NL2HLTL2PLAN: Scaling Up Natural Language Understanding for Multi-Robots Through Hierarchical Temporal Logic Task Representation

Shaojun Xu[1,*,†], Xusheng Luo[2,*], Yutong Huang[2], Letian Leng[2], Ruixuan Liu[2], Changliu Liu[2]

***Abstract*—To enable non-experts to specify long-horizon, multi-robot collaborative tasks, language models are increasingly used to translate natural language commands into formal specifications. However, because translation can occur in multiple ways, such translations may lack accuracy or lead to inefficient multi-robot planning. Our key insight is that concise hierarchical specifications can simplify planning while remaining straightforward to derive from human instructions. We propose NL2HLTL2PLAN, a framework that translates natural language commands into hierarchical Linear Temporal Logic (LTL) and solves the corresponding planning problem. The translation involves two steps leveraging Large Language Models (LLMs). First, an LLM transforms instructions into a Hierarchical Task Tree, capturing logical and temporal relations. Next, a fine-tuned LLM converts sub-tasks into flat LTL formulas, which are aggregated into hierarchical specifications, with the lowest level corresponding to ordered robot actions. These specifications are then used with off-the-shelf planners. Our NL2HLTL2PLAN demonstrates the potential of LLMs in hierarchical reasoning for multi-robot task planning. Evaluations in simulation and real-world experiments with human participants show that NL2HLTL2PLAN outperforms existing methods, handling more complex instructions while achieving higher success rates and lower costs in task allocation and planning. Additional details are available at nl2hltl2plan.github.io.**

***Index Terms*—Formal Methods in Robotics and Automation; Human-Robot Interaction; Multi-Robot Systems**

I. INTRODUCTION

Large Language Models (LLMs), trained on vast text corpora, display common sense reasoning abilities that enable them to handle routine tasks expressed in human language. The development of LLMs has opened up accessible ways for non-experts to instruct and interact with robots through natural language [1]. One approach is the **neuro-symbolic paradigm** [2], in which an intermediate formal specifiaction is derived from natural language input and subsequently used by existing solvers for planning, offering a structured and consistent interpretation of tasks [3]. This approach is also data-efficient, especially considering the limited availability of robotic data. A core requirement in this line of work is that specifications should be accurate, concise and enhance the downstream planners' effectiveness. It is widely recognized that hierarchical models outperform flat models in interpretability and efficiency [4, 5]. However, effectively incorporating human-derived hierarchical insights into algorithms necessitates careful engineering, posing a challenge to leveraging hierarchical planners.

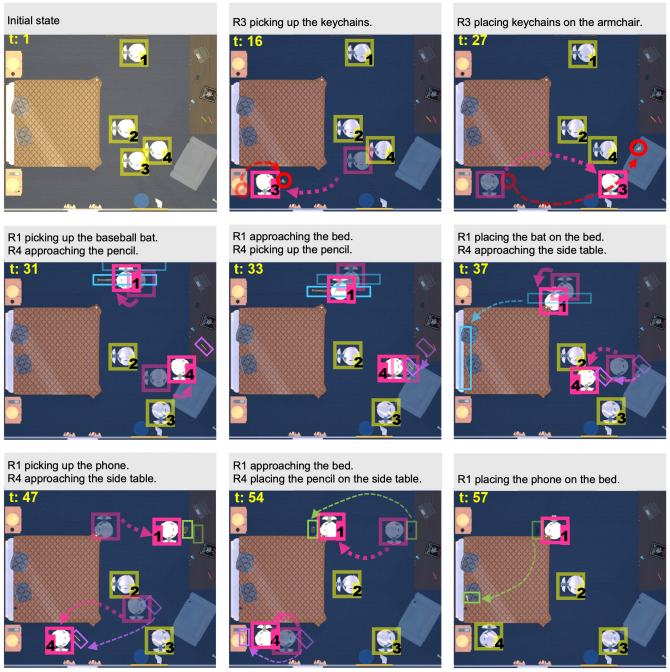


Fig. 1: A sequence of images, arranged from left to right and top to bottom, depicts the task "First, put a set of keychains on the armchair. Retrieve a pencil and put it on the side table. Move the phone and the bat to the bed in any order", objects and their trajectories are marked with different colors as follows, keychains (red), bat (blue), pencil (purple) and phone (green). t represents the discrete time steps in simulation.

Recently, [6] introduced the use of Linear Temporal Logic (LTL) as a standardized framework for specifying goals in embodied decision making, highlighting its expressiveness and compactness. Nonetheless, their approach is limited to addressing simple goals defined by an ordered sequence. Our insight is that task hierarchy can be progressively obtained from language instruction with the help of a LLM. In light of the above, we propose harnessing LLMs as language-based task hierarchy extractors. Hierarchical Linear Temporal Logic, a variant of formal languages introduced in work [7], is adopted as intermediate task specification, which is succinct and results in efficient planners compared to its flat counterpart, aligning well with hierarchically represented human instructions and has been applied to robotics [8, 9]. With hierarchy extraction, we can handle multiple-sentences instructions involving multiple robots, while related work primarily

*Equal contribution.
[1]Shaojun Xu is with Department of Precision Instrument, Tsinghua University, Beijing, 100084, China xusj24@mails.tsinghua.edu.cn
[2]Xusheng Luo, Yutong Huang, Letian Leng, Ruixuan Liu and Changliu Liu are with Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA {xushengl, yutongh3, lleng, ruixuanl, cliu6}@andrew.cmu.edu
†Shaojun Xu was an intern at CMU when this work was conducted.

