2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Ensuring Safety in LLM-Driven Robotics: A Cross-Layer Sequence Supervision Mechanism

Ziming Wang, Qingchen Liu, Jiahu Qin*, Man Li

***Abstract*— Integrating Large Language Models (LLMs) into robotics significantly enhances autonomous task planning. However, ensuring that multi-step task plans (action sequence) generated by LLMs comply with pre-defined safety constraints during planning and execution remains a challenge, limiting their adaptability in complex environments. To address this issue, a mechanism that can monitor and adjust the plan generated by the LLM-driven task planner and guide the motion planner to avoid potential risks during action execution is required. Therefore, this paper proposes a cross-layer sequence supervision mechanism. Specifically, we employ linear temporal logic syntax to express safety constraints and convert them into a set of nondeterministic Büchi automatons to build a cross-layer safety supervisor. For the task planning layer, the safety supervisor provides a closed-loop correction mechanism that can identify violations in the task plan in real time and guide LLM-driven planners to correct this plan to ensure compliance. For the motion planning layer, the safety supervisor introduces virtual "obstacle" information into the task plan to form the task plan tuple. Based on this plan tuple, the motion planner can proactively prevent unsafe behaviors during action execution. Extensive experimentation demonstrates significant improvements in safety with this cross-layer supervision mechanism, highlighting its potential to enhance LLM-driven robotic technology. Experiment details can be found in `https://youtu.be/BDdSSEP6HJw`.**

I. Introduction

In recent developments, the integration of Large Language Models (LLMs) into robotics has made significant progress, endowing robots with Natural Language comprehension and the ability to generate the task plan directly from task textual instructions [1], [2], [3], [4], [5]. Models like GPT-4 [6] have demonstrated remarkable proficiency in understanding and generating Natural Language, promising to revolutionize autonomous task planning for robot.

Despite the expansive application potential of integrating LLMs into robotics, a prominent challenge persists: LLM-driven robotics often lack the capability to adhere to safety constraints in task planning [7], [8], [9]. Specifically, at the task planning layer, multi-step task plans (action sequence) generated based on task textual instructions frequently overlook safety constraints, leading to scenarios where the robot may operate in a unsafe condition [10], [7]. This limitation hampers their adaptability in real-world environments, where adherence to human-defined safety rules is crucial.

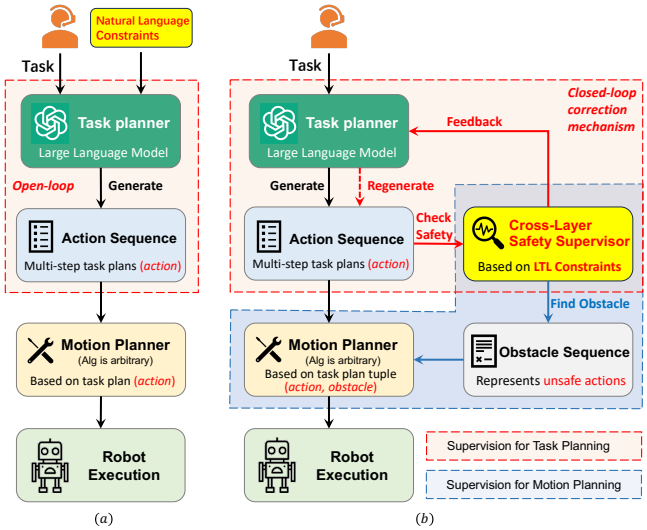

Fig. 1. illustrates of an LLM-driven robotic system with safety constraints. Approach (a) involves defining safety constraints in natural language for input into LLMs, as outlined in [2], [3]. Approach (b) introduces our proposed method, where safety constraints are described in LTL syntax and serve as a safety supervisor to guide both task and motion planning.

One approach to address this issue involves using Natural Language (NL) to describe safety constraints and input these constraints directly into the LLM-driven task planner, leveraging the powerful ability of LLM to understand the meaning of safety constraints in Natural Language, thereby generating multi-step task plans considering these constraints [2], [3], as shown in Fig.1(a). However, this open-loop approach does not guarantee compliance with safety constraints for the generated task plan and lacks a mechanism for dynamically adjusting task plan upon detecting violations of these constraints. Moreover, this open-loop strategy fails to address safety concerns at the execution level. Even if a task plan theoretically complies with safety constraints, the robot might inadvertently violate these constraints during motion planning, introducing potential safety risks. For instance, it is unsafe at the motion planning layer if a robot accidentally passes through a designated hazardous "Area1" while executing the plan "Go to Tool" that complies with safety constraints (as seen in Fig. 5).

To address the aforementioned challenges and unlock the potential of LLM-based robotic technology, a mechanism

This work was supported in part by the National Natural Science Foundation of China under Grant 62303435, in part by the Natural Science Foundation of Anhui Province under Grant 2308085QF203. *(Corresponding author: Jiahu Qin.)*

Ziming Wang, Qingchen Liu, and Man Li are with the Department of Automation, University of Science and Technology of China, Hefei 230027, China (e-mail: wang_ziming@mail.ustc.edu.cn; qingchen_liu@ustc.edu.cn; man.li@ustc.edu.cn).

Jiahu Qin is with the Department of Automation, University of Science and Technology of China, Hefei 230027, China, and also with the Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei 230088, China (e-mail: jhqin@ustc.edu.cn).

that ensures safety constraints at both task and motion planning layers is required. This mechanism can monitor and correct the task plan generated by LLM in real time, and can guide motion planner to avoid potential risks during action execution, as shown in Fig. 1(b). Therefore, this paper proposes a cross-layer sequence supervision mechanism. Specifically, we build a cross-layer safety supervisor using linear temporal logic syntax to articulate safety constraints and convert these constraints into a set of nondeterministic Büchi automatons (NBAs). Thanks to the powerful sequence judgment capabilities provided by NBA [11], [12], this safety supervisor can correct and improve multi-step task plans (action sequence), effectively ensuring safety at both task and motion planning layers. For the task planning layer, this safety supervisor provides a closed loop correction mechanism to ensure the safety of the task plan. That is, the supervisor evaluates each action generated by the LLM-driven task planner and provides feedback containing constraint information when violations are detected, guiding the task planner to generate a new action that comply with safety constraints. For the motion planning layer, the safety supervisor identifies areas linked to unsafe actions during task plan execution as "obstacles" and incorporates them into the task plan, creating a task plan tuple to minimize the potential risk of violating safety constraints. Our primary contributions can be summarized as follows:

- We introduce a cross-layer sequence supervision mechanism for LLM-driven robotics. Its cross-layer features enhance safety across both the generation and execution stages of the task plan, thereby ensuring safety at both task and motion planning layers.
- We propose a novel closed-loop correction mechanism for the task planning layer to identify safety violation actions and feedback violation information in real time. This ensures that the task plan generated by LLM can be continuously corrected based on this feedback.
- Extensive experiments confirm the notable advantages of proposed safety supervisor. It not only provides dual-layer safety assurance, significantly enhancing safety performance, but also stands out as a module that can be deployed across various robot platforms, highlighting its potential for further research and application.

