Environment-Driven and LLM-Guided Multi-Robot Task Inference and Allocation Under Temporal Logic Specifications

Lin Li, *Graduate Student Member, IEEE*, Ziyang Chen, and Zhen Kan, *Senior Member, IEEE*

***Abstract*—In multi-robot systems, the successful execution of tasks typically depends on predefined instructions. However, existing approaches encounter substantial challenges in dynamic environments, particularly in autonomously reasoning, generating task instructions, and allocating tasks. These challenges are further exacerbated by the need to address complex temporal, spatial, and heterogeneous task constraints. To address these limitations, inspired by the success of the Large Language Models (LLMs) in natural language understanding and logical inference, this paper proposes an environment-driven and LLM-guided task inference and allocation framework with dual-system temporal logics. The framework consists of three key modules: the Environment Module, which employs environment LTL to continuous monitor and verify environmental resource constraints; the Inference Module, leveraging LLMs for autonomous generation and verification of robotic tasks in response to resource changes; and the Robot Module, which explores the feasible task allocation for the multi-robot system. When the resources in the environment do not satisfy the specification, the Inference Module is used to analyze and infer the feasible actions to be executed by the multi-robot system, so as to alleviate the environmental resource problem. Experimental results demonstrate the scalability, efficiency, and autonomy of our framework across varying task environments and robot configurations.**

***Note to Practitioners*—This work presents an environment-driven and LLM-guided framework for multi-robot task inference and allocation, providing practical guidance for deploying autonomous decision-making in dynamic domains such as logistics, manufacturing, and agriculture. We recognize that successful real-world implementation requires deep integration with existing robotic middleware and systematic handling of three key challenges: seamless coordination among heterogeneous platforms, mitigation of real-time reasoning disturbances caused by intermittent communications, and robustness against perception noise that may affect temporal logic translation. In our design, heterogeneous capabilities are quantitatively modeled to facilitate robot cooperation, while a cyclic environment verification and "Generation–Verification–Regeneration" closed loop mechanism are introduced to alleviate communication delays and interference. Furthermore, the framework's modular architecture, comprising continuous verification of environmental requirements, autonomous inference of task strategies, and optimized allocation among robot teams, ensures high flexibility and extensibility. This design allows practitioners to embed robustness enhancements (e.g., communication retry mechanisms, state estimation filters) and safety assurance modules according to specific deployment needs, without altering the core logical structure. Numerical and simulation studies have preliminarily validated the scalability of the framework in large-scale task settings. Beyond a theoretical construct, this work provides a transferable methodological foundation for building large-scale, environment-aware, and cooperatively intelligent multi-robot systems.**

***Index Terms*—Linear temporal logic, large language models, multi-robot task allocation, task inference.**

I. Introduction

TASKS performed by multi-robot systems are typically carried out in two stages: task instruction publication and task execution. For example, robots engaged in tasks such as search and rescue [1], urban traffic [2], or agricultural production [3] must execute actions according to the instructions they receive. However, in practical multi-robot task scenarios, explicit instructions are often unavailable or manually generated by humans on a case-by-case basis, leading to inefficient task execution. Imagine a chemical laboratory scenario, where a specific workspace must consistently maintain an adequate supply of instruments and reagents. When supplies fall below the required levels, task instructions must be sent to the robots, guiding them to take actions that restore supplies to the required level. In such a scenario, humans typically struggle to monitor supply statuses in real time, resulting in delays in issuing instructions. Furthermore, these instructions must adapt to varying conditions, such as supply shortages or surpluses, necessitating diverse temporal, spatial, and heterogeneous tasks. Therefore, a major challenge for multi-robot systems is autonomously generating robot instructions based on complex and dynamic environmental states, guiding robots to perform corresponding actions to ensure that the environment continuously meets specific thresholds. Motivated by this challenge, this work proposes an environment-driven and LLM-guided multi-robot task inference and allocation framework, aiming to autonomously generate instructions based on environmental states and guide the robots to continuously meet environmental requirements.

One of the most direct methods of transmitting instructions is through natural language. Previous works have parsed task instructions expressed in natural language into low-level robot operations [4], [5]. To enhance the parsing capability of natural language instructions for complex tasks, recent studies have integrated Large language models (LLMs) with environmental perception [6], [7], [8], enabling the generation of low-level

Received 27 January 2025; revised 17 October 2025; accepted 26 January 2026. Date of publication 28 January 2026; date of current version 5 February 2026. This article was recommended for publication by Associate Editor F. Pascucci and Editor T. Nishi upon evaluation of the reviewers' comments. This work was supported in part by the National Natural Science Foundation of China under Grant U25A20473 and Grant 62173314. *(Corresponding author: Zhen Kan.)*

The authors are with the Department of Automation, University of Science and Technology of China, Hefei, Anhui 230026, China (e-mail: zkan@ustc.edu.cn).

Digital Object Identifier 10.1109/TASE.2026.3659055

executable plans from vague or abstract high-level instructions while also offering dynamic adaptability. LLMs can also be used to convert natural language instructions into Planning Domain Definition Language (PDDL) [9], providing a natural language interface for robot task planning based on LLMs. On the other hand, task instructions may involve spatial, temporal, and heterogeneous attributes. Specifically, spatial attributes refer to the spatial constraints imposed on tasks, temporal attributes denote tasks that must be executed in a specific temporal sequence, and heterogeneous attributes pertain to tasks requiring robots with specialized capabilities. However, natural language descriptions of such complex tasks often lack precision and conciseness, potentially leading to ambiguities. To address this, we adopt temporal logic to represent complex robotic tasks. Due to their expressiveness and compactness, temporal logics are widely used to describe both temporal and spatial task instructions for robots.

Linear temporal logic (LTL) [10] leverages Boolean and temporal operators to effectively express tasks with a specified temporal order. For instance, LTL has been used to represent long-horizon manipulation tasks [11], or to guide robot path planning [12]. In multi-robot systems, LTL is also widely applied in various applications [13], [14], [15]. Furthermore, temporal logic can be extended to capture additional attributes and features of task requirements. For example, LTL has been extended to cLTL to incorporate the number of robots involved in multi-robot tasks [16], while the incorporation of task batches into LTL can describe instructions with internal constraints among tasks [17], [18]. Additionally, transformation the "flat" LTL representation into a hierarchical form can enhance expressive power of temporal logic [19]. However, the temporal logic approaches described above primarily focus on robotic tasks, overlooking the temporal requirements related to the dynamic nature of the environment. To address this limitation, our work introduces the dual-system temporal logics that explicitly decouple the environment and the robotic system. More specifically, the proposed dual-system temporal logics contain the environment LTL and the robot LTL. The environment LTL specifies temporal properties of resource requirements, such as "the resource r_1 in the task region a is eventually always no more than x_1, while the resource r_2 in the task region b is always no less than x_2". Robot LTL, on the other hand, describes the tasks that the robots should execute, which contain the temporal, spatial and heterogeneous information. For example, "a team of robots with individual capability c, where the required total sum of this capability c is v, is required to visit the task region b before visiting the task region a". Through this explicit separation, the dual-system framework provides a closed-loop formulation that jointly characterizes environmental requirements and robotic responses, thereby enhancing both expressiveness and adaptability compared to existing single-system logics.

Enabling accessible and interpretable temporal logic instructions is critical to ensuring the successful execution of tasks. To autonomously generate temporal logic task instructions, numerous studies have focused on converting natural language into temporal logic representations [20], [21], [22]. However, these methods are limited in handling complex instruments and require extensive task-specific datasets and manually designed features to accurately parse natural language into LTL. LLMs such as GPT-3/4, with the robust logical reasoning abilities, have been used in many studies for generating temporal logic tasks. For instance, LLMs have been combined with data augmentation techniques to translate natural language instruments into LTL formulas, optimizing the translation model through fine-tuning while minimizing reliance on manually annotated data [23], [24], [25]. Similarly, Lang2LTL is proposed to leverage LLMs for decomposing the transition process from natural language to LTL [26]. Moreover, Cook2LTL has been introduced, combining LLMs with a dynamic catching mechanism to parse cooking steps in natural language into LTL formulas [27]. Yet, these methods rely on human-provided natural language instructions as input to LLMs for generating corresponding LTL formulas, resulting in limited autonomy for the task generation. To bridge this gap, this work proposes an environment-driven and LLM-guided task inference and allocation framework that guarantees the autonomous generation of tasks based on the environmental states, with verification and exploration capabilities for dual-system temporal logics.

Based on the dual-system temporal logics, our proposed environment-driven and LLM-guided task inference and allocation framework contains three module: 1) Environment Module; 2) Inference Module; and 3) Robot Module. The Environment Module, a resource-guided process, is used to verify whether the environment LTL is satisfied, which encodes the environment LTL and environmental information into a resource planning tree. If the resource requirements are not satisfied, the Inference Module, which consists of two parts - task generation and task verification - will be triggered. LLM is leveraged in task generation to generate a robot LTL and verify whether the generated LTL formula can resolve the current environmental resource issues in task verification, thus ensuring the fulfillment of the corresponding sub-task. The feasible formula is then passed to the Robot Module, which is responsible for solving the corresponding allocation scheme to determine which robots should be selected to execute the tasks, what capabilities those robots should possess, and the corresponding values for those capabilities. The robots follow the allocation scheme to execute the generated robot LTL, providing feedback on resource changes to both environmental and robotic systems until the environment LTL is satisfied. To solve the dual-system temporal logics, we adopt a decoupled and incremental tree-search mechanism. The global problem is decomposed into high-level temporal verification (environment LTL) and local task allocation (robot LTL), which are addressed through a dual-tree structure, namely, the resource planning tree and the robotic planning tree, thus avoiding the state-space explosion and prohibitive computational cost of formulating and solving a monolithic global optimization problem. The resource planning tree expands nodes along feasible automaton transitions to verify resource conditions. When violations occur, the Inference Module is triggered to generate and validate robot tasks. These validated tasks are then incrementally solved on the robotic planning tree as smaller-scale allocation problems. This "explore-while-

allocating" structure enables the system to react adaptively. Instead of recomputing a global plan from scratch under environmental changes, our proposed closed-loop process ensures the satisfaction of the environmental requirements, even under changing resource conditions.

The main contributions are summarized as follows:

- We introduce the dual-system temporal logics that decouple the environment LTL and the robot LTL, enabling the modeling of the dynamic environment-robot interaction at a higher level of abstraction and thereby enhancing the expressiveness of task specifications.
- We propose an environment-driven and LLM-guided task inference and allocation framework, which enables the system to independently generate LTL-based task instructions, enhancing both flexibility and autonomy of continuously validating the environment and guiding robots to restore environmental resources.
- We design a decoupled and incremental tree-search mechanism that effectively mitigates the state-space explosion problem inherent in traditional temporal-logic planning, improving both efficiency and scalability.
- Numerical and simulation experiments demonstrate the scalability, effectiveness and autonomy of our method.

II. Preliminaries and Problem Formulation

In this section, we first review relevant preliminaries and then formulate the problem to be addressed. Denote by $\mathbb{N}$ and $[N]$ the set of integers and the shorthand notation for $\{1,\dots,N\}$, respectively. For a sequence $\mathbf{s} = s_0 s_1 \dots$, we define $\mathbf{s}[j\dots] = s_j s_{j+1} \dots$ as the subsequence starting at index j, and $\mathbf{s}[\dots j]$ as the subsequence from the beginning up to and including s_j. For a set A, $|A|$ denotes its cardinality, and 2^A represents its power set. Given two sets A and B, the set difference $A\backslash B = \{x | x \in A, x \notin B\}$ denotes the set of elements that belong to A but not to B.

A. Environment and Multi-Robot Systems

1) Environment System: Consider an environment $\mathcal{M}$ consisting of n_t non-overlapped task regions $\mathcal{O} = \{o_1, o_2, \dots, o_{n_t}\}$ and non-task regions $\mathcal{O}'$. Let o_0 be the void task region. The environment also contains n_r types of resources, collectively represented by the set $Res = \{res_1, res_2, \dots, res_{n_r}\}$. Each task region is characterized by a tuple $o_i = (p_i^o, \mathbf{r}_i, \mathcal{R}_i)$, where p_i^o denotes the center of o_i, $\mathbf{r}_i = \{r_i^{res_j}\}_{i\in n_t, res_j \in Res_i}$ represents the current amounts of resources $res_j \in Res_i$, where $Res_i \subseteq Res$ is the set of all available resource types in o_i. Let $\mathcal{R}_i : Res_i \to \mathbb{R}$ be a function mapping each resource type to its corresponding threshold. Specifically, if $\mathcal{R}_i(res_j) < 0$, o_i requires no more than $|\mathcal{R}_i(res_j)|$ of the resource res_j. If $\mathcal{R}_i(res_j) > 0$, the amount of res_j in o_i must meet or exceed $\mathcal{R}_i(res_j)$. If $\mathcal{R}_i(res_j) = 0$, no specific requirement exists for res_j in o_i.