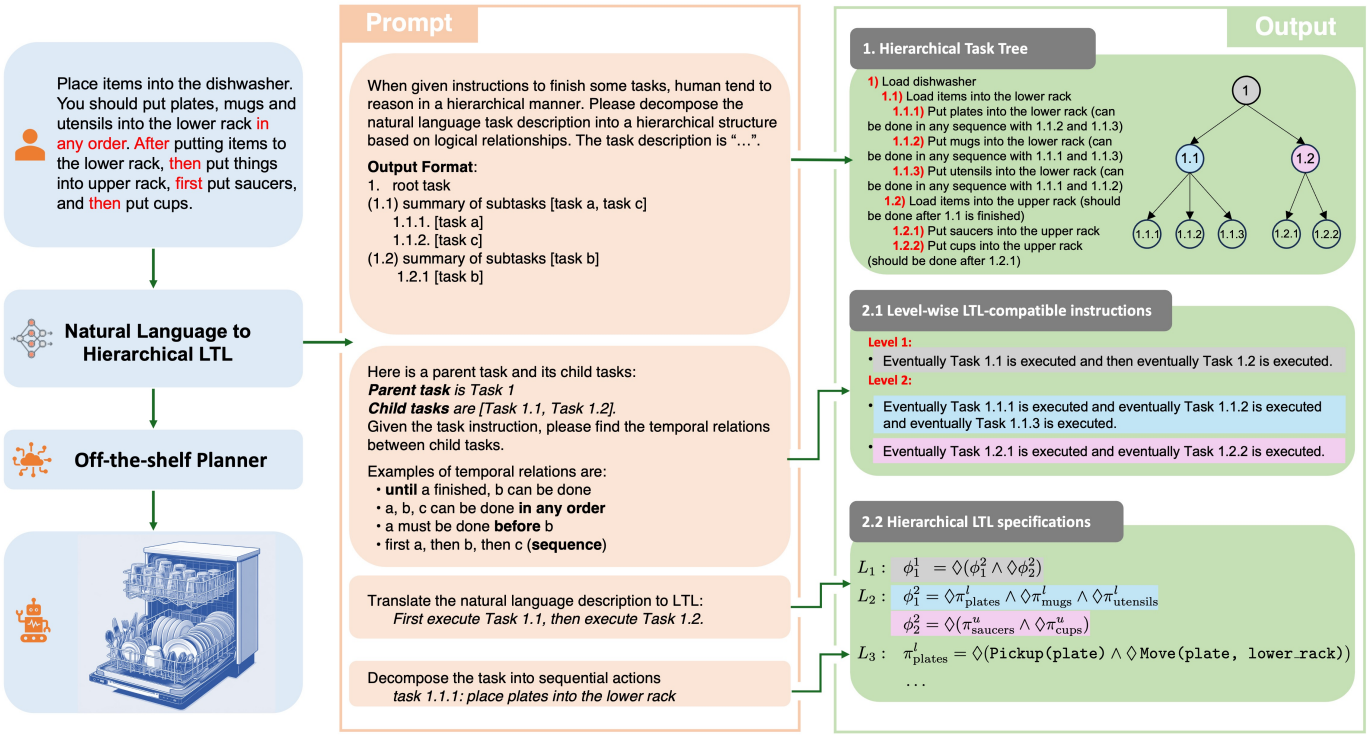


Fig. 2: Overview of the framework NL2HLTL2PLAN. The non-leaf nodes in the Hierarchical Task Tree (see Section IV-A), the language descriptions of subtasks, and the flat specifications are color-coded to indicate one-to-one correspondence. Summary snippets of the prompts are provided, with more information accessible on the project page nl2hltl2plan.github.io.

focuses on short instructions for single robot.

Via fine-tuned LLM, a naive approach to converting language instructions directly into hierarchical LTL is easy to implement. However, this technique tends to perform poorly as LLMs are still not good at logical reasoning [10], which is crucial for crafting logical formulas. Furthermore, LTL formulas in the dataset for learning translations generally have between 2 and 4 propositions [11], rendering them unsuitable for instructions that involve multiple lengthy sentences. We propose a two-step approach NL2HLTL2PLAN to unlock the expressive prowess of temporal logic, converting instructions into hierarchical LTL. Initially, upon receiving an instruction, we prompt an LLM to generate and gradually refine a task representation which is a simplified version of Hierarchical Task Network [12]. Subsequently, in the second phase, subtasks of each task can be translated into a single flat LTL via a fine-tuned LLM. Through iterative processing of all sub-tasks of every task in the intermediary phase, we can construct hierarchical LTL specifications, where the lowest level corresponds to sequentially ordered robot actions. This paradigm of using a formal representationis is data efficient and interpretable [3].

With NL2HLTL2PLAN, human instructions are ready for use by off-the-shelf hierarchical LTL planners, and applied to multi-robot systems with specified objective like cost optimization, which differs from most works that only consider finding feasible solutions rather than optimize under specific objectives. The translation of hierarchical task 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 [11, 13, 14].

Contributions: 1) We proposed a neuro-symbolic method NL2HLTL2PLAN to extract task hierarchies from instructions to facilitate multi-robot planning for long-horizon tasks; 2) We developed a method that transforms language into hierarchical LTL, thus integrating human-derived hierarchical knowledge in planning solvers; 3) We validated our method through simulations and real-world experiments using instructions to formulate plans for multi-robot mobile manipulation tasks.

II. RELATED WORK

Language-Conditioned Robotic Planning: Given instructions, there are two primary methods for generating actions [3]. The first uses deep-learning techniques to translate instructions into low-level actions, such as joint states. Systems on this have shown capabilities across multiple modalities [1, 15–17], but they depend on large volumes of data. Others translate instructions into an intermediate representation, then employing off-the-shelf solvers to generate actions, which limits the solution space, further reducing the need for extensive data. The intermediate representations employed can vary from formal planning formalisms such as Planning Domain Definition Language (PDDL) and temporal logics, to less formal structures like code or predefined skills.

LLMs have been used to extract goal states and domain descriptions from instructions via prompting [18–20]. Their capacity to generate low-level code or call APIs has been

verified [21–25]. An updatable skill library, instead of calling fixed APIs, are introduced by Voyager [26] and SAYCAN [27], and enhanced by INNERMONOLOGUE [28], KNOWNO [29] through integrating feedback or help seeking ability. A commonality is their focus on single-robot scenarios, however, extension to multi-robot scenarios remains largely unexplored.

