# AdaTAMP: Adaptive Task and Motion Planning with LLM-based Embodied Agents


## Abstract

In this paper, we present AdaTAMP, a novel LLM-based adaptive task and motion planning framework that combines continuous task planning and motion planning with a self-feedback loop for real-time correction and multi-agent cooperation for embodied agents in household environments. AdaTAMP enables embodied agents to handle dynamic environments, correct errors, and collaborate effectively in long-horizon, multi-agent tasks. Tested on the VirtualHome simulator, AdaTAMP outperforms baseline planners in success rates and efficiency, particularly in complex scenarios.


## 1 Introduction and Related Work

Autonomous agents capable of performing tasks accurately while being able to perceive and respond in a dynamic environment has been an important goal in artificial intelligence (AI) [12, 17]. Sequential long-horizon planning for embodied agents requires not only reasoning and formulating a series of dependent actions over extended time scales, but also ensuring the feasibility of executing these actions under environment and physical constraints [8, 27]. Task and Motion Planning (TAMP) is a methodological framework that hierarchically divides planning into two stages: high-level symbolic planning to reason over abstract action sequences and low-level continuous motion planning to compute feasible trajectories subject to physical constraints [6, 22]. Classical solutions like the Planning Domain Definition Language (PDDL) [5] typically use specific modules to interface symbolic and continuous representations by generating real-valued parameters for symbolic actions and providing numerical goals to the motion planner [4, 10]. However, these approaches often require manual domain-specific design and struggle with long-horizon task planning and real-time adaptability, limiting generalization in dynamic environments [22].

Recent advancements in Large Language Models (LLMs) [1, 21], such as GPT-4 [1], have demonstrated capabilities in reasoning [18, 23], problem-solving [11, 15], and embodied task planning [9, 19, 24]. LLMs have shown the ability to serve as task planners by decomposing long-horizon tasks into a sequence of sub-tasks based on natural language instructions [14]. For instance, Huang et al. translates high-level plans (e.g., "make breakfast") into mid-level actionable steps (e.g., "open fridge") using language models [9]. Some researchers have also proposed iterative self-refinement strategies that refine initial plans through environmental feedback to improve complex sequential decision-making where each mid-level action influences subsequent ones [15, 20, 27]. While LLMs perform reasonably well at high-level planning, they often struggle to translate these plans into precise low-level actions (i.e., motion control) and account for real-world constraints and dynamic environments, which must be optimized together for many embodied tasks [2, 3]. Chen et al. [2] developed a framework that converts few-shot translation of natural language task descriptions into an intermediate task representation through auto-regressive re-prompting for task and motion planning. Moreover, Zhang et al. [26] proposed CoELA, a modular framework for cooperative embodied agents for multi-agent planning, communication, and interaction while integrating cognitive-inspired modules like perception, memory, and execution to address challenges in decentralized control and partial observations. However, these methods did not effectively explore the integration of *continuous* motion planning necessary for handling complex embodied environments, and lack a robust mechanism for *real-time self-correction* based on motion execution feedback. Additionally, the above methods mainly focus on robot manipulation rather than navigation and execution of long-horizon tasks in a dynamic, multi-room environment with multiple agents and objects.

To address these limitations, we present AdaTAMP, an LLM-based adaptive TAMP framework that combines continuous motion planning with a self-feedback loop for real-time correction to enhance the integration of symbolic task planning and motion execution for embodied agents in household environments. We tested our framework on VirtualHome [16], a multi-agent simulator platform for household activities. Agents are represented as humanoid avatars capable of interacting with their surroundings using structured, high-level instructions. More details about the simulator is discussed in the appendix.

