

# LiP-LLM: Integrating Linear Programming and Dependency Graph With Large Language Models for Multi-Robot Task Planning

Kazuma Obata<sup>1</sup>, Tatsuya Aoki<sup>1</sup>, Takato Horii<sup>1</sup>, *Member, IEEE*, Tadahiro Taniguchi<sup>2</sup>, *Member, IEEE*, and Takayuki Nagai<sup>1</sup>

**Abstract**—This study proposes LiP-LLM: integrating linear programming and dependency graph with large language models (LLMs) for multi-robot task planning. For multi-robots to efficiently perform tasks, it is necessary to manage the precedence dependencies between tasks. Although multi-robot decentralized and centralized task planners using LLMs have been proposed, none of these studies focus on precedence dependencies from the perspective of task efficiency or leverage traditional optimization methods. It addresses key challenges in managing dependencies between skills and optimizing task allocation. LiP-LLM consists of three steps: skill list generation and dependency graph generation by LLMs, as well as task allocation using linear programming. The LLMs are utilized to generate a comprehensive list of skills and to construct a dependency graph that maps the relationships and sequential constraints among these skills. To ensure the feasibility and efficiency of skill execution, the skill list is generated by calculated likelihood, and linear programming is used to optimally allocate tasks to each robot. Experimental evaluations in simulated environments demonstrate that this method outperforms existing task planners, achieving higher success rates and efficiency in executing complex, multi-robot tasks. The results indicate the potential of combining LLMs with optimization techniques to enhance the capabilities of multi-robot systems in executing coordinated tasks accurately and efficiently. In an environment with two robots, a maximum success rate difference of 0.82 is observed in the language instruction group with a change in the object name.

**Index Terms**—Multi-robot systems, task planning.

## I. INTRODUCTION

THE integration of multi-robot systems into various environments enhances the execution of a broad spectrum of tasks. For an efficient performance, it is necessary to manage

the precedence dependencies between tasks and identify the tasks to be executed in parallel. A key challenge in multi-robot task planning is managing these dependencies from natural language instructions, ensuring that tasks are executed in the correct sequence, thereby increasing the success rate. Additionally, actions without dependencies can be parallelized, further boosting efficiency. In addition, it is also important to consider the diverse characteristics of each robot, such as the different grippers attached to the robotic arms.

The complete automation of the entire process of multi-robot task planning, including task decomposition, remains a significant challenge. Brian et al. [1] proposed SayCan, a task planning method for a single robot that uses LLMs to decompose language instructions into more detailed tasks and accomplish them. However, the relationships between the decomposed tasks are simple sequence structures, and their extension to multiple robots has not been discussed. Kutter et al. [2] modeled tasks as structures with precedence dependencies, and then proposed an approach to execute independent tasks in parallel. However, the modeling is performed manually, and requires human intervention to decompose complex tasks into simpler subtasks. Extensive research has been conducted on multi-robot systems [3], [4]. Most multi-robot task planning systems focus on two main aspects: task decomposition and task-allocation. Task allocation is frequently automated using optimal assignment problems. However, as stated earlier, human involvement is necessary in several cases to ensure accurate task decomposition and the handling of dependencies.

Recently, research on multi-robot task planning has explored the application of LLMs. Two primary approaches have been proposed: decentralized processing, such as RoCo [5], in which multiple agents determine tasks through dialogue [6], [7], and centralized processing, such as SMART-LLM [8], which uses multistage reasoning to assign tasks to each agent [9]. In both approaches, LLMs manage all stages of task planning, including task decomposition and task allocation. Decentralized processing requires textual information exchange between robots, which increases the dialogue history and expands the prompt size with the number of robots. Issues with hallucinations and task allocation were common to both groups. Although LLMs can be applied to various tasks, they struggle to effectively address domain-specific problems, and their performance does not match that of task-specific algorithmic approaches [10].

Received 31 July 2024; accepted 22 November 2024. Date of publication 16 December 2024; date of current version 26 December 2024. This article was recommended for publication by Associate Editor G. A. Sartoretti and Editor M. A. Hsieh upon evaluation of the reviewers' comments. This work was supported by Japan Science and Technology Agency (JST) Moonshot R&D under Grant JPMJMS2011. (Corresponding author: Kazuma Obata.)

Kazuma Obata, Tatsuya Aoki, and Takato Horii are with the Department of Systems Innovation, Graduate School of Engineering Science, Osaka University, Osaka 565-0871, Japan (e-mail: k.obata@rlg.sys.es.osaka-u.ac.jp; t.aoki@rlgsys.es.osaka-u.ac.jp; takato@sys.es.osaka-u.ac.jp).