Natural Language to Temporal Logic: Early attempts at translating natural language into temporal logics relied on grammar-based methods, which excel at processing structured inputs [30]. Recently, the use of LLMs for this task has gained traction, leveraging tools like GPT to generate LTL formulas [14, 31]. While these approaches focus on the translation process, they often overlook the critical issue of grounding language in robotics—linking linguistic instructions to physical actions and environments. To address this, [32] fine-tuned an LLM using a synthetic dataset that pairs natural language instructions with temporal logic formulas designed for quadrotor tasks. Similarly, weakly supervised semantic parsers have been developed to learn from execution trajectories without requiring explicit LTL annotations [33, 34]. Systems such as LANG2LTL [35], NL2TL [36], and others [37] employ LLMs to convert domain-specific commands (e.g., for navigation or motion planning) into formal specifications. In contrast, [38] adopts a predefined LTL specification approach, where predicates are defined using succinct human instructions. Our NL2HLTL2PLAN extends these capabilities, supporting more complex specifications with over 10 atomic propositions and enabling task allocation across multiple robots—surpassing the scope of prior works, which typically handle fewer than five atomic propositions.

LLMs to Multi-Robots: Recently, there has been a notable trend in adapting LLMs for use in multi-robot systems. SMART-LLM [24] uses an LLM to synthesize code that facilitates task decomposition, coalition formation, and task allocation. Multiple intermediate approaches have been implemented in multi robot planning, such as dialogue-based framework [39], behavior trees [40], batch of multi communication frameworks (centralized, decentralized, or hybrid) [41], and address deadlock resolution in navigation scenarios [42]. Decentralized LLM-based planner [43] and global LLM-based planners [44] are introduced to enhance the efficiency of target searches or make individual decisions autonomously However, the works mentioned above focus on finding feasible solutions. In contrast, our research can optimize the cost and time required to complete tasks.

III. HIERARCHICAL 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 ($\neg$), temporal operators like next ($\bigcirc$) and until ($\mathcal{U}$) [45]:

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

where $\top$ stands for a true statement, and π is a boolean valued atomic proposition. Other temporal operators can be derived from $\mathcal{U}$, such as $\Diamond\phi$ that implies ϕ will be true at a future time. We focus on a subset of LTL known as syntactically co-safe formulas (sc-LTL) [46]. Any LTL formula encompassing only the temporal operators $\Diamond$ and $\mathcal{U}$ and written in positive normal form (where negation is exclusively before atomic propositions) is classified under sc-LTL formulas [46], which can be satisfied by finite sequences followed by any infinite repetitions. This makes sc-LTL apt for reasoning about robot tasks with finite duration.

Definition 3.1 (Hierarchical sc-LTL [7]): Hierarchical sc-LTL is structured into K levels, labeled as $L_1, \ldots, L_K$, arranged from the highest to the lowest. Each level L_k, where $k \in [K]$, contains n_k sc-LTL formulas. The hierarchical sc-LTL can be represented as $\Phi = \left\{\phi_k^i \mid k \in [K], i \in [n_k]\right\}$, where ϕ_k^i denotes the i-th sc-LTL formula at level L_k. The hierarchical sc-LTL adheres to the following rules:

1) Each formula at a given level L_k, for $k \in [K-1]$, is derived from the formulas at the next lower level L_{k+1}.
2) Every formula at any level other than the highest (i.e., $k = 2, \ldots, K$) is included in exactly one formula at the next higher level L_{k-1}.
3) Atomic propositions are used exclusively within the formulas at the lowest level L_K.

Let Φ^k denote the set of formulas at level L_k with $k \in [K]$. We refer to each specification ϕ_i^k in Φ as the "flat" specification, which can be organized in a tree-like specification hierarchy graph, where each node represents a flat sc-LTL specification. Edges between nodes indicate that one specification belongs to another as a *composite proposition*. The K-th level leaf nodes represent *leaf specifications* that consist only of atomic propositions, while non-leaf nodes represent *non-leaf specifications* made up of composite propositions.

Example 1 (Dishwasher Loading Problem): Consider the following instruction: "Place items into the dishwasher. 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." The hierarchical LTL is:

$$\begin{aligned} L_1: \quad & \phi_1^1 \;= \Diamond(\phi_2^1 \wedge \Diamond\phi_2^2) \\ L_2: \quad & \phi_2^1 = \Diamond\pi^l_{\text{plates}} \wedge \Diamond\pi^l_{\text{mugs}} \wedge \Diamond\pi^l_{\text{utensils}} \\ & \phi_2^2 = \Diamond(\pi^u_{\text{saucers}} \wedge \Diamond\pi^u_{\text{cups}}), \end{aligned} \qquad (2)$$

where ϕ_2^1 and ϕ_2^2 are composite propositions, and the formula $\Diamond(\phi_2^1 \wedge \Diamond\phi_2^2)$ specifies that ϕ_2^1 should be fulfilled before moving on to ϕ_2^2. π_{i}^j represents atomic propositions, denoting the act of placing a specific type of dishware. Note that the lowest level L_2 only includes atomic propositions.

IV. METHODOLOGY: NL2HLTL2PLAN

LLMs excel in common sense reasoning but perform poorly in logical reasoning and lack grounding in the available robot skills [10, 47]. Therefore, we propose a two-stage method for translating natural language into hierarchical LTL using an intermediary structure known as the Hierarchical Task Tree.

A. Conversion from instructions to Hierarchical Task Tree

Definition 4.1 (Hierarchical Task Tree (HTT)): A Hierarchical Task Tree (HTT) is a tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where

- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ denotes the set of nodes. Each node is associated with an instruction of its respective task;

- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents the edges, indicating a decomposition relationship between tasks. An edge $e = (v_1, v_2) \in \mathcal{E}$ implies that *child* task v_2 is in sub-tasks set of *parent* task v_1. The node set $\mathcal{V}$ can be partitioned into multiple disjoint subsets $\{\mathcal{V}_1, \ldots, \mathcal{V}_m\}$, such that all nodes within the same subset $\mathcal{V}_i$ share the same parent node.
- $\mathcal{R} \subseteq \mathcal{V} \times \mathcal{V}$ defines the set of temporal relations between *sibling* tasks, which are decompositions of the same parent task. A relation $(v_1, v_2) \in \mathcal{R}$, where $v_1, v_2 \in \mathcal{V}_i$ for some $i \in \{1, \ldots, m\}$, indicates that task v_1 should be completed before task v_2.

The HTT is a simplified version of the hierarchical task network (HTN) as it is specifically designed to align with the structure of hierarchical LTL. The tree unfolds level by level, where each child task is a decomposition of its parent task. The relation $\mathcal{R}$ specifically captures the temporal relationships between sibling tasks that share the same parent. The temporal relationship between any two tasks can be inferred by tracing their lineage back to their common ancestor. The primary distinction between HTT and HTN is that HTN includes interdependencies between sub-tasks under different parent tasks and each node in the HTT is solely focused on the sub-task goal and does not incorporate other properties like preconditions and effects that are found in HTN. A LLM is employed to construct the HTT through a two-step process from given task instruction, as outlined in step 1 of Fig. 2.

