MAP-THOR: Benchmarking Long-Horizon Multi-Agent Planning Frameworks 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

Evaluating embodied multi-agent planners necessitates robust and versatile benchmarks. We introduce MAP-THOR (Multi-Agent Planning in AI2-THOR), a benchmark specifically designed to assess the performance of embodied multi-agent planning systems in realistic, partially observable environments within the AI2-THOR environment. Existing benchmarks offer extensive environments for single-agent tasks, but fail to capture the complexities inherent in multi-agent interactions, non-stationarity, partial observability and long-horizon planning. Addressing these gaps, MAP-THOR facilitates the development of frameworks that allocate tasks and enable coordination among multiple agents. MAP-THOR introduces a comprehensive suite of household tasks demanding collaboration and adaptation to dynamic environmental changes, mirroring real-world scenarios. Our benchmark includes detailed metrics for success rate, efficiency, and collaborative effectiveness, setting a new standard for evaluating multi-agent planning systems. Through rigorous experiments, we show that MAP-THOR offers a robust evaluation framework for language model (LM)-based multi-agent planning systems. Ultimately, we hope that MAP-THOR serves as a standard benchmark to identify embodied multi-agent planning frameworks that systematically improve generalization for long-horizon partially observable planning.

[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).

1. Introduction

Building cooperative embodied agents that can engage in and help humans in multi-agent tasks is a valuable yet challenging endeavor. This requires developing effective multi-agent planning frameworks for advancing the capabilities of robotic systems to handle complex real-world tasks such as household chores (Brodeur et al., 2017; Wu et al., 2018; Kolve et al., 2017; Chang et al., 2017; Carroll et al., 2019; Gan et al., 2020; Shridhar et al., 2019), search and rescue missions (Rahman et al., 2022), and industrial and workspace operations (Mandi et al., 2023; Sengar et al., 2022; Baghel et al., 2021; Kapoor et al., 2023) alongside other agents and humans. Humans are skilled at solving complex tasks in a group by cooperating and communicating with others (Woolley et al., 2010). Thus, evaluating such planners requires robust benchmarks that accurately reflect the challenges of real-world environments such as the complexity of perception, partial observation, long planning horizon, natural language communication, and so on (Deitke et al., 2022). However, existing benchmarks primarily focus on single-agent tasks and fail to capture the complexities of multi-agent systems. This gap in the literature hinders the development of more advanced and capable multi-agent planning frameworks.

Multi-agent planning in realistic environments presents several challenges (Albrecht et al., 2024). Firstly, the interactions between agents introduce non-stationarity, as the actions of one agent can influence the observations and actions of others (Oliehoek & Amato, 2016; Tan, 1993; Tampuu et al., 2015). Secondly, partial observability requires agents to make decisions based on incomplete information (Amato, 2024), often leading to suboptimal actions. Thirdly, long-horizon planning demands the ability to sequence actions over extended periods, making it difficult to maintain coherent and effective strategies (Zhang et al., 2021). These factors make it challenging to develop and evaluate multi-agent planners that can generalize across diverse tasks and environments.

To address these challenges, we introduce MAP-THOR, a benchmark specifically designed for evaluating embodied multi-agent planning systems. MAP-THOR is built within the AI2-THOR environment and features a comprehensive suite of household tasks that require multiple agents to collaborate and adapt to dynamic changes . Our benchmark includes detailed metrics for success rate, efficiency, and collaborative effectiveness, setting a new standard for evaluating multi-agent planning systems.

We verify the effectiveness of MAP-THOR through rigorous experiments that benchmark various language model (LM)-based multi-agent planning frameworks. Our results demonstrate that MAP-THOR provides a comprehensive and challenging evaluation environment, allowing for meaningful comparisons between different approaches.

Our main contributions are as follows:

- We introduce **MAP-THOR (Multi-Agent Planning in THOR)**, a comprehensive benchmark suite of household tasks within the AI2-THOR simulator under partial observability to standardize methodologies and metrics for evaluating multi-agent planning effectiveness and robustness.
- Detailed metrics for success rate, efficiency, and collaborative effectiveness among agents.
- Rigorous experiments demonstrating the effectiveness of MAP-THOR in benchmarking state-of-the-art LM-based multi-agent planning frameworks.

In the subsequent sections, we refer to language models as LMs, large language models as LLMs and vision language models as VLMs.

2. Related Work

The AI community has developed numerous platforms to drive algorithmic advances in both single-agent and multi-agent reinforcement learning. Notable environments include the Arcade Learning Environment (Bellemare et al., 2013), OverCookedAI (Carroll et al., 2019), Minecraft-based Malmo (Johnson et al., 2016), maze-based DeepMind Lab (Beattie et al., 2016) , Doom-based ViZDoom (Kempka et al., 2016), Google football (Kurach et al., 2020), StarCraft II (Vinyals et al., 2019), DOTA (Berner et al., 2019). These platforms have primarily focused on developing and testing reinforcement learning algorithms in controlled, often static settings.