2) Multi-Robot System: There exists a multi-robot system $\mathcal{A}$ operating within the environment $\mathcal{M}$. Consider a set of n_a robots, denoted as $\mathcal{A} = \{a_1, a_2, \dots, a_{n_a}\}$, which is categorized into n_c classes $C = \{c_1, c_2 \dots, c_{n_c}\}$ based on their functional capabilities such as carrying, grasping or detecting. Each robot $a_i \in \mathcal{A}$ is represented as a tuple $(p_i, c_i, cap_i)_{i\in n_a}$, where p_i denotes its position, c_i represents its class and cap_i indicates its capability value. To describe the robot allocation scheme, we define $alloc = [l_1, l_2, \dots, l_{n_a}]$, a vector of n_a binary variables, where $l_i = 1$ indicates that a_i is assigned a task and $l_i = 0$ indicates otherwise.

For clarity, in the following sections, we will adopt data structures to represent the attributes of the multi-robot and the environment systems. For instance, the position of a_i is denoted as $a_i.p$, while the resources available in the task region o_i are $o_i.\mathbf{r}$.

B. Temporal Logic Specifications

Linear Temporal Logic (LTL) consists of a set of atomic propositions $\mathcal{AP}$, boolean operators $\neg$ (negation), $\wedge$ (conjunction) and $\vee$ (disjunction), and temporal operators $\bigcirc$ (next), $\cup$ (until), $\Diamond$ (eventually) and $\Box$ (always). An LTL formula is thus defined over $\mathcal{AP}$ and operators as follows:

$$\varphi ::= true \mid \pi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \cup \varphi_2$$

where $\pi \in \mathcal{AP}$ is an atomic proposition, *true* is the boolean value, and $\varphi_1, \varphi_2 \in \phi$ are LTL formulas, where ϕ is the set of LTL formulas. An infinite word over the alphabet $\Sigma = 2^{\mathcal{AP}}$ is defined as $\sigma = \sigma_0\sigma_1 \dots \in (2^{\mathcal{AP}})^\omega$, where ω denotes an infinite repetition and $\sigma_i \in 2^{\mathcal{AP}}$, $\forall i \in \mathbb{N}$. The semantics of LTL formulas are defined over the word σ as follows [10],

- $\sigma \models true$
- $\sigma \models \pi \Leftrightarrow \pi \in \sigma_0$
- $\sigma \models \varphi_1 \wedge \varphi_2 \Leftrightarrow \sigma \models \varphi_1$ and $\sigma \models \varphi_2$
- $\sigma \models \varphi_1 \vee \varphi_2 \Leftrightarrow \sigma \models \varphi_1$ or $\sigma \models \varphi_2$
- $\sigma \models \neg\varphi \Leftrightarrow \sigma \not\models \varphi$
- $\sigma \models \bigcirc\varphi \Leftrightarrow \sigma[1\dots] \models \varphi$
- $\sigma \models \varphi_1 \cup \varphi_2 \Leftrightarrow \exists i, j \in \mathbb{N}$ s.t. $\sigma_i \models \varphi_2$, $\forall j \in [0, i)$, $\sigma_j \models \varphi_1$

where the satisfaction relation is denoted as $\models$. An LTL formula φ can be translated to a Nondeterministic Büchi Automaton (NBA) [28], which is defined as a tuple $\mathcal{B}(\varphi) = (\mathcal{S}, \mathcal{S}^0, \Sigma, \delta, \mathcal{S}^F)$, where $\mathcal{S}$ is the set of states, $\mathcal{S}^0$ is the set of initial states, $\delta \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is the transition relation and $\mathcal{S}^F$ is the set of accepting states. An infinite run $\mathbf{s}^{\mathcal{B}} = s_0 s_1 \dots$ of $\mathcal{B}(\varphi)$ generated by an infinite word σ satisfies $s_0 \in \mathcal{S}^0$ and $(s_i, \sigma_i, s_{i+1}) \in \delta$, $\forall i \in \mathbb{N}$. $\mathbf{s}^{\mathcal{B}}$ is called accepting if $\mathbf{s}^{\mathcal{B}}$ intersects with $\mathcal{S}^{\mathcal{F}}$ infinitely many times.

C. Dual-System Temporal Logics

This section introduces the dual-system temporal logics, which consists of the environment LTL $\varphi^{\mathcal{M}}$ for representing the temporal resource requirements in the environment and the robot LTL $\varphi^{\mathcal{A}}$ for describing spatial, temporal, and heterogeneous tasks of the multi-robot system.

The resource requirements in the task regions can be described by $\varphi^{\mathcal{M}}$, and the propositions $\pi^{\mathcal{M}} \in \Sigma^{\mathcal{M}}$ in $\varphi^{\mathcal{M}}$ are grounded to specific task regions. For example, $\pi_i^{\mathcal{M}}$ represents the resource thresholds to be satisfied in the task region o_i. Given a collection of resource states $\mathbf{r}(t) = \{\mathbf{r}_1(t), \mathbf{r}_2(t), \dots, \mathbf{r}_{n_t}(t)\}$ at time t, where $\mathbf{r}_i(t)$ represents resource states in the task region o_i at time t. The qualitative semantics of $\varphi^{\mathcal{M}}$ are defined over $\mathbf{r}(t)$ in task regions and the satisfaction of $\pi_i^{\mathcal{M}}$ at time t is defined as

$$\mathbf{r}(t) \models \pi_i^{\mathcal{M}} \Leftrightarrow \forall res_j \in Res_i,$$

$$\Big((\mathcal{R}_i(res_j) < 0 \Rightarrow r_i^{res_j} \leq |\mathcal{R}_i(res_j)|) \wedge (\mathcal{R}_i(res_j) > 0 \Rightarrow r_i^{res_j} \geq \mathcal{R}_i(res_j))\Big). \quad (1)$$

We denote by $\mathbf{r}(0) \models \varphi^{\mathcal{M}}$ if the sequence of resource states satisfies the environment LTL $\varphi^{\mathcal{M}}$.

Example 1: Suppose there exist two task regions $\mathcal{O} = \{o_1, o_2\}$ and two types of resources $Res = \{r_1, r_2\}$, where $\mathcal{R}_1(r_1) = 3$, $\mathcal{R}_1(r_2) = -2$, $\mathcal{R}_2(r_1) = -3$ and $\mathcal{R}_2(r_2) = 2$. An environment LTL $\varphi^{\mathcal{M}} = \square\Diamond\pi_1^{\mathcal{M}} \wedge \square\Diamond\pi_2^{\mathcal{M}}$ indicates that the task region o_1 requires the amount of r_1 to be no less than 3 and r_2 to be no more than 2, similar for the task region o_2.

The robot LTL $\varphi^{\mathcal{A}}$ consists of propositions $\pi^{\mathcal{A}} \in \Sigma_{\mathcal{A}}$, where $\pi^{\mathcal{A}} \triangleq (\pi, \{c_i, cap_i\}_{i \in n_c})$ is defined similarly to the cLTL [16] and CaTL [29]. $\pi^{\mathcal{A}}$ is true if there exist at least capabilities $\{c_i, cap_i\}_{i \in n_c}$ possessed by the robots in the task region corresponding to π. Specifically, for an individual robot a_k with the trajectory $\boldsymbol{\sigma}_k = \sigma_k^0 \sigma_k^1 \ldots$, where σ_k^t is the trajectory of a_k at step t, $\sigma_k^t \models \pi$ if and only if $\pi \in \boldsymbol{\sigma}_k[\ldots t]$. The collection of trajectories about all robots at time t is thus represented as $(\tau_{\mathcal{A}}, t) = (\{\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \ldots, \boldsymbol{\sigma}_{n_a}\}, t)$. For each atomic proposition corresponding to a task region in the environment, we introduce a mapping $\mathcal{L}_{\mathcal{M}} : \Sigma^{\mathcal{M}} \cup \Sigma^{\mathcal{A}} \to \mathcal{M}$, where $\pi^{\mathcal{M}} \in \Sigma^{\mathcal{M}}$ and $\pi^{\mathcal{A}} \in \Sigma^{\mathcal{A}}$ capture all task regions under $\mathcal{L}_{\mathcal{M}}(\pi^{\mathcal{M}}) = o_1, o_2, \ldots, o_{n_t}$ and $\mathcal{L}_{\mathcal{M}}(\pi^{\mathcal{A}}) = o_1, o_2, \ldots, o_{n_t}$. Therefore, the satisfaction of $\pi^{\mathcal{A}}$ is defined as

$$(\tau_{\mathcal{A}}, t) \models \pi^{\mathcal{A}} \Leftrightarrow \left(\sum_{i \in n_a} \{a_i.cap_j | \sigma_i^t \models \pi\} \geq \pi^{\mathcal{A}}.cap_j\right)_{j \in n_c}$$

For simplicity of description, we elicit the task state $q \in \mathcal{Q}$, which is the abstract expression of the task, where $\mathcal{L} : \mathcal{Q} \to \Sigma^{\mathcal{M}}$ maps the task states to the environment proposition. Similarly, the robot system tasks are abstracted as $\mathcal{Q}^{\mathcal{A}}$ and $\mathcal{L}_{\mathcal{A}} : \mathcal{Q}^{\mathcal{A}} \to \Sigma^{\mathcal{A}}$ maps the robotic task states to its corresponding propositions.

Remark 1: Existing extensions such as cLTL and CaTL are essentially *single-system logics* that focus on the execution layer of robotic tasks. They extend LTL or STL to directly prescribe collective robot behaviors, where cLTL emphasizes the "counting" for robots and CaTL emphasizes robot "capabilities". In these formalisms, "resources" are intrinsic properties of robots. In contrast, our dual-system temporal logics deliberately decouple the problem into environment LTL, which specifies temporal persistence requirements of environment resources (problem definition), and robot LTL, which encodes spatio-temporal and heterogeneous demands of robots in response (solution definition). This separation enables a closed-loop representation of the environment-robot interaction, rather than constraining robots alone.

Problem 1: Given a heterogeneous multi-robot system $\mathcal{A}$, an environment LTL $\varphi^{\mathcal{M}}$ with the resource requirement, and the environment $\mathcal{M}$, the goal is to develop a task inference and allocation framework to autonomously infer and verify feasible robot LTL formulas $\varphi^{\mathcal{A}}$ if $\varphi^{\mathcal{M}}$ is not satisfied, and solve their task allocation and execution schemes to meet $\varphi^{\mathcal{M}}$.

III. Environment-Driven and LLM-Guided Task Inference and Allocation Synthesis

This section proposes an environment-driven and LLM-guided task inference and allocation synthesis to address Problem 1, which contains three main components: (i) **Environment Module**: An environment-driven module that constantly verifies whether the environment LTL is satisfied currently and determines whether the Inference Module needs to be triggered. (ii) **Inference Module**: A main component for task inference, encompassing the task generation and task verification, which generates executable robot LTL. (iii) **Robot Module**: A multi-robot task allocation module that adheres to the robot LTL generated by the Inference Module, efficiently solves the corresponding allocation schemes.

A. Environment Module

The Environment Module constantly and periodically verifies whether the resource requirements are met. Note that the environmental resources in this work need to meet temporal requirements such as $\square$ and $\Diamond$ as in Example 1. Thus, its satisfaction cannot be simply checked by the resource values at a given time in a particular task region. Temporal logic offers a concise way to describe tasks with temporal constraints. However, the resources can be time-varying and automaton-based model checking methods are generally computationally expensive and thus not effective in continuous monitoring of environmental resources. Tree search [30], on the other hand, has demonstrated significant potential for efficiently expanding and exploring temporal logic planning problems.

Inspired by the tree search methods, a resource planning tree $Tree^{\mathcal{M}}$ is designed for the verification of $\varphi^{\mathcal{M}}$ in the requirements. To track the progress of $\varphi^{\mathcal{M}}$, we convert it into an NBA $\mathcal{B}(\varphi^{\mathcal{M}})$ via translation tools in [28] and encode the automaton states in $\mathcal{B}(\varphi^{\mathcal{M}})$ to $Tree^{\mathcal{M}}$.

Definition 1: The resource planning tree is defined as $Tree^{\mathcal{M}} = (\mathcal{V}^{\mathcal{M}}, \mathcal{E}^{\mathcal{M}}, J^{\mathcal{M}})$, where $\mathcal{V}^{\mathcal{M}} = \{v_0^{\mathcal{M}}, v_1^{\mathcal{M}}, \ldots\}$ is the set of nodes with $v_0^{\mathcal{M}}$ being the root node of $Tree^{\mathcal{M}}$, $\mathcal{E}^{\mathcal{M}} \subset \mathcal{V}^{\mathcal{M}} \times \mathcal{V}^{\mathcal{M}}$ is the set of edges which reflects the parent-child relationships among nodes and the edge $(v_a^{\mathcal{M}}, v_b^{\mathcal{M}}) \in \mathcal{E}^{\mathcal{M}}$ if $v_a^{\mathcal{M}}$ is the parent node of $v_b^{\mathcal{M}}$, and $J^{\mathcal{M}} : \mathcal{V}^{\mathcal{M}} \to \mathbb{R}_{\geq 0}$ is the cost from the root node to the current node. The node in $Tree^{\mathcal{M}}$ is defined as $v^{\mathcal{M}} = (s, q, g)$, where $s \in \mathcal{B}(\varphi^{\mathcal{M}}).\mathcal{S}$ is the automaton states in $\mathcal{B}(\varphi^{\mathcal{M}})$, $q \in \mathcal{Q}$ is the task state and $g \in \mathcal{G}$ is the state stages of $v^{\mathcal{M}}$.