1) HTT without temporal relations $\mathcal{R}$: The first step involves generating the nodes $\mathcal{V}$ and edges $\mathcal{E}$, excluding the temporal relations $\mathcal{R}$. The LLM is employed to decompose the whole task into a structured hierarchy and the decomposition continues until a task consists solely of sequential operations performed on a single object.

2) Add temporal relations $\mathcal{R}$: For each non-leaf node v, we consider $\mathcal{V}'$, which represents its child tasks at the level directly beneath it. Then temporal relations between sibling tasks within $\mathcal{V}'$ is determined by LLM.

B. Generation of task-wise flat LTL specifications

Once the HTT representation is obtained, a flat LTL is generated for each node via a breadth-first search; see Alg. 1.

1) Logical search: For every non-leaf node v, we gather its child tasks $\mathcal{V}'$ and the temporal relations among them, defined by $\mathcal{R}' \subseteq \mathcal{V}' \times \mathcal{V}'$. We then use an LLM to rephrase these child tasks with their temporal relations into syntactically correct sentences aligned with the semantics of LTL specifications (as illustrated in step 2.1 in Fig. 2). A fine-tuned LLM is then used as a translator to obtain single LTL formula from reformulated sentences (as depicted in step 2.2 in Fig. 2). To this end, we first developed a dataset comprising pairs of natural language descriptions and their corresponding LTL formulas, and then fine-tune a language model for translation, `Mistral-7B-Instruct-v0.2` [48]. Training datasets were synthesized from sources including Efficient-Eng-2-LTL [32], Lang2LTL [35], nl2spec [14], and NL2TL [36]. Given the domain-specific nature of these datasets, we substituted specific tasks with generic symbols such as "task 1.1 should be completed before task 1.2" paired with the LTL

Algorithm 1: Construction of hierarchical LTL

Input: HTT $\mathcal{T}$
Output: Hierarchical LTL specifications

$\mathcal{V}_{\text{front}} = \varnothing$, $\Phi = \varnothing$; ▷ $\mathcal{V}_{\text{front}}$ is a stack that contains nodes to be expanded
$\mathcal{V}_{\text{front}}$.push($v_{\text{root}}$) ; ▷ Add root node
while $\mathcal{V}_{\text{front}} \neq \varnothing$ **do**
  $v = \mathcal{V}_{\text{front}}$.pop();
  k = GetDepth(v) ; ▷ Get the depth of node v in $\mathcal{T}$, GetDepth(v_{root}) = 1
  i = Count(Φ, k) ; ▷ Count the number of specifications at level k in Φ
  if v is a leaf node **then**
    ϕ^k_{i+1} = ActionCompletion(v);
  else
    $\mathcal{V}'$ = GetChildren($\mathcal{T}, v$);
    $\mathcal{V}_{\text{front}}$.push($\mathcal{V}'$)
    $\mathcal{R}'$ = GetTemporalRelations($\mathcal{T}, \mathcal{V}'$);
    ϕ^k_{i+1} = GenerateLTL($\mathcal{V}', \mathcal{R}'$) ; ▷ Generate the single LTL
  Φ.add(ϕ^k_{i+1});
return Φ;

ϕ = $\Diamond$(`task1.1` $\wedge$ $\Diamond$`task1.2`), which allows the fine-tuned LLM act as a task unrelated translator, as demonstrated in [32, 36]. Next, we ask an LLM to reinterpret these "lifted" LTL specifications, creating a domain-agnostic dataset containing approximately 509 unique LTL formulas and 10621 natural language descriptions produced by the LLM.

2) Action completion: Given an HTT, each leaf node represent a simple task on certain objects, such as "task 1.1.1: place plates into the lower rack" in Fig. 2. Viewing such simple task as a sequence of action steps, LLM is asked to expand the short instruction into a sequence of pre-defined APIs. This approach helps improve alignment with robot skills and has demonstrated effectiveness [21]. For instance, the symbol π^l_{plates} that represents task 1.1.1 can be replaced with LTL specification composed of sequential APIs: π^l_{plates} = $\Diamond$(`Pickup(plate)` $\wedge$ $\Diamond$`Move(plate, lower_rack)`); see step 2.2 in Fig. 2. After this step, a complete hierarchical LTL specifications is generated.

Remark 4.2: Assuming the HTT contains n_1 non-leaf nodes and n_2 leaf nodes, our method queries the LLM $2(n_1+n_2)+1$ times. Firstly, an LLM are queried once to create the HTT without temporal relations. Subsequently, in $n_1 + n_2$ times, temporal relations for non-leaf nodes and serial actions for leaf nodes are derived. Finally, nodes are tranlsated to flat LTL formulas in $n_1 + n_2$ times via a fine-tuned LLM.

V. Experimental Results

We evaluate the performance of NL2HLTL2PLAN both in simulated and real-world environments. For simulation, we use the AI2-THOR simulator [49], an interactive 3D environment that models various domestic settings, coupled with the ALFRED dataset [50], which focuses on natural language comprehension and embodied actions. In real-world

experiments, we arrange objects on a tabletop using single or multiple robotic arms via handover. We employ GPT-4 [51] and aim to answer three key questions:

Q1. Is NL2HLTL2PLAN capable of reasoning over complex human instructions effectively?

Q2. Does NL2HLTL2PLAN achieve higher success rates while maintaining high solution quality?

Q3. Is NL2HLTL2PLAN flexible enough to adjust to the verbal styles of various users?

A. Mobile manipulation tasks in AI2-THOR

1) Tasks: The ALFRED dataset contains task instructions with strictly sequential steps, which we classify as *base* tasks. To create more complex tasks, we procedurally combine base tasks in same scenes to generate *derivative* tasks. Specifically, the tasks are firstly identified by a LLM to ensure the same object is not included in multiple base tasks simultaneously. The base tasks that involve distinct objects are then randomly combined with randomly generated temporal relationships. Subsequently, the randomly combined tasks are then reformulated into *derivative* tasks by the LLM to align more naturally with human expression patterns. The number of base tasks, varied from 1 to 4, are used to reflect the complexity of the derivative task. 50 derivative tasks are generated for each category; and one of them is shown in Fig. 1. We then assign 1, 2, or 4 robots, each with randomly chosen initial positions within the floor plan, leading to $4\times50\times3 = 600$ test scenarios. For simultaneous task allocation and planning, a search-based planner [7] for a multi-robot system is employed.