Our contribution can be summarized as follows:

1. **Comprehensive AdaTAMP Framework:** We present the AdaTAMP framework, which integrates symbolic task sequences with continuous motion planning, addressing both navigation and task execution for embodied agents in a multi-agent setting. AdaTAMP allows for any valid natural language instruction and develops executable task plans, making it generalizable and easily deployable.
2. **Real-time Adaptability through Self-Feedback:** We incorporate a self-feedback loop that allows for real-time corrections based on motion feedback to adapt to failures and dynamic environment changes.
3. **Extensive Evaluation Metrics in Household Scenarios:** We perform a comprehensive analysis of the framework's effectiveness in navigating and handling dynamic scenarios in household environments, demonstrating its scalability and practicality for multi-agent and human-agent interactions across different metrics.

## 2 Method

**Problem Description.** As shown in Figure 1, we aim to convert natural language task instructions into an executable motion plan for multiple embodied agents operating collaboratively. This motion plan of each agent should adhere to both the spatial and temporal constraints from the natural language instructions and physical constraints of the environment. The environment state $S$ is encoded as set of named objects and their attributes (e.g., position, orientation, and possible actions) as context. A self-feedback loop is used to refine each agent's motion plan in real-time whenever an action fails based on environmental feedback. The updated motion plan at the next step $t+1$ can be expressed as $\tau_{t+1} = \tau_t + \Delta(\tau_t, f_t)$, where $\Delta(\tau_t, f_t)$ represents the correction based on feedback $f_t$. The objective (i.e. a goal function $g : S \to \{0, 1\}$ that checks whether the task goal is achieved at state $s$) is to succesfully execute a sequence of feasible trajectories $T$ and actions $A$ that transitions the agent from the initial environment state to the goal state while ensuring motion feasibility (e.g., maximum velocity and time limits).

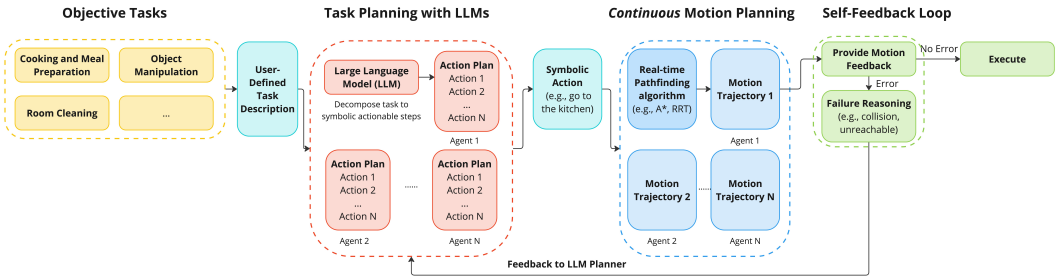


Figure 1: AdaTAMP framework for continuous task and motion planning. It consists of three steps: high-level task planning with LLMs for multi-agents, motion planning for low-level actions, and self-feedback loop for failure correction.

**Multi-agent Task Planning with LLM.** We use GPT-4 [1] to perform high-level task planning for multiple agents. Given natural language task instructions, the pre-trained LLM generates a sequence of symbolic actions, which correspond to a series of steps required to complete the task (e.g., clean the room). The LLM uses common-sense and sequential reasoning to decompose the task into actionable steps [9]. To enable the LLM to understand the environment information symbolically, we leverage grounding information from the simulated environment. This includes translating the graph representations of nodes (i.e., objects) and edges (i.e., relationships between objects) into natural

language descriptions that encapsulate object properties, locations, and spatial relationships (e.g., mug is *on* the kitchen table). For implementation details, please see the appendix.

**Continuous Motion Planning.** We integrate the A* motion planning algorithm [7, 13] into the navigation mesh (NavMesh) structure supported by the simulator. The NavMesh provides a representation of the traversable areas within the environment, which is crucial for pathfinding and collision avoidance. A* is applied on this NavMesh representation to move the agent from its current position to final destination, which is determined by the symbolic actions derived from task plans. A* generates a collision-free trajectory that is also the optimal path since it takes shortest distance to reach from the current location to the target location. Our motion planner dynamically re-evaluates and adjusts paths based on real-time environmental feedback to address inter-agent conflicts, such as agents converging on the same location or obstructing each other's paths, thereby maintaining smooth coordination and execution. For a multi-agent setting, this allows different agents to execute their tasks simultaneously.