Note that an accepting run $\mathbf{s}^{\mathcal{B}} = s_0 s_1 \ldots$ of an NBA can be divided into a prefix part $\mathbf{s}_{pre}$ and an infinite suffix parts $\mathbf{s}_{suf}$, which can be represented as $\mathbf{s}^{\mathcal{B}} = \mathbf{s}_{pre}[\mathbf{s}_{suf}]^{\omega}$. Drawing from this prefix-suffix structure of an accepting run in a Büchi automaton, the state stages of nodes associated with automaton states can thus be divided into a prefix and many suffix stages, with a final stage *end* designated to terminate the verification process of $\varphi^{\mathcal{M}}$. These stages can be denoted as $\mathcal{G} = \{pre, suf^1, suf^2, \ldots, suf^{n_s}, end\}$, where n_s ($n_s \geq 2$) is the number of suffixes to detect. The prefix state stage is used to verify whether the environmental system admits a feasible trajectory leading to an accepting state, i.e., whether there

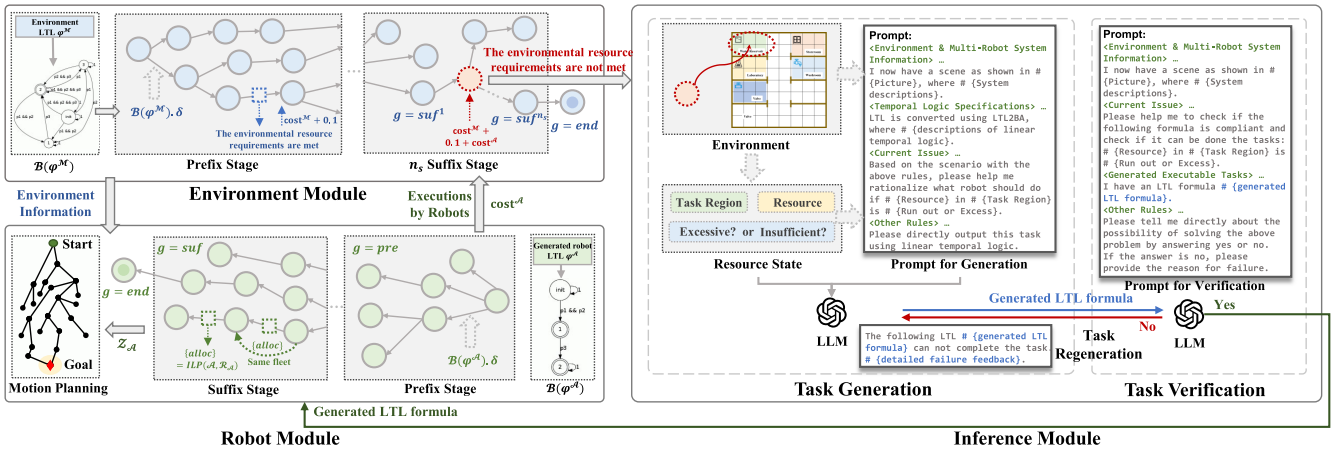

Fig. 1. The framework of environment-driven and LLM-guided task inference and allocation synthesis, which contains three main components: the Environment Module, the Inference Module and the Robot Module. The Environment Module continuously verifies resource conditions based on the environment LTL formula. When resources fail to meet the threshold requirements, the Inference Module is triggered to generate robot-executable tasks. Then, the Robot Module allocates these tasks and executes them, subsequently feeding the resource changes back to the Environment Module.

exists a feasible verification path that satisfies the environment LTL formula. Once an accepting state is reached, the system transitions into the suffix stage, where cyclic visits to the accepting state are verified for a finite number of iterations. In this way, the corresponding environmental resources specified in the environment LTL formula can be examined under cyclic access. Considering the real-world systems are subject to computational resource limitations and task deadlines, it is infeasible to perform infinite detection of environmental resource states. Therefore, we introduce a termination state stage *end* as a controllable stopping condition for environmental detection. When the n_sth suffix stage is completed, the state stage reaches *end*, indicating that the specified number of detection iterations has been completed and marking the termination of the entire process.

$Tree^{\mathcal{M}}$ provides an intuitive representation between resource states and task processes, while it expands according to the feasible transition in the automaton. For example, the node $v_a^{\mathcal{M}} = (s, q, g)$ in $Tree^{\mathcal{M}}$ indicates the task region $\mathcal{L}_{\mathcal{M}}\big(\mathcal{L}(v_a^{\mathcal{M}}.q)\big)$, where the resources and their threshold function are $\mathcal{L}_{\mathcal{M}}\big(\mathcal{L}(v_a^{\mathcal{M}}.q)\big).r$ and $\mathcal{L}_{\mathcal{M}}\big(\mathcal{L}(v_a^{\mathcal{M}}.q)\big).\mathcal{R}$, respectively. By continuously expanding and traversing $Tree^{\mathcal{M}}$, it is possible to determine whether resources in the task region corresponding to the task state of each node satisfy required thresholds, as detailed in Alg. 1.

To begin with constructing $Tree^{\mathcal{M}}$, we initialize the root node as $v_0^{\mathcal{M}} = (s_0, q_0, g_0)$, where $s_0 \in \mathcal{B}(\varphi^{\mathcal{M}}).\mathcal{S}^0$, $q = q_0$, $\mathcal{L}_{\mathcal{M}}(\mathcal{L}(q_0)) = o_0$ and $g_0 = pre$ (line 2). For all nodes in $Tree^{\mathcal{M}}$, we classify them into two sets: the traversed set $TraTree^{\mathcal{M}}$ and the untraversed set $Tree^{\mathcal{M}} \backslash TraTree^{\mathcal{M}}$. The nodes in the untraversed set will be explored and then extended to its corresponding branches. Suppose the current node $v^{\mathcal{M}}$ is the parent node which may have multiple child nodes. We denote the set of its child nodes as $SubTree^{\mathcal{M}}$, which is initialized as an empty set (line 5). To expand $SubTree^{\mathcal{M}}$, we first define $\Gamma(v^{\mathcal{M}})$ as the set of nodes in the branch from the root node to $v^{\mathcal{M}}$ that share the same state stage as $v^{\mathcal{M}}$. Additionally, let $\Gamma(v^{\mathcal{M}}).s$ represent the set of automaton

Algorithm 1 Environment Module

1 $\mathcal{B}(\varphi^{\mathcal{M}}) \leftarrow$ Büchi$(\varphi^{\mathcal{M}})$;
2 Construct $Tree^{\mathcal{M}} \leftarrow \{v_0^{\mathcal{M}}\}$;
3 **while** *true* **do**
4 **for** $v^{\mathcal{M}} \in Tree^{\mathcal{M}} \backslash TraTree^{\mathcal{M}}$ **do**
5 Initialize $SubTree^{\mathcal{M}} \leftarrow \emptyset$;
6 $v_{sub}^{\mathcal{M}} \leftarrow$ Expand_SubTree$(SubTree^{\mathcal{M}}, \mathcal{B}(\varphi^{\mathcal{M}}))$;
7 **while** CheckResource$(v_{sub}^{\mathcal{M}})$ **do**
8 $\mathcal{R}_{\mathcal{A}}, \varphi \leftarrow$ Inference_Module$(v_{sub}^{\mathcal{M}}, \mathcal{M}, \mathcal{P})$ // Alg. 2
9 $\mathcal{M}, \mathcal{A} \leftarrow$ Robot_Module$(\varphi, \mathcal{A}, \mathcal{M}, \mathcal{R}_{\mathcal{A}})$ // Alg. 3
10 Update $v_{sub}^{\mathcal{M}}$, $SubTree^{\mathcal{M}}$ and $\Gamma(v^{\mathcal{M}})$;
11 $SubTree^{\mathcal{M}} \leftarrow$ Prune$(SubTree^{\mathcal{M}})$;
12 $Tree^{\mathcal{M}}, TraTree^{\mathcal{M}} \leftarrow$ Expand$(SubTree^{\mathcal{M}}, v^{\mathcal{M}})$;

states corresponding to the nodes in $\Gamma(v^{\mathcal{M}})$, which records the traversed automaton states under the same state stage. With $v^{\mathcal{M}}$ as the parent node, the untraversed automaton states in the current state stage and task states are traversed to find all feasible s' and q' such that $\big(v^{\mathcal{M}}.s, \mathcal{L}(q'), s'\big) \in \mathcal{B}(\varphi^{\mathcal{M}}).\delta$, where $s' \in \mathcal{B}(\varphi^{\mathcal{M}}).\mathcal{S} \backslash \Gamma(v^{\mathcal{M}}).s$. Thus, the found s' and q' are the automaton state and task state of the child node $v_{sub}^{\mathcal{M}} \in SubTree^{\mathcal{M}}$, respectively. This ensures that the edges in $Tree^{\mathcal{M}}$ are generated according to the feasible transition in $\mathcal{B}(\varphi^{\mathcal{M}})$, so that each brach satisfies the temporal requirements in the environment. In other words, the child nodes satisfy $v_{sub}^{\mathcal{M}}.q = q'$, $v_{sub}^{\mathcal{M}}.s = s'$, $(v^{\mathcal{M}}, v_{sub}^{\mathcal{M}}) \in \mathcal{E}^{\mathcal{M}}$, and $\big(v^{\mathcal{M}}.s, \mathcal{L}(v_{sub}^{\mathcal{M}}.q), v_{sub}^{\mathcal{M}}.s\big) \in \mathcal{B}(\varphi^{\mathcal{M}}).\delta$.

While ensuring temporal requirements, we need to determine whether the current resources in the task region $\mathcal{L}_{\mathcal{M}}(v_{sub}^{\mathcal{M}}.q)$ meet the required resource thresholds. Specifi-

cally, for each child node $v_{sub}^{\mathcal{M}}$, `CheckResource` (line 7) means that the resource $\mathcal{L}_{\mathcal{M}}\big(\mathcal{L}(v_{sub}^{\mathcal{M}}.q)\big).r$ in the corresponding task region must meet the specified threshold requirements $\mathcal{L}_{\mathcal{M}}\big(\mathcal{L}(v_{sub}^{\mathcal{M}}.q)\big).\mathcal{R}$ in (1). If the current resources satisfy the specified thresholds, the process proceeds, and this child node is assigned a cost value for verification in the Environment Module. Otherwise, the Inference Module is invoked to generate appropriate tasks for the multi-robot system to fulfill the resource requirements (line 8), as specified in Section III-B. Subsequently, the Robot Module allocates and executes tasks based on the generated tasks from the Inference Module, while updating the environment and the robotic systems to reflect changes in resource status (line 9), as detailed in Section III-C.

For each extended child node in $Tree^{\mathcal{M}}$, its corresponding total cost refers to the accumulate exploration time spent starting from the root node to the current child node along the branch. Assuming that the cost of each verification and exploration step in Environment Module is $c_v \in \mathbb{R}^+$, the cost of each child node is defined as

$$J^{\mathcal{M}}(v_{sub}^{\mathcal{M}}) = \begin{cases} J^{\mathcal{M}}(v^{\mathcal{M}}) + c_v, & \mathcal{M} \models \mathcal{L}(v_{sub}^{\mathcal{M}}.q) \\ J^{\mathcal{M}}(v^{\mathcal{M}}) + c_v + J^{\mathcal{A}}(v_{sub}^{\mathcal{M}}), & \text{else}, \end{cases} \tag{2}$$

where $J^{\mathcal{A}}$ is the cost of performing tasks by the robots. As indicated in (2), no actions are required if the current resources satisfy the task state, and thus the cost of this node is only the time spent on tree expansion. Once the resources fail to meet their thresholds, robots need to restore the resources. Hence, the cost of this node includes not only the time spent on tree expansion, but also the additional time spent by robots to allocate tasks.

Next, the state stage of the child node is determined using the state updating rule

$$f_g(g, s) = \begin{cases} g, & s \notin \mathcal{B}.\mathcal{S}^F, \\ suf^1, & s \in \mathcal{B}.\mathcal{S}^F, g = pre, \\ \dots & \dots \\ suf^{n_s}, & s \in \mathcal{B}.\mathcal{S}^F, g = suf^{n_s-1}, \\ end, & g = suf^{n_s}. \end{cases} \tag{3}$$

The state updating rule (3) indicates that the state stage of a child node is related to its automaton state and the state stage of its parent node. If the automaton state of the child node is not an accepting state, its state stage remains the same as the parent node. If the automaton state is an accepting state, its state stage is updated according to the current state stage of its parent node. Specifically, when the automaton state of the child node is an accepting state and the state stage of the parent node is pre, it indicates that the child node has transitioned to the next state stage, suf^1. Similarly, we can infer the state stage of the child node based on the relationship between the current child node and its parent node. When its parent node's stage is suf^{n_s}, indicating the final stage to be explored, the child node's stage will be end, signifying the completion of the continuous validation and exploration process.

By the stage updating rule (3), the state stage of $v_{sub}^{\mathcal{M}}$ is updated according to

$$v_{sub}^{\mathcal{M}}.g = f_g(v^{\mathcal{M}}.g, v_{sub}^{\mathcal{M}}.s) \tag{4}$$

Also, the elements in $\Gamma(v^{\mathcal{M}})$ are updated by

$$\Gamma(v^{\mathcal{M}}) = \begin{cases} \Gamma(v^{\mathcal{M}}) \cup \{v_{sub}^{\mathcal{M}}\}, & v_{sub}^{\mathcal{M}}.g = v^{\mathcal{M}}.g, \\ \emptyset, & \text{else}. \end{cases} \tag{5}$$

If $v_{sub}^{\mathcal{M}}$ belongs to the same state stage as its parent node $v^{\mathcal{M}}$, this child node $v_{sub}^{\mathcal{M}}$ is added to $\Gamma(v^{\mathcal{M}})$. However, if it does not belong to the same state stage but steps to the next stage, $\Gamma(v^{\mathcal{M}})$ will be reset to an empty set to initiate the exploration of a new state stage. Besides, each child node is added to $SubTree^{\mathcal{M}}$ (line 10).

