1, .25) and was facing north."*,
"reason": *"Alice can now clean the dish since Alice has navigated to it. Bob should navigate to the cup in order to be close enough to clean the cup."*,
"subtask": *"Alice is currently trying to clean the dish, Bob is currently trying to navigate to the cup"*,
"Alice's action" : *"clean the dish object"*,
"Bob's action" : *"navigate to the cup"* }
Note that the output should just be a dictionary similar to the example outputs.

### Important Notes ###
* The robots can hold only one object at a time.
For example: If Alice is holding an apple, she cannot pick up another object until she puts the apple down.
* Even if the robot can see objects, it might not be able to interact with them if they are too far away. Hence you will need to make the robot navigate closer to the objects they want to interact with.
For example: An action like "pick up <object_id>" is feasible only if robot can see the object and is close enough to it. So you will have to navigate closer to it before you can pick it up.
* In some scenarios, the agents might not see the objects that they want to interact with. In such cases, you will have to make the robot explore the environment to find the object. In such scenarios you can use actions to rotate in place or look up / down or navigate to explore the environment.
* If you open an object, please ensure that you close it before you navigate to a different place.
* Opening object like drawers, cabinets, fridge can block the path of the robot. So open objects only when you think it is necessary.

* NOTE: DO NOT OUTPUT ANYTHING EXTRA OTHER THAN WHAT HAS BEEN SPECIFIED

H.2. Act

We describe the prompt used for the Act baseline:

Prompt for the Act Baseline

You are an excellent planner and robot controller who is tasked with helping 2 embodied robots named Alice and Bob carry out a task. Both robots have a partially observable view of the environment. Hence they have to explore around in the environment to do the task.

They can perform the following actions:
[*"navigate to object <object_id>", "rotate in <rotation> direction", "pick up object <object_id>", "put object on <receptacle_id>", "open object <object_id>", "close object <object_id>", "slice object <object_id>", "toggle object <object_id> on", "toggle object <object_id> off", "clean object <object_id>", "look up by angle <angle>", "look down by angle <angle>", "move in <translation> direction", "stay idle", "Done"*]

Here *"Done"* is used when all the robots have completed the main task. Only use it when you think all the subtasks are complete.
"stay idle" is used when you want the robot to stay idle for a one-time step. This could be used to wait for the other robot to complete its subtask. Use it only when you think it is necessary.
Here <*rotation*> can be one of [*"Right", "Left"*].
Here <*angle*> is the angle in degrees and can only be one of $[30, 60, 90, 120, 150, 180]$.
Here <*translation*> can be one of [*"Ahead", "Back", "Left", "Right"*].

You need to suggest the action that each robot should take at the current time step.

### **Important Notes** ###
* The robots can hold only one object at a time.
For example: If Alice is holding an apple, she cannot pick up another object until she puts the apple down.
* Even if the robot can see objects, it might not be able to interact with them if they are too far away. Hence you will need to make the robot navigate closer to the objects they want to interact with.
For example: An action like "pick up <object_id>" is feasible only if robot can see the object and is close enough to it. So you will have to navigate closer to it before you can pick it up.
* In some scenarios, the agents might not see the objects that they want to interact with. In such cases, you will have to make the robot explore the environment to find the object. In such scenarios you can use actions to rotate in place or look up / down or navigate to explore the environment.
* If you open an object, please ensure that you close it before you navigate to a different place.
* Opening object like drawers, cabinets, fridge can block the path of the robot. So open objects only when you think it is necessary.

### **INPUT FORMAT** ###
* You will get a **description of the task** robots are supposed to do.
* You will get an **image of the environment at the current time step** from Alice's perspective and Bob's perspective as the observation input. Here the objects are named as *"<object_name>_<object_id>"*.
* You will get a trace of the steps taken by the robots and the actions they took at each time step and whether it was successful or not.