II. Related Work

A. LLM-driven Robot Task Planning

The integration of Large Language Models (LLMs) into robotic task planning has gained a fast increasing attention. Models like GPT-4 [6] showcase the potential for LLMs to revolutionize task planning with their nuanced understanding and generation of Natural Language. Projects such as ProgPrompt [2] and SayCan [3] leverage LLMs to create context-sensitive, dynamic task sequence, marking a shift towards more intelligent and adaptable robotic systems. However, the challenge of integrating robust safety constrains into plans generated by Large Language Models (LLM) persists. Some approaches, such as Plug in the Safety Chip [7], enhance task plan safety by encoding safety constraints into logical language at the task planning layer, yet they seldom address safety concerns at the motion planning layer.

B. LTL as Safety Constraints for Robot

Linear Temporal Logic (LTL) has emerged as a pivotal tool for formulating temporal task specifications across a broad spectrum of planning and learning endeavors, notably in the domains of robotics and artificial intelligence [13], [14]. Its utility extends to embedding robust safety mechanisms within robotic operations, offering a methodical approach to evaluate whether sequence of actions align with predefined LTL constraints for ensuring operational safety [15], [16]. The successful use of LTL in projects such as [17] demonstrates its effectiveness in defining action constraints for ensuring safety in complex and dynamic tasks.

C. Hierarchical Task and Motion Planning

Reflecting on advancements in hierarchical task and motion planning, we see a crucial need for integrating high-layer planning with low-layer execution. Studies like [18] highlight the development of planner-independent interfaces that bridge task and motion planning, while works such as [19] and [20] focus on enabling robots to navigate and fulfill tasks reactively in dynamic environments. Despite these advances, a common challenge across these efforts, including [18], [19], and [20], is the potential for motion planning to inadvertently overlook high-layer constraints, thereby risking constraint violations. [21] addresses this by modifying automata for task-layer constraints, but the risk of violating motion layer constraints persists.

III. Preliminaries

A. Linear Temporal Logic

In this work, Linear Temporal Logic (LTL) serves as an expressive language for defining complex task requirements. The syntax of an LTL formula Φ is defined over a set of atomic propositions AP as $\Phi ::= \text{true}\ |ap|\neg| \wedge |X|U|F|G$, where true is the Boolean value, $ap \in AP$ is an atomic proposition, $\neg$(negation) and $\wedge$ (conjunction) are standard Boolean operators, X (next), U (until), F (finally) and G (always) are temporal operators.

An LTL formula can be converted into a Nondeterministic Büchi Automaton (NBA) using existing tools, such as LTL2BA [12]. Such an NBA is defined as a tuple $B_\phi = (2^{AP}, S, \delta, s_0^\phi, S_F)$, where 2^{AP} is the alphabet represented the power set of AP, S is a finite set of states, $s_0^\phi \in S$ is the initial state, $\delta \subseteq S \times S$ is a set of labels between state transition (s_i^ϕ, s_{i+1}^ϕ), $s_i^\phi, s_{i+1}^\phi \in S$, and $S_F \in S$ is the set of accepting states. Let B_ϕ denote the NBA generated from ϕ. The word $\mathbf{\Sigma} = \sigma_0\sigma_1\sigma_2\ldots$ is an infinite sequence where $\sigma_i \in 2^{AP}$, $\forall\ i \geq 0$. We denote by $s_i^\phi \xrightarrow{\sigma_i} s_{i+1}^\phi$ if $\sigma_i \in \delta(s_i^\phi, s_{i+1}^\phi)$. Over B_ϕ, the word $\mathbf{\Sigma}$ can generate the corresponding trajectory $\boldsymbol{s}^\phi = s_0^\phi s_1^\phi s_2^\phi \ldots$ If the input word $\mathbf{\Sigma}$ can generate at least one run $\boldsymbol{s}$ that intersects S_F infinitely many times, B_ϕ is said to accept $\mathbf{\Sigma}$. Therefore, the way to judge whether a word $\mathbf{\Sigma}$ is satisfied with task constraint ϕ is whether word $\mathbf{\Sigma}$ can be accepted by B_ϕ.

