

# Obtaining Hierarchy from Human Instructions: an LLMs-based Approach

Xusheng Luo<sup>1,†</sup>, Shaojun Xu<sup>2,†,‡</sup>, and Changliu Liu<sup>1</sup>

**Abstract**—Long-horizon planning in robotics is often hindered by challenges such as uncertainty accumulation, computational complexity, delayed rewards and incomplete information. This work proposes an innovative approach to exploit the inherent task hierarchy from human instructions to aid in planning. Utilizing Large Language Models (LLMs), we propose a two-step approach to translate multi-sentence human instructions into a structured language, Hierarchical Linear Temporal Logic, which serves as an intermediary for planning. Initially, LLMs transform the human instructions into a Hierarchical Task Network (HTN)-like representation, capturing the logical and temporal relations among tasks. Following this, a domain-specific fine-tuning of LLMs or human expert can translate sub-tasks of each task into flat LTL formulas, aggregating them to form hierarchical LTL specifications. These specifications are then leveraged for planning. Our framework not only bridges the gap between human instructions and algorithmic planning but also showcases the potential of LLMs in harnessing human-like hierarchical reasoning to automate complex task planning. Through simulated experiments, we exhibit the efficacy of our approach in generating executable plans from human instructions, fostering a more intuitive and user-friendly robotic assistance in everyday scenarios.

## I. INTRODUCTION

The challenge of long-horizon planning arises from factors such as uncertainty accumulation, computational complexity, delayed rewards and incomplete information. A viable strategy to address computational complexity and delayed rewards is by leveraging the hierarchy inherent in the task structure. Hierarchical models have demonstrated a notable edge over flat models in facets like interpretability and efficiency [1], [2]. Take for instance a robot assigned with the chore of loading a dishwasher. A user might opt to position cups and saucers on the top rack, reserving the bottom rack for plates, utensils, and mugs. Additionally, the preference might extend to loading the bottom rack prior to the top one. This exemplifies how commonplace household chores are imbued with hierarchical structures. However attractive, how to obtain hierarchy information still remains an open problem.

One pathway is to deduce hierarchy through the observation of task execution [3], which alleviates human effort yet poses a challenge due to the extensive scope of solution candidates and a high requirement for abstraction reasoning. Conversely, the

other route, which entails acquiring hierarchy directly from human input, appears straightforward. Humans excel at hierarchical reasoning and are used to articulate hierarchically through language effortlessly [4]. For example, one can effortlessly voice their preference for loading the lower rack prior to the upper one. Nonetheless, the hierarchical insights from humans cannot be readily integrated by algorithms without meticulous engineering. This gap between human preferences and algorithmic formulation impedes the application of hierarchy-based planning algorithms. However, with the advent of LLMs [5], we foresee a bright prospect of narrowing this gap.

LLMs, being trained on extensive textual corpus, exhibit common sense reasoning abilities, thereby efficiently managing everyday task specifications articulated in human languages. Our key observation is that hierarchy can be straightforwardly obtained from human input with the help of a LLM, as inferring hierarchy directly from observations may be challenging. The decision-making process of humans is hierarchical [4], which is reflected in verbal languages. Thus, the focal point here to transmit this hierarchy to downstream planners effectively. In light of the above, our work proposes harnessing LLMs as translators to extract task hierarchy from human languages. What sets us apart from existing work on language-conditioned planning is that while existing work primarily focuses on planning subject to single-sentence instructions, our objective is to handle multiple-sentences long instructions with relatively clear hierarchical structure.

The structured language that is translated from human instructions is Hierarchical LTL, a variant of formal languages introduced in our prior work [6]. Unlike the unstructured and ambiguous nature of natural languages, formal languages are structured and yield a unique interpretation, of which LTL has been applied to robotics [7]–[9]. Hierarchical LTL is succinct and interpretable compared to its flat counterpart, aligning well with hierarchical human instructions. While temporal logic is expressive in articulating complex tasks, the learning curve associated with it is steep; even expert may require time to draft suitable formulas. For those not well-versed in the field, expressing tasks through natural language is considerably more intuitive. We perceive LLMs as an efficacious tool to unlock the expressive prowess of temporal logic. The translation of hierarchical human instructions into hierarchical LTL proves to be more straightforward and dependable compared to translating into a cumbersome flat formula, a challenge not solved by existing works [10]–[12].

A naive approach to converting human instructions into hierarchical LTL is by employing an end-to-end methodology. However, this technique tends to perform poorly as LLMs are

<sup>1</sup>Xusheng Luo and Changliu Liu are with Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA (e-mail: {xushengl, cliu6}@andrew.cmu.edu)

<sup>2</sup>Shaojun Xu is with College of Control Science and Engineering, Zhejiang University, Hangzhou, 310058, China (e-mail: sjxu@zju.edu.cn).

<sup>†</sup>Equal contribution.

<sup>‡</sup>Work done during internship at Carnegie Mellon University.

still not good at logical reasoning [13], which is crucial for crafting logical formulas. In this work, we propose a two-step approach to convert human instructions into hierarchical LTL. Initially, upon receiving a human instruction, we prompt LLMs to yield a task representation akin to Hierarchical Task Network (HTN) [14]. This representation deviates from HTN as it involves complex logical and temporal relations amongst tasks at varying levels of abstraction. The integration of an HTN-like task specification as an intermediary phase aims to harness the common sense reasoning capabilities of LLMs fully. Subsequently, in the second phase, sub-tasks of each task can be translated into a single flat LTL via fine-tuned LLMs or human expert. Through iterative processing of all sub-tasks of every task in the intermediary phase, we are able to construct a hierarchical LTL. With our methodology, human instructions abundant in hierarchical information are ready for use by hierarchical LTL planners.

**Contributions:** Our contributions can be summarized as follows:

- 1) We proposed a framework for deriving task hierarchy from human instructions, utilizing hierarchical LTL as an intermediate step;
- 2) We introduced an algorithm for converting human languages into hierarchical LTL, thereby facilitating the leverage of human’s hierarchical knowledge in subsequent planning solvers;
- 3) We showcased simulations that employ human instructions to generate executable plans for navigation tasks.

## II. RELATED WORK

### A. LLMs-based Robotic Planning

In traditional robotics workflows, an expert is typically essential for optimizing robot performance in specific scenarios. However, LLMs allows us to interpret high-level language instructions and engage in intricate long-horizon task planning.