Once all possible child nodes have been explored, the set of children can be further pruned to minimize subsequent redundant exploration (line 11). There are three pruning principles: (i) Cost-Based Pruning: Nodes with the same automaton state and the same state stage are pruned away from nodes with greater costs. (ii) Termination Node Pruning: Child nodes are retained until their state stage reaches end. And (iii) Repeated State Pruning: Nodes with the automaton states already traversed in $\Gamma(v^{\mathcal{M}}).s$ and an unchanged state stage are pruned.

The pruned $SubTree^{\mathcal{M}}$ and the current node are added to $Tree^{\mathcal{M}}$ and the traversed set, respectively (line 12). The environment validation will continue until the state stage of $Tree^{\mathcal{M}}$ reaches end, which means the tree search have completed the manually specified number of detection cycles.

It is worth noting that the concept of state stages in our framework is derived from the theoretical structure of accepting runs in the Büchi automaton corresponding to the environment LTL, which is an algorithmic control mechanism tailored for finite-horizon verification. Their primary purpose is to support node expansion, pruning, and termination decisions during the tree search process. According to the standard semantics of Büchi automaton, the first arrival at an accepting state marks the end of a prefix. However, under our framework, when an automaton state of a node in $Tree^{\mathcal{M}}$ is first accepting, its state stage is marked as the first suffix state stage. This "one-step-ahead" phenomenon facilitates efficient forward search without effecting the semantic correctness of the final path. When the traversal reaches the leaf node $v_{end}^{\mathcal{M}}$ whose state stage is end, the tree search in Environment Module terminates. Subsequently, a backtracking procedure is performed along the feasible path from $v_{end}^{\mathcal{M}}$, during which each node is reassigned to its actual semantic stage. Let $\Pi_g^{\mathcal{M}}$ denote the stage-labeled path of nodes in $Tree^{\mathcal{M}}$ whose actual stage is g and $v_0^{\mathcal{M}} \in \Pi_{pre}^{\mathcal{M}}$. The backtracking-based partition follows the rule:

$$\Pi_h^{\mathcal{M}} = \langle v_i^{\mathcal{M}} | parent(v_i^{\mathcal{M}}).g = h, v_i^{\mathcal{M}} \in Tree^{\mathcal{M}}, i \geq 1 \rangle, \tag{6}$$

where $parent(v_i^{\mathcal{M}})$ is the parent node of $v_i^{\mathcal{M}}$ and $h \in \{pre, suf^1, \dots, suf^{n_s}\}$. By executing this backtracking procedure, all nodes along the feasible path are reassigned into stage sets fully consistent with Büchi automaton theory, thereby ensuring semantic alignment.

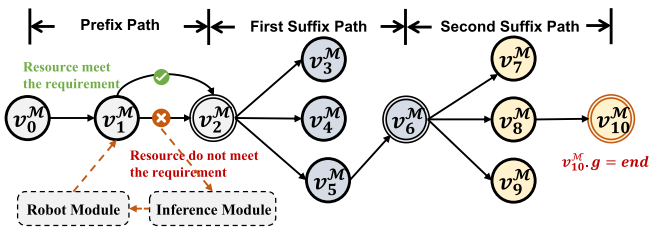

Fig. 2. The resource planning tree corresponding the environment LTL $\varphi^{\mathcal{M}} = \square\Diamond\pi_1^{\mathcal{M}} \wedge \square\Diamond\pi_2^{\mathcal{M}}$.

Example 2: Continuing with Example 1, assume $n_s = 2$ with the corresponding tree expansion process illustrated in Fig. 2. The root node is first initialized as $v_0^{\mathcal{M}} = (s_0, q_0, g_0)$, where $s_0 \in \mathcal{B}(\varphi)$, and $g_0 = pre$ indicates that the system is currently in the prefix stage. Next, according to the feasible state transition of the Büchi automaton $(v_0^{\mathcal{M}}.s, \mathcal{L}(q_1), s_1) \in \mathcal{B}(\varphi).\delta$, the task state and the automaton state of $v_1^{\mathcal{M}}$ are q_1 and s_1, respectively. The system then checks the resource state of the task region $\mathcal{L}_{\mathcal{M}}(\mathcal{L}(q_1))$: if the resources do not meet the thresholds specified by $\mathcal{R}_1$, the Inference Module is triggered to generate corresponding robotic tasks, which are subsequently allocated and executed by the Robot Module; if the resources already satisfy the thresholds, both the Inference and Robot Module are skipped, and the process proceeds directly to the next expansion step. The cost of the node is updated according to (2), while the state stage is updated by (3). Since $v_1^{\mathcal{M}}$ is not an accepting state, its state stage remains the same as that of its parent node, i.e., $v_1^{\mathcal{M}}.g = pre$. At this point, $v_1^{\mathcal{M}}$ is still in the prefix stage and can thus be added to $\Gamma(v_0^{\mathcal{M}})$. The node $v_1^{\mathcal{M}}$ is then added to the subtree. Its child node is generated in the same manner. When the automaton state of $v_2^{\mathcal{M}}$ is an accepting state, the prefix-stage process is completed, and its child nodes $v_3^{\mathcal{M}}$, $v_4^{\mathcal{M}}$ and $v_5^{\mathcal{M}}$ with the same automaton state s_1 enter the first suffix stage. The resources of $\mathcal{L}_{\mathcal{M}}(\mathcal{L}(q_1))$ are re-verified in the first suffix stage to determine whether the Inference Module and Robot Module should be triggered. Similarly, the costs and state stages of $v_3^{\mathcal{M}}$, $v_4^{\mathcal{M}}$ and $v_5^{\mathcal{M}}$ are updated accordingly, and each of them is added to the subtree of $v_2^{\mathcal{M}}$. Since these three child nodes share the same automaton state and state stage, the pruning principles ensure that only the node with the minimum cost (i.e., $v_5^{\mathcal{M}}$) is retained for subsequent expansion. When the expansion reaches node $v_{10}^{\mathcal{M}}$, its state stage is updated to *end*. According to the pruning principles, no further expansion is performed. The final environmental verification path is $v_0^{\mathcal{M}} v_1^{\mathcal{M}} v_2^{\mathcal{M}} v_5^{\mathcal{M}} v_6^{\mathcal{M}} v_9^{\mathcal{M}} v_{10}^{\mathcal{M}}$. Based on the backtracking formula (6), since the parent node $v_8^{\mathcal{M}}$ of $v_{10}^{\mathcal{M}}$ is in the second suffix stage, then $v_{10}^{\mathcal{M}} \in \Pi_{suf^2}$. The stage-labeled paths of remaining nodes can be recursively determined accordingly, i.e., $\Pi_{pre}^{\mathcal{M}} = \langle v_0^{\mathcal{M}}, v_1^{\mathcal{M}}, v_2^{\mathcal{M}} \rangle$, $\Pi_{suf^1} = \langle v_5^{\mathcal{M}}, v_6^{\mathcal{M}} \rangle$ and $\Pi_{suf^2} = \langle \pi_8^{\mathcal{M}}, \pi_{10}^{\mathcal{M}} \rangle$.

B. Inference Module

When the resources fail to meet the environment requirement, the Inference Module is required to infer tasks for the multi-robot system to restore the environment resources. LLMs, due to their extensive knowledge, can serve as a robot task planner. Therefore, we use LLMs as the core of this subsection, dividing the Inference module into three parts: task generation, task verification and task regeneration.

Algorithm 2 Inference Module

Input: $v_{sub}^{\mathcal{M}}$, $\mathcal{M}$, $\mathcal{P}$
Output: $\mathcal{R}_{\mathcal{A}}$, φ

```
φ ← TaskGeneration(v_sub^M, M, P_1);
while not TaskVerification(φ, M, P_2) do
    φ ← TaskRegeneration(φ, M, P_3);
R_A ← ComputeResource(M, v_sub^M);
```

An executable robot LTL can be decomposed into two parts: a traditional LTL formula and its required capabilities. With the continuous updates and iterations of LLMs, traditional LTL formulas can be inferred using current general-purpose LLMs such as GPT-4 [31] and Claude [32]. Inspired by the prompt engineering [33], we select *GPT-4o* for the Inference Module and design prompt examples for generating robotic tasks according to the environment states, while verifying and modifying them. Meanwhile, in the majority of tasks, environment resources are closely related to the capabilities equipped by robots. For example, the water in the environment are linked to the robots' capability to carry water. Therefore, the required capabilities can be determined based on the current resource states in the environment and their corresponding thresholds, i.e., the amount of resources required.

More specifically, the task generation component integrates the current environmental and robotic information into our designed prompts, utilizing both image and textual data to generate tasks that are executable by the multi-robot system. The task verification component then takes the executable tasks generated by the task generation component, along with system information and the syntactical rules, and incorporates them into prompts to determine whether this generated tasks can effectively address the issues related to the environmental resources. If the verification fails, the task regeneration is triggered by exploring previously failed tasks with its failure feedback and then generates a new task for re-verification. Together, "Generation–Verification–Regeneration" forms an LLM-driven closed-loop mechanism that continuously enhances task reasoning robustness through iterative generation, verification, and regeneration.

Remark 2: In this work, we choose to use prompt engineering rather than fine-tuning models like Llama3 [34]. This is because LLMs are evolving rapidly and already exhibit strong natural language understanding and temporal logic reasoning capabilities [35]. Fine-tuning models, on the other hand, are limited by the constraints of training data. Therefore, leveraging prompt engineering in combination with general-purpose LLMs provides scalability and adaptability in our work.

Alg. 2 outlines the mechanism of the Inference module, where the input $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$ represents the set of prompts. First, the designed task generation prompt $\mathcal{P}_1$, along with the current environment and node information, is input into the task generation component. The LLM uses environmental

image and textual information to generate an executable LTL formula φ for robotic system (line 1). Subsequently, the task verification component evaluates the generated formula φ along with the task verification prompt and current environmental information to determine whether φ can solve the current environmental resource issues (disregarding resource quantities).

Algorithm 3 Robot Module

Input: φ, $\mathcal{A}$, $\mathcal{M}$, $\mathcal{R}_{\mathcal{A}}$
Output: the updated $\mathcal{M}$, $\mathcal{A}$
1 $\varphi^{\mathcal{A}} \leftarrow \texttt{RoboticSystemLTL}(\varphi, \mathcal{R}_{\mathcal{A}})$;
2 $\mathcal{B}(\varphi^{\mathcal{A}}) \leftarrow \texttt{Buchi}(\varphi^{\mathcal{A}})$;
3 Construct $Tree^{\mathcal{A}} \leftarrow \{v_0^{\mathcal{A}}\}$;
4 **while** *true* **do**
5 **for** $v^{\mathcal{A}} \in Tree^{\mathcal{A}} \backslash TraTree^{\mathcal{A}}$ **do**
6 Initialize $SubTree^{\mathcal{A}} \leftarrow \emptyset$;
7 $v_{sub}^{\mathcal{A}} \leftarrow \texttt{Expand\_SubTree}(SubTree^{\mathcal{A}}, \mathcal{B}(\varphi^{\mathcal{A}}))$;
8 **if** $Fleet\_Check(v_{sub}, \mathcal{L}_F, Tree^{\mathcal{A}})$ **then**
9 $v_{sub} \leftarrow \texttt{Fleet\_Allocation}(Tree^{\mathcal{A}})$;
10 **else**
11 $v_{sub} \leftarrow \texttt{ILP}(\mathcal{A}, \mathcal{R}_{\mathcal{A}})$;
12 Update $v_{sub}^{\mathcal{A}}$, $SubTree^{\mathcal{A}}$ and $\Gamma(v^{\mathcal{A}})$;
13 $SubTree^{\mathcal{A}} \leftarrow \texttt{Prune}(SubTree^{\mathcal{A}})$;
14 $Tree^{\mathcal{A}}, TraTree^{\mathcal{A}} \leftarrow \texttt{Expand}(SubTree^{\mathcal{A}}, v^{\mathcal{A}})$;
15 $\{alloc\} \leftarrow \texttt{Retro\_Search}(Tree^{\mathcal{A}})$;
16 $\mathcal{M}, \mathcal{A} \leftarrow \texttt{RobotExecution}(\{alloc\}, \varphi^{\mathcal{A}})$;

If the verification passes, the process continues to the next step. If it fails, a task regeneration prompt, designed to include the failed task φ and failure reason, is input into the LLM. This prompt guides the LLM to generate a new task, and the process repeats until the generated formula φ passes the task verification component (lines 2-3). A validated robot system task from the Inference Module will be processed by the Robot Module for task allocation and planning. Before this, the required resources and their corresponding amount of adjustments ${\mathcal{R}_{\mathcal{A}}}^{Res} \in \mathcal{R}_{\mathcal{A}}$ are determined by

$$ {\mathcal{R}_{\mathcal{A}}}^{Res} = |r_i^{Res} - |\mathcal{R}_i(Res)||, \ \ \mathcal{L}_{\mathcal{M}}(\mathcal{L}(v_{sub}^{\mathcal{M}}.q)) = o_i, \tag{7} $$

which is taken as the input for Alg. 3.