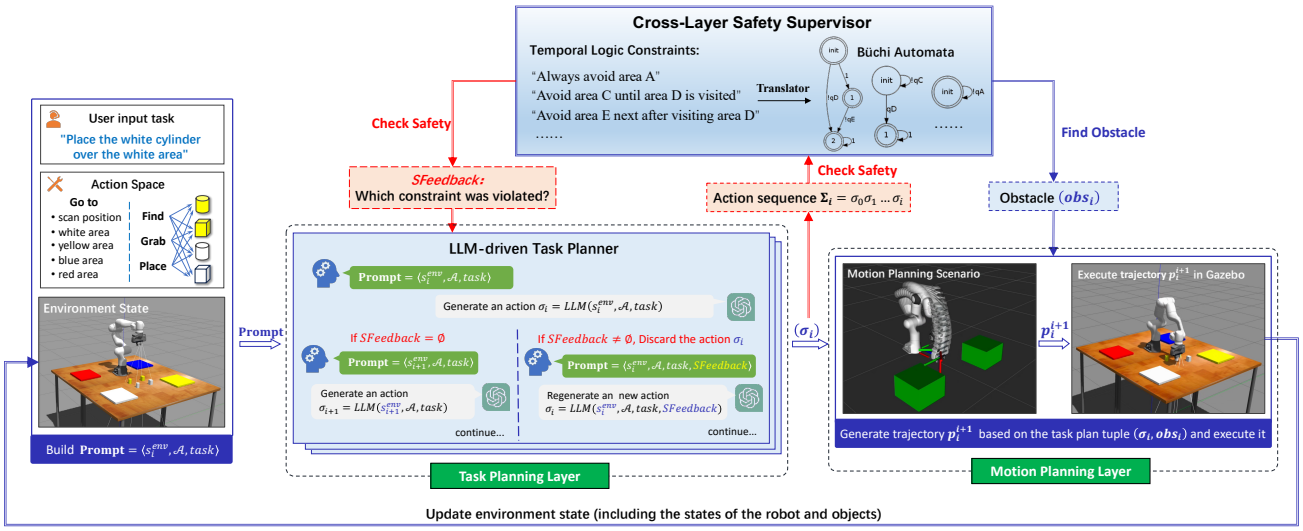

Fig. 2. Our method begins with the user providing a task command in Natural Language, forming a prompt tuple with the action space and environment state. The task planner employs LLM's semantic understanding to generate a reference task plan (action). A cross-layer safety supervisor detects potential violations of safety constraints with each action. If a violation is detected, the action is discarded at the task planning layer, and a new action is generated based on feedback information from supervisor. During action execution, the supervisor identifies "obstacles", representing areas restricted by LTL constraints. The resulting task plan tuple, comprising action and obstacle, is sent to the motion planner for computing a safe trajectory.

B. Task Planning Formulation

In this work, describing the operation of the LLM-driven task planner involves the utilization of the concept of a state transition system. Here, we model the task planning process as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, s_0^{env}, task, prompt)$. Specifically, $\mathcal{S}$ represents the set of environment states (including the state of the robot and objects), with s_0^{env} denoting the initial state. $\mathcal{A}$ is a set of executable actions, defined based on the current environment state $s_i^{env} \in \mathcal{S}$. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denotes the transition relationship between states. $task$ is the task description provided by the user. $prompt$ is a prompt tuple $(s_i^{env}, task, \mathcal{A})$ constructed for LLM.

Furthermore, building upon the knowledge in Section III-A, to ensure that the robot's actions result in state transitions in both the environment and the automaton, we define the action set $\mathcal{A}$ as a subset of the atomic proposition set AP, i.e., $\mathcal{A} \subseteq AP$. Specifically, for an action $\sigma_i \in \mathcal{A}$, it holds that $s_i^{\phi_j} \xrightarrow{\sigma_i} s_{i+1}^{\phi_j}$ and $s_i^{env} \xrightarrow{\sigma_i} s_{i+1}^{env}$. This alignment creates a connection between environmental state transitions and NBA state transitions through the action σ_i generated by LLM-driven task planner. Therefore, we can determine task completion by assessing whether the final state of the environment reaches the goal state. Simultaneously, we can identify whether an action violates safety constraints by assessing whether all NBAs reach accepting states. In Section IV-B, we will delve into this point in detail and propose a corrective action mechanism for LLM-driven task planner to ensure safety in the task planning.

C. Task Plan Tuple

We define task plan as a tuple $\mathbf{\Pi} = (\mathbf{\Sigma}, \boldsymbol{\mathcal{O}})$, where $\mathbf{\Sigma} = \sigma_0\sigma_1\sigma_2 \ldots$ is a sequence of actions (atomic propositions), with σ_0 representing *False*, and $\boldsymbol{\mathcal{O}} = obs_0 obs_1 obs_2 \ldots$ is a sequence of "obstacles". We define $\pi_i = (\sigma_i, obs_i)$ as a task plan tuple for the i-th step, signifying the need to avoid obstacle obs_i while executing the action σ_i. Thus, the results of task planning can be reformulated as $\mathbf{\Pi} = \pi_0\pi_1\pi_2 \ldots$.

Furthermore, we strive to distinguish the task plan of our proposed method from traditional results, which are often limited to mere sequence of actions. We introduce "obstacles" elements into task plan, expecting to guide motion planner through safety supervisor. It should be noted that "obstacles" do not refer to physical obstructions but rather denotes areas that the robot is not allowed to enter based on safety constraints restrictions, corresponding to actions that violate these constraints. We aim to introduce illegal actions identified by the safety supervisor as "obstacles" into the motion planning scenario to ensure safety at the motion planning layer, as discussed more in-depth in Section IV.

IV. Methodology

This section introduces the proposed method in detail. The overall method is to divide the safe planning problem into a LLM-driven task planning layer and a motion planning layer and add a cross-Layer safety supervisor to guide the behavior of the task planner and motion planner respectively. Specifically, Section IV-A outlines the safety supervisor's key roles: "CheckSafety" for the task planning layer and "FindObstacle" for the motion planning layer, based on Linear Temporal Logic (LTL) constraints. Section IV-B discusses how the task planner, under the "CheckSafety" function guidance, ensures the task plan comply with safety constraints, correcting it as needed. Section IV-C details how the "Find Obstacle" function assists the motion planner by designating unsafe actions as "obstacles".

A. LTL Constraints as a Safety Supervisor

This work uses LTL syntax to describe safety constraints. Suppose there is a set of task constraints $\Phi = \{\phi_1, \phi_2, \ldots \phi_n\}$, according to the previous discussion in Section III, these n constraints can be converted into a

Algorithm 1: CheckSafety Function

Input: Action Sequence $\mathbf{\Sigma}_i$, Action σ_{i+1}, NBA Set $\mathcal{B}_\Phi$;
Output: Violated Constraint Information *SFeedback*;
SFeedback ← Empty list to store violated constraints;
for *each* $B_\phi \in \mathcal{B}_\Phi$ **do**
  if $B_\phi(\mathit{Append}(\mathbf{\Sigma}_i, \sigma_{i+1})) = \mathit{False}$ **then**
    Append constraint information of ϕ to *SFeedback*;
if *SFeedback* $\neq \emptyset$ **then**
  Discard action σ_{i+1};
  return *SFeedback*;
else
  return *True*;

Algorithm 2: FindObstacle Function

Input: Action Sequence $\mathbf{\Sigma}_{i-1}$, Action σ_i, NBA Set $\mathcal{B}_\Phi$, Atomic Proposition Set AP;
Output: Obstacle obs_i;
$\boldsymbol{\sigma}_i$ ← Extract action sequence from $\mathbf{\Pi}_i$;
obs_i ← Empty set to store i-th step obstacle ;
for *each* $B_\phi \in \mathcal{B}_\Phi$ **do**
  for ap *in* AP/σ_i **do**
    if $\mathbf{\Sigma}_{i-1} = \mathbf{\Sigma}_0 = \sigma_0$ **then**
      Add $\emptyset$ to obs_0
    if $ap \wedge \sigma_i \notin \delta_{B_\phi}(s_{i-1}, s_i)$ **then**
      Add ap to obs_i;
return obs_i;

set of NBAs $\mathcal{B}_\Phi = \{B_{\phi_1}, B_{\phi_2}, \dots B_{\phi_n}\}$. $\mathcal{B}_\Phi$ is defined as the safety supervisor and plays different roles in the task planning and motion planning.