**Synthesizing Environment Feedback.** We incorporate a closed-loop feedback mechanism that supports multi-agent coordination. If a motion plan fails due to collisions, conflicts with other agents, or unreachability, the system provides semantically meaningful agent-specific feedback as well as global task-level updates. The feedback loop allows agents to refine their individual plans while maintaining synchronization with the team's overall goal. This approach ensures that agents adapt their behaviors dynamically and maintain collaborative task progress without requiring a complete re-plan of the entire task.

## 3 Experiments

**Experiment Setup.** We evaluated agent performance through 20 tasks across three categories of household tasks: easy (n=6), medium (n=7), and hard (n=7). We chose a subset of tasks provided in the VirtualHome documentation, which includes ground truth task plans for each task with a single agent. Easy tasks involved basic pick-or-place interactions, such as "pick up an apple." Medium tasks required navigation and interaction, such as "place the wine on the coffee table." Hard tasks presented long-horizon challenges involving multiple sequential steps, such as "get the water glass from dining table and plate from kitchen and put them on the coffee table." More information is provided in the appendix. We assessed agent performance using single-agent, two-agent, and multi-agent scenarios. Due to the complexity of the tasks, the two-agent scenario was evaluated exclusively on medium and hard tasks and the multi-agent scenario was evaluated exclusively on hard tasks. For multi-agent tasks, 4 agents were initialized and the LLM determined how many agents to employ to complete the task.

The agents were initialized with varying start positions in a setting containing distinct regions with specific objects, locations, and properties (e.g., type, function, position). For each task, the LLM was prompted with a task description (e.g., "pick up a water glass" or "set up the table"). We compared our framework, which incorporates motion planning and feedback mechanisms to dynamically adjust paths and recover from execution failures against a baseline planner that generates a one-time prompt-based plan without the ability to adapt to environmental changes. More information on the baseline can be found in Appendix A.3.

**Evaluation Metrics.** We evaluated planning performance using several key metrics. Final Success Rate if execution completed the overall task and goal condition. Subgoal Success Rate measured the percentage of intermediate objectives successfully completed during the task. Action Count Efficiency (ACE) evaluated how closely the number of actions taken matched the minimum number required for execution efficiency (obtained from ground truth task plan). In the two-agent and multi-agent setting, we measure ACE relative to the single-agent setting that has ground-truth task plans. Here, 00% would mean same performance as single-agent. Levenshtein Distance [25] quantifies the similarity between the generated action sequences and the ground-truth plan by calculating the minimum number of edit operations needed to transform one sequence into the other (e.g., insertion, deletion, substitution). For the multi-agent setting, we look at further metrics. Completion Time measured the percentage decrease in time taken by all agents to complete the final task relative to a single agent, with lower times indicating more efficient execution. The Coordination Score quantified the degree of parallelization and collaboration between agents by measuring percentage of overlapping steps for task execution.

**Results.** In the single-agent scenario, AdaTAMP demonstrated superior performance compared to the baseline across varying task difficulties, as can be seen in Figure 2. This emphasizes the critical role of our feedback loop and TAMP strategy for better task and motion planning. AdaTAMP exhibited better adaptability compared to baseline, as can be seen from its lower Levenshtein Distance scores. These results underscored the framework's capability to execute sequential tasks with greater precision and adaptability in dynamic environments for single agent.

| Task Level | | Final Success Rate | Subgoal Success Rate | Action Count Efficiency | Completion Time | Levenshtein Distance |
|---|---|---|---|---|---|---|
| **Easy** | AdaTAMP | **100%** | **91.7%** | 89% | -20.4% | **0.67** |
| | baseline | 66.7% | 83.3% | 94.5% | **-15.2%** | 0.83 |
| **Medium** | AdaTAMP | **71.4%** | **70.3%** | 84.3% | -35.3% | **4.7** |
| | baseline | 57.1% | 64.7% | 85% | **-40.2%** | 5.1 |
| **Hard** | AdaTAMP | **42.8%** | **55.6%** | **78.1%** | **-49.2%** | **4.7** |
| | baseline | 14.3% | 52% | 75.6% | -45.2% | 4.9 |
| **Overall** | AdaTAMP | **70%** | **72.5%** | 83.6% | -33% | **3.5** |
| | baseline | 45% | 65.5% | **84.5%** | **-45%** | 3.8 |