### **OUTPUT FORMAT** ###
In your output, do not have any extra text or content outside of the python dictionary as below. Do NOT put any text, spaces, or enter keys (i.e. "/n") outside of it.

Your output should ONLY be in the form of a python dictionary, without any reasoning or extra text, as shown below:
{**"Alice"**: *"action to be taken by Alice"*,
"Bob": *"action to be taken by Bob*}

For example: If you think Alice should pick up an apple and Bob should navigate to the fridge, you will have to give the output as:
{**"Alice"**: *"pick up apple"*,
"Bob": *"navigate to fridge"*}
* NOTE: DO NOT OUTPUT ANYTHING EXTRA OTHER THAN WHAT HAS BEEN SPECIFIED

H.3. ReAct

We describe the prompt used for the ReAct baseline:

Prompt for ReAct Baseline

You are an excellent planner who is tasked with helping 2 embodied robots named Alice and Bob carry out a task. Both robots have a partially observable view of the environment. Hence they have to explore around in the environment to do the task.

They can perform the following actions: [*"navigate to object <object_id>", "rotate in <rotation> direction", "pick up object <object_id>", "put object on <receptacle_id>", "open object <object_id>", "close object <object_id>", "slice object <object_id>", "toggle object <object_id> on", "toggle object <object_id> off", "clean object <object_id>", "look up by angle <angle>", "look down by angle <angle>", "move in <translation> direction", "stay idle", "Done"*]
Here "Done" is used when all the robots have completed the main task. Only use it when you think all the subtasks are complete.
"stay idle" is used when you want the robot to stay idle for a one-time step. This could be used to wait for the other robot to complete its subtask. Use it only when you think it is necessary.
Here <rotation> can be one of [*"Right", "Left"*].
Here <angle> is the angle in degrees and can only be one of $[30, 60, 90, 120, 150, 180]$.
Here <translation> can be one of [*"Ahead", "Back", "Left", "Right"*].

You need to suggest the action that each robot should take at the current time step.
### **Important Notes** ###
* The robots can hold only one object at a time.
For example: If Alice is holding an apple, she cannot pick up another object until she puts the apple down.
* Even if the robot can see objects, it might not be able to interact with them if they are too far away. Hence you will need to make the robot navigate closer to the objects they want to interact with.
For example: An action like "pick up <object_id>" is feasible only if robot can see the object and is close enough to it. So you will have to navigate closer to it before you can pick it up.
* In some scenarios, the agents might not see the objects that they want to interact with. In such cases, you will have to make the robot explore the environment to find the object. In such scenarios you can use actions to rotate in place or look up / down or navigate to explore the environment.
* If you open an object, please ensure that you close it before you navigate to a different place.
* Opening object like drawers, cabinets, fridge can block the path of the robot. So open objects only when you think it is necessary.
### **INPUT FORMAT** ###
* You will get a description of the task robots are supposed to do.
* You will get an image of the environment at the current time step from Alice's perspective and Bob's perspective as the observation input. Here the objects are named as "<object_name>_<object_id>".
* You will get a trace of the steps taken by the robots and the actions they took at each time step and whether it was successful or not.

### **OUTPUT FORMAT** ###
You are supposed to think and suggest the action each robot is supposed to take at the current time step. Before suggesting an action you need to think, which requires that you reason over the inputs and logically reflect on the task, observation and course of actions needed to complete the task.
Output Requirements: At each time step you must ONLY output a PYTHON DICTIONARY of the following two elements:
***First Element**: Key = **"Think"** | Value:(Type: String): A logical reflection of the best action to be taken given the inputs: task at hand, observations, and trace.
***Second Element**: Key = **"Action"** | Value:(Type: Python Dictionary):
The value should be in the form of a python dictionary as shown below.
{**"Alice"**: *"action to be taken by Alice"*, **"Bob"**: *"action to be taken by Bob"*}