Remark 3: Note that the robot LTL consists of $\pi^{\mathcal{A}}$, each of which can be further decomposed into traditional atomic proposition and its corresponding capability requirements. However, LLMs output the traditional LTL formula first, followed by solving for the required capabilities. This is because it is straightforward to deduce that, for addressing the environmental resource issues of the node in $Tree^{\mathcal{M}}$, the capability requirements for each proposition in the LTL formula generated by the LLM should be identical. Consequently, the entire LTL formula can be directly generated, and the corresponding capability requirements only need to be solved once. Furthermore, in the Robot Module, we will include a check to determine whether the sub-task requires the same fleet of robots to perform.

In the following, the prompts used in each part of Inference Module are detailed, where the workflow of Inference Module is illustrated through an example in Fig. 3.

1) Task Generation: To enable LLMs to effectively generate executable robot LTL formulas in response to environmental challenges, the task generation consists of two parts, task generation and task regeneration. The structure of prompts are similar in both parts, where the prompts consist of the following main components.

- **Task Background Descriptions**: The image information and textual descriptions of the structured environment for task execution.
- **Linear Temporal Logic Specifications**: Provide the syntax rules of LTL, present the output in LTL2BA format, and include an example of an LTL formula as the output.
- **Current Issue**: The descriptions of the resource problem to be solved by the current node. This content is automatically generated by filling a predefined text template with structured information (e.g., resource name, region, and states such as "insufficient" or "excessive") to create a qualitative, natural language description that provides the core problem context for the LLM.
- **Output Rules**: The desired output format for task generation. In the task regeneration, descriptions of the generated tasks and detailed failure feedback (e.g., syntactical issues or inability to complete tasks) that failed verification should also be included in this component. The generated LTL formula must be re-generated based on the aforementioned conditions and the failed generated LTL formula.

Remark 4: It is worth noting that the tendency of our Task Generation module to produce sequential LTL formulas reflects a design principle that prioritizes verifiability and executability, which is crucial in safety-critical systems. This is not an intrinsic limitation of our framework's expressive power. While this approach simplifies formal verification against high-level semantics, we also introduce an improved prompting strategy to enhance flexibility. For instance, the LLM can be guided to generate a declarative formula, such as $Fp_1 \wedge Fp_2$, or a disjunctive formula connecting all possible valid execution orders, such as $F(p_1 \wedge Fp_2) \vee F(p_2 \wedge Fp_1)$. This strategy delegates planning flexibility to the downstream module, thereby improving adaptability without compromising formal guarantees.

Based on the experimental results in section IV, the task generation is capable of generating valid LTL formulas in the vast majority of cases. However, to mitigate the risk of incorrect outputs due to hallucination in LLMs [36], we introduce the task regeneration to ensure the robustness of the Inference Module.

2) Task Verification: Once the task generation component generates the executable LTL tasks, the task verification is triggered to determine whether the tasks can address the challenges present in the environment or if the generated LTL formula contains any syntax errors. Specifically, the prompts for this module consist of the following main components. It is worth noting that in the task verification, the **Task Background Descriptions** and **Linear Temporal Logic Specifications** in the prompt are the same as those in the task generation. Therefore, we only introduce the different parts below.

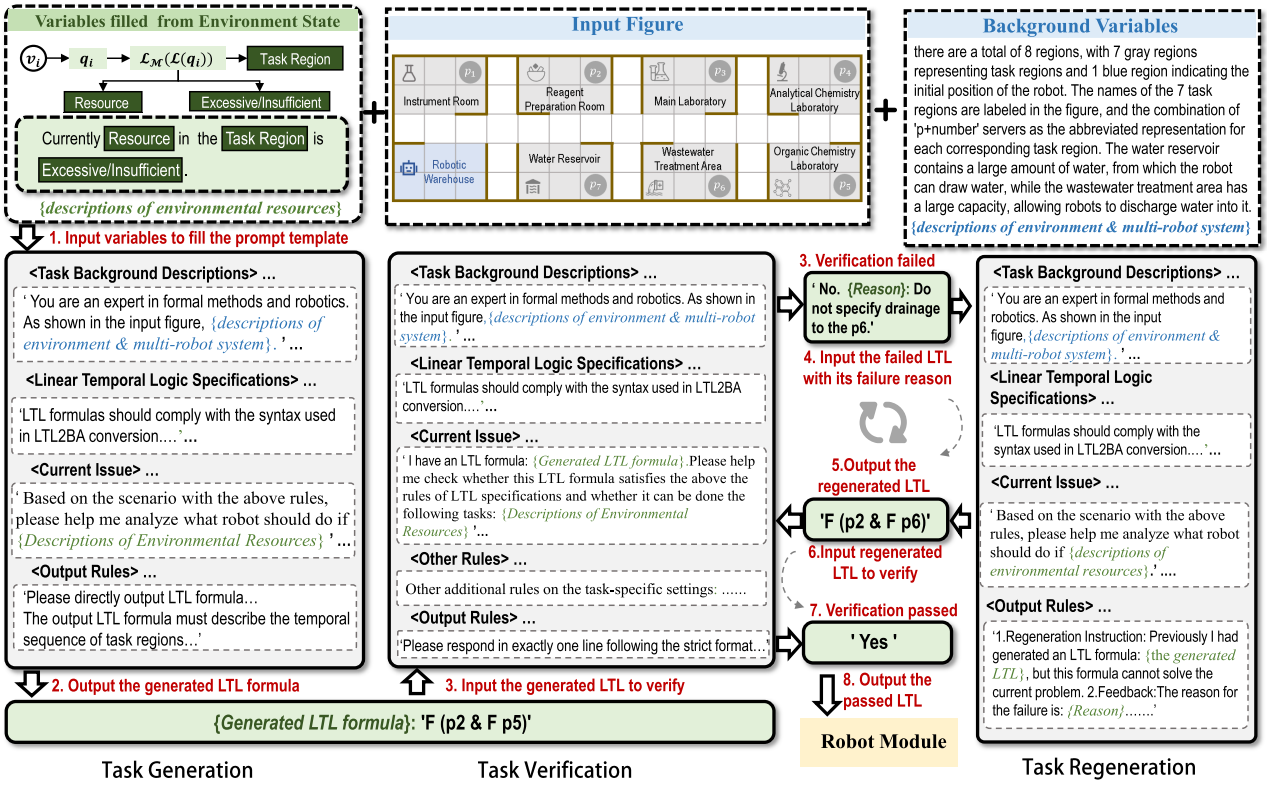

Fig. 3. An illustrative workflow of the LLM-guided Inference Module, showcasing the closed-loop "Generation-Verification-Regeneration" process in a chemical laboratory scenario. The variables filled from the environment state are automatically extracted from real-time environmental information, while the input figure and background variables are predefined by the user as initial inputs to the Inference Module. The prompt templates in the Inference Module are available at: https://llm-with-mrta.github.io/environment-driven-MRTA.github.io/

- **Current Issue**: The generated LTL formula from the task generation and the descriptions of the verification criteria.
- **Other Rules**: Other rules for the task settings, such as the definition of adversarial verification rules to guide the LLM to check the formula for logical loopholes. For example, *"Please explicitly assess whether this formula can be bypassed: Is it possible for the robot to satisfy the formula without actually executing the core recovery actions? For instance, if the robot never visits a critical location, would the formula still be valid? If so, please point out this flaw."*
- **Output Rules**: The output format for task verification.

3) Task Regeneration: The specific composition of the prompts has been described in the Task Generation (Section III-B1). To ensure the correctness of regenerated results and the overall system stability, a multi-level safety and verification mechanism is established. Task regeneration operates in an iterative "Generation–Verification–Regeneration" loop: each LTL formula generated by the LLM, whether initial or regenerated, is verified for semantic validity and executability. If verification fails, the failed formula and its reason are fed back to the LLM to guide the next regeneration.

Moreover, a failure-memory and adversarial regeneration strategy is introduced, where all failed formulas and error reasons are stored as negative examples in the prompt to encourage the LLM to produce logically revised solutions, reducing repeated errors. To avoid infinite loops, a maximum regeneration attempt number N_m is defined. If N_m consecutive iterations fail to pass verification, a fail-safe strategy is automatically activated, such as instructing the robot to return to its standby area or requesting human intervention, to ensure the overall operational safety of the system.

C. Robot Module

The Robot Module performs task allocation and planning for the multi-robot system based on the generated formula with corresponding resource requirements from the Inference Module. Similar to the mechanism of the resource planning tree, for the multi-robot system, we build a robotic planning tree to search the feasible task allocation. This robotic planning tree is denoted as $Tree^{\mathcal{A}} = (\mathcal{V}^{\mathcal{A}}, \mathcal{E}^{\mathcal{A}}, J^{\mathcal{A}})$, where its root node is initialized as $v_0^{\mathcal{A}} \in \mathcal{V}^{\mathcal{A}}$, the set of edges is $\mathcal{E}^{\mathcal{A}} \subset \mathcal{V}^{\mathcal{A}} \times \mathcal{V}^{\mathcal{A}}$ and $J^{\mathcal{A}} : \mathcal{V}^{\mathcal{A}} \to \mathbb{R}_{\geq 0}$ indicates the cost from the root node to the current node. Different from the resource planning tree, the node in $Tree^{\mathcal{A}}$ is represented as $(s^{\mathcal{A}}, q^{\mathcal{A}}, alloc, g^{\mathcal{A}})$, where $s^{\mathcal{A}} \in \mathcal{B}(\varphi^{\mathcal{A}}).\mathcal{S}$ is the automaton state, $q^{\mathcal{A}} \in \mathcal{Q}^{\mathcal{A}}$ is the corresponding task state, *alloc* represents the task allocation corresponding to the node, and $g^{\mathcal{A}} \in \mathcal{G}^{\mathcal{A}} = \{pre, suf, end\}$ is the the state stage.

As indicated in Alg. 3, the task allocation and planning first explores all nodes that have not been traversed in $Tree^{\mathcal{A}} \backslash TraTree^{\mathcal{A}}$. For each $v^{\mathcal{A}}$ to be traversed, we initiate a search for its set of reachable child nodes $SubTree^{\mathcal{A}}$ which is initialized as an empty set, followed by the expansion process of $SubTree^{\mathcal{A}}$ (lines 6-7). We also define $\Gamma(v^{\mathcal{A}}).s$ to be the automaton states of all nodes on the branch from the root node

to $v^{\mathcal{A}}$ with the same state stage. Then, based on the automaton state of the current traversed node $v^{\mathcal{A}}$, we find all reachable automaton states $s^{\mathcal{A}} \in \mathcal{B}(\varphi^{\mathcal{A}}).\mathcal{S}\backslash\Gamma(v^{\mathcal{A}}).s$ and corresponding robotic task states $q^{\mathcal{A}}$ that satisfy $(v^{\mathcal{A}}.s, \mathcal{L}_{\mathcal{A}}(q^{\mathcal{A}}), s^{\mathcal{A}}) \in \mathcal{B}(\varphi^{\mathcal{A}}).\delta$, where these $s^{\mathcal{A}}$ and $q^{\mathcal{A}}$ are assigned as the automaton states and the task states of the childe nodes, respectively.

Next, the allocation schemes corresponding to the robotic task states of these child nodes are addressed (lines 8-11). For different robotic task states, considering that in real-world scenarios some tasks require the same fleet of robots to perform actions, such as "fetch a bucket at o_a and then fill it with water at o_b," we design a mapping $\mathcal{L}_F : \mathcal{Q}^{\mathcal{A}} \to \mathbb{N}_{\geq 0}$. Each robotic task state is associated with an index, and task states corresponding to the same index can be executed by the same fleet of robots. It is obvious that each proposition $\pi^{\mathcal{A}}$ in the same generated LTL formula shares the same fleet of robots, thus their $\mathcal{L}_F(\cdot)$ are the same. Therefore, before assigning robots, we need to check whether there is a node with the same $\mathcal{L}_F(v^{\mathcal{A}}_{sub})$ on the branch from the root node to the current child node (line 8). If such a node exists, we directly assign its allocation scheme to $v^{\mathcal{A}}_{sub}$ (line 9). Otherwise, we solve the allocation scheme using Integer Linear Programming (ILP) (line 11).

$$\min \sum_{i=1}^{n_a} v^{\mathcal{A}}_{\text{sub}} \cdot \text{alloc}(i) \times \frac{\left\| a_i \cdot p - \mathcal{L}_{\mathcal{M}}\left(\mathcal{L}_{\mathcal{A}}\left(v^{\mathcal{A}}_{\text{sub}} \cdot q^{\mathcal{A}}\right)\right)\right\|_2}{v_{\text{robot}}},$$
$$\text{s.t.} \left(\sum_{i=1}^{n_a} v^{\mathcal{A}}_{\text{sub}} \cdot \text{alloc}(i) \times a_i \cdot \text{cap}_j \geq \mathcal{R}_{\mathcal{A}^j} \right)_{\mathcal{R}_{\mathcal{A}^j} \in \mathcal{R}_{\mathcal{A}}}, \quad (8)$$

where v_{robot} is the average linear velocity of robots.