For the LLM-driven task planning layer, the safety supervisor monitors LLM's decisions in real time, and every action generated by the task planner is detected. If an action violates safety constraints, the action will be discarded, and the safety supervisor will immediately feedback the constraint information violated by the action to the task planner. This information assists the LLM-driven task planner in regenerating a new action that comply with safety constraints. This process is referred to as the safety supervisor's "CheckSafety" function. Alg. 1 outlines how this function works. Suppose we have a sequence of task plan tuples $\mathbf{\Pi}_{\mathit{current}} = \pi_0\pi_1\pi_2\dots\pi_{\mathit{current}}$. The determination of whether the current task plan tuple π_{current} violates constraints depends on the action sequence $\mathbf{\Sigma}_{\mathit{current}} = \sigma_0\sigma_1\sigma_2, ...\sigma_{\mathit{current}}$ within $\mathbf{\Pi}_{\mathit{current}}$. For any $\phi \in \Phi$, if the action sequence $\mathbf{\Sigma}_{\mathit{current}}$ violates the constraint ϕ (indicated by $B_\phi(\mathbf{\Sigma}_{\mathit{current}}) = \text{false}$), this implies that the action $\sigma_{\mathit{current}}$ is considered unsafe. In response, the safety supervisor will provide corresponding constraint information associated with ϕ to assist the LLM task planner in correcting errors.

For the motion planning layer, the safety supervisor provide motion planner with obstacle sequence (where cannot go). This process is called the safety supervisor's "FindObstacle" function. Alg. 2 outlines how this function works. Suppose we have a sequence of "obstacles" $\boldsymbol{\mathcal{O}}_{\mathit{current}} = obs_0 obs_1 obs_2 \dots obs_{\mathit{current}}$, where obs_i indicates the constraints that should be taken into account during the transition from π_{i-1} to π_i. For each $\pi_i = (\sigma_i, obs_i)$, if $ap \wedge \sigma_i, \forall ap \in AP$ is not in $\delta_{B_\phi}(s_{i-1}, s_i)$ for all $B_\phi \in \mathcal{B}_\Phi$, it indicates that when the robot executes π_i, action ap is not executable. In other words, the key area corresponding to action ap should not be accessible to the robot. Therefore, action ap is added to obs_i, and it is subsequently used in the motion planning for executing action σ_i. This ensures that the robot avoids violating constraints during its motion by updating the "obstacles" information obs_i in the motion planning.

B. LLM-driven Task Planning with Correction Mechanism

In our proposed approach, the LLM serves as the central link between the robot and the environment, acting as a task planner responsible for generating actions for the entire task or correcting actions for the current step based on prompts. To implement this process, as mentioned in Section III-B , we model these prompts for LLM using the tuple $(s_i^{\mathit{env}}, task, \mathcal{A})$, where $s_i^{\mathit{env}} \in \mathcal{S}$ represents the current environment state (including the state of the robot and objects), $task$ represents the task text instructions defined by the user for robot, and $\mathcal{A}$ represents the current set of executable actions. The process of LLM generating action sequence can be expressed as:

$$\begin{cases} LLM(\mathcal{S}, task, \mathcal{A}) = \{\sigma_i \mid i = 0, 1, \dots, n\} \\ \sigma_i \in \mathcal{A} = AP \end{cases} \tag{1}$$

where $LLM(\mathcal{S}, task, \mathcal{A})$ denotes the LLM task planner, σ_i represents the generated action at the i-th step, and it has the ability to trigger state transitions in the environment state and NBAs, as mentioned in Section III-B. This way generates a chain transfer system as follows:

$$\begin{cases} \mathcal{S} : s_0^{\mathit{env}} \xrightarrow{\sigma_0} s_1^{\mathit{env}} \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_n} s_{\mathit{success}}^{\mathit{env}} \\ B_{\phi_1} : s_0^{\phi_1} \xrightarrow{\sigma_0} s_1^{\phi_1} \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_n} s_n^{\phi_1} \\ \quad \vdots \\ B_{\phi_n} : s_0^{\phi_n} \xrightarrow{\sigma_0} s_1^{\phi_n} \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_n} s_n^{\phi_n} \end{cases} \tag{2}$$

where $s_i^{\mathit{env}} \in \mathcal{S}$ denotes the environment state and $s_i^{\phi_j}$ denotes the state of B_{ϕ_j}. $s_{\mathit{success}}^{\mathit{env}}$ and $s_k^{\phi_j} \in S_F^{\phi_j}$ represent the final success environment state and the accepted state of B_{ϕ_j}, respectively. However, due to limitations of LLM knowledge base, each decision it makes may not consistently comply with specified safety constraints. Therefore, it becomes crucial to evaluate the legality of LLM's decisions and utilize this information to assist LLM in correcting action sequence.

As discussed in Section IV-A, the cross-layer safety supervisor has ability to assess whether the actions σ_i generated by LLM violate safety constraints. If the check fails, this safety supervisor will provide feedback containing constraint information. Upon capturing this feedback, it is necessary