Efforts to integrate LLMs into robotic planning can largely be categorized into two distinct approaches: employing LLMs as planners and using them as translators. As planner, LLMs are always adopted to divided instructions into sub-tasks sequences with specialized prompting functions [15], [16], or mapped to specific control policy affordance functions [17]. Additionally, they can be part of recursive policy code generation methods [18], [19]. While the absence of real-time environmental feedback from the robots, make it challenging to ascertain the feasibility of the generated sub-task sequences. Vision-language architectures like Voxposer [20] and AlphaBlock [21] have been introduced to overcome this limitation by integrating sensory feedback in multi-step reasoning. But they all encounter difficulties in generating long-term planning sequences that account for temporal events.

Other Studies have suggested that LLMs are more aptly suited for translation tasks than for planning [22]. Utilizing LLMs as translators bridge the gap between natural language and structured planning languages. Implementations including translating natural language into Planning Domain Definition Language (PDDL) to derive optimal solutions through classical planners [23], LLM-PDDL based heuristic search planners [24]

and Signal Temporal Logic trajectory planners [25] all decompose robotic planning into formal verification procedures, thereby enhancing the planning process’s transparency and predictability, and facilitating easier adjustments.

### B. Natural Language to Temporal Logic

Numerous variations of temporal logic exist, generally sharing similarities in syntactic structure. Initial efforts in translating natural language to temporal logics mainly adopted grammar-based methodologies, capable of processing structured natural language [26]. Traditional techniques usually involve a three-stage process: 1) preliminary analysis of English input to obtain syntactical features, 2) the establishment of patterns or rules specific to temporal logic through classification, and 3) the use of an attribute grammar-based parser to convert into the desired logical format. Such methods are limited to handling only narrowly defined natural language input.

The application of neural networks for translating into temporal logics has emerged more recently. Examples include training models from scratch specifically for Signal Temporal Logic (STL) [27], fine-tuning existing language models [11], [28], and implementing one-shot approaches using GPT-3 [29], [30]. While LLMs-based methods outperform traditional ones, current research mainly focuses on “flat” forms of temporal logic, which raises questions about the applicability of these methods to more intricate, hierarchical structures. Notably, when humans express tasks over extended time horizons in natural language, pronouns are frequently employed. For example, someone might say, “First load the lower rack. After that, finish loading the upper rack”. These pronouns serve as both a blessing and a curse in the translation process. On the positive side, they offer clear cues for breaking down the task into a hierarchical structure. On the flip side, they introduce the issue of co-reference, which requires careful resolution. Existing research has not adequately addressed this challenge, resulting in a lack of targeted training data.

## III. PRELIMINARY

### A. Linear Temporal Logic

Linear Temporal Logic (LTL) is composed of basic statements, referred to as atomic propositions  $\mathcal{AP}$ , along with boolean operators such as conjunction ( $\wedge$ ) and negation ( $\neg$ ), as well as temporal operators like next ( $\bigcirc$ ) and until ( $\mathcal{U}$ ) [31]. LTL formulas, constructed using these elements over the set of atomic propositions  $\mathcal{AP}$ , follow the syntax outlined below:

$$\phi := \top \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 \mathcal{U} \phi_2, \quad (1)$$

where  $\top$  stands for an unconditionally true statement, and  $\pi$  refers to a boolean valued atomic proposition. Other temporal operators can be derived from  $\mathcal{U}$ , such as  $\diamond\phi$  that implies  $\phi$  will ultimately be true at a future time, and  $\square\phi$  that denotes  $\phi$  will perpetually be true from the current point onwards.

*Example 1: (Dishwasher Loading Problem (DLP))* Place items into the dishwasher. Robot should put plates, mugs and utensils into the lower rack in any order. After putting items to the lower rack, then put things into upper rack, first put saucers, and then put cups.

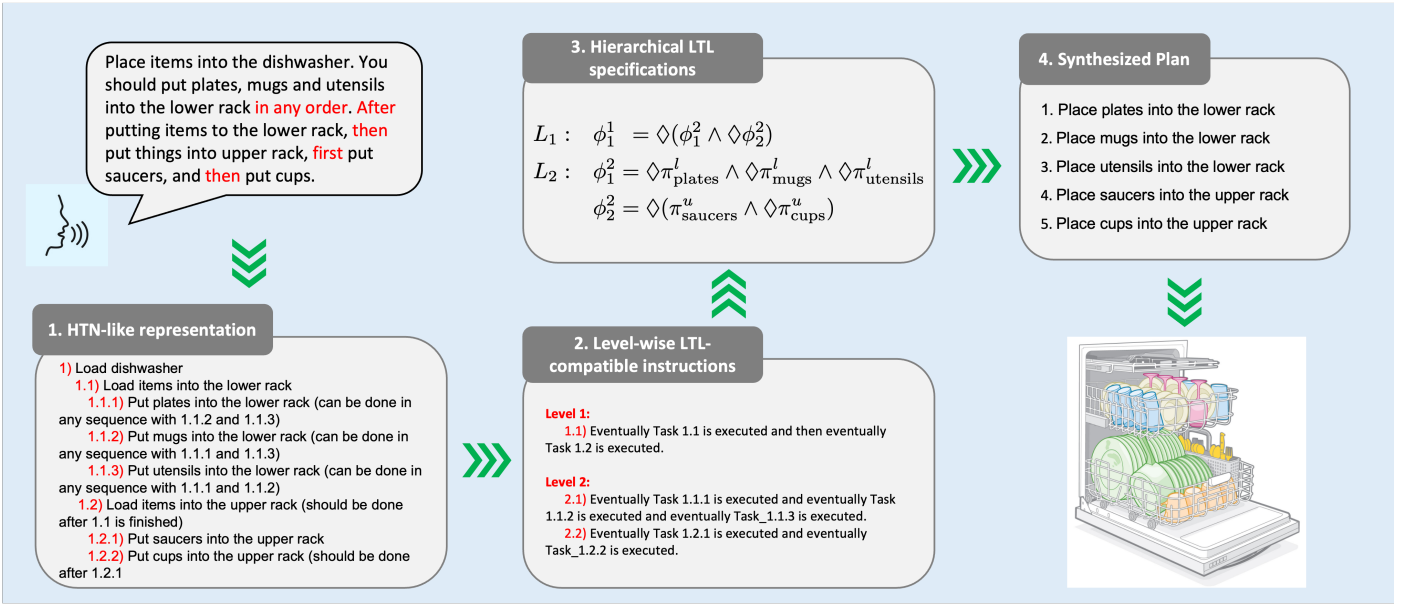


Fig. 1: Overview of the framework. Use the dishwasher loading problem as a case study.