2) Comparison: We compare our method with SMART-LLM [24], which uses an LLM to generate Python scripts invoking predefined APIs of actions for the purposes of task decomposition and task allocation. The diagram comparison of these two pipelines is displayed in Fig. 3. Other approaches, such as those based on PDDL or LTL, face significant challenges in solving the tasks discussed in this paper. Translating instructions into PDDL fails to account for temporal constraints, while methods that expand *derivative* tasks into flat LTL representations become excessively complex and are therefore unsuitable for managing the tasks presented here.

3) Metrics: We consider the following metrics. 1) Success rate, which measures whether the target goal states of objects are achieved and if the order in which these states occur satisfies the specified temporal requirements. For a detailed analysis, we further break it down into two separate components: a) conversion, b) planning. 2) Travel cost, measured in meters, is defined as the total distance traveled by all robots, assuming no movements in manipulation. 3) Completion time, quantified as the number of discrete time steps used to complete the tasks.

4) Results: The dimensions of grid maps range from (25∼30)×(25∼30) based on scene size. The statistical results are shown in Tab. I, which are organized based on the number of base tasks included in the derivative tasks. This provides affirmative answers to our first two questions **Q1** and **Q2**. SMART-LLM is limited to solving derivative tasks with only one base task, whereas our method can handle up to 4 tasks. For tasks comprising more than two base tasks, SMART-LLM's output exceeds the context window of GPT-4 (as its reasoning relies on the whole context), indicating that it uses a significant number of tokens to generate Python scripts. To address this, we introduced an additional layer atop SMART-LLM, providing a satisfying sequence of base tasks decomposed from derivative tasks. Each base task is then sequentially processed through SMART-LLM to obtain a viable solution. In general, our approach not only achieves a higher success rate but also results in plans that are more cost-effective and require shorter amount of time to complete. For derivative tasks comprising 4 base tasks, SMART-LLM exhibits a considerably lower success rate. However, NL2HLTL2PLAN still attains a success rate of approximately 84% when converting to hierarchical LTL. As the number of robots increases, both travel costs and completion times decrease due to the parallel execution of base tasks. However, the success rate slightly decreases during the planning phase when more robots are involved as the off-the-shelf planning search time exceeds the five-minute timeout. We hypothesize that the planner [7] employs a best-first search strategy to ensure optimality, facing a substantial challenge due to the vast search space involving long-horizon tasks, action spaces (both navigation and manipulation), and map dimensions. More robots can be handled by upgrading to a more capable downstream planner. This flexibility in utilizing off-the-shelf planners differentiates our approach from existing studies where LLMs are primarily used for task allocation. Note that only travel cost and completion time for successfully completed tasks are recorded. Therefore, the data for SMART-LLM are not fully representative due to its lower success rate.

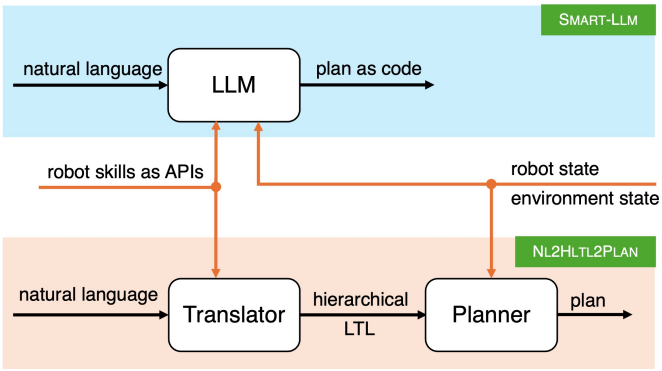


Fig. 3: Comparison of pipelines from natural language to plans between SMART-LLM and NL2HLTL2PLAN.

Tasks	Success rate %		Travel cost (m)		Completion time	
-robots	ours (HLTL, PL)	[24]	ours	[24]	ours	[24]
1-1	**92** (100,92)	48	**2.9±2.0**	4.7±3.5	**11.5±7.9**	14.1±9.6
1-2	**90** (100, 90)	36	**2.7±1.9**	6.3±4.7	**10.6±8.0**	17.8±14.1
1-4	**88** (100, 88)	22	**2.4±2.0**	6.0±5.9	**9.3±7.8**	18.5±22.4
2-1	**84** (92, 91)	16	**7.1±6.2**	10.1±4.8	**28.4±24.8**	28.8±16.9
2-2	**82** (92, 89)	10	**7.0±6.2**	7.3±5.0	**22.7±21.2**	29.0±19.8
2-4	**74** (92, 81)	8	5.9±5.7	**3.8±3.3**	18.6±17.7	**15.2±13.1**
3-1	**78** (86, 91)	46	13.1±8.6	**12.7±5.9**	52.5±34.4	**41.0±22.3**
3-2	**76** (86, 88)	38	**12.6±8.3**	16.8±10.3	**35.0±23.0**	50.4±43.1
3-4	**68** (86, 79)	18	**12.4±9.0**	14.1±7.2	**35.0±25.8**	41.6±17.7
4-1	**74** (84, 88)	36	**12.5±8.3**	21.8±5.1	**50.1±33.1**	70.9±35.6
4-2	**74** (84, 88)	26	14.0±9.4	**13.0±4.8**	**38.1±26.8**	71.5±33.5
4-4	**64** (84, 77)	18	**12.6±7.8**	13.2±1.3	**33.7±24.8**	81.8±31.6

TABLE I: Performance comparison. The success rate column first presents the overall success rate, with the success rates for conversion and planning in parentheses.