Figure 2: Results comparing baseline planning methods and AdaTAMP in single-agent scenario.

| Task Level | | Final Success Rate | Subgoal Success Rate | Action Count Efficiency | Completion Time | Coordination (parallelization) |
|---|---|---|---|---|---|---|
| **Medium** | AdaTAMP | 57.1% | 75.3% | **87.4%** | 24% | 75% |
| | baseline | 57.1% | 75.3% | 86.4% | 24.7% | 75% |
| **Hard** | AdaTAMP | **85.7%** | 88.6% | **117.57%** | 39.1% | **69.1%** |
| | baseline | 71.4% | 93% | 110.7% | 42.4% | 65.6% |
| **Overall** | AdaTAMP | **76.9%** | 88.2% | **107.3%** | 33.9% | **0.776** |
| | baseline | 64.3% | **90.6%** | 106.2% | **36.1%** | 0.756 |

Figure 3: Results comparing baseline planning methods and AdaTAMP in two-agent scenario.

| Task Level | | Final Success Rate | Subgoal Success Rate | Action Count Efficiency | Completion Time | Coordination (parallelization) | Average Number of Agents |
|---|---|---|---|---|---|---|---|
| **Hard** | AdaTAMP | **100%** | **100%** | **140.7%** | 52.8% | **66.7%** | 2.8 |
| | baseline | 85.7% | 90.8% | 136.7% | 44.2% | 65.2% | 2.8 |

Figure 4: Results comparing baseline planning methods and AdaTAMP in multi-agent scenario.

In the multi-agent scenario, AdaTAMP showed notable performance improvements over the baseline. As no ground truth exists for direct comparison, Levenshtein Distance could not be evaluated in this context. Action count efficiency was generally comparable between AdaTAMP and the baseline; however, AdaTAMP outperformed the baseline specifically in hard tasks and demonstrated better performance in multi-agent settings. This improvement aligns with expectations, as the integration of environmental feedback enables AdaTAMP to select more optimal actions, a capability that becomes particularly critical for handling the complexity of hard tasks.

AdaTAMP most often failed due to inadequate understanding of the environment from the LLM. For example, given a task "pick up the cup," the LLM may give a subtask to pick up the water cup from a place where the cup doesn't exist. Another place of failure, albeit rare, is due to incorrect sequence planning (e.g., taking out fridge item before opening fridge).

When transitioning from single-agent to two-agent scenarios, AdaTAMP maintained consistent performance for medium tasks. However, the two-agent and multi-agent setting saw a notable improvement in final success rates (85.7%) compared to the single-agent scenario (42.8%) in hard tasks, demonstrating its ability to effectively handle long-horizon planning and inter-agent dependencies. Metrics of coordiation and final success rate show that similar to humans, splitting and executing tasks to solve a common goal becomes more efficient for TAMP with multiple agents.

## 4 Conclusion

In this paper, we create a comprehensive framework for task and motion planning with environmental feedback to improve performance. Additionally, we expand and test AdaTAMP in the multi-agent setting, where we observe a significant boost in performance for hard-level tasks that require long-horizon planning, multi-agent collaboration, and navigation.

**Limitations and Future Work.** In the multi-agent setting, multiple valid task plans may exist for a given instruction, and our framework does not guarantee selecting the optimal one. A future direction could involve training a Reinforcement Learning algorithm on action sequences from task plans to determine the optimal sequence for multi-agent collaboration, though this may reduce the generalizability and real-time adaptability of AdaTAMP. Additionally, since our experiments were conducted solely in simulation, it would be valuable to evaluate the AdaTAMP framework on real-world robots for both robot-robot and human-robot collaboration scenarios.