## B. Hierarchical LTL

We briefly introduce the hierarchical LTL that enables the specification of more intricate tasks than the flat structure and we refer readers to work [6] for more details.

*Definition 3.1: (Hierarchical LTL [6])* A hierarchical linear temporal logic specification includes  $K$  levels such that each proposition at level  $k \in \{1, \dots, K - 1\}$  is constructed from specifications at the next lower level  $k + 1$ .

*Example 1: continued (DLP)* The primary task constraint is that the robot should first focus on loading the lower rack and subsequently load the upper rack:

$$\begin{aligned}
 L_1 : \quad \phi_1^1 &= \diamond(\phi_1^2 \wedge \diamond\phi_2^2) \\
 L_2 : \quad \phi_1^2 &= \diamond\pi_{\text{plates}}^l \wedge \diamond\pi_{\text{mugs}}^l \wedge \diamond\pi_{\text{utensils}}^l \\
 \phi_2^2 &= \diamond(\pi_{\text{saucers}}^u \wedge \diamond\pi_{\text{cups}}^u),
 \end{aligned} \tag{2}$$

where  $\phi_i^k$  denotes the  $i$ -th specification at level  $k$ ,  $\phi_1^2$  and  $\phi_2^2$  are *composite propositions*, representing the events of loading items onto the lower and upper racks,  $\pi_i^j$  are atomic propositions, denoting the act of placing a specific type of dishware on either the lower or upper rack. The formula  $\diamond(\phi_1^2 \wedge \diamond\phi_2^2)$  specifies that  $\phi_1^2$  should be fulfilled before moving on to  $\phi_2^2$ .  $L_1$  and  $L_2$  denote different levels of the task hierarchy. Note that the lowest level  $L_2$  only includes atomic propositions. A flat LTL specification of this task is

$$\begin{aligned}
 \phi &= \diamond\pi_{\text{plates}}^l \wedge \diamond\pi_{\text{mugs}}^l \wedge \diamond\pi_{\text{utensils}}^l \wedge \diamond(\pi_{\text{saucers}}^u \wedge \diamond\pi_{\text{cups}}^u) \\
 &\wedge \neg\pi_{\text{saucers}}^u \mathcal{U} \pi_{\text{plates}}^l \wedge \neg\pi_{\text{saucers}}^u \mathcal{U} \pi_{\text{mugs}}^l \wedge \neg\pi_{\text{saucers}}^u \mathcal{U} \pi_{\text{utensils}}^l.
 \end{aligned}$$

## IV. NATURAL LANGUAGE TO HIERARCHICAL LTL

LLMs excel in common sense reasoning yet behavior poorly in logical reasoning [13], [32]. Hence, we propose a two-step approach. Initially, given human instructions, we exploit the common sense reasoning capability of LLMs to transform

human instructions into an HTN-like representation; following which, we generate a flat LTL for each task. The overview of the proposed framework is displayed in Fig. 1.

### A. Conversion from NL to HTN-like Representation

By using in-context learning, we prompt LLMs to generate an HTN-like representation, shown as step 1 in Fig. 1. The representation should abide by the following format:

1) At the bottom level, every task should include only actions or events usually represented as atomic propositions. It's noteworthy that we do not necessitate the tasks at this level to be indecomposable, such as sub-task 1.2.1) in Fig. 3 of Appendix A, resulting in a flexible hierarchical task structure. Hence, the number of levels and tasks per level can vary across different LLMs, a scenario anticipated as the abstraction granularity varies based on distinct understanding of the task.

2) The parent task ought to summarize the objectives its child tasks aim to accomplish. It's imperative to note that the preciseness of summarization is not crucial as parent tasks solely function as abstractions, which is more pertinent to interpretation than to the planning process itself.

3) At each level, logic and temporal constraints, if any, are encapsulated in parentheses at the end of certain tasks, describing the relations with other tasks on the same level. We noticed a drop in performance when amalgamating all such information alongside their parent task, possibly stemming from the challenge of summarizing the relations of child tasks all at once.

After generating the HTN-like representation, we further prompt LLMs to rephrase it into a JSON format, which helps format the next step of generating task-wise flat LTL specifications. We observed that LLMs fall short in performance when tasked to generate the JSON format in a singular step without initially prompting it to generate the HTN-like representation. The detailed prompt for generating HTN-like

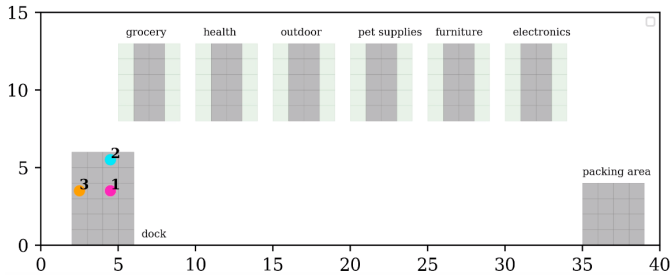


Fig. 2: Topological map of the supermarket.

representation and generating JSON format are shown in Figs. 3 and 4 in Appendix A. In the JSON format, a notable field is “sibling\_nodes\_condition”, which specifies the logical and temporal relations between sub-tasks associated with the same parent task.

### B. Generation of task-wise Flat LTL Specifications

Once the HTN-like representation in JSON format is obtained, we proceed to generate a single flat LTL specification for all sub-tasks of one task, leveraging LLMs-based techniques that convert natural languages into flat LTL; shown as steps 2 and 3 in Fig. 1. The input for these methods is constituted by human instructions. For this objective, we prompt LLMs to concentrate on one task at a time, delineating all of its sub-tasks along with the logical and temporal relations among them in a manner commonly employed for interpreting LTL. To achieve this, we regard each task as a solitary entity without delving into the internals of every task, and discern between safe and co-safe properties whilst synthesizing the instruction. Co-safe property implies that something good will eventually happen, whereas a safe property signifies that something bad will never occur. We discovered that reasoning over safe and co-safe properties first inclines LLMs towards opting for suitable wording in elucidating LTL. This prompt is exhibited in Fig. 5 in Appendix A.