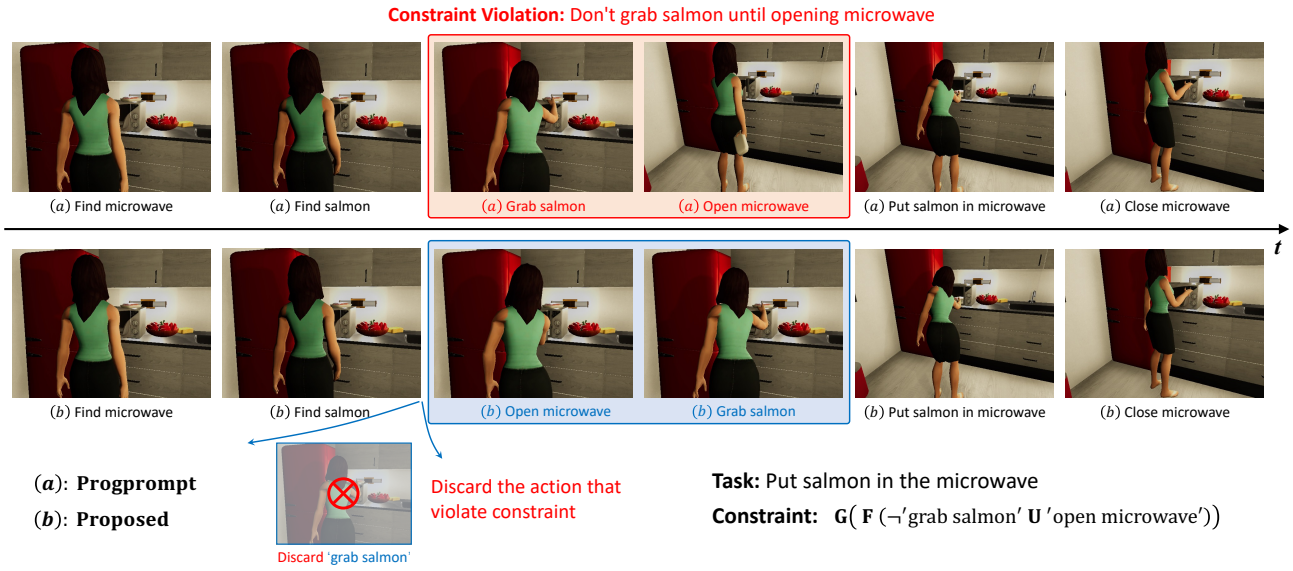

Fig. 3. Compared with ProgPrompt [2], the proposed method shows better performance in VH simulator. In the case where the action "*Grab salmon*" violates the constraint, the proposed method learns from the constraint information, discards the action and regenerates a new action "*Open microwave*". This error correction prevents constraint violations from occurring during task completion.

TABLE I

PERFORMANCE COMPARISON BETWEEN PROPOSED METHOD (TASK PLANNER PART) AND PROGPROMPT IN VIRTUAL HOME TESTING TASKS.

Task Description	$\|A\|$	Proposed Method (Task Planner Part)			ProgPrompt [2]		
		Safety rate	Success rate	Exec	Safety rate	Success rate	Exec
Put salmon in the microwave	7.2	1.00 ± 0.00	0.40 ± 0.55	0.73 ± 0.26	0.40 ± 0.55	0.40 ± 0.55	0.91 ± 0.08
Watch TV	4.0	1.00 ± 0.00	0.00 ± 1.00	1.00 ± 0.00	0.20 ± 0.45	0.40 ± 0.55	0.80 ± 0.21
Turn off light	4.2	1.00 ± 0.00	0.60 ± 0.55	0.90 ± 0.14	0.00 ± 0.00	0.80 ± 0.45	0.95 ± 0.11
Brush teeth	8.4	1.00 ± 0.00	0.40 ± 0.55	0.91 ± 0.09	0.00 ± 0.00	0.20 ± 0.45	0.82 ± 0.11
Throw away apple	8.6	1.00 ± 0.00	0.20 ± 0.45	0.66 ± 0.27	0.40 ± 0.55	0.40 ± 0.55	0.83 ± 0.16
Make toast	8.2	1.00 ± 0.00	0.00 ± 0.00	0.61 ± 0.30	0.20 ± 0.45	0.00 ± 0.00	0.60 ± 0.10
Eat chips on the sofa	6.4	1.00 ± 0.00	0.80 ± 0.45	0.00 ± 0.00	0.00 ± 0.00	0.80 ± 0.45	0.94 ± 0.13
Wash the plate	15.0	1.00 ± 0.00	0.00 ± 0.00	0.55 ± 0.13	0.00 ± 0.00	0.00 ± 0.00	0.72 ± 0.08

to discard the action σ_i, add the constraint information to prompts, and prompt LLM to regenerate a new action. The formula for action correction is as follows:

$$\begin{cases} LLM(s_i^{env}, task, \mathcal{A}, SFeedback) = \sigma_{correct} \\ \sigma_{correct} \in \mathcal{A} = AP/\sigma_i \end{cases} \tag{3}$$

where $\sigma_{correct}$ represents the corrected action, and AP/σ_i denotes the set of actions excluding the unsafe action σ_i.

It should be noted that since the action generated in task planning are not immediately executed, the unsafe action σ_i does not alter the environment state s_i^{env}. Therefore, the correction of the erroneous decision by LLM still depends on the state s_i^{env}. The action $\sigma_{correct}$ regenerated by LLM based on constraint information will replace the original unsafe action σ_i, thereby preventing the robot from entering an unsafe state at the task planning layer.

C. Safety-constrained Motion Planning

In this work, another focus is on ensuring the safety of the robot during the task execution, i.e. ensuring safety at the motion planning layer. Section IV-B ensures the generation of action sequence compliant with safety constraints at the task planning layer. However, during the execution of actions, the robot's trajectory may still intersect with unsafe regions. Therefore, the motion planning layer also needs to consider safety constraints defined by LTL syntax.

As discussed in Section IV-A, the safety supervisor can obtain "obstacles" obs_i during the execution of action σ_i based on LTL constraints. Consequently, the motion planner obtains the i-th step task plan tuple $\pi_i = (\sigma_i, obs_i)$ from both the task planner and the safety supervisor. To ensure the generalization capability of the proposed method, we do not explicitly specify the trajectory generation algorithm used at the motion planning layer. However, the trajectory plan p_i^{i+1} generated by the algorithm must adhere to the task plan tuple $\pi_i = (\sigma_i, obs_i)$. In other words, when executing action σ_i, the robot must avoid the region where "obstacles" obs_i is marked in workspace M.