Tadahiro Taniguchi is with the Department of Informatics, Kyoto University, Kyoto 606-8501, Japan (e-mail: taniguchi@i.kyoto-u.ac.jp).

Takayuki Nagai, deceased, was with the Department of Systems Innovation, Graduate School of Engineering Science, Osaka University, Osaka 565-0871, Japan (e-mail: nagai@sys.es.osaka-u.ac.jp).

This letter has supplementary downloadable material available at <https://doi.org/10.1109/LRA.2024.3518105>, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3518105

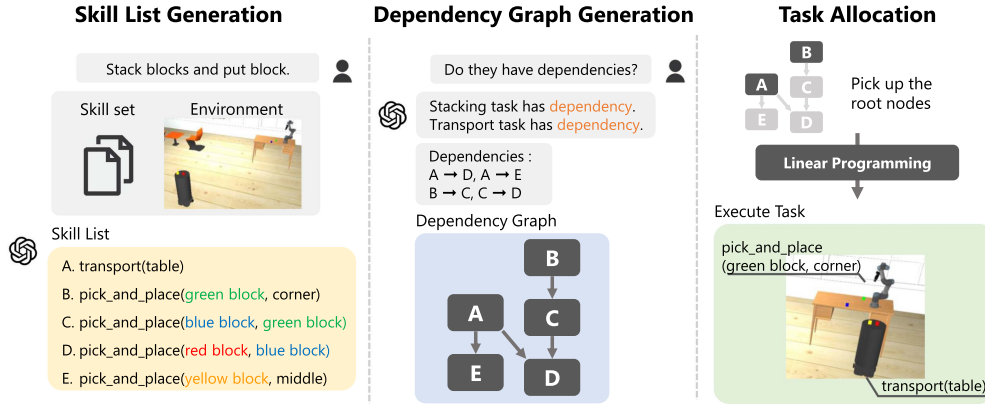


Fig. 1. **LiP-LLM** generates a dependency graph from a skill list and allocates the actions to multiple robots using linear programming. In the skill list generation phase, it creates a skill list from a predefined skill set to accomplish the given instructions. Next, it analyzes the skills in the skill list for precedence dependencies and generates a dependency graph. Finally, it allocates the tasks to each robot using linear programming and executes them.

Leveraging external planners may be useful for addressing this issue. The task allocation problem addressed in this study is classified as Single-Task robots, Single-Robot tasks, Time-extended Assignment(ST-SR-TA) [11], [12]. Because the ST-SR-TA is NP-hard, approaches that approximate it as Single-Task robots, Single-Robot tasks, Instantaneous Assignment(ST-SR-IA) have been proposed, which can be solved using optimization, or heuristic solution. McIntirel et al. [13] proposed an approximation approach for the ST-SR-IA in the context of the ST-SR-TA with precedence constraints between tasks. In this study, the task execution time was not considered, and the difference between the numbers of robots and of assignable tasks in each step was considerably small. Therefore, the approximation approach yields a solution that is sufficiently close to the optimal [12].

This study proposes a multi-robot task planning method, LiP-LLM, which combines linear programming with LLMs. As shown in Fig. 1, LiP-LLM divides task planning into three steps: skill list generation, dependency graph generation, and task allocation. LLMs manage task decomposition and dependencies that are traditionally managed by humans. First, the LLMs generate the required skill list for a particular language instruction using SayCan’s likelihood calculation method to ensure feasible skills. To address the hallucinations caused by LLMs, we employed the likelihood calculation used in SayCan [1] to generate a skill list. Subsequently, a dependency graph is created, linking skills as nodes and dependencies as edges. Finally, tasks are optimally allocated to robots via linear programming, by removing the completed nodes from the graph after execution. This process enables effective multi-robot task allocation.

The contributions of this study are as follows

- 1) We proposed LiP-LLM to handle task dependencies by structuring a graph such that tasks with precedence dependencies are executed in the correct order. This allows us to structure the precedence dependencies between tasks and allocate parallelizable tasks using linear programming.
- 2) The multi-robot task system improves the performance of decentralized and centralized task planners, as proposed in related studies.

## II. RELATED WORKS

### A. Application of LLMs to Robotics

Recently, LLMs have been widely applied to task planning in robotics [14]. Brian et al. [1] proposed SayCan, a system that integrates skill prediction with the LLM and a value function to guide the robotic actions. SayCan works by combining the prediction probability from an LLM, which estimates the usefulness of a skill, with the execution probability from a value function, which assesses the likelihood that a skill can be executed correctly. When commanded, the robot detects a skill based on these combined probabilities, resulting in appropriate and feasible actions, given the language instructions. This method leverages the extensive knowledge embedded in LLMs and supplements it with real-world information, which is a challenge on LLMs. This integration enables generalized action generation, even for commands that contain ambiguity.