*Example 1: continued (DLP)* The solutions to the dishwasher loading problem can be found in Figs. 6 through 8 and Tab. I in Appendix B. It’s important to note that for a given task description, there is not a single “optimal” LTL specification that perfectly captures the essence of the task. In other words, the translation of natural language into LTL is not unique. Various LTL specifications exist that, to some extent, represent the task, in that some of their solutions fulfill the task description. For example, in the second row of Tab. I, the instruction for tasks 1.1 and 1.2 state, “Always task\_1.1 must precede task\_1.2 and eventually task\_1.1 must be executed.” Here, tasks 1.1 and 1.2 refer to placing dishes in the lower and upper racks, respectively (see Fig. 1). Based solely on the LLM-generated description, the corresponding LTL is  $\diamond T_{1.1} \wedge \square(T_{1.1} \rightarrow \diamond T_{1.2})$ . According to the original task outlined in Example 1, a more suitable version would be  $\diamond(T_{1.1} \wedge \diamond T_{1.2})$ . Nevertheless, the specification  $\diamond T_{1.1} \wedge \square(T_{1.1} \rightarrow \diamond T_{1.2})$  remains valid to some extent, as it ensures that task 1.2 will eventually follow task 1.1.

## V. NUMERICAL RESULTS

In this section, we showcase some preliminary results to illustrate the effectiveness of our proposed framework, which takes natural language instructions as input and outputs a plan. We focus on the order picking problem (OPP) in a supermarket setting, as in [33]. As depicted in Fig. 2, the supermarket is divided into six areas: grocery, health, outdoor, pet supplies, furniture, and electronics. Robots initiate from the docking area, gather items, and then transport them to a designated packing zone. Although we use a household cleanup scenario in the prompts for the LLMs, our test task originates from the navigation domain, thereby highlighting the framework’s versatility. GPT-4 [34] serves as the LLM in our experiments. It’s noteworthy that due to their complexity, none of the tasks here can be addressed by existing methodologies that directly translate instructions into flat LTL specifications [10]–[12].

In our study, we examine three tasks from our previous work [6], but with a modification: we clearly designate which robot is responsible for which task. The dialog sequences for task 2 with GPT-4 are detailed in Figs. 9 to 10 and Tab.II in Appendix C-A. The dialog sequences for task 2 with GPT-4 are detailed in Figs. 11 to 12 and Tab.III in Appendix C-B. Similarly, the dialog sequences for task 3 are presented in Figs. 13 to 14 and Tab.IV in Appendix C-C. While attempting to formulate flat LTL formulas for each task, we tested existing models, but found that none could process the human-like instructions generated by GPT-4. In this work, we manually convert these instructions into flat LTL, strictly adhering to the GPT-4 generated instructions. Future efforts will involve fine-tuning a pre-trained LLM using a dataset that pairs GPT-4 generated instructions with corresponding flat LTL specifications. We employ the planning algorithm from [6] to derive solutions. Videos showcasing the generated plans for all tasks can be accessed at [https://youtu.be/4m6rHAY\\_Jqk](https://youtu.be/4m6rHAY_Jqk).

## VI. CONCLUSIONS AND FUTURE WORK

We proposed using LLMs to translate human instructions into hierarchical LTL, thereby transforming unstructured language into a structured format without losing its hierarchical nature. Our simulation outcomes, involving inputting language and outputting plans, demonstrated that the framework offers an intuitive and user-friendly approach for deploying robots in daily situations. Future work will focus on:

- 1) Gather a dataset comprising GPT-4-generated instructions paired with flat LTL specifications, and fine-tune a pre-trained LLM for the purpose of translation.
- 2) Incorporate both a syntax verifier and a semantic verifier to form a closed-loop framework that continuously interact with LLMs. The syntax verifier ensures compliance with hierarchical LTL structure for the HTN-like representation, while the semantic verifier provides error feedback if planner could not find a solution.
- 3) Conduct ablation study and compare with LLM-based and NL-to-PDDL-based planner.
- 4) Conduct experiments involving robotic manipulation tasks in real-world settings