V. SIMULATION AND EXPERIMENTS

To validate the effectiveness and versatility of our proposed method across diverse scenarios and objects, we conducted a series of experiments, including both virtual simulations and real-world experiments. Specifically, Section V-A demonstrates the performance of task plan correction mechanism brought by safety supervisor to task planning layer in the Virtual Home Simulation. Section V-B and

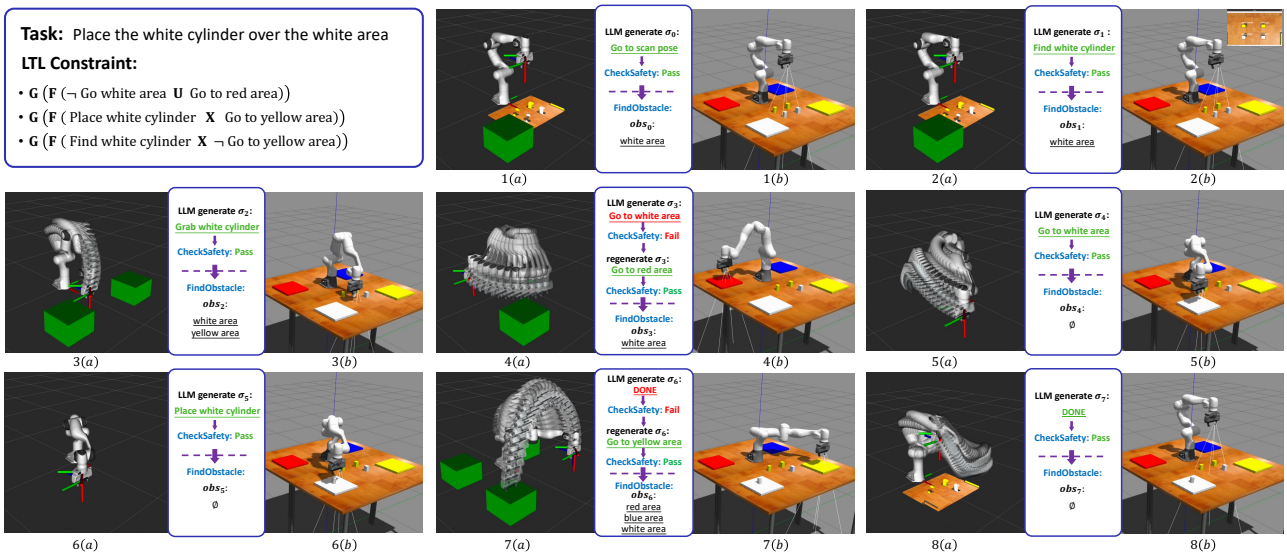

Fig. 4. Implementation of proposed method in gazebo engine. This framework provides an action sequence to complete the task without violating LTL constraints, and presents constraint-violating behaviors in the motion planning scenario in the form of "obstacles". This method ensures that the robot does not violate constraints at both task and motion planning layers.

V-C showcase the cross-layer characteristics of the safety supervisor, providing safety assurance at both task and motion planning layers. However, Section V-B is deployed on the Franka Robotic Arm in the Gazebo engine, while Section V-C is deployed on a real TurtleBot mobile robot. All experiments were performed using GPT-4.0, with Gazebo simulations built on ROS Noetic and Ubuntu 20.04 as the underlying infrastructure.

Evaluation Metrics: We use three metrics to evaluate system performance: safety rate, success rate, and executability (Exec). **Safety rate** is defined as the fraction of the number of times task constraints are satisfied during task execution to the total number of tasks. **Success rate** is defined as the fraction of the number of successful task completions to the total number of tasks. **Executability (Exec)** is defined as the fraction of actions in the task plan that are executable in the environment, even if they are not relevant for the task.

A. Virtual Home Simulation

First, we validate our proposed method on a virtual home environment containing a large number of daily scenes and objects. Here, we focus on evaluating the performance of safety supervisor at the task planning layer, specifically the effectiveness of the task plan correction mechanism for LLM-driven task planner. In the simulation environment, the agent executed multi-step task plans (action sequence) based on the output generated by LLM, and occasionally the task plan (action) violated predefined safety constraints. Through a series of task tests, our proposed method effectively copes with these challenges compared with ProgPrompt [2].

As depicted in Fig. 3, consider that the task text input is "$Put\ salmon\ in\ the\ microwave$", and the task safety constraint is expressed in LTL syntax as $\phi = G\,(\,F\,(\neg\ Grab\ salmon\ U\ Open\ microwave)\,)$. This constraint, set by humans based on specific constraint information, implies that the agent cannot open the microwave oven while holding the salmon, making it crucial to avoid this unsafe operation. ProgPrompt generates an action sequence capable of completing the task based on the predefined action space. However, errors occur during the execution of the "$Grab\ salmon$" and "$Open\ microwave$" actions. The reason for this failure is that, despite inputting the constraints of Linear Temporal Logic (LTL) into ProgPrompt in Natural Language (NL) format, there is still a probability that the task planner may overlook the existence of this NL constraint during task planning. Specifically, when ProgPrompt improperly generates the action "$Grab\ salmon$", there is no feedback mechanism in place to indicate that this action violates the specified constraint. In the method we proposed, after obtaining an action, the safety supervisor determines whether this action violates the safety constraints. The action that violate constraints will be discarded, and a new action will be regenerated based on the supervisor's feedback with constraint information. As shown in Fig. 3(b), the system discards the action "$Grab\ salmon$" after acquiring it and provides a new action "$Open\ microwave$", thereby avoiding the possibility of violating safety constraints at the task planning layer. Table I shows the evaluation results of the matrix using ProgPrompt and the proposed method.

Throughout the experiments, we tested eight task scenarios, each with a safety constraint. Five iterations were performed for each task using both ProgPrompt and the proposed method, with the results averaged.

B. Franka Robotic Arm Gazebo Simulation

To comprehensively evaluate the cross-layer characteristics of the safety supervisor, providing dual-layer safety assurance for both task and motion planning layers, we devised an object sorting task for the Franka Panda robot in the Gazebo physics engine. We pre-defined the Panda robot's action skill library, including "Go", "$Find$", "$Grab$", and "$Place$". The LLM-driven task planner selects actions from this skill

TABLE II

PERFORMANCE EVALUATION OF PROPOSED METHOD (ABLATION STUDY).