In recent years, several photorealistic simulator environments have been developed to simulate more complex, real-world scenarios. Examples include AI2-THOR (Kolve et al., 2017), Habitat (Savva et al., 2019; Szot et al., 2021; Puig et al., 2023), HoME (Brodeur et al., 2017), MatterPort3D (Chang et al., 2017), IGibson (Li et al., 2021; Shen et al., 2021), ThreeDWorld (Gan et al., 2020) and VirtualHomeSocial (Puig et al., 2018). These environments have enabled the creation of frameworks for single embodied agent tasks. For instance, FactorWorld (Xie et al., 2023), KitchenShift (Xing et al., 2021) and Colosseum (Pumacay et al., 2024) focus on policy generalization for manipulation tasks by evaluating all potential perturbations. However, these frameworks do not address the complexities of multi-agent interactions and planning.

ALFRED (Shridhar et al., 2019), built on top of AI2-THOR, focuses on single-agent tasks where an agent must follow language instructions to perform household activities in a visually rich environment. While ALFRED excels in evaluating single-agent capabilities, it does not address the complexities of multi-agent interactions, non-stationarity, or collaborative planning. MAP-THOR extends this line of research by incorporating multi-agent scenarios, necessitating coordination and dynamic adaptation among agents.

RoCo (Mandi et al., 2023) leverages LLMs to enable dialectic collaboration between multiple robots in structured environments where detailed knowledge about the environment is available. In contrast, MAP-THOR challenges planners in partially observable settings without privileged information, focusing on real-world applicability and adaptability to dynamic changes.

SMART-LLM (Kannan et al., 2023) presents a multi-agent planning framework using LLMs to generate and coordinate plans for multiple robots. This approach assumes a static and fully observable environment, limiting its applicability to real-world scenarios. While SMART-LLM introduces a dataset of 36 tasks within AI2-THOR, these tasks are limited to single floor plans and do not support multi-agent collaboration for certain tasks, such as `Pick up the pillow`, `Open the laptop to turn it on`, and `Turn off the lamp`. MAP-THOR, on the other hand, introduces tasks that require agents to explore and gather information dynamically, providing a more realistic evaluation of multi-agent planning systems.

While prior research has significantly advanced the evaluation of multi-agent embodied-AI frameworks, there remains a need for a standardized and systematic assessment of these proposed solutions. To address this gap, we introduce MAP-THOR, a benchmark designed to build upon previous efforts. MAP-THOR aims to evaluate multi-agent embodied task planners, focusing on challenges such as partial observability, non-stationarity, and long planning horizons.

3. MAP-THOR

To simulate open-ended, long-horizon tasks that resemble everyday activities, we build MAP-THOR on the AI2Thor simulator (Kolve et al., 2017), which supports a diverse

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

Figure 1. Photorealistic rendering of household scenarios in the AI2Thor simulator enables the usage of multiple autonomous robots to carry out daily tasks.

set of interactions and photorealistic rendering. Figure ?? shows a few scenes from the simulator. MAP-THOR comprises of 45 tasks, each defined for five distinct floor plans. The tasks vary in the ambiguity in the natural language instruction given as input to the agents. These tasks are solvable by both single and multiple agents. We also include automatic checker modules to verify subtask completion and evaluate plan quality.

An increase in ambiguity in the language task instruction increases the difficulty level of the tasks. The complete task list of each category can be found in the Appendix C.

- **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.

A detail list of the task types is provided in Appendix C

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 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.
- **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.
- **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.

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.

We highlight the distinction between task, subtask and high-level action in the Appendix A

4. Experiments

We instantiate the problem with multiple agents cooperating to accomplish a long-horizon rearrangement task (Batra et al., 2020) in an indoor environment. The agents do not know the objects present in the environment *a priori* and are encouraged to explore the environment to gather more information to complete the task. We list the set of observation and action space in the Appendix B.

Baselines

For replication real-world setup, we make modifications to the baselines to make them work in partially observable settings with limited reliance on the simulator. More details about implementations can be found in Appendix D.

- **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.

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) for details about CMAS and DMAS).

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)

Table 1. Comparison of evaluation metrics against baselines averaged across all tasks.

Baseline Comparisons: Table 1 compares the baselines in a 2-agent scenario using GPT-4V as the underlying VLM. Act and ReAct show similar performance, with Act struggling due to its lack of strategic planning or correction and ReAct performing 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 failure 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). Previous research (Chen et al., 2023) confirms CMAS frameworks are more effective than DMAS frameworks.