## REFERENCES

- [1] J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman, “How to grow a mind: Statistics, structure, and abstraction,” *science*, vol. 331, no. 6022, pp. 1279–1285, 2011.
- [2] C. Kemp, A. Perfors, and J. B. Tenenbaum, “Learning overhypotheses with hierarchical bayesian models,” *Developmental science*, vol. 10, no. 3, pp. 307–321, 2007.
- [3] B. Hayes and B. Scassellati, “Autonomously constructing hierarchical task networks for planning and human-robot collaboration,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 5469–5476.
- [4] B. A. Purcell and R. Kiani, “Hierarchical decision processes that operate over distinct timescales underlie choice and changes in strategy,” *Proceedings of the national academy of sciences*, vol. 113, no. 31, pp. E4531–E4540, 2016.
- [5] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, “Challenges and applications of large language models,” *arXiv preprint arXiv:2307.10169*, 2023.
- [6] X. Luo, S. Xu, R. Liu, and C. Liu, “Robotic planning under hierarchical temporal logic specifications,” *arXiv preprint arXiv:2308.10393*, 2023.
- [7] X. Luo, Y. Kantaros, and M. M. Zavlanos, “An abstraction-free method for multirobot temporal logic optimal control synthesis,” *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1487–1507, 2021.
- [8] X. Luo and M. M. Zavlanos, “Temporal logic task allocation in heterogeneous multirobot systems,” *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3602–3621, 2022.
- [9] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, “Robots that use language,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 25–55, 2020.
- [10] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah, “Grounding complex natural language commands for temporal tasks in unseen environments,” in *7th Annual Conference on Robot Learning*, 2023.
- [11] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, “NI2tl: Transforming natural languages to temporal logics using large language models,” *arXiv preprint arXiv:2305.07766*, 2023.
- [12] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, “nl2spec: Interactively translating unstructured natural language to temporal logics with large language models,” *arXiv preprint arXiv:2303.04864*, 2023.
- [13] F. Xu, Q. Lin, J. Han, T. Zhao, J. Liu, and E. Cambria, “Are large language models really good logical reasoners? a comprehensive evaluation from deductive, inductive and abductive views,” *arXiv preprint arXiv:2306.09841*, 2023.
- [14] M. Ghallab, D. Nau, and P. Traverso, *Automated planning and acting*. Cambridge University Press, 2016.
- [15] S. S. Raman, V. Cohen, E. Rosen, I. Idrees, D. Paulius, and S. Tellex, “Planning with large language models via corrective re-prompting,” in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.
- [16] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, “Progprompt: program generation for situated robot task planning using large language models,” *Autonomous Robots*, pp. 1–14, 2023.
- [17] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman et al., “Do as i can, not as i say: Grounding language in robotic affordances,” *arXiv preprint arXiv:2204.01691*, 2022.
- [18] W. Chen, S. Hu, R. Talak, and L. Carlone, “Leveraging large language models for robot 3d scene understanding,” *arXiv preprint arXiv:2209.05629*, 2022.
- [19] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser, “Tidybot: Personalized robot assistance with large language models,” *arXiv preprint arXiv:2305.05658*, 2023.
- [20] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, “Voxposer: Composable 3d value maps for robotic manipulation with language models,” *arXiv preprint arXiv:2307.05973*, 2023.
- [21] C. Jin, W. Tan, J. Yang, B. Liu, R. Song, L. Wang, and J. Fu, “Alphablock: Embodied finetuning for vision-language reasoning in robot manipulation,” *arXiv preprint arXiv:2305.18898*, 2023.
- [22] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh, “Translating natural language to planning goals with large-language models,” *arXiv preprint arXiv:2302.05128*, 2023.
- [23] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, “Llm+ p: Empowering large language models with optimal planning proficiency,” *arXiv preprint arXiv:2304.11477*, 2023.
- [24] T. Silver, V. Hariprasad, R. S. Shuttlesworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling, “Pddl planning with pretrained large language models,” in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.
- [25] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan, “Autotamp: Autoregressive task and motion planning with llms as translators and checkers,” *arXiv preprint arXiv:2306.06531*, 2023.
- [26] S. Konrad and B. H. Cheng, “Real-time specification patterns,” in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 372–381.
- [27] J. He, E. Bartocci, D. Ničković, H. Isakovic, and R. Grosu, “Deepstl: from english requirements to signal temporal logic,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 610–622.
- [28] C. Hahn, F. Schmitt, J. J. Tillman, N. Metzger, J. Siber, and B. Finkbeiner, “Formal specifications from natural language,” *arXiv preprint arXiv:2206.01962*, 2022.
- [29] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah, “Lang2tl: Translating natural language commands to temporal robot task specification,” *arXiv preprint arXiv:2302.11649*, 2023.
- [30] F. Fuggitti and T. Chakraborti, “NI2tl—a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas,” in *AAAI Conference on Artificial Intelligence*, 2023.
- [31] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press Cambridge, 2008.
- [32] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, and A. Miller, “Language models as knowledge bases?” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 2463–2473.
- [33] L. Pansart, N. Catusse, and H. Cambazard, “Exact algorithms for the order picking problem,” *Computers & Operations Research*, vol. 100, pp. 117–127, 2018.
- [34] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg et al., “Sparks of artificial general intelligence: Early experiments with gpt-4,” *arXiv preprint arXiv:2303.12712*, 2023.

APPENDIX A  
PROMPT FOR LLMs

Prompt for generating HTN-like representation

Convert the natural language task description into Hierarchical Task Network (HTN), meanwhile expanding pronouns [e.g., it, them, they] to refer to specific objects.

The example task description is given as follows: “There are blocks, toys, and clothes on the table, you are suggested to put some of them into the red, green, and blue boxes. For blocks, if there are yellow blocks on the table, then first put the biggest block into the red box, and then put the rest of yellow blocks into the red box. After putting blocks to red box, you can start to putting toys into the green box. If their are toy bears on the table, directly put them into the green box. If there are dolls on the table, you need to dress them up before lay them in the green box. You can put clothes into the blue box in any time, but you need to place all skirts and pants into the blue box before putting in socks. Eventually, you should pile the red, green, and blue boxes in sequence.”

The HTN-like representation is as follows:

- 1) Sort things
  - 1.1) Place blocks, toys, and clothes into red, green, and blue boxes.
    - 1.1.1) Place blocks into red boxes
      - 1.1.1.1) If there are yellow blocks on the table, then first put the biggest block into the red box.
      - 1.1.1.2) Put the yellow block into the yellow box (after 1.1.1.1 finished).
    - 1.1.2) Move toys into the green box (after 1.1.1 finished).
      - 1.1.2.1) If there are toy bears on the table, directly put them into the green box.
      - 1.1.2.2) If there are dolls on the table, you need to dress them up before laying them in the green box.
    - 1.1.3) Move clothes into the blue box.
      - 1.1.3.1) Place all skirts into the blue box.
      - 1.1.3.2) Place all pants into the blue box.
      - 1.1.3.3) Putting in socks into the blue box (after 1.1.3.1 and 1.1.3.2 finished).
  - 1.2) Pile the red, green, blue boxes in sequence (after 1.1 finished)
    - 1.2.1) First move the red box, then move the green box, then move the blue box.

Below is a new task:

Fig. 3: Prompt for generating HTN-like representation.

Prompt for generating JSON file

Next, write the HTN into JOSN format where each task is in the following format:

```
1 "Task_1.1.2": {
2   "task_id": "Task_1.1.2",
3   "task_instruction": "move toys into green box",
4   "task_related_description": "you can start to putting toys into the green box",
5   "sibling_nodes_condition": "After putting blocks to red box",
6   "subtasks_of_this_node": ["Task_1.1.2.1", "Task_1.1.2.2"]
7 },
```

where “task\_id” denotes the unique task id, “task\_instruction” summarize the task along with all of its subtasks, “task\_related\_description” is the part of sentence corresponding to this task\_i, “subtasks\_of\_this\_node” is the set of subtasks that “task\_i” is decomposed into, only including task ids.

Please write the complete JSON. DO NOT IDLE For brevity.

Fig. 4: Prompt for generating JSON file.

### Prompt for extracting task-wise flat LTL specifications

First get "sibling\_nodes\_condition" field of Task\_1.1 and Task\_1.2 that describe the relations between siblings, then determine their respective property (co-safe or safe). Next describe tasks themselves (without diving deep into the internal part of tasks) along with their relationship using one sentence, in a way that is normally used to interpret the safe or co-safe property in linear temporal logic. The paraphrased sentence uses task id, e.g., Task\_1.1, Task\_1.2, to refer to concrete tasks, and return it in the JSON dialog box.

```
1 {
2   "LTL_description: Eventually Task_1.1 happens, then eventually Task_1.2
3   happens."
```

Let's think step by step.