Task	Success rate (%)		Travel cost		Runtimes (s)	
ID	ours	LLM	ours	LLM	ours	LLM
1	100	100	111.2±21.6	111.2±21.6	5.9±0.5	**3.5±0.7**
2	100	100	**150.6±26.2**	160.7±20.4	7.1±0.5	**6.7±3.2**
3	100	100	**172.5±36.4**	211.3±27.8	11.5±0.3	**5.4±0.3**
4	100	100	**232.7±37.8**	235.3±35.4	25.7±2.4	**6.4±0.4**

ID	Task description
1	Place the green apple in the blue plate and the orange in the yellow plate.
2	Place the green apple in the pink plate, the orange in the yellow plate and the red apple in the blue plate. The order of placement is not specified and can be chosen freely.
3	Place the carrot in the blue plate, the orange in the yellow plate, the green apple in the green plate, and the red apple in the pink plate in any order.
4	Begin by placing the green apple and orange in the yellow plate. Once done, place the carrot and red apple in the blue plate also in any order.

TABLE II: Statistical results from tabletop experiments.

Tasks of higher complexity, which typically involve greater travel costs and longer completion times, are more likely to fail and are thus excluded from the data. A series of snapshots capturing task execution is displayed in Fig. 1.

B. Real-world rearrangement involving human participants

The real-world tabletop experiment is with a robotic arm placing fruits and vegetables onto colored plates. Given the 2D nature of the task, we convert the environment into a discrete grid world, and use the planner [7]. The use of one arm simplifies the task compared to the multi-robot scenarios, as it eliminates task allocation. Our evaluation has two aspects: a) the adaptability to verbal tones and styles from various users; and b) the comparative effectiveness of the plan generated from our method against existing methods. To explore the first aspect, we conduct a user study with 4 participants, asking each to rephrase the task instructions according to personal style, while maintaining the original semantics. For the second aspect, we employ an LLM as the task planner, explicitly prompting it to minimize trajectory length based on the provided initial 2D coordinates of all objects and robotic arms. This approach directly generates a sequence of API calls, similar to the method used in PROGPROMPT [21]. We developed a dataset containing instructions for eight arrangement tasks, each specified with temporal constraints. To address the probabilistic behavior of the LLM, we conducted 5 queries to the LLM for each rephrased instruction, resulting in a total of 25 test cases per task. In each scenario, object locations are randomized. The cost metric used is the projected travel distance of the robotic arm within a 2D space.

The results are presented in Tab. II, which positively answers **Q3**. As observed, both NL2HLTL2PLAN and the LLM achieve a high success rate, which aligns with the expectations given the task complexities. Regarding cost, with multiple feasible solutions, NL2HLTL2PLAN consistently produces lower-cost paths, with the exception of task 1. In this task, the LLM manages to create an optimal plan given the placement of fruits. The runtimes include the time to obtain the executable action sequence. NL2HLTL2PLAN experienced slightly longer runtimes compared to the LLM because querying times varies in different HTT structure. Comparison between NL2HLTL2PLAN and the LLM is displayed in Fig. 4.

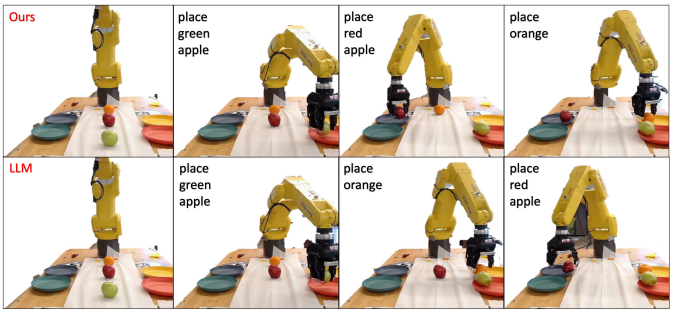


Fig. 4: Comparative snapshots between NL2HLTL2PLAN and an LLM for task 6. NL2HLTL2PLAN generates an optimal trajectory, whereas the LLM follows the sequence in which the fruits are mentioned in the instructions.

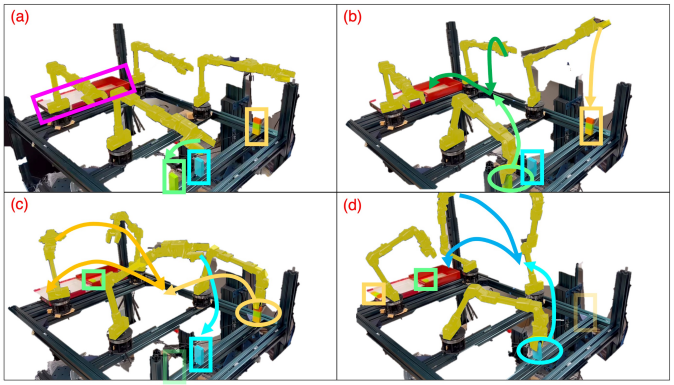
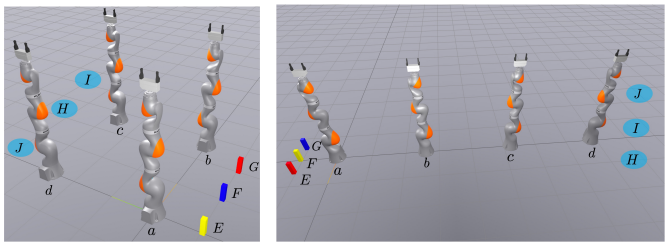


Fig. 5: Snapshots depict four arms performing real-world tasks of picking and placing objects via handover. The instruction given is, "Please move the blue, green, and multi-colored blocks to the two opposite boxes, place the colored ones after the green ones." Target areas are colored in magenta. The block being selected is emphasized with an ellipse, and the remaining blocks are contained within rectangles. The colored curves with arrows illustrate the trajectories of the end-effectors, where head-to-head arrows indicate handovers between robotic arms.

C. Multi-robot handover tasks

We examine the execution of pick-and-place tasks involving multiple objects by four fixed robot arms, which are either aligned in a straight line or arranged in a square configuration; see Fig. 6. Certain tasks might necessitate the transfer of objects between robots, depending on their proximity. The planner our approach uses, inspired from work [52], produces collision-free trajectories by simultaneously considering task and motion planning. The prompt for the baseline that directly uses the LLM as task planner is illustrated in Fig. 2.