Task Description	Obstacle	$\|A\|$	LTL Constraint for LLM Task Planner			NL Constraint for LLM Task Planner		
			Safety rate	Success rate	Exec	Safety rate	Success rate	Exec
Place white cylinder over white area	by LTL	7.8	1.00 ± 0.00	0.60 ± 0.55	0.73 ± 0.38	0.20 ± 0.45	0.60 ± 0.55	0.77 ± 0.32
Place all cylinders over white area	by LTL	13.2	1.00 ± 0.00	0.40 ± 0.55	0.58 ± 0.40	0.20 ± 0.45	0.20 ± 0.45	0.61 ± 0.26
Place all boxes over yellow area	by LTL	14.4	1.00 ± 0.00	0.40 ± 0.55	0.65 ± 0.33	0.00 ± 0.00	0.40 ± 0.55	0.63 ± 0.35
Place all objects according to color	by LTL	19.8	1.00 ± 0.00	0.00 ± 0.00	0.43 ± 0.09	0.00 ± 0.00	0.00 ± 0.00	0.54 ± 0.23
Place white cylinder over white area	None	7.8	0.40 ± 0.55	0.60 ± 0.55	0.91 ± 0.20	0.00 ± 0.00	0.60 ± 0.55	0.93 ± 0.09
Place all cylinders over white area	None	14.0	0.40 ± 0.55	0.80 ± 0.45	1.00 ± 0.00	0.00 ± 0.00	0.40 ± 0.55	0.91 ± 0.13
Place all boxes over yellow area	None	14.8	0.20 ± 0.45	0.80 ± 0.45	0.92 ± 0.17	0.00 ± 0.00	0.60 ± 0.55	0.89 ± 0.15
Place all objects according to color	None	20.4	0.40 ± 0.55	0.00 ± 0.00	0.84 ± 0.25	0.00 ± 0.00	0.00 ± 0.00	0.88 ± 0.18

library to generate action sequence for completing the user-inputted task. As depicted in Fig. 4, consider that the task input is "*Place white cylinder over white area*", and the task constraints are explicitly formalized using LTL synatx, represented by the following formulas:

$$\begin{cases} \phi_1 = G\,(\,F\,(\neg\, Go\ to\ white\ area\ U\ Go\ to\ red\ area\,)\,) \\ \phi_2 = G\,(\,F\,(\,Place\ white\ cylinder\ X\ Go\ to\ yellow\ area\,)\,) \\ \phi_3 = G\,(\,F\,(\,Find\ white\ cylinder\ X \neg\, Go\ to\ yellow\ area)\,) \end{cases}$$

As depicted in Fig. 4 - 4, LLM-driven task planner generated the action "*Go to white area*", which was flagged by the safety supervisor's "CheckSafety" function as violating safety constraint ϕ_1. Consequently, the action was replaced with "*Go to red area*", which meets safety requirements. However, executing the action "*Go to red area*" may still inadvertently traverse the "*white area*" posing a risk of safety violation. To address this, the safety supervisor's "FindObstacle" function identifies and marks the "*white area*" as an "obstacle" in the motion planning scenario to ensure safety at the motion layer, as shown in Figure 4-4(b). Thus, the safety supervisor provides dual-layer safety assurance by applying "CheckSafety" to task planning and "FindObstacle" to motion planning.

To rigorously evaluate the safety supervisor, we test each sorting task through five iterations , presenting the results in Table II. The evaluation focused on how the safety supervisor's key functionalities (CheckSafety and FindObstacle) influenced the success rate and safety rate. As shown in Table II, the cross-layer safety supervisor enables the robot to exhibit stronger safety characteristics at both task and motion planning layers, thereby achieving the 100% safety rate.

C. TurtleBot Real World Experiment

To further validate the generalization capability of our proposed method, we designed a search and rescue task for TurtleBot mobile robot. The task scenario is illustrated in Fig. 5, where the robot must search and rescue in damaged Areas1 and Area2. The task input is defined as "*Cover Area1 and Area2*". Before initiating the rescue, the robot must first visit the tool store to acquire equipment, and prior to that, it must recharge at the base. Entering damaged areas without these steps poses safety risks, making it essential to follow the sequence of visiting the base, then the tool store, before proceeding with the rescue. These safety constraints are expressed in LTL syntax as follows:

$$\begin{cases} \phi_1 = G\,(\,F(\neg\,(\,Go\ to\ Area1 \vee Go\ to\ Area2)\ U\ Go\ to\ Tool\,)\,) \\ \phi_2 = G\,(\,F(\neg\, Go\ to\ Tool\ U\ Go\ to\ Base\,)\,) \end{cases}$$

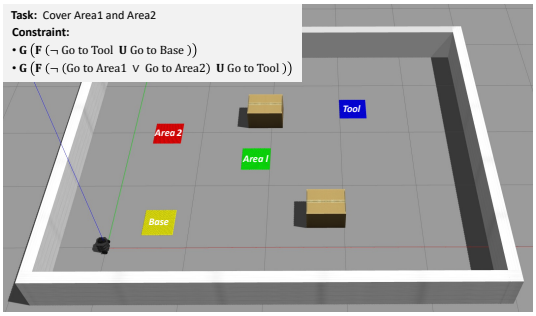

Fig. 5. The working environment of the rescue robot.

Fig. 6 illustrates the safety assurance that our proposed method can achieve. Additionally, we conducted TurtleBot experiments in the real-world environment, as depicted in Fig. 7. In Fig. 6(a), the path planned by our framework is depicted, while Fig. 6(b) shows the path planned without the safety supervisor's "FindObstacle" function. Fig. 6(c) displays the path planned after converting LTL constraints to NL constraints, i.e., removing the "CheckSafety" function. Finally, Fig. 6(d) demonstrates the path planned with only NL constraints. Based on the generated multi-step task plan tuples $\Pi = \pi_0\pi_1\ldots$, it is evident that our proposed approach fully reflects the intention of safety constraints during task completion. NL constraints cannot provide the "CheckSafety" function brought by LTL constraints (determining whether the action in the task plan tuple violates safety constraints), nor can they convey the information of "where can not go" to the motion planner. Therefore, our proposed cross-layer safety supervisor, based on LTL synatx to describe safety constraints, enables the task plan to have more comprehensive safety assurance.