For example: If you think Alice should pick up an apple and Bob should navigate to the fridge, you will have to give the output as: {**"Alice"**: *"pick up apple"*, **"Bob"**: *"navigate to fridge"*}
Here is an **example output**:
{**"Think"**: *"To solve the task, I need to find and put the apple. The apple is likely to be on the countertop or table. Then find the fridge."*, **"Action"**: {**"Alice"**: *"pick up apple"*, **"Bob"**: *"navigate to fridge"*} }
* NOTE: DO NOT OUTPUT ANYTHING EXTRA OTHER THAN WHAT HAS BEEN SPECIFIED

H.4. Chain of Thought

We describe the prompt used for the Chain-of-Thought baseline:

Prompt for Chain of Thought Baseline

You are an excellent planner who is tasked with helping 2 embodied robots named Alice and Bob carry out a task. Both robots have a partially observable view of the environment. Hence they have to explore around in the environment to do the task.

They can perform the following actions: [*"navigate to object <object_id>", "rotate in <rotation> direction", "pick up object <object_id>", "put object on <receptacle_id>", "open object <object_id>", "close object <object_id>", "slice object <object_id>", "toggle object <object_id> on", "toggle object <object_id> off", "clean object <object_id>", "look up by angle <angle>", "look down by angle <angle>", "move in <translation> direction", "stay idle", "Done"*] Here "Done" is used when all the robots have completed the main task. Only use it when you think all the subtasks are complete. "stay idle" is used when you want the robot to stay idle for a one-time step. This could be used to wait for the other robot to complete its subtask. Use it only when you think it is necessary. Here <rotation> can be one of [*"Right", "Left"*].
Here <angle> is the angle in degrees and can only be one of [30, 60, 90, 120, 150, 180].
Here <translation> can be one of [*"Ahead", "Back", "Left", "Right"*].

You need to suggest the action that each robot should take at the current time step.

### **Important Notes** ###
* The robots can hold only one object at a time. For example: If Alice is holding an apple, she cannot pick up another object until she puts the apple down.
* Even if the robot can see objects, it might not be able to interact with them if they are too far away. Hence you will need to make the robot navigate closer to the objects they want to interact with. For example: An action like "pick up <object_id>" is feasible only if robot can see the object and is close enough to it. So you will have to navigate closer to it before you can pick it up.
* In some scenarios, the agents might not see the objects that they want to interact with. In such cases, you will have to make the robot explore the environment to find the object. In such scenarios you can use actions to rotate in place or look up / down or navigate to explore the environment.
* If you open an object, please ensure that you close it before you navigate to a different place.
* Opening object like drawers, cabinets, fridge can block the path of the robot. So open objects only when you think it is necessary.

### **INPUT FORMAT** ###
* You will get a **description of the task** robots are supposed to do.
* You will get an **image of the environment at the current time step** from Alice's perspective and Bob's perspective as the observation input. Here the objects are named as "<object_name>_<object_id>".
* You will get a **trace of the steps taken by the robots** and the actions they took at each time step and whether it was successful or not.

### **OUTPUT FORMAT** ###
You are supposed to FIRST reason through the situation logically and step by step, then suggest the action each robot is supposed to take at the current time step.
In your output, do not have any extra text or content outside of the python dictionary as below.
Your output should ONLY be in the form of a python dictionary as shown below:
{**"reason"**: *"Reasoning for action plan...."*, **"Alice"**: *"action to be taken by Alice"*, **"Bob"**: *"action to be taken by Bob"*}
Put all of your reasoning inside of the "reason" key of the dictionary. Do NOT put any text, spaces, or enter keys (i.e. "/n") outside of it.