Tab. III displays eight multi-stage pick-and-place tasks with temporal constraints. For the LLM-based planner, a planning scheme is deemed successful if it allows for the sequential actions of multiple robots to be executed successfully while adhering to the temporal constraints. It is evident that for tasks involving robot handovers, the success rate of the LLM-based planner decreases due to the need for cooperative planning. Considering the probabilistic output of GPT-4, we conducted 10 tests per task to enhance the diversity of the LLM's responses. The results indicate that by dividing the planning process into task hierarchical extraction and LTL-based optimization, we can effectively bypass direct control of robots' low-level movements, thereby improving completion of multi-stage handover tasks. Moreover, we conduct experiments in a real-world setting with four robotic arms, and a series of



(a) Straight line configuration (b) Square configuration

Fig. 6: Four robot arms in straight line or square configurations, where symbols E, F and G represent source locations and H, I and J denote target locations.

Success rate (%)	Task ID							
	1	2	3	4	5	6	7	8
Ours	**100**	**100**	**80**	**80**	**80**	**90**	**100**	**100**
LLM	90	30	20	20	0	40	10	40

ID	task description
1	Transfer three blocks into the opposite. The red block should be placed before the blue block.
2	Put the three blocks into three goal region at region H at, region I and region J each goal region can only be placed with one object.
3	Please plan a solution to move the three pillars to the placement area (region E, F, G) to the side of the arm a and arm b. Each object cannot be placed on the area other than the target region.
4	Transfer three blocks into the opposite regions. The robot arm cannot touch blue block until any of the robot arm touched the red block. After all of these, you should place yellow block back to region G (where the red block is currently locating).
5	The red block should be placed before the blue block. Please exchange the position of the red block and blue block. At any time, put the yellow block into the opposite side.
6	The red block should be placed before the blue block. Please place the three objects separately to region G, H and I.
7	The red block should be placed before the blue block. Please place the three objects one after another to region I, no objects can be placed on region F, G and H during this period.
8	Please first place the yellow block to region I, then place the yellow block to region G, and move red and blue blocks to any empty region at any time.

TABLE III: Statistical results for multi-robot handover. The first five tasks involve scenarios where four arms are arranged in a square formation, while the last three tasks involve scenarios where the four arms are aligned in a straight line.

snapshots are presented in Fig. 5.

D. Analysis of failure reasons

We categorize the causes of failure into four groups, each aligned with the four stages described in Sections IV-A and IV-B. These failure rates are presented in Tab. IV.

1) Task decomposition: The breakdown of tasks might omit certain subtasks. For instance, a decomposition of "Heat a sliced tomato in microwave." could be "1.1 slice a tomato, 1.2 heat the tomato in microwave". However, there is a necessary intermediate step absent between 1.1 and 1.2, which should be 1.2 place the tomato in microwave.

2) Temporal extraction: Ambiguous wording might cause an LLM to recognize only a subset of temporal relations. Consider the sequence: "Put a bread in the oven [1.1], place a pot in the pool [1.2]. At any time, move a bowl to the desk [1.3]." The output suggests 1.2 and 1.3 can occur in any order, and 1.2 follows 1.1. This inference arises from the absence of clear directives on order. A better inference would indicate that 1.1 and 1.2 can be performed in any order.

Error type	task decomposition	temporal extraction	LTL translation	action completion
Failure rate %	1.33	1.83	2.83	2.50

TABLE IV: Statistics of failure cases. The results derive from scenarios in Sections V-A and V-C, encompassing 208 task descriptions altogether. The rate is determined across the total instances in the HTT. An HTT comprising n non-leaf nodes and m leaf nodes accounts for m instances in action completion and n instances across the other three categories.

3) LTL translation: The conversion process may result in inaccuracies. For example, the specified task sequence "First 1.1, then 1.2, 1.3, and 1.4 can be completed in any sequence" is erroneously translated as $\Diamond(p_{1.1} \wedge (\Diamond(p_{1.2} \wedge (p_{1.3} \wedge p_{1.4}))))$. The correct formula should be $\Diamond(p_{1.1} \wedge \Diamond p_{1.2} \wedge \Diamond p_{1.3} \wedge \Diamond p_{1.4})$.

4) Action completion: Redundant actions that duplicate previous ones may occur. For instance, the phrase "Place **the** sliced tomato on the pan," which functions as a leaf node in HTT, implies that the tomato has already been sliced. A redundant sequence like "pick(tomato), slice(tomato), put(pan, tomato)" would be inappropriate here, as it reflects the actions for a non-leaf node, such as "Place **a** sliced tomato on the pan."

VI. Conclusions and Limitations

We proposed NL2HLTL2PLAN to transform unstructured language into a structured, hierarchical formal representation–hierarchical LTL, where the lowest level corresponds to sequentially ordered robot actions. The task representation is ready to be used by off-the-shelf planners for multi-robot systems. Our simulation and real-world experiment outcomes demonstrated that the framework offers an intuitive and user-friendly approach for deploying robots in daily situations.

Limitations: The NL2HLTL2PLAN operates as an open loop without feedback. To transition to a closed-loop one, it is essential to integrate a syntax checker and a semantic checker. The syntax checker verifies adherence to the hierarchical LTL structure necessary for HTT representation. Meanwhile, the semantic checker offers feedback on errors when the planner fails to identify a solution. Another limitation is that once created, the HTT representation remains unchanged. We derive an LTL specification by extracting child tasks from a parent task. As more child tasks are included, the accuracy of translation drops. Therefore, to handle tasks with more base tasks, it is necessary to restructure the HTT to restrict the number of child tasks a single parent task has.

References

[1] A. Padalkar et al., "Open x-embodiment: Robotic learning datasets and rt-x models," arXiv preprint arXiv:2310.08864, 2023.
[2] V. Belle et al., "Neuro-symbolic ai+ agent systems: A first reflection on trends, opportunities and challenges," in International Conference on Autonomous Agents and Multiagent Systems. Springer, 2023, pp. 180–200.
[3] V. Cohen et al., "A survey of robotic language grounding: Tradeoffs between symbols and embeddings," arXiv preprint arXiv:2405.13245, 2024.
[4] J. B. Tenenbaum et al., "How to grow a mind: Statistics, structure, and abstraction," science, vol. 331, no. 6022, pp. 1279–1285, 2011.
[5] C. Kemp et al., "Learning overhypotheses with hierarchical bayesian models," Developmental science, vol. 10, no. 3, pp. 307–321, 2007.