VI. CONCLUSION

In this paper, we propose a cross-layer sequence supervision mechanism to enhance safety in LLM-driven robotic systems, covering both task and motion planning. We use Linear Temporal Logic (LTL) to define safety constraints and convert them into a Nondeterministic Büchi Automatons (NBAs). This supervisor detects and corrects actions that violate safety constraints, preventing unsafe behaviors and

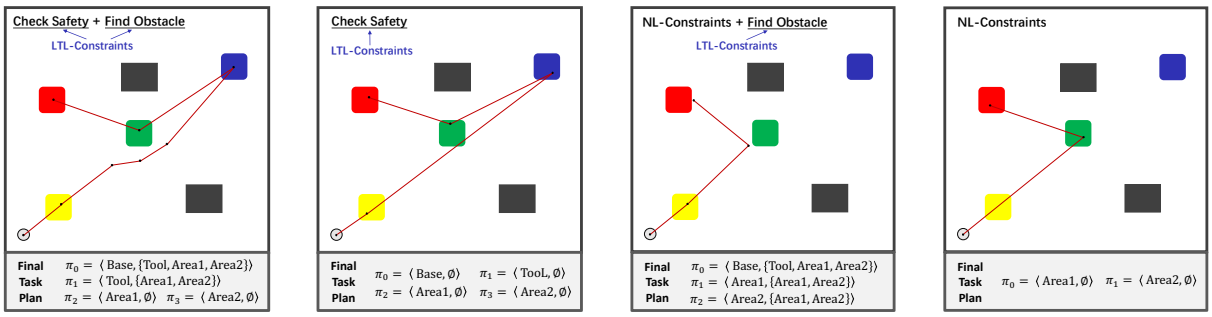

Fig. 6. Illustration of the safety assurance provided by the proposed framework for LLM-driven robots. (a) Represents the path planned by our proposed method and the planned path from Base to Tool avoids Area1. (b) Represents the path planned without the "FindObstacle" function and the planned path from Base to Tool is through Area1. (c) Represents the path planned without the "CheckSafety" function and the planned path does not pass through the Tool before going to Area 1 and the planned path does not eventually enter Area1 and Area2 because they are marked as obstacle. (d) Represents the path planned using only NL constraints and the planned path did not pass through Tool before heading to Area1.

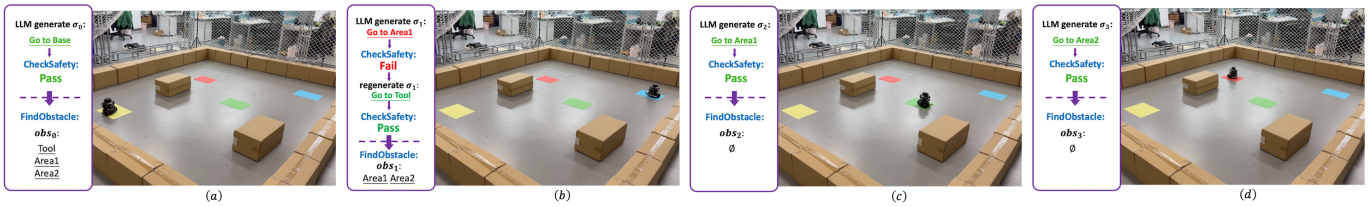

Fig. 7. Our proposed method can operate on the real TurtleBot robot. If the action generated by LLM-driven task planner violates safety constraints, the action will be discarded. The safety supervisor guides task planner to regenerate a new action and provide "obstacles" information to the motion planner.

improving safety in complex environments. Our experiments validate the method's effectiveness and significant safety improvements, highlighting its potential for future research and application in LLM-driven robotics.

References

[1] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "LLM-planner: Few-shot grounded planning for embodied agents with large language models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2998–3009, 2023.

[2] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530, IEEE, 2023.

[3] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning*, pp. 287–318, PMLR, 2023.

[4] C. Ming, J. Lin, P. Fong, H. Wang, X. Duan, and J. He, "HiCRISP: A hierarchical closed-loop robotic intelligent self-correction planner," *arXiv preprint arXiv:2309.12089*, 2023.

[5] S. Sharan, F. Pittaluga, M. Chandraker, *et al.*, "LLM-assist: Enhancing closed-loop planning with language-based reasoning," *arXiv preprint arXiv:2401.00125*, 2023.

[6] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[7] Z. Yang, S. S. Raman, A. Shah, and S. Tellex, "Plug in the safety chip: Enforcing constraints for LLM-driven robot agents," *arXiv preprint arXiv:2309.09919*, 2023.

[8] C. Zheng, F. Yin, H. Zhou, F. Meng, J. Zhou, K.-W. Chang, M. Huang, and N. Peng, "Prompt-driven LLM safeguarding via directed representation optimization," *arXiv preprint arXiv:2401.18018*, 2024.

[9] X. Wu, R. Xian, T. Guan, J. Liang, S. Chakraborty, F. Liu, B. Sadler, D. Manocha, and A. S. Bedi, "On the safety concerns of deploying LLMs/VLMs in robotics: Highlighting the risks and vulnerabilities," *arXiv preprint arXiv:2402.10340*, 2024.

[10] T. Yuan, Z. He, L. Dong, Y. Wang, R. Zhao, T. Xia, L. Xu, B. Zhou, F. Li, Z. Zhang, *et al.*, "R-Judge: Benchmarking safety risk awareness for LLM agents," *arXiv preprint arXiv:2401.10019*, 2024.

[11] E. Clarke, O. Grumberg, and K. Hamaguchi, "Another look at LTL model checking," in *Computer Aided Verification: 6th International Conference, CAV'94 Stanford, California, USA, June 21–23, 1994 Proceedings 6*, pp. 415–427, Springer, 1994.

[12] P. Gastin and D. Oddoux, "Fast LTL to büchi automata translation," in *Computer Aided Verification: 13th International Conference, CAV 2001 Paris, France, July 18–22, Proceedings 13*, pp. 53–65, Springer, 2001.

[13] Z. Chen, Z. Zhou, S. Wang, and Z. Kan, "A hierarchical decoupling approach for fast temporal logic motion planning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1579–1585, IEEE, 2023.

[14] H. Zhang, H. Wang, and Z. Kan, "Exploiting transformer in sparse reward reinforcement learning for interpretable temporal logic motion planning," *IEEE Robotics and Automation Letters*, 2023.

[15] M. Foughali, S. Bensalem, J. Combaz, and F. Ingrand, "Runtime verification of timed properties in autonomous robots," in *2020 18th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pp. 1–12, IEEE, 2020.

[16] J. Schumann, P. Moosbrugger, and K. Y. Rozier, "R2U2: Monitoring and diagnosis of security threats for unmanned aerial systems," in *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22-25, 2015. Proceedings*, pp. 233–249, Springer, 2015.

[17] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1525–1532, IEEE, 2014.

[18] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *2014 IEEE international conference on robotics and automation (ICRA)*, pp. 639–646, IEEE, 2014.

[19] Y. Kantaros, M. Malencia, V. Kumar, and G. J. Pappas, "Reactive temporal logic planning for multiple robots in unknown environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11479–11485, IEEE, 2020.

[20] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 1002–1028, 2020.

[21] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3602–3621, 2022.