## References


[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6695–6702. IEEE, 2024.

[3] Jae-Woo Choi, Youngwoo Yoon, Hyobin Ong, Jaehong Kim, and Minsu Jang. Lota-bench: Benchmarking language-oriented task planners for embodied agents. *arXiv preprint arXiv:2402.08178*, 2024.

[4] E Allen Emerson. Temporal and modal logic. In *Formal Models and Semantics*, pages 995–1072. Elsevier, 1990.

[5] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[6] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[7] Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. Reach for a*: Efficient point-to-point shortest path algorithms. In *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 129–143. SIAM, 2006.

[8] Valentin N Hartmann, Andreas Orthey, Danny Driess, Ozgur S Oguz, and Marc Toussaint. Long-horizon multi-robot rearrangement planning for construction assembly. *IEEE Transactions on Robotics*, 39(1):239–252, 2022.

[9] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.

[10] Ziyuan Jiao, Yida Niu, Zeyu Zhang, Song-Chun Zhu, Yixin Zhu, and Hangxin Liu. Sequential manipulation planning on scene graph. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8203–8210. IEEE, 2022.

[11] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

[13] Ehsan Latif. 3p-llm: Probabilistic path planning using large language model for autonomous robot navigation. *arXiv preprint arXiv:2403.18778*, 2024.

[14] Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. Delta: Decomposed efficient long-term robot task planning using large language models. *arXiv preprint arXiv:2404.03275*, 2024.

[15] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

[16] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8494–8502, 2018.

[17] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[18] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[19] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009, 2023.

[20] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[21] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[22] Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. Llmˆ 3: Large language model-based task and motion planning with motion failure reasoning. *arXiv preprint arXiv:2403.11552*, 2024.

[23] Yue Wu, Yewen Fan, Paul Pu Liang, Amos Azaria, Yuanzhi Li, and Tom M Mitchell. Read and reap the rewards: Learning to play atari with the help of instruction manuals. *Advances in Neural Information Processing Systems*, 36, 2024.

[24] Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied task planning with large language models. *arXiv preprint arXiv:2307.01848*, 2023.

[25] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.

[26] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485*, 2023.

[27] Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2081–2088. IEEE, 2024.

# A Appendix

## A.1 VirtualHome Simulation

VirtualHome is a multi-agent platform to simulate activities in a household. VirtualHome also includes a Knowledge Base, providing instructions to perform a large set of activities. We use tasks from this knowledge base for our experiments. The simulator supports a comprehensive range of interaction actions, including “pick up,” “open,” “close,” “switch on/off,” and “put down.” Object interaction skills can only be executed when the interacting object is within close proximity to the

agent. For instance, the action “pick up the plate” will only succeed if the distance between the plate and the agent is less than a predefined threshold. We connect terms such as “find” and “go to” with the motion planner to allow users to navigate and explore their environment.

VirtualHome allows you to quickly generate humanoid avatars and objects. We can define where to place these objects and their relationships through a graph representation that VirtualHome maintains to keep track of all items in the simulation. This graph representation consists of nodes describing the items in the entire house and edges decribing the relationship (i.e, position and orientation) between two nodes. The simulator is implemented in Unity and can be viewed through their python API. We are providing some images describing the simulation.