[6] M. Li et al., "Embodied agent interface: Benchmarking llms for embodied decision making," in The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

[7] X. Luo and C. Liu, "Simultaneous task allocation and planning for multi-robots under hierarchical temporal logic specifications," arXiv preprint arXiv:2401.04003, 2024.

[8] 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.

[9] 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.

[10] F. Xu et al., "Are large language models really good logical reasoners? a comprehensive evaluation from deductive, inductive and abductive views," arXiv preprint arXiv:2306.09841, 2023.

[11] J. X. Liu et al., "Grounding complex natural language commands for temporal tasks in unseen environments," in 7th Annual Conference on Robot Learning, 2023.

[12] M. Ghallab, D. Nau, and P. Traverso, Automated planning and acting. Cambridge University Press, 2016.

[13] Y. Chen et al., "Nl2tl: Transforming natural languages to temporal logics using large language models," arXiv preprint arXiv:2305.07766, 2023.

[14] M. Cosler et al., "nl2spec: Interactively translating unstructured natural language to temporal logics with large language models," in International Conference on Computer Aided Verification. Springer, 2023, pp. 383–396.

[15] O. M. Team et al., "Octo: An open-source generalist robot policy," arXiv preprint arXiv:2405.12213, 2024.

[16] Y. Jiang et al., "Vima: General robot manipulation with multimodal prompts," in NeurIPS 2022 Foundation Models for Decision Making Workshop, 2022.

[17] X. Li et al., "Vision-language foundation models as effective robot imitators," arXiv preprint arXiv:2311.01378, 2023.

[18] 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.

[19] 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.

[20] K. Valmeekam et al., "Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change," Advances in Neural Information Processing Systems, vol. 36, 2024.

[21] I. Singh et al., "Progprompt: program generation for situated robot task planning using large language models," Autonomous Robots, pp. 1–14, 2023.

[22] J. Liang et al., "Code as policies: Language model programs for embodied control," in 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023, pp. 9493–9500.

[23] 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," in Conference on Robot Learning. PMLR, 2023, pp. 540–562.

[24] S. S. Kannan et al., "Smart-llm: Smart multi-agent robot task planning using large language models," arXiv preprint arXiv:2309.10062, 2023.

[25] Z. Hu et al., "Deploying and evaluating llms to program service mobile robots," IEEE Robotics and Automation Letters, 2024.

[26] G. Wang et al., "Voyager: An open-ended embodied agent with large language models," arXiv preprint arXiv:2305.16291, 2023.

[27] A. Brohan et al., "Do as i can, not as i say: Grounding language in robotic affordances," in Conference on robot learning. PMLR, 2023, pp. 287–318.

[28] W. Huang et al., "Inner monologue: Embodied reasoning through planning with language models," in Conference on Robot Learning. PMLR, 2023, pp. 1769–1782.

[29] A. Z. Ren et al., "Robots that ask for help: Uncertainty alignment for large language model planners," in 7th Annual Conference on Robot Learning, 2023.

[30] S. Konrad and B. H. Cheng, "Real-time specification patterns," in Proceedings of the 27th international conference on Software engineering, 2005, pp. 372–381.

[31] F. Fuggitti and T. Chakraborti, "Nl2ltl–a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas," in AAAI Conference on Artificial Intelligence, 2023.

[32] J. Pan, G. Chou, and D. Berenson, "Data-efficient learning of natural language to linear temporal logic translators for robot task specification," in 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023, pp. 11 554–11 561.

[33] R. Patel, R. Pavlick, and S. Tellex, "Learning to ground language to temporal logical form," in Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), 2019.

[34] C. Wang, C. Ross, Y.-L. Kuo, B. Katz, and A. Barbu, "Learning a natural-language to ltl executable semantic parser for grounded robotics," in Conference on Robot Learning. PMLR, 2021, pp. 1706–1718.

[35] J. X. Liu et al., "Lang2ltl: Translating natural language commands to temporal robot task specification," arXiv preprint arXiv:2302.11649, 2023.

[36] Y. Chen et al., "NL2TL: Transforming natural languages to temporal logics using large language models," in Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 15 880–15 903.

[37] J. Hsu et al., "What's left? concept grounding with logic-enhanced foundation models," Advances in Neural Information Processing Systems, vol. 36, 2024.

[38] J. Wang et al., "Conformal temporal logic planning using large language models: Knowing when to do what and when to ask for help," arXiv preprint arXiv:2309.10092, 2023.

[39] Z. Mandi et al., "Roco: Dialectic multi-robot collaboration with large language models," arXiv preprint arXiv:2307.04738, 2023.

[40] A. Lykov et al., "Llm-mars: Large language model for behavior tree generation and nlp-enhanced dialogue in multi-agent robot systems," arXiv preprint arXiv:2312.09348, 2023.

[41] Y. Chen et al., "Scalable multi-robot collaboration with large language models: Centralized or decentralized systems?" arXiv preprint arXiv:2309.15943, 2023.

[42] K. Garg et al., "Large language models to the rescue: Deadlock resolution in multi-robot systems," arXiv preprint arXiv:2404.06413, 2024.

[43] J. Wang, G. He, and Y. Kantaros, "Safe task planning for language-instructed multi-robot systems using conformal prediction," arXiv preprint arXiv:2402.15368, 2024.

[44] B. Yu et al., "Co-navgpt: Multi-robot cooperative visual semantic navigation using large language models," arXiv preprint arXiv:2310.07937, 2023.

[45] C. Baier and J.-P. Katoen, Principles of model checking. MIT press Cambridge, 2008.

[46] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," Formal methods in system design, vol. 19, pp. 291–314, 2001.

[47] F. Petroni et al., "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.

[48] A. Q. Jiang et al., "Mistral 7b," arXiv preprint arXiv:2310.06825, 2023.

[49] E. Kolve et al., "Ai2-thor: An interactive 3d environment for visual ai," arXiv preprint arXiv:1712.05474, 2017.

[50] M. Shridhar et al., "Alfred: A benchmark for interpreting grounded instructions for everyday tasks," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 10 740–10 749.

[51] J. Achiam et al., "Gpt-4 technical report," arXiv preprint arXiv:2303.08774, 2023.

[52] V. Kurtz and H. Lin, "Temporal logic motion planning with convex optimization via graphs of convex sets," IEEE Transactions on Robotics, 2023.