Fig. 5: Prompt for extracting task-wise flat LTL specifications.

## APPENDIX B

### EXAMPLE 1: DISHWASHER LOADING PROBLEM

#### Dialog 1: Generating HTN-like representation

**Prompt:** *Same as that in Fig. 3.* "Place items into the dishwasher. You should put plates, mugs and utensils into the lower rack in any order. After putting items to the lower rack, then put things into upper rack, first put saucers, and then put cups."

**GPT-4:** Hierarchical Task Network (HTN) - Load Dishwasher

1. Load dishwasher
  - 1.1 Load items into the lower rack
    - 1.1.1 Put plates into the lower rack (can be done in any sequence with 1.1.2 and 1.1.3)
    - 1.1.2 Put mugs into the lower rack (can be done in any sequence with 1.1.1 and 1.1.3)
    - 1.1.3 Put utensils into the lower rack (can be done in any sequence with 1.1.1 and 1.1.2)
  - 1.2 Load items into the upper rack (should be done after 1.1 is finished)
    - 1.2.1 Put saucers into the upper rack
    - 1.2.2 Put cups into the upper rack (should be done after 1.2.1)

This HTN provides a hierarchical breakdown of the given task into its various sub-tasks. The tasks are broken down into sequential or parallel steps, with sequential steps being represented by the use of "after" and parallel steps allowing for any sequence.

Fig. 6: Example 1 Dialog 1: generating HTN-like representation.

#### Dialog 2: Generating JSON file

**Prompt:** *Same as that in Fig. 4.*

**GPT-4:** Alright, let's break down the previously mentioned HTN for loading the dishwasher into JSON format. First, I will structure the main tasks and the most prominent sub-tasks:

```
1 {
2   "Task_1": {
3     "task_id": "Task_1",
4     "task_instruction": "Load dishwasher with items in the specified order",
5     "task_related_description": "Place items into the dishwasher",
6     "sibling_nodes_condition": "",
```

```

7     "sub-tasks_of_this_node": ["Task_1.1", "Task_1.2"]
8   },
9   "Task_1.1": {
10    "task_id": "Task_1.1",
11    "task_instruction": "Load items into the lower rack in any order",
12    "task_related_description": "first load items into the lower rack",
13    "sibling_nodes_condition": "",
14    "sub-tasks_of_this_node": ["Task_1.1.1", "Task_1.1.2", "Task_1.1.3"]
15  },
16  "Task_1.2": {
17    "task_id": "Task_1.2",
18    "task_instruction": "After putting items to the lower rack, load items into
19    the upper rack in the specified sequence",
20    "task_related_description": "after that load other things into the upper
21    rack",
22    "sibling_nodes_condition": "After loading items into the lower rack",
23    "sub-tasks_of_this_node": ["Task_1.2.1", "Task_1.2.2"]
24  }
25 }

```

Next, I will expand upon the sub-tasks for the lower and upper racks:

```

1 {
2   "Task_1.1.1": {
3     "task_id": "Task_1.1.1",
4     "task_instruction": "Put plates into the lower rack",
5     "task_related_description": "put plates into the lower rack",
6     "sibling_nodes_condition": "can be done in any sequence with Task_1.1.2 and
7     Task_1.1.3",
8     "sub-tasks_of_this_node": []
9   },
10  The rest is omitted for brevity...
11 }

```

This format encapsulates the task structure from the given HTN into the specified JSON format. You can integrate the two code blocks for a complete JSON representation.

Fig. 7: Example 1 Dialog 2: Generating JSON file.

Dialog 3: Extracting task-wise specification between Task<sub>1.1</sub> and Task<sub>1.2</sub>

**Prompt:** Same as that in Fig. 5.

**GPT-4:** Alright, let's tackle this step by step.

Firstly, let's define the concepts of *safe* and *co-safe*:

- **Safe:** A property is “safe” if a violation can be observed in finite time. It pertains to something “bad” that must never happen. In Linear Temporal Logic (LTL), a typical operator to express safety properties is  $\square$  (always).
- **Co-safe:** A property is “co-safe” if its satisfaction can be observed in finite time. It pertains to something “good” that must eventually happen. In LTL, a typical operator to express co-safety properties is  $\diamond$  (eventually).

Using the provided HTN:

- 1) For Task<sub>1.1</sub>: “first load items into the lower rack” has no specific constraint on when it needs to occur in relation to its sibling tasks. However, it does have constraints related to its own sub-tasks (they can occur in any order).
- 2) For Task<sub>1.2</sub>: “after that load other things into the upper rack”, it explicitly mentions that it should be done after Task<sub>1.1</sub>.



Based on the above:

- Task\_1.1 has a co-safe property with respect to its sibling because it has to be done before Task\_1.2.
- Task\_1.2 has a safe property with respect to its sibling because it should not start until Task\_1.1 is complete.

Now, summarizing using LTL:

LTL\_Description: "Always (Task\_1.1 must precede Task\_1.2) and Eventually (Task\_1.1 must be executed)."

So, in JSON dialog box format:

```

1 {
2   "LTL_description": "Always (Task_1.1 must precede Task_1.2) and Eventually (
3     Task_1.1 must be executed)."
```

The LTL sentence is a combination of safe and co-safe properties, expressed in a natural language manner using the task ids.

Fig. 8: Example 1 Dialog 3: Extracting task-wise specification between Task\_1.1 and Task\_1.2.

TABLE I: Extracting task-wise specifications for Example 1

Tasks	LLMs-generated human instructions
sub-tasks of Task_1	Always Task_1.1 must precede Task_1.2 and eventually Task_1.1 must be executed.
sub-tasks of Task_1.1	Eventually Task_1.1.1 is executed and eventually Task_1.1.2 is executed and eventually Task_1.1.3 is executed.
sub-tasks of Task_1.2	Always Task_1.2.1 precedes Task_1.2.2 and eventually Task_1.2.1 is executed and eventually Task_1.2.2 is executed.