5. Conclusion

In this paper, we introduced MAP-THOR, a benchmark designed to evaluate the performance of multi-agent planning systems in realistic, partially observable environments within the AI2-THOR simulator. MAP-THOR addresses a critical gap in the existing benchmarks, which predominantly focus on single-agent tasks and fail to capture the complexities of multi-agent interactions, non-stationarity, and long-horizon planning. Our test suite sets a new standard for evaluating multi-agent planning systems by including detailed metrics for success rate, efficiency, and collaborative effectiveness. We benchmarked several state-of-the-art language model (LM) based multi-agent planning frameworks. The results showed that MAP-THOR offers a rigorous and comprehensive evaluation, facilitating the identification of frameworks that significantly improve generalization for long-horizon, partially observable planning tasks. Future work will involve expanding MAP-THOR to include more diverse tasks and environments, further enhancing its applicability and robustness. We also plan to integrate more advanced evaluation metrics and explore the use of MAP-THOR in real-world scenarios, bridging the gap between simulation and practical deployment.

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

Albrecht, S. V., Christianos, F., and Schäfer, L. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. URL `https://www.marl-book.com`.

Amato, C. (a partial survey of) decentralized, cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2405.06161*, 2024.

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.

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`.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

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`.

Carroll, M., Shah, R., Ho, M. K., Griffiths, T., Seshia, S., Abbeel, P., and Dragan, A. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., and Zhang, Y. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

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.

Deitke, M., Batra, D., Bisk, Y., Campari, T., Chang, A. X., Chaplot, D. S., Chen, C., D'Arpino, C. P., Ehsani, K., Farhadi, A., et al. Retrospectives on the embodied ai workshop. *arXiv preprint arXiv:2210.06849*, 2022.

Gan, C., Schwartz, J., Alter, S., Mrowca, D., Schrimpf, M., Traer, J., De Freitas, J., Kubilius, J., Bhandwaldar, A., Haber, N., et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020.

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. The malmo platform for artificial intelligence experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pp. 4246–4247. AAAI Press, 2016. ISBN 9781577357704.

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.

Kapoor, A., Swamy, S., Bachiller, P., and Manso, L. J. Socnavgym: A reinforcement learning gym for social navigation. In *2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pp. 2010–2017, 2023. doi: 10.1109/RO-MAN57019.2023.10309591.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)*, pp. 1–8. IEEE, 2016.

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.

Kurach, K., Raichuk, A., Stańczyk, P., Zajac, M., Bachem, O., Espeholt, L., Riquelme, C., Vincent, D., Michalski, M., Bousquet, O., et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 4501–4510, 2020.

Li, C., Xia, F., Martín-Martín, R., Lingelbach, M., Srivastava, S., Shen, B., Vainio, K., Gokmen, C., Dharan, G., Jain, T., et al. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. *arXiv preprint arXiv:2108.03272*, 2021.

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

Oliehoek, F. A. and Amato, C. A concise introduction to decentralized pomdps. In *SpringerBriefs in Intelligent Systems*, 2016. URL https://api.semanticscholar.org/CorpusID:3263887.

Puig, X., Ra, K. K., Boben, M., Li, J., Wang, T., Fidler, S., and Torralba, A. Virtualhome: Simulating household activities via programs. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8494–8502, 2018. URL https://api.semanticscholar.org/CorpusID:49317780.

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.

Pumacay, W., Singh, I., Duan, J., Krishna, R., Thomason, J., and Fox, D. The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint arXiv:2402.08191*, 2024.

Rahman, A., Bhattacharya, A., Ramachandran, T., Mukherjee, S., Sharma, H., Fujimoto, T., and Chatterjee, S. Adversar: Adversarial search and rescue via multi-agent reinforcement learning. In *2022 IEEE International Symposium on Technologies for Homeland Security (HST)*, pp. 1–7. IEEE, 2022.

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.

Shen, B., Xia, F., Li, C., Martín-Martín, R., Fan, L., Wang, G., Pérez-D'Arpino, C., Buch, S., Srivastava, S., Tchapmi, L., et al. igibson 1.0: a simulation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7520–7527. IEEE, 2021.

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

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

Szot, A., Clegg, A., Undersander, E., Wijmans, E., Zhao, Y., Turner, J., Maestre, N., Mukadam, M., Chaplot, D., Maksymets, O., Gokaslan, A., Vondruš, V., Dharur, S., Meier, F., Galuba, W., Chang, A., Kira, Z., Koltun, V., Malik, J., Savva, M., and Batra, D. Habitat 2.0: Training Home Assistants to Rearrange their Habitat. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. Multiagent Cooperation and Competition with Deep Reinforcement Learning. *CoRR*, abs/1511.08779, 2015. URL http://arxiv.org/abs/1511.08779.