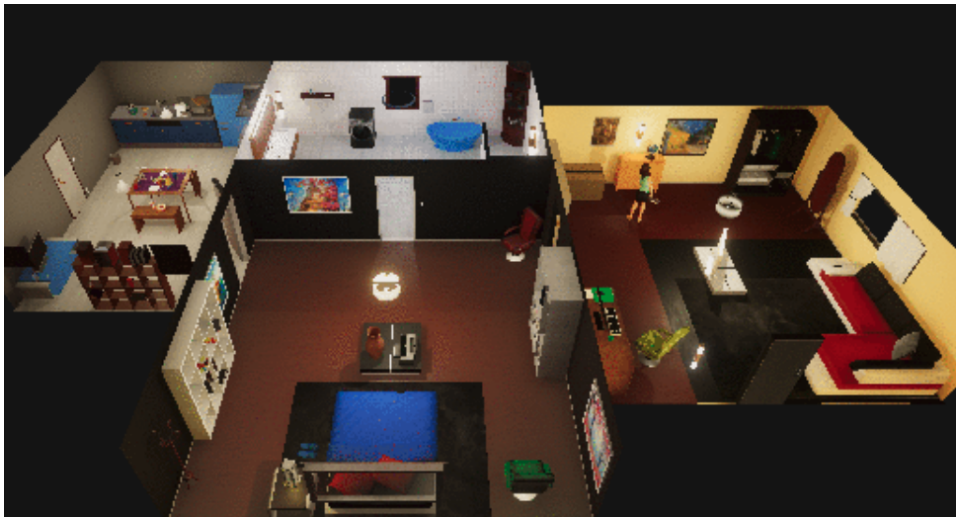

Figure 5: A complete view of a home. We can see an agent interacting with objects in this environment.

Figure 6: An agent moving around a home. We show different camera angles of a female agent moving in simulation.

Our LLM generates task plans in the form of executable action programs that we can interface with the VirtualHome Simulator and motion planner to execute subtasks. The LLM output can be seen in Appendix A.2.

### A.2 Task examples

Below are examples of tasks and task plans generated by AdaTAMP. Here, char<i> is an agent, where i=0,1,2,3. The provided LLM output is the end output after passing the LLM response through structured outputs and then the grounding dictionary as described in Appendix A.4.

**Easy task:** "Find a wine glass"

Task Plan: 1. Char1 walks to the coffee table. 2. Char1 looks at the wineglass 3. Char1 looks at another wineglass

LLM Output: ['character': 'char1', 'action': 'Walk', 'object': 'coffeetable', 'id': 113, 'character': 'char1', 'action': 'Look', 'object': 'wineglass', 'id': 199, 'character': 'char1', 'action': 'Look', 'object': 'wineglass', 'id': 200]

**Medium task:** "Place the wine on the coffee table"

Task Plan: 1. Char1 walks to the wineglass. 2. Char1 picks up the wineglass. 3. Char1 walks to the living room. 4. Char1 walks to the coffee table. 5. Char1 places the wineglass on the coffee table.

LLM Output: ['character': 'char0', 'action': 'Walk', 'object': 'wineglass', 'id': 199, 'character': 'char0', 'action': 'Grab', 'object': 'wineglass', 'id': 199, 'character': 'char0', 'action': 'Walk', 'room': 'livingroom', 'id': 12, 'character': 'char0', 'action': 'Walk', 'object': 'coffeetable', 'id': 113, 'character': 'char0', 'action': 'Place', 'object': 'wineglass', 'id': 199]

**Hard Task:** "Put 1 cupcake, 1 juice, 1 pound cake, and 1 pudding on the kitchen table"

Task Plan (with 4 agents): 1. Char1 walks to the cupcake. 2. Char2 walks to the juice. 3. Char3 walks to the pound cake. 4. Char4 walks to the pudding. 5. Char1 grabs the cupcake. 6. Char2 grabs the juice. 7. Char3 grabs the pound cake. 8. Char4 grabs the pudding. 9. Char1 walks to the kitchen table. 10. Char2 walks to the kitchen table. 11. Char3 walks to the kitchen table. 12. Char4 walks to the kitchen table. 13. Char1 places the cupcake on the kitchen table. 14. Char2 places the juice on the kitchen table. 15. Char3 places the pound cake on the kitchen table. 16. Char4 places the pudding on the kitchen table.