## APPENDIX C

### HIERARCHICAL ORDER PICKING PROBLEM

#### A. Task 1

Dialog 1: Generating HTN-like representation

**Prompt:** Same as that in Fig. 3.

"Several robots are tasked with finding, retrieving, and packing items within a market. Each robot is assigned specific responsibilities. Robot\_1 is assigned to the furniture area, where they will remain until Robot\_3 arrives to assist in retrieving furniture together. After successfully retrieving the furniture, Robot\_3 will proceed to the packing area and then to the dock area. Meanwhile, after successfully retrieving the furniture, Robot\_1 is instructed to proceed to the outdoor area and then to the pet area. After Robot\_1 has collected the required items, it should head to the packing area and subsequently to the dock area. Robot\_2 is responsible for visiting both the medicine area and the grocery area, it is important to note that Robot\_2 must first locate the medicine area before moving on to the grocery area. After fetching the necessary items, Robot\_2 is directed to make his way to the packing area and then proceed to the dock area."

**GPT-4:**

1. Complete market tasks (1.1, 1.2, 1.3 can be done in any order)
  - 1.1. Robot\_1 and Robot\_3 retrieve furniture (Robot\_1 will remain in the furniture area until Robot\_3 arrives)
    - 1.1.1. Robot\_1 goes to the furniture area
    - 1.1.2. Robot\_3 arrives at the furniture area
    - 1.1.3. Robot\_1 and Robot\_3 retrieve furniture together
  - 1.2. Robot\_3 proceeds to the packing area and then to the dock area (after 1.1.3 is completed)
    - 1.2.1. Robot\_3 goes to the packing area
    - 1.2.2. Robot\_3 goes to the dock area
  - 1.3. Robot\_1 proceeds to the outdoor area, then to the pet area, then to the packing area, and finally to the dock area (after 1.1.3 is completed)
    - 1.3.1. Robot\_1 goes to the outdoor area
    - 1.3.2. Robot\_1 goes to the pet area
    - 1.3.3. Robot\_1 goes to the packing area

- 1.3.4. Robot\_1 goes to the dock area
- 1.4. Robot\_2 visits the medicine area, then the grocery area, then the packing area, and finally the dock area
  - 1.4.1. Robot\_2 goes to the medicine area
  - 1.4.2. Robot\_2 goes to the grocery area (after 1.4.1 is completed)
  - 1.4.3. Robot\_2 goes to the packing area
  - 1.4.4. Robot\_2 goes to the dock area

Fig. 9: Task 1 Dialog 1: generating HTN-like representation.

Dialog 2: Generating JSON format

**Prompt:** Same as that in Fig. 4.

**GPT-4:**

```

1 {
2   "Task_1": {
3     "task_id": "Task_1",
4     "task_instruction": "Complete market tasks",
5     "task_related_description": "Several agents are tasked with finding,
6       retrieving, and packing items within a market.",
7     "sibling_nodes_condition": "None",
8     "sub-tasks_of_this_node": ["Task_1.1", "Task_1.2", "Task_1.3", "Task_1.4"]
9   },
10  "Task_1.1": {
11    "task_id": "Task_1.1",
12    "task_instruction": "Agent_1 and Agent_3 retrieve furniture",
13    "task_related_description": "Agent_1 is assigned to the furniture area, where
14      they will remain until Agent_3 arrives to assist in retrieving furniture
15      together.",
16    "sibling_nodes_condition": "None",
17    "sub-tasks_of_this_node": ["Task_1.1.1", "Task_1.1.2", "Task_1.1.3"]
18  },
19  "Task_1.1.1": {
20    "task_id": "Task_1.1.1",
21    "task_instruction": "Agent_1 goes to the furniture area",
22    "task_related_description": "Agent_1 is assigned to the furniture area",
23    "sibling_nodes_condition": "remain in the furniture area until Agent_3 arrives",
24    "sub-tasks_of_this_node": []
25  },
26  },
27 }
28 The rest is omitted for brevity...

```

Fig. 10: Task 1 Dialog 2: generating JSON file

TABLE II: Extracting task-wise specifications for Task 1

Tasks	LLMs-generated human instructions
sub-tasks of Task_1	Task_1.1 and Task_1.4 can be executed at any time, while Task_1.2 and Task_1.3 must be executed after Task_1.1 is completed.
sub-tasks of Task_1.1	Task_1.1.1 will keep on until Task_1.1.2 happens, after which Task_1.1.3 will eventually happen.
sub-tasks of Task_1.2	Task_1.2.1 can be executed independently, after which Task_1.2.2 can be executed.
sub-tasks of Task_1.3	Task_1.3.1 must be completed first, followed by Task_1.3.2, then Task_1.3.3, and finally Task_1.3.4.
sub-tasks of Task_1.4	Task_1.4.1 is always executed first, followed by Task_1.4.2, then Task_1.4.3, and finally Task_1.4.4.

B. Task 2

**Dialog 1: Generating HTN-like representation**

**Prompt:** *Same as that in Fig 3.*

"Robot\_1 is tasked with collecting various items in a marketplace. The items include furniture, outdoor goods, pharmaceuticals, groceries, electronics, and pet supplies. After collecting all the necessary items, the robot will proceed to the packing zone and subsequently move to the docking area. Additionally, there is a specific sequence that must be followed when retrieving items. The robot will prioritize collecting furniture and outdoor goods first. Following this, it will focus on obtaining pharmaceutical and grocery items. Lastly, it will gather electronics and pet supplies."

---

**GPT-4:**

1. Collect and move items to docking area
  - 1.1 Collect various items in marketplace
    - 1.1.1 Collect furniture and outdoor goods
      - 1.1.1.1 Collect furniture items
      - 1.1.1.2 Collect outdoor goods
    - 1.1.2 Collect pharmaceutical and grocery items (after 1.1.1 is finished)
      - 1.1.2.1 Collect pharmaceutical items
      - 1.1.2.2 Collect grocery items
    - 1.1.3 Collect electronics and pet supplies (after 1.1.2 is finished)
      - 1.1.3.1 Collect electronics
      - 1.1.3.2 Collect pet supplies
  - 1.2 Proceed to the packing zone (after 1.1 is finished)
  - 1.3 Move to the docking area (after 1.2 is finished)

Fig. 11: Task 2 Dialog 1: generating HTN-like representation

**Dialog 2: Generating JSON file**

**Prompt:** *Same as that in Fig 4.*

---

**GPT-4:** (The rest is omitted for brevity.)

```

1 {
2   "Task_1": {
3     "task_id": "Task_1",
4     "task_instruction": "Collect and move items to docking area",
5     "task_related_description": "Robot_1 is tasked with collecting various items
6       in a marketplace. After collecting all the necessary items, the robot will
7       proceed to the packing zone and subsequently move to the docking area.",
8     "sibling_nodes_condition": "Root Task",
9     "subtasks_of_this_node": ["Task_1.1", "Task_1.2", "Task_1.3"]
10  },
11  "Task_1.1": {
12    "task_id": "Task_1.1",