Tan, M. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pp. 330–337. Morgan Kaufmann, 1993.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350 – 354, 2019. URL https://api.semanticscholar.org/CorpusID:204972004.

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.

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.

Woolley, A., Chabris, C., Pentland, A., Hashmi, N., and Malone, T. Evidence of a collective intelligence factor in the performance of human groups. *Science (New York, N.Y.)*, 330:686–8, 10 2010. doi: 10.1126/science.1193147.

Wu, Y., Wu, Y., Gkioxari, G., and Tian, Y. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018.

Xie, A., Lee, L., Xiao, T., and Finn, C. Decomposing the generalization gap in imitation learning for visual robotic manipulation. *arXiv preprint arXiv:2307.03659*, 2023.

Xing, E., Gupta, A., Powers, S., and Dean, V. Kitchenshift: Evaluating zero-shot generalization of imitation-based policy learning under domain shifts. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2021. URL https://openreview.net/forum?id=DdglKo8hBq0.

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

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.

Zhang, K., Yang, Z., and Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

A. 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 satisfied 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 mid-level 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

B. Environment

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).

B.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.

B.2. Action Space

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

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

C. Task Types

The complete list of task for each task type:

- **Explicit object type, explicit quantity and target**:
  - put bread, lettuce, and a tomato in the fridge
  - Put the pots and pans on the stove burners
  - Slice the bread and tomato and crack the egg
  - Put the butter knife, bowl, and mug in the sink
  - Turn off the faucet and light if either is on
  - Put the tissue box, keys, and plate in the box
  - Put the computer, book, and pen on the couch
  - Put the bowl and tissue box on the table
  - Put apple in fridge and switch off the light
  - Put the watch and Keychain inside the drawer
  - Wash the bowl, mug, pot, and pan
  - Put the Box on the sofa and the bowl in the box
- **Explicit object type and explicit target**: Here we explicitly describe the object type but keep the quantity of the objects ambiguous. E.g. `Put all the apples in the fridge`. For this, the agents have to explore the environment to ensure that they find all of them.
  - Open all the drawers (make sure they are closed initially)
  - Open all the cabinets
  - Turn on all the stove knobs
  - Put all the vases on the table
  - Put all the potatoes in the bowl
  - Put all pencils and pens in the box
  - Move all lamps next to the door
  - Turn off all light switches
  - Turn on all light switches
- **Explicit target, implicit object types**: The object types are implicitly defined whereas the target is explicitly defined. E.g. `Put all groceries in the fridge`. This tests whether the model can identify objects of certain categories.
  - Put all groceries in the fridge (should identify the tomato, bread, apple, potato, and lettuce)
  - Put all shakers in the closest drawer (should identify the salt shaker and pepper shaker)
  - Put all tableware on the countertop (should identify the bowl, plate, mug)
  - Put all food on the countertop (should identify the tomato, bread, apple, potato, and lettuce)
  - Put all school supplies on the couch (should identify the pencil, computer, and book)
  - Put all kitchenware in the cardboard box (should move the bowl and plate)
  - Put all silverware in the sink
  - Move everything on the table to the desk (should move the laptop, pencil, pen, plate, credit card, book, and newspaper)
  - Slice the lettuce, trash the mug and switch off the light
  - Put all electronics on the couch
  - Make a dish by microwaving eggs and tomato
  - Put all readable objects on the sofa
  - Wash all fruits
- **Implicit target and object types**: Here both the object type and the target are explicitly defined. E.g. `Clear the floor by placing the items at their appropriate positions.` Here the model is expected to keep items like pens, book, laptop on the study table, litter in the trash can, etc.
  - Clear the floor by placing items at their appropriate positions (depending on what's on the floor)
  - Clear the table by placing the items in their appropriate positions (depends on the floorplan, e.g. bread, apple, tomato, knife, bowl, book)
  - Clear the countertop by placing items in their appropriate positions (should move the lettuce, mug, and paper towel

roll)
- Clear the desk by placing the items in other appropriate positions (should move the statue, watch, and remote control)
- Clear the table by placing the items in other appropriate positions (should move the book, credit card, laptop, plate, newspaper, pen, and pencil)
- Clear the couch by placing the items in other appropriate positions (should move the pillow)
- Make the living room dark
- Make a mug of coffee and toast the bread
- Trash all groceries
- Slice all sliceable objects

D. Baselines

While there are a lot of impressive LLM-based multi-agent planners, 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). 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.

D.1. 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

D.2. 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

D.3. 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

D.4. 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.

D.5. 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.