LLM Output: ['character': 'char0', 'action': 'Walk', 'object': 'cupcake', 'id': 197, 'character': 'char1', 'action': 'Walk', 'object': 'juice', 'id': 199, 'character': 'char2', 'action': 'Walk', 'object': 'pound cake', 'id': 198, 'character': 'char3', 'action': 'Walk', 'object': 'pudding', 'id': 200, 'character': 'char0', 'action': 'Grab', 'object': 'cupcake', 'id': 197, 'character': 'char1', 'action': 'Grab', 'object': 'juice', 'id': 199, 'character': 'char2', 'action': 'Grab', 'object': 'pound cake', 'id': 198, 'character': 'char3', 'action': 'Grab', 'object': 'pudding', 'id': 200, 'character': 'char0', 'action': 'Walk', 'object': 'kitchen table', 'id': 113, 'character': 'char1', 'action': 'Walk', 'object': 'kitchen table', 'id': 113, 'character': 'char2', 'action': 'Walk', 'object': 'kitchen table', 'id': 113, 'character': 'char3', 'action': 'Walk', 'object': 'kitchen table', 'id': 113, 'character': 'char0', 'action': 'Place', 'object': 'cupcake', 'id': 197, 'character': 'char1', 'action': 'Place', 'object': 'juice', 'id': 199, 'character': 'char2', 'action': 'Place', 'object': 'pound cake', 'id': 198, 'character': 'char3', 'action': 'Place', 'object': 'pudding', 'id': 200]

### A.3 Baseline

Our baseline is a modified version from the algorithm developed in the LotaBENCH paper[3]. The LLM prompting used in LotaBENCH, however, is more constrained in nature and does not use the grounding dictionary and structured outputs. Thus, LotaBENCH works for only a fixed task and environment setting. Our baseline is more generalizable and works for any natural language instruction. Additionally, we did not find any previous work to incorporate multi-agent collaboration with the TAMP framework for VirtualHome. Hence, we decided to set up our own baseline which consisted of the same LLM prompting and motion planner used in AdaTAMP without any environment feedback.

### A.4 Deployment

We provided as input a user-defined natural language instruction to complete an overarching task to the LLM (here, we use GPT-4o). The LLM takes this input and outputs a sequence of subtasks to reach the final goal state given the environment graph and number of total agents in the environment. We use structured outputs, an OpenAI feature, to enforce GPT to generate subtasks in the form of programs (similar to that in Appendix A.2). We noticed that providing in-context examples significantly boosted the LLM performance in generating appropriate subtasks.

We construct a structured dictionary to bridge the gap between natural language task instructions and the simulated environment. This dictionary translates objects and agents described in a subtask to symbolic representations of the environment, ensuring that the instructions are grounded under the

physical and spatial constraints of our environment. Figure 7 shows some of these implementation details.

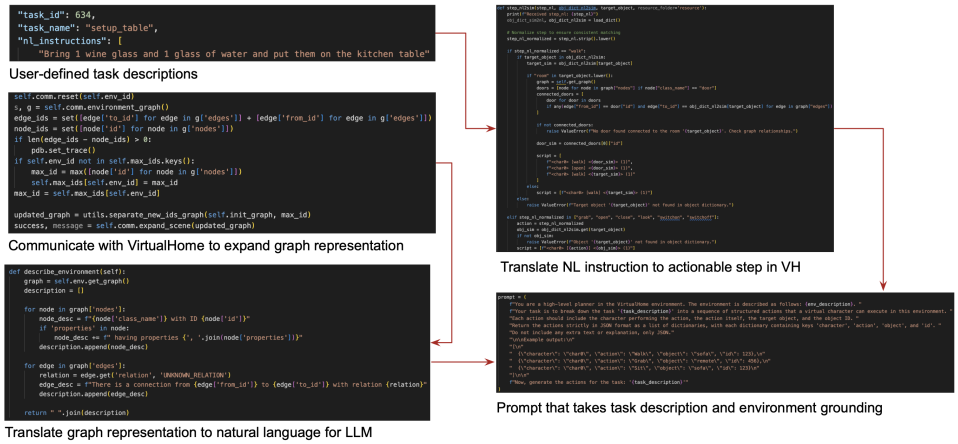


Figure 7: Deployment details for Task planning