```

```

11   "task_instruction": "Collect various items in marketplace",
12   "task_related_description": "The robot is tasked with collecting various items
    in a marketplace.",
13   "sibling_nodes_condition": "First Task",
14   "subtasks_of_this_node": ["Task_1.1.1", "Task_1.1.2", "Task_1.1.3"]
15 },
16 "Task_1.1.1": {
17   "task_id": "Task_1.1.1",
18   "task_instruction": "Collect furniture and outdoor goods",
19   "task_related_description": "The robot will prioritize collecting furniture
    and outdoor goods first.",
20   "sibling_nodes_condition": "After finishing the first task",
21   "subtasks_of_this_node": ["Task_1.1.1.1", "Task_1.1.1.2"]
22 },
23 "Task_1.1.1.1": {
24   "task_id": "Task_1.1.1.1",
25   "task_instruction": "Collect furniture items",
26   "task_related_description": "Collect furniture items",
27   "sibling_nodes_condition": "",
28   "subtasks_of_this_node": []
29 },
30 The rest is omitted for brevity...
31 }

```

Fig. 12: Task 2 Dialog 2: generating JSON file

TABLE III: Extracting task-wise specifications for Task 2

Tasks	LLMs-generated human instructions
subtasks of Task_1	Task_1.1 must be completed before Task_1.2 begins, and Task_1.2 must be completed before Task_1.3 begins.
subtasks of Task_1.1	Task_1.1.1 must be completed before Task_1.1.2 can begin, and Task_1.1.2 must be completed before Task_1.1.3 can begin.
subtasks of Task_1.1.1	Either Task_1.1.1.1 or Task_1.1.1.2 can be executed first, and their completion is independent of each other, satisfying co-safe properties.
subtasks of Task_1.1.2	Either Task_1.1.2.1 or Task_1.1.2.2 can be performed first, and completing one does not restrict the initiation or completion of the other.
subtasks of Task_1.1.3	Task_1.1.3.1 and Task_1.1.3.2 can be executed independently, and both should eventually be completed.

### C. Task 3

#### Dialog 1: Generating HTN-like representation

**Prompt:** *Same as that in Fig. 3.*

"Three robots are tasked with retrieving items from a market. Robot\_1 is responsible for collecting medicine, groceries, electronics, and pet supplies. These items can be gathered in any sequence. Once Robot\_1 has secured all the necessary items, it will proceed to the packing area before heading to the docking zone. On the other hand, Robot\_2 and Robot\_3 operate under a redundant configuration. This means that the task is considered successfully completed if either Robot\_2 or Robot\_3 finishes all assigned tasks. However, both must actually attempt the task. Their assignment is to first acquire outdoor products, then move to the packing area, and finally proceed to the docking area."

#### GPT-4:

- 1 Retrieve items and proceed to docking zone (1.1, 1.2, and 1.3 can be done in parallel)
  - 1.1 Robot\_1 tasks
    - 1.1.1 Collect medicine
    - 1.1.2 Collect groceries
    - 1.1.3 Collect electronics

- 1.1.4 Collect pet supplies
- 1.1.5 Move to packing area (only after 1.1.1, 1.1.2, 1.1.3, and 1.1.4 are completed)
- 1.1.6 Move to docking zone (only after 1.1.5 is completed)

1.2 Robot\_2 tasks

- 1.2.1 Acquire outdoor products
- 1.2.2 Move to packing area (only after 1.2.1 is completed)
- 1.2.3 Move to docking zone (only after 1.2.2 is completed)

1.3 Robot\_3 tasks

- 1.3.1 Acquire outdoor products
- 1.3.2 Move to packing area (only after 1.3.1 is completed)
- 1.3.3 Move to docking zone (only after 1.3.2 is completed)

Condition: For 1.2 and 1.3, the task is considered successfully completed if either Robot\_2 or Robot\_3 finishes all assigned tasks (1.2.1 to 1.2.3 or 1.3.1 to 1.3.3). Both Robot\_2 and Robot\_3 must attempt the task.

Fig. 13: Task 3 Dialog 1: Generating HTN-like representation

Dialog 2: Generating JSON format

**Prompt:** Same as that in Fig. 4.

**GPT-4:**

```

1 {
2   "Task_1": {
3     "task_id": "Task_1",
4     "task_instruction": "Retrieve items and proceed to docking zone",
5     "task_related_description": "Three robots are tasked with retrieving items
6       from a market.",
7     "sub-tasks_of_this_node": ["Task_1.1", "Task_1.2", "Task_1.3"]
8   },
9   "Task_1.1": {
10    "task_id": "Task_1.1",
11    "task_instruction": "Robot_1 tasks",
12    "task_related_description": "Robot_1 is responsible for collecting medicine,
13      groceries, electronics, and pet supplies.",
14    "sub-tasks_of_this_node": ["Task_1.1.1", "Task_1.1.2", "Task_1.1.3", "Task_1
15      .1.4", "Task_1.1.5", "Task_1.1.6"]
16  },
17  "Task_1.1.1": {
18    "task_id": "Task_1.1.1",
19    "task_instruction": "Collect medicine",
20    "task_related_description": "Collect medicine",
21    "sub-tasks_of_this_node": []
22  },
23  The rest is omitted for brevity...
24 }

```

Fig. 14: Task 3 Dialog 2: generating JSON file

TABLE IV: Extracting task-wise specifications for Task 3

Tasks	LLMs-generated human instructions
sub-tasks of Task_1	Task_1.1, Task_1.2, and Task_1.3 can execute in parallel, and either Task_1.2 or Task_1.3 completing ensures the overall success of their shared objective.
sub-tasks of Task_1.1	Task_1.1.1, Task_1.1.2, Task_1.1.3, and Task_1.1.4 can execute in any order, but Task_1.1.5 must occur after all these tasks are completed, followed by Task_1.1.6.
sub-tasks of Task_1.2	Task_1.2.1 must be completed before Task_1.2.2 can start, and Task_1.2.2 must be completed before Task_1.2.3 can start.
sub-tasks of Task_1.3	Task_1.3.1 must be completed before Task_1.3.2 can start, and Task_1.3.2 must be completed before Task_1.3.3 can start.