According to (3), the state stage of $v^{\mathcal{A}}_{sub}$ is updated by the state updating rule

$$f_g(g, s) = \begin{cases} g, & s \notin \mathcal{B}.\mathcal{S}^F, \\ suf, & s \in \mathcal{B}.\mathcal{S}^F, g = pre, \\ end & g = suf. \end{cases} \quad (9)$$

Unlike the resource planning tree, we specify that robots perform tasks with only one prefix aud one suffix, thus $n_s = 1$.

Next, we update the cost of the current child node. The estimated task completion time for the multi-robot system at the current node is considered as the cost of the node. Specifically, for robots assigned to the task at the current node, their estimated task execution time is $\frac{\|a_i.p - \mathcal{L}_{\mathcal{M}}(\mathcal{L}_{\mathcal{A}}(v^{\mathcal{A}}_{sub}.q^{\mathcal{A}}))\|_2}{v_{robot}}$, while for unassigned robots, the estimated task execution time is set as 0. Therefore, the estimated task completion time for each robot at the node $v^{\mathcal{A}}_{sub}$ is defined as the sum of the current node's task execution time and the cost of the preceding node in the branch, i.e.,

$$t^{a_i}\left(v^{\mathcal{A}}_{sub}\right) = \begin{cases} \frac{\left\|a_i \cdot p - \mathcal{L}_{\mathcal{M}}\left(\mathcal{L}_{\mathcal{A}}\left(v^{\mathcal{A}}_{sub} \cdot q^{\mathcal{A}}\right)\right)\right\|_2}{v_{\text{robot}}} + J^{\mathcal{A}}\left(v^{\mathcal{A}}\right), & v^{\mathcal{A}}_{\text{sub}} \cdot \text{alloc}\ (i) = 1 \\ J^{\mathcal{A}}\left(v^{\mathcal{A}}\right), & v^{\mathcal{A}}_{\text{sub}} \cdot \text{alloc}\ (i) = 0 \end{cases} \quad (10)$$

The cost of $v^{\mathcal{A}}_{sub}$ is thus defined as the maximum of the estimated task completion times among all robots, i.e.,

$$J^{\mathcal{A}}(v^{\mathcal{A}}_{sub}) = max\{t^{a_i}(v^{\mathcal{A}}_{sub})\}_{a_i \in \mathcal{A}} \quad (11)$$

Additionally, $\Gamma(v^{\mathcal{A}})$ and $SubTree^{\mathcal{A}}$ are updated and pruned in a manner similar to $Tree^{\mathcal{M}}$ (lines 12-13). $v^{\mathcal{A}}$ and the pruned $SubTree^{\mathcal{A}}$ are added to $TraTree^{\mathcal{A}}$ and $Tree^{\mathcal{A}}$, respectively (line 14).

Then, we identify the leaf node $v^{\mathcal{A}*}$ with the minimum cost and backtrack along its branch, i.e.,

$$v^{\mathcal{A}*} = \underset{v^{\mathcal{A}} \in Tree^{\mathcal{A}}}{\arg\min}\, J^{\mathcal{A}}(v^{\mathcal{A}}). \quad (12)$$

We follow its branch up to the root node. The similar backtracking procedure is applied to partition the feasible path nodes in $Tree^{\mathcal{A}}$. The actual stage node paths of $Tree^{\mathcal{A}}$ are denoted as $\Pi^{\mathcal{A}}_h = \langle v^{\mathcal{A}}_i | parent(v^{\mathcal{A}}_i).g = h \rangle, v^{\mathcal{A}}_i \in Tree^{\mathcal{A}}, i \geq 1$, where $parent(v^{\mathcal{A}}_i)$ is the parent node of $v^{\mathcal{A}}_i$ and $h \in \{pre, suf\}$. And the corresponding allocation scheme for each node on this branch is extracted to form a sequence $\{alloc\}$, which is the feasible task allocation scheme responding to $\varphi^{\mathcal{A}}$, with the cost of $J^{\mathcal{A}}(v^{\mathcal{A}*})$ (line 15).

Remark 5: In section II-A, we assume that each robot possesses only a single capability. However, our method can also handle scenarios where each robot has multiple capabilities. It can be achieved by modifying the capabilities assigned to each robot and incorporating these multiple capabilities into the constraint formulas during the ILP solving process.

Finally, the multi-robot system operates according to $\varphi^{\mathcal{A}}$ and its corresponding obtained sequence of allocation schemes $\{alloc\}$, and then feeds back the processed resource quantities to both the environmental and robotic systems (line 16).

It is important to note that in multi-robot systems, a complete task execution process typically consists of two layers: high-level task allocation and low-level motion planning. The low-level motion planning aims to generate concrete, collision-free trajectories for multiple robots, thereby addressing the path-level problem of *"how to move"*. In contrast, the contribution of this work focuses on the higher abstraction level of task allocation, which address the question of *"which robots should execute the tasks"*. The core lies in generating an available allocation sequence based on environmental requirements and the heterogeneous capabilities of the robot fleet. In the implementation of the Robot Module, we adopt standard motion planners at the low level to avoid obstacles, while potential inter-robot conflicts are mitigated at the task level through temporal logic constraints, sequential execution and native navigation stack. Hence, our task allocation and motion planning components form a hierarchical and complementary relationship: the allocation schemes generated by the robotic planning tree can be directly used as inputs to the low-level motion planner. Integrating this high-level decision framework with more advanced multi-robot cooperative motion planners constitutes an important direction for our future work.

D. Discussion

As previously discussed, the Inference Module is activated when the available environmental resources fail to meet the specified requirements. However, this approach can lead to repetitive task generation and verification for similar scenarios, resulting in unnecessary token consumption. This subsection introduces an alternative approach to reduce redundant

token consumption. Initially, the Inference Module generates a preliminary set of executable tasks that cope with potential resource constraint, denoted as $\mathcal{T}^{\mathcal{A}}$. When the resource state corresponding to a node in $Tree^{\mathcal{M}}$ falls below the required threshold, the proposed method matches these tasks to the current environmental state to identify feasible robotic tasks that can address the resource limitations. If a resource state deviates from the set of potential resource constraints, the Inference Module is reactivated to generate and verify tasks. These newly generated tasks, along with their associated environmental states, are then added to $\mathcal{T}^{\mathcal{A}}$. This approach treats the task generation and verification process outlined in Section III-B as an independent module. Specifically, lines 1-3 in Alg. 2 are executed at the outset to generate tasks for the multi-robot system, accounting for all potential environmental states and resource conditions. When the available resources in the environment fail to meet the task requirements, Alg. 2 first searches within $\mathcal{T}^{\mathcal{A}}$ to identify whether an LTL formula aligns with the current environmental and resource state. If not, the Inference Module will generate tasks.

Moreover, to address uncertainty that may arise in downstream allocation (e.g., ILP infeasibility due to an insufficient number of robots or limited capabilities), we provide a closed-loop recovery scheme for infeasible assignments. When the ILP solver in the Robot Module fails to find a feasible solution under physical constraints, the system does not terminate. Instead, it treats the infeasible result as verification feedback and returns it to the Inference Module. This feedback triggers the task regeneration process, guiding the LLM to propose an alternative that is more consistent with physical realities. In this way, a closed loop from high-level reasoning to physical verification enhances the ultimate executability of the plan.

The complexities of the Environment and Robot Modules are analyzed as follows. For the Environment Module, the structure of $Tree^{\mathcal{M}}$ consists of one prefix stage, n_s suffix stages, and one final stage. Through Repeated State Pruning, it is established that the automaton states in the prefix and suffix stages can differ. Consequently, the maximum number of nodes in these stages is $|\mathcal{S}|$. Thus, the maximum depth of $Tree^{\mathcal{M}}$ is given by $(n_s+1)|\mathcal{S}|+1$. Additionally, the rule of Cost-Based Pruning ensures that only nodes with lower costs are selected, which constrains the tree's maximum width to $|\mathcal{S}|+n_s\times(|\mathcal{S}^F|\times|\mathcal{S}|)$. This is because, for nodes with the same automaton state and state stage, the method exclusively explores those with lower costs. Therefore, the prefix stage contains at most $|\mathcal{S}|$ nodes, while the suffix stage contains up to $|\mathcal{S}^F|\times|\mathcal{S}|$ nodes. Furthermore, each node can generate at most $|\mathcal{S}|\times|\mathcal{Q}|$ child nodes according to the expansion mechanism. Thus the maximum number of explorations in the Environment Module is

$$\big((n_s+1)|\mathcal{S}|+1\big)\times\big(|\mathcal{S}|+n_s\times(|\mathcal{S}^F|\times|\mathcal{S}|)\big)\times|\mathcal{S}|\times|\mathcal{Q}|. \quad (13)$$

Since $|\mathcal{S}^F|$ is typically much smaller than $|\mathcal{S}|$, and the number of generated child nodes is nearly proportional to $|\mathcal{Q}|$, the complexity of the Environment Module is approximately $\mathcal{O}(n_s^2|\mathcal{S}|^2)$.

The complexity of Robot Module is similar to that of the Environment Module, with the key difference being that the nodes in $Tree^{\mathcal{A}}$ have only one prefix stage, one suffix stage, and one final stage. Therefore, the maximum tree length is $2\times|\mathcal{S}|+1$. Similarly, the maximum tree width is $|\mathcal{S}|+|\mathcal{S}^F|\times|\mathcal{S}|$, and each parent node can generate at most $|\mathcal{Q}_{\mathcal{A}}|\times|\mathcal{S}|$ child nodes. Besides, solving the ILP problem for robot allocation at each node is inherently NP-hard. In the theoretical worst case, the computational complexity grows exponentially with the number of robots n_a, i.e., $\mathcal{O}(2^{n_a}n_c)$. However, our framework effectively alleviates this potential bottleneck through the incremental search and pruning mechanisms of the robotic planning tree. Specifically, instead of solving a single global ILP that simultaneously encodes all robots and task possibilities, our approach decomposes the problem into much smaller local ILP instances, each solved only for the current subtask at a given tree node. This strategy enables the method to efficiently handle large-scale robot systems, as verified by the experimental results in Section IV. Therefore, the overall computational complexity of the Robot Module is primarily determined by the size of the planning tree and the cost of ILP solving, and can be approximated as $\mathcal{O}(2^{n_a}n_c|\mathcal{S}|^2)$.

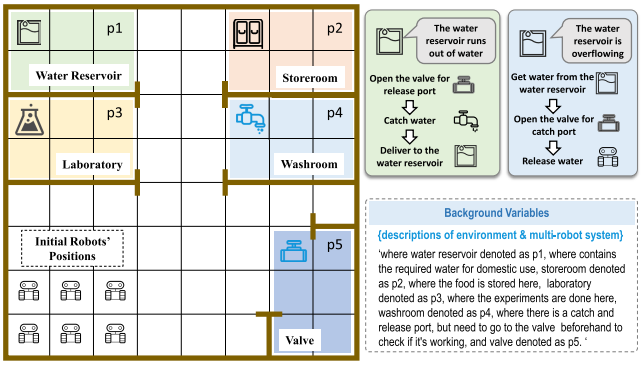

Fig. 4. The chemical experiment scenario for the numerical experiments and the background variables in the input prompts in the Inference Module.

IV. Experiment

In this section, we evaluate the scalability, efficiency, and autonomy of our proposed method through numerical and simulation experiments, which are executed on a computing platform equipped with Intel i9-13900k, 64 GB RAM, and an NVIDIA RTX 3090 GPU. The numerical experiments are conducted using MATLAB 2024a, while the simulation experiments are performed on Ubuntu 18.04. Besides, LTL2BA is utilized to convert LTL into Büchi automaton.

A. Numerical Experiments

1) Resource Conditions: In the scenario depicted in Fig. 4, we test the generated tasks, along with the generation time, verification time, regeneration time and number of regenerations under five different environment scenarios, where detailed descriptions can be found in Table I. When faced with different resource conditions, the Inference Module generates feasible multi-robot system tasks. As shown in Table I, the generated LTL formulas comply with LTL specifications and meet the format requirements for LTL2BA conversion.

TABLE I

PERFORMANCE EVALUATION UNDER DIFFERENT SCENARIOS.GT DENOTES THE GENERATION TIME (S); VT DENOTES THE VERIFICATION TIME (S);RGT DENOTES THE REGENERATION TIME (S); AND NR REPRESENTS THE NUMBER OF REGENERATIONS

Sce.	Generated LTL Formula	GT (s)	VT (s)	RGT (s)	NR
S1	$F(p_1 \& X(F(p_4 \& X(Fp_1))))$	3.73	2.47	2.53	1
S2	$F(p_1 \& F(p_5 \& Fp_4))$	2.56	2.52	2.13	2
S3	$F(p_5 \& X(p_4 \& Xp_2))$	2.81	2.79	2.74	1
S4	$F(p_5 \& F(p_2 \& Fp_3))$	2.44	2.43	0.00	0
S5	$F(p_5) \& F(p_4) \& F(p_2) \& F(p_1) \& F(p_3)$	3.42	2.18	2.79	1

S1: The water reservoir runs out of water.
S2: The water reservoir is overflowing.
S3: There is a water pipe leak in the storage room and there may be a lot of water in the storage room.
S4: The laboratory requires an urgent delivery of food from the storeroom. However, a new safety protocol requires that the main water valve must be inspected before any new materials are brought into the lab.
S5: The laboratory needs a reagent and water, but the storeroom where reagents are stored is locked. The key is in the washroom.