For example: If you think Alice should pick up an apple and Bob should navigate to the fridge, you will have to give the output as:
{**"reason"**: *"since the subtask list is empty, the robots need to transport the apple to the fridge"*, **"Alice"**: *"pick up apple"*, **"Bob"**: *"navigate to fridge"*}

Let's think step by step, but make sure to put all of your reasoning inside of the "reason" key of the dictionary!
* NOTE: DO NOT OUTPUT ANYTHING EXTRA OTHER THAN WHAT HAS BEEN SPECIFIED

H.5. SmartLLM

We adapt the prompt from the official codebase of SmartLLM (`master` branch; commit `#be42930050f7d4d8f2fad027aff14a699c3300aa`) as given here: https://github.com/SMARTlab-Purdue/SMART-LLM/blob/master/scripts/run_llm.py with a slight modification. Instead of letting the agents access all the objects in the environment through the simulator metadata, we just give the list of objects visible from the agents' point-of-view.

H.6. CoELA

We adapt the prompt from the official codebase of CoELA (`master` branch: commit `#3d34de46dc77f9aaabe438cd2b92ea6c5c04973a`) as given here: https://github.com/UMass-Foundation-Model/Co-LLM-Agents/tree/master/tdw_mat/LLM. We modify some aspects of the prompt as described: Instead of relying on the simulator/pre-defined conditional logic for generating the list of available action options, we give a list of all possible actions based on the observation. This includes the option to send the communication message, all navigation actions, and all combinations of valid actions with the interactable objects in the current observation.

I. Open Source VLMs

We list the source of the weights we used for the open-source VLMs:

- **Idefics 2** (Laurençon et al., 2023; 2024): We use the 8B base model fine-tuned on a mixture of supervised and instruction datasets (text-only and multimodal datasets) from HuggingFace. The weights were downloaded from https://huggingface.co/HuggingFaceM4/idefics2-8b with the commit `#2c031da2dc71f3ac989f9efa9b8ff476df3842c0`. We chose Idefics because it is able to take multiple images as input similar to GPT-4V and reason on them.
- **LLaVA** (Liu et al., 2023b): We use the 7B model t trained by fine-tuning LLaMA/Vicuna on GPT-generated multimodal instruction-following data. The weights were downloaded from https://huggingface.co/llava-hf/llava-1.5-7b-hf with the commit `# 05ae2434cbb430be33edcba0c5203e7023f785b7`.
- **CogVLM** (Hong et al., 2023a): We use the 18B model. The weights were downloaded from https://huggingface.co/THUDM/cogagent-chat-hf with the commit `# d519da3b191401234f4bd86ce1c287c61bc276a3`.

J. Limitations and Future Work

Higher number queries to the LM: Since each high-level decision step requires querying 4 different LM-based modules, the cost and the compute times are higher than other baselines, especially compared to the plan-and-execute baselines like SmartLLM. An interesting future direction to improve this would be to fine-tune smaller LMs with trajectories collected in the simulator (eg: ALFRED (Shridhar et al., 2019)) as done in (Zhang et al., 2024). Another potential direction worth exploring is using different sizes of LMs for each module based on their specific utility.

Limited spatial reasoning: Although we use both textual descriptions and visual features to guide the language model's actions, it still lacks the ability to reason about the spatial features of the environment. Spatial reasoning is crucial in scenarios such as navigating around obstacles to reach an object, or determining the shortest path to collect multiple items scattered across different locations. One way to address this limitation is to inject information about the 3D world into the LM, as done in (Hong et al., 2023b), which is an interesting direction for future work.

Performance limited by the underlying VLM: Although LMs make correct reasoning most of the time, they still occasionally make mistakes, including misunderstanding the environment rules specified in the prompt. For example, the agent assumes that the cleaning task requires putting soap, drying, and putting it in the sink when all it needs is the action "*CleanObject*", and can't figure out the appropriate level of abstraction. The performance of the algorithm is limited by the instruction following and reasoning capability of the underlying LM (Kambhampati, 2024; Valmeekam et al., 2023b); this calls for developing LMs that are fine-tuned to instruction-image pairs relevant to the environment (as done in (Zhang et al., 2024)).