Moreover, the multi-robot system, by executing the generated LTL formulas, can address the issues described by the current environmental resource requirements. For the Scenario 1, when the water reservoir runs out of water, the robots are expected to first turn on the valve, then fetch water from the washroom, and finally deliver the water to the water reservoir. As can be seen from the generated LTL formula, robots can accomplish the task as above by executing this LTL formula. For the amount of water resources fetched and delivered, it can be solved during the tree search, as detailed in Section III-B and III-C. Besides, the time spent by the generation module and verification module for different issues is summarized in Table I.

2) Multi-Robot System: Continuing with Fig. 4, with the environment LTL of $GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}} \wedge GF\pi_3^{\mathcal{M}}$, we test the solution times for multi-robot systems with different scales, including times of tree search (the total solving time of the Environment Module and Robot Module) and the Inference Module. Suppose the current resource condition is *the water reservoir runs out of water*. Fig. 5 (a) indicates the solution times of our method for different multi-robot system scales. In Fig. 5, the problem is solved 10 times for each scenario and the average time, maximum time and minimum time are shown. Specifically, when the number of robots is 6, 60, 150, 300, 3000, 9000 and 15000, the solution time for the tree search module is 2.638s, 2.947s, 3.403s, 4.115s, 23.918s, 292.805s and 1218.149s, respectively. Meanwhile, the time for the Inference Module is 17.931s, 17.132s, 15.291s, 17.869s, 13.692s, 17.420s and 21.013s, respectively. Fig. 5 indicates that the solution times for the Inference Module exhibit minimal fluctuation, as it primarily depends on the task generation and verification. As the scale of the robotic system increases, the solution time of our method increases correspondingly. For the robotic system with up to 300 robots, our method can be completed in approximately 20 seconds, with a significant portion of this time attributed to

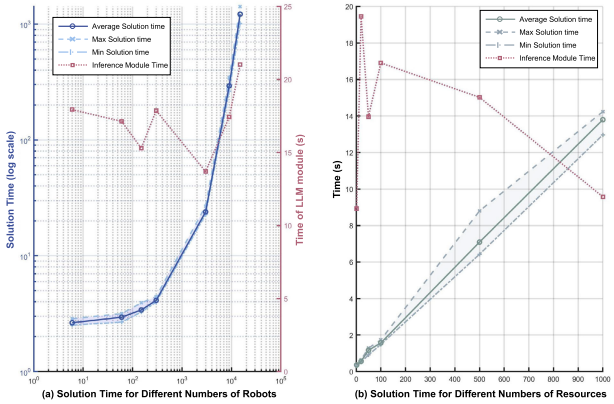

Fig. 5. (a) The solution times for different scales of multi-robot system, and (b) the solution times for different number of resource types.

the task generation and verification of the LTL formula, which demonstrates that our method is both efficient and scalable. It should be emphasized that the solution times reported in the numerical experiments do not reflect the performance of physical robotic systems, but rather demonstrate the scalability and effectiveness of our framework at the strategic level. Our experimental results provide the evidence of our framework's computational feasibility for such large-scale applications.

3) Resource Scale: Given an environment LTL $GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}} \wedge GF\pi_3^{\mathcal{M}}$, we further test the solution times for the system with 30 robots under varying numbers of resource types, where the robots are categorized into three groups based on their differing capability values. Suppose the current resource condition is *the water reservoir runs out of resource* $1, \ldots,$ *resource* n, and the extraction and replenishment methods for these resources are the same as those for the water resource. Hence, we only need to apply the Inference Module to generate the task corresponding to the robotic system for one type of resource. This means that the water reservoir will simultaneously require n types of resources to meet the task requirements specified by the environment. We set n to 2, 20, 50, 100, 500, and 1000, and solved each case 10 times. The average solution times for the tree search module are 0.362s, 0.558s, 1.169s, 1.569s, 7.091s, and 13.798s, respectively, while the average solution times for the Inference Module are 8.922s, 19.458s, 13.960s, 16.906s, 15.022s, and 9.576s, respectively. The results are shown in Fig. 5 (b), where the solution time of the tree search module increases linearly with the number of resource types. Additionally, we test the case when the number of resource types is 10,000, where the solution time for the tree search module is 141.833 ± 3.873s and the solution time for the large model module is 19.458s. This demonstrates that our method still maintains good scalability with respect to the number of resource types.

4) Task Regions: Continuing with the example in Fig. 3, we consider 8 different task regions, each with one type of resource, while the multi-robot system has 30 robots. The solution times are computed for both the tree search and the Inference Module for task regions ranging from 1 to 5. The resources conditions are more involved with the increase of the number of task regions. For example, when the number of task

TABLE II
SOLUTION FOR DIFFERENT ENVIRONMENT LTL AND TASK REGIONS

Environment LTL	Added Descriptions of Environmental Resource	Task Regions	Inference Module Time (s)	Solution Time (s)
$GF\pi_1^{\mathcal{M}}$(2)	The water in the instrument room is insufficient.	1	12.162	0.330 ± 0.007
$GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}}$ (3)	The water in the Reagent Preparation Room is excessive.	2	15.866	0.843 ± 0.056
$GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}} \wedge GF\pi_3^{\mathcal{M}}$ (4)	The water in the main laboratory Preparation Room is excessive.	3	25.563	1.135 ± 0.033
$GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}} \wedge GF\pi_3^{\mathcal{M}} \wedge GF\pi_4^{\mathcal{M}}$ (5)	The water in the analytical chemistry laboratory is insufficient.	4	39.724	1.372 ± 0.007
$GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}} \wedge GF\pi_3^{\mathcal{M}} \wedge GF\pi_4^{\mathcal{M}} \wedge GF\pi_5^{\mathcal{M}}$ (6)	The water in the organic chemistry is excessive.	5	43.783	1.414 ± 0.021

TABLE III
SOLUTION FOR TASK REGENERATION (NR: NUMBER OF REGENERATIONS; RGT: REGENERATION TIME)

Sce.	Failed LTL	Failure Analysis	NR	Re-generated LTL	RGT (s)
S1	G p2 & G p4	invalid syntax and missing eventual visit operator.	1	F (p2 & F p4)	2.14
S2	F (p7 & F p4)	missing reagent source p2 visit.	1	F (p7 & F (p2 & F (p4)))	1.58
S3	F (p7 & F p4 & F p1)	missing discharge step at wastewater area.	2	F (p4 & F (p6 & F (p1 & F p4)))	2.56
S4	F (p2 & (p7 & p1 & p4))	invalid parentheses and missing sequential visit semantics.	1	F (p1 & F (p2 & F (p7 & F (p4))))	1.93

S1: The Reagent is insufficient in the Analytical Chemistry Laboratory.
S2: Both the water and reagent are insufficient in the Analytical Chemistry Laboratory.
S3: The water is excessive and the instruments are insufficient in the Analytical Chemistry Laboratory.
S4: The reagent, water and instruments are insufficient in the Analytical Chemistry Laboratory.

regions is 2, we want to ensure that the environment satisfies $GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}}$. However, at this point, the environmental resources include both *the water in the instrument room are insufficient* and *the water in the Reagent Preparation Room is excessive*. As shown in Table II, with the increase in the number of task regions and the corresponding environmental resource descriptions, both the solution times of the tree search and the Inference Module increase. Specifically, the solution time of the tree search remains within 1.5s, demonstrating the effectiveness of our method.

5) Task Regeneration: Further to the scenario of Fig. 3 in the previous subsection, the environment LTL formula is $GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}} \wedge GF\pi_3^{\mathcal{M}} \wedge GF\pi_4^{\mathcal{M}}$ and we test the effectiveness of task regeneration. Suppose that, at some point, certain laboratory resources do not meet the thresholds specified in the environment, as shown in Table III. This triggers the Inference Module to generate executable tasks for the robots. We assume that the task generation in the Inference Module produces a failed LTL that can not pass the task verification, here we set these failed LTL formulas artificially and pass them into the task regeneration, where the results are shown in Table III. According to the results, the re-generated LTL formulas are valid for the current resource situations, demonstrating that the task regeneration can enhance the robustness of the task generation to a certain extent.

6) Comparison and Ablation Experiments: To further validate the effectiveness and robustness in task generation of our proposed Inference Module, we conduct both comparison and ablation studies under diverse task scenarios. Specifically, four types of task scenarios are designed to reflect different levels of environmental and resource complexity: Single-resource/Single-region (SS), Single-resource/Multi-region (SM), Multi-resource/Single-region (MS), and Multi-resource/Multi-region (MM). For each scenario type, 10 task requirements are randomly generated, and each requirement is executed 20 times to ensure statistical reliability, resulting in 800 independent trials per method.

TABLE IV
QUANTITATIVE COMPARISON AND ABLATION STUDY UNDER VARYING TASK SCENARIOS

Method	**Time (s)↓**	**SR↑**	**SR-SS↑**	**SR-SM↑**	**SR-MS↑**	**SR-MM↑**
Comparison baselines						
Rule-based Method	**2.41**	0.42	0.68	0.26	0.64	0.08
Heuristic Optimization	2.51	0.70	0.80	0.60	0.76	0.64
Ablation variants						
w/o verification	7.62	0.81	0.82	0.78	0.96	0.66
w/o regeneration	5.25	0.51	0.62	0.56	0.44	0.40
Ours	8.17	**0.99**	**1.00**	**1.00**	**1.00**	**0.96**

w/o verification: Without feedback mechanism in Task Verification.
w/o regeneration: Without both Task verification and Task Regeneration in the Inference Module.

Table IV summarizes the quantitative results, where Time is the total time for task generation, SR is the success rate, SR-SS, SR-SM, SR-MS and SR-MM stand for success rate for SS, SM, MS and MM, respectively. Among the comparison baselines, the Rule-based Method relies on pre-defined logical templates and therefore struggles with generalization in complex multi-resource tasks. The Heuristic Optimization baseline, which constructs the visiting sequence based on task priorities and precedence constraints, achieves improved performance but remains limited in adaptability. In contrast, our LLM-driven framework exhibits consistent advantages even when certain components are removed.

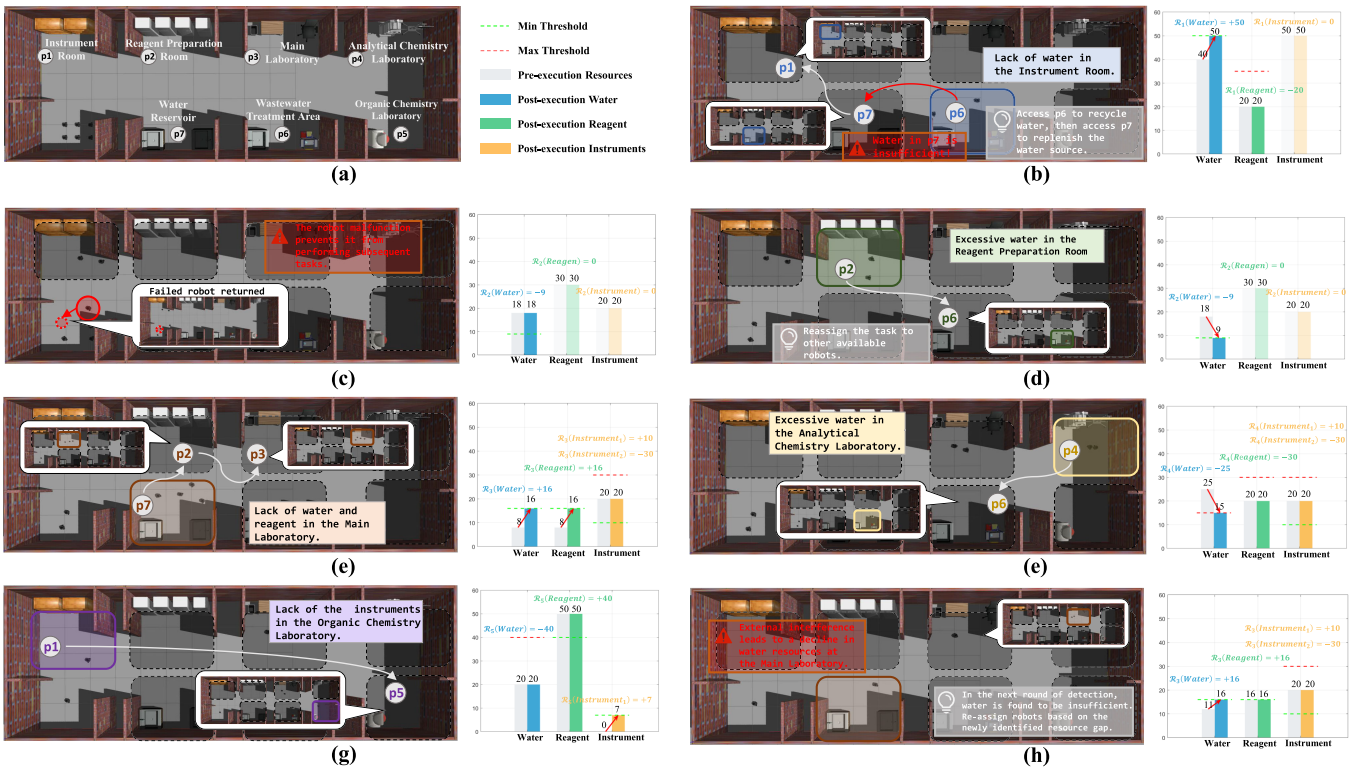

Fig. 6. (a) The schematic diagram of a chemistry lab scene and the legends in the following bar graphs. (b) Execution process for resolving water shortage in the Instrument Room but the water in the Water Reservoir is insufficient. (c) Robot failure occurs en route to p7 and the system reassigns the task to another robot. (d) Execution of the reassigned task to transfer excessive water from the Reagent Preparation Room. (e) Execution process for water and reagent shortage in the Main Laboratory. (f) Execution process for excessive water in the Analytical Chemistry Laboratory. (g) Execution process for instrument shortage in the Organic Chemistry Laboratory. (h) Execution process for water shortage in the Main Laboratory due to external disturbance. The transparent portion of the bar graph indicates that there are no threshold requirements for this resource. The simulation experiment video is available at: https://youtu.be/vgUGGd4fAYM.

Among the ablation experiments, disabling the feedback mechanism in Task Verification slightly decreases overall success (SR=0.81), while removing the whole regeneration module (both Task Verification and Regeneration) leads to a sharp drop (SR=0.51). These results confirm that both components are critical for ensuring task correctness and recovery. With all components enabled, our proposed task Inference Module achieves nearly perfect success across all scenario categories, demonstrating strong robustness and adaptability with only a marginal increase in generation time. Moreover, the results highlight that our closed-loop "Generation–Verification–Regeneration" mechanism serves as the core strategy for mitigating LLM hallucination risks and enhancing overall system robustness.

B. Simulation Experiment

In this section, the effectiveness and robustness of our method are demonstrated through a simulation in Gazebo, following the scenario illustrated in Fig. 3. The environment LTL is $\varphi^{\mathcal{M}} = GF\pi_1^{\mathcal{M}} \wedge GF\pi_2^{\mathcal{M}} \wedge GF\pi_3^{\mathcal{M}} \wedge GF\pi_4^{\mathcal{M}} \wedge GF\pi_5^{\mathcal{M}}$ and the corresponding executions of the multi-robot system are shown in Fig. 6. During the validation of the environmental system, it is found that there is a lack of water resource in the instrument room, which causes the violation of $\pi_1^{\mathcal{M}}$. In response, the Inference Module generates the executable LTL formula φ_1 = F(p7&Fp1) (i.e., an LTL formula $\varphi_1^{\mathcal{A}} = F(\pi_7^{\mathcal{A}} \wedge F\pi_1^{\mathcal{A}})$). This ensures that the robots will first proceed to the water reservoir to fetch water, and then transport it to the instrument room, thereby restoring the water resource to meet the required threshold. However, it discovers that the reservoir's own water level is insufficient. Instead of failing, the Inference Module is re-triggered and reasons about the deeper resource dependency. It generates a new, more complex task: the robot must first visit the Wastewater Treatment Area (p6) to recycle water, use it to replenish the Water Reservoir (p7), and then proceed to the Instrument Room (p1) to deliver the water, as shwon in Fig. 6 (b).

Similarly, when faced with the presence of excessive water in the reagent preparation room, the generated robot LTL is $\varphi_2^{\mathcal{A}} = F(\pi_2^{\mathcal{A}} \wedge F\pi_6^{\mathcal{A}})$ with the purpose of going to the reagent preparation room and transferring the excess water resource to the wastewater treatment area. However, during the execution of the task, an assigned robot experiences a failure and becomes inoperable. In response, the Robot Module re-initiates the task allocation process, assigning the unfinished task to another available robot, as shown in Fig. 6 (c-d). The current reagent and water in the main laboratory are 8g and 8L, respectively, which is contrary to the requirements in $\varphi^{\mathcal{M}}$ that both reagent and water here should not be less than 16. Therefore, the generated task $\varphi_3^{\mathcal{A}} = F(\pi_7^{\mathcal{A}} \wedge F(\pi_2^{\mathcal{A}} \wedge F\pi_3^{\mathcal{A}}))$ directs the robot to go to the reagent preparation room and water reservoir first to get the reagent and water, and then to the main laboratory to replenish these resources. Likewise, due to excess water in the analytical chemistry room, as well

as the lack of instruments in the organic chemistry laboratory, our method generates measures $\varphi_4^{\mathcal{A}} = F(\pi_4^{\mathcal{A}} \wedge F\pi_6^{\mathcal{A}})$ and $\varphi_5^{\mathcal{A}} = F(\pi_1^{\mathcal{A}} \wedge F\pi_5^{\mathcal{A}})$ corresponding to the robotic system, respectively, thus achieving resource-satisfiability in the environment. After a successful operational cycle, a sudden external disturbance depletes the water in the Main Laboratory. When a sudden external disturbance creates a water shortage in the Main Laboratory, the system autonomously generates a corrective task to restore the resource, as shown in Fig. 6 (h). From the simulation results presented above, it can be concluded that our method demonstrates effectiveness, robustness, and autonomy in continuous validation and restoration of environmental resources.

V. Conclusion

This work presents an environment-driven and LLM-guided task inference and allocation framework that integrates dual-system temporal logics to address dynamic resource management challenges in multi-robot systems. By coupling temporal logic-based environment verification with LLM-guided robotic task generation, the proposed approach ensures continuous satisfaction of environmental requirements under the temporal constraints. Experimental validations demonstrate that our method effectively addresses resource insufficiency and excess scenarios, ensuring autonomous and efficient task execution across varying scales of robots and resources. Although our proposed framework demonstrates strong potential and effectiveness in bridging high-level logical specifications with LLM reasoning, we also recognize several directions that merit deeper exploration in future work. First, we plan to further enhance the robustness of the framework in real-world deployments by integrating active execution monitoring, model predictive control (MPC)-based task replanning, and probabilistic state estimation methods, enabling it to effectively handle physical uncertainties such as sensor noise and communication delays. Second, we plan to investigate broader integration of dynamic task attributes and adaptive learning mechanisms, and will aim to combine our proposed task allocation framework with advanced multi-robot cooperative motion planning algorithms, such as global path-planning solvers (e.g., CBS/ECBS) and local collision-avoidance controllers (e.g., ORCA), to achieve a tighter coupling between task-level allocation and trajectory-level execution. This will enable the system to address more complex scenarios that require intensive and synchronized operations among multiple robots, thus further enhancing its responsiveness and execution efficiency.

Acknowledgment

The AI-driven experiments, simulations, and model training were performed on the robotic AI-scientist platform of Chinese Academy of Sciences.

References

[1] B. Chen et al., "DIBNN: A dual-improved-BNN based algorithm for multi-robot cooperative area search in complex obstacle environments," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 2361–2374, 2025.

[2] Z. Xu, Y. Kang, Y. Cao, and Z. Li, "Spatiotemporal graph convolution multifusion network for urban vehicle emission prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3342–3354, Aug. 2021.

[3] J. Liu et al., "Intermittent deployment for large-scale multi-robot forage perception: Data synthesis, prediction, and planning," *IEEE Trans. Autom. Sci. Eng.*, vol. 21, no. 1, pp. 27–47, Jan. 2024.

[4] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to parse natural language commands to a robot control system," in *Proc. 13th Int. Symp. Exp. Robot.*Cham, Switzerland: Springer, 2013, pp. 403–415.

[5] V. Blukis, C. Paxton, D. Fox, A. Garg, and Y. Artzi, "A persistent spatial semantic representation for high-level natural language instruction execution," in *Proc. Conf. Robot Learn.*, 2021, pp. 706–717.

[6] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2Motion: From natural language instructions to feasible plans," *Auto. Robots*, vol. 47, no. 8, pp. 1345–1365, Dec. 2023.

[7] D. Shah, B. Osinski, B. Ichter, and S. Levine, "LM-Nav: Robotic navigation with large pre-trained models of language, vision, and action," in *Proc. Conf. Robot Learn.*, 2022, pp. 492–504.

[8] Z. Li et al., "Language-guided dexterous functional grasping by LLM generated grasp functionality and synergy for humanoid manipulation," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 10506–10519, 2025.

[9] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh, "Translating natural language to planning goals with large-language models," 2023, *arXiv:2302.05128*.

[10] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[11] H. Wang, H. Zhang, L. Li, Z. Kan, and Y. Song, "Task-driven reinforcement learning with action primitives for long-horizon manipulation skills," *IEEE Trans. Cybern.*, vol. 54, no. 8, pp. 4513–4526, Aug. 2023.

[12] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1487–1507, Oct. 2021.

[13] M. Guo and D. V. Dimarogonas, "Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 797–808, Apr. 2017.

[14] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," *IEEE Trans. Robot.*, vol. 38, no. 6, pp. 3602–3621, Dec. 2022.

[15] Z. Chen, M. Cai, Z. Zhou, L. Li, and Z. Kan, "Fast motion planning in dynamic environments with extended predicate-based temporal logic," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 5293–5307, 2025.

[16] Y. E. Sahin, P. Nilsson, and N. Ozay, "Multirobot coordination with counting temporal logics," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1189–1206, Aug. 2020.

[17] L. Li, Z. Chen, H. Wang, and Z. Kan, "Fast task allocation of heterogeneous robots with temporal logic and inter-task constraints," *IEEE Robot. Autom. Lett.*, vol. 8, no. 8, pp. 4991–4998, Aug. 2023.

[18] L. Li, Z. Chen, H. Wang, and Z. Kan, "Task allocation of heterogeneous robots under temporal logic specifications with inter-task constraints and variable capabilities," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 14030–14047, 2025.

[19] X. Luo and C. Liu, "Simultaneous task allocation and planning for multirobots under hierarchical temporal logic specifications," *IEEE Trans. Robot.*, vol. 41, pp. 5040–5059, 2025.

[20] N. Gopalan, D. Arumugam, L. L. S. Wong, and S. Tellex, "Sequence-to-sequence language grounding of non-Markovian task specifications," in *Proc. Robot., Sci. Syst.*, Jan. 2018.

[21] M. Berg, D. Bayazit, R. Mathew, A. Rotter-Aboyoun, E. Pavlick, and S. Tellex, "Grounding language to landmarks in arbitrary outdoor environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 208–215.

[22] C. Wang, C. Ross, Y. Kuo, B. Katz, and A. Barbu, "Learning a natural-language to LTL executable semantic parser for grounded robotics," in *Proc. Conf. Robot Learn.*, 2020, pp. 1706–1718.

[23] J. Pan, G. Chou, and D. Berenson, "Data-efficient learning of natural language to linear temporal logic translators for robot task specification," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 11554–11561.

[24] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "NL2TL: Transforming natural languages to temporal logics using large language models," 2023, *arXiv:2305.07766*.
[25] S. Xu, X. Luo, Y. Huang, L. Leng, R. Liu, and C. Liu, "Nl2Hltl2Plan: Scaling up natural language understanding for multi-robots through hierarchical temporal logic task specifications," *IEEE Robot. Autom. Lett.*, vol. 10, no. 10, pp. 10482–10489, Oct. 2025.
[26] J. X. Liu et al., "Lang2LTL: Translating natural language commands to temporal specification with large language models," in *Proc. Workshop Lang. Robot. CoRL*, 2022.
[27] A. Mavrogiannis, C. Mavrogiannis, and Y. Aloimonos, "Cook2LTL: Translating cooking recipes to LTL formulae using large language models," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2024, pp. 17679–17686.
[28] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Proc. 13th Int. Conf. Comput. Aided Verif.*Cham, Switzerland: Springer, 2001, pp. 53–65.
[29] K. Leahy et al., "Scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS)," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2516–2535, Aug. 2022.
[30] Z. Chen and Z. Kan, "Real-time reactive task allocation and planning of large heterogeneous multi-robot systems with temporal logic specifications," *Int. J. Robot. Res.*, vol. 44, no. 4, pp. 640–664, Apr. 2025.
[31] J. Achiam et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.
[32] *Meet Claude*. [Online]. Available: https://www.anthropic.com/claude
[33] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Comput. Surveys*, vol. 55, no. 9, pp. 1–35, Sep. 2023.
[34] H. Touvron et al., "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.
[35] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: Interactively translating unstructured natural language to temporal logics with large language models," in *Proc. Int. Conf. Comput. Aided Verif.*, 2023, pp. 383–396.
[36] Y. Zhang et al., "Siren's song in the ai ocean: A survey on hallucination in large language models," *Comput. Linguist.*, vol. 2025, pp. 1–46, Aug. 2025.

Lin Li (Graduate Student Member, IEEE) received the B.E. degree in mechanical engineering and automation from Hefei University of Technology, Hefei, China, in 2021. She is currently pursuing the Ph.D. degree in control engineering with the University of Science and Technology of China, Hefei. Her current research interests include multirobot systems and formal methods.

Ziyang Chen received the B.E. degree from Beihang University, Beijing, China, in 2020, and the Ph.D. degree in control science and engineering from the University of Science and Technology of China, Hefei, China, in 2025. He currently works for Momenta. His current research interests include robotics and autonomous vehicles.

Zhen Kan (Senior Member, IEEE) received the Ph.D. degree from the Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL, USA, in 2011.

He is currently a Professor with the Department of Automation, University of Science and Technology of China, Hefei, China. His research interests include control theory, formal methods, and robotics. He currently serves on program committees for several internationally recognized scientific and engineering conferences and is an Associate Editor of *International Journal of Robotics Research*, IEEE TRANSACTIONS ON AUTOMATIC CONTROL, and IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.