Some approaches combine LLMs with classical planners, specifically the Planning Domain Definition Language (PDDL), to perform a variety of planning tasks [15], [16], [17], [18]. These studies demonstrate that converting natural language commands into PDDL using LLMs and then using a conventional planner can provide solutions for a wide range of problems. In their work using LLMs for robot task representation, Cao et al. [19] utilized pre-trained LLMs to generate behavior-tree-based tasks for robots. Similarly, Lin et al. [20] proposed the “Plan Like a Graph,” which calculated the optimal execution time of tasks by transforming a sequence of tasks with ordering constraints into a graph structure using LLMs. Additionally, efforts have been made to leverage LLMs to create robot application programming interfaces and other behavioral programs [21], [22], [23], determine robot behavior [24], [25], [26], and improve the accuracy of robotic actions [27]. As these studies indicate, the LLM can be effectively applied to robotics, suggesting its potential for enhancing robot task planning through human-robot interactions.

### B. Utilization of LLMs for Multi-Robot Systems

Zhao et al. [5] proposed RoCo, a system in which multiple robots interactively discuss task strategies by using LLMs to

perform tasks and trajectory planning. RoCo leverages the role-setting and interactive nature of ChatGPT by assigning names to each robot and providing environmental feedback, resulting in a highly interpretable and flexible system. This approach demonstrated a high success rate in six benchmark tasks that emphasized the sequential and overlapping nature of a robot workspace. However, the benchmarks involved a relatively small number of robots (2-3), and action decisions were made in a constrained environment with limited action options for each robot. The scalability of the number of robots and limitations on the number of actions were not addressed. Shyam et al. [8] proposed SMART-LLM, a task planning framework for multiple robots using LLMs. This method involves task decomposition, team formation, and task assignment based on input language instructions, subdividing tasks and assigning them according to robot characteristics. However, the task decomposition granularity in this method is coarse, and the framework was not evaluated for instructions requiring complex task decomposition.

Chen et al. [28] evaluated the agent structures and prompt configurations of a multi-robot system using LLMs. They proposed and compared four types of structures: a conventionally distributed structure and two centralized hybrid types. The study found that one of the hybrid types achieved a higher performance. Chen et al. [29] proposed Autotamp, an approach that uses LLMs and STL for multi-robot planning; however, it is limited to mobile robots and does not address problem settings that include manipulators. Zhang et al. [6] introduced a method for determining actions through natural language communication using LLMs in a multi-agent system. Yu et al. [9] developed Co-NavGPT, a cooperative navigation system for multiple robots using LLMs. In this system, environmental information, such as obstacles and uncharted areas on a map, is converted into text, thereby enabling cooperative navigation. Furthermore, research on the application of LLMs in multi-agent systems extends beyond robotics, with various studies exploring their applications in different domains, such as virtual game worlds and education [30].

Thus, various approaches to multi-robot task planning using LLMs have been proposed. However, these studies did not comprehensively compare the different methods or adequately evaluate their performance. In this study, a series of task planning with natural language as input were realized in a multi-robot task plan by having LLMs handle task dependencies that are previously managed by humans.

### III. PROPOSED METHOD

#### A. Skill List Generation

The algorithmic procedure for skill list generation is outlined in Algorithm 1. In the skill list generation process, we used LLMs to generate a list of skills necessary to accomplish the given language instructions from a predefined skill set. First, a set of executable robot skills  $\Pi$  is defined. Each skill  $\pi \in \Pi$  has an associated language description  $s_\pi$ , where  $s_\pi$  is represented by a command in the form of a function + argument, such as “pick\_and\_place(red block, middle)” that the robot can execute.  $p(s_\pi|i)$  is likelihood of the token sequence for the skill’s

---

#### Algorithm 1: Skill List Generation.

---

**Input:** Language instruction  $i$ , predefined skill set  $\Pi$

```

1:  $n = 0$ 
2: while  $s_{\pi_{n-1}} \neq \text{"done"}$  do
3:    $C = \emptyset$ 
4:   for  $s_\pi \in s_\Pi$  do
5:      $p(s_\pi|i, s_{n-1}, \dots, s_0) \leftarrow \text{LLMs}(\text{prompt} + i + s_0 + \dots + s_n)$ 
6:      $C = C \cup p(s_\pi|i, s_{n-1}, \dots, s_0)$ 
7:   end for
8:    $s_n = \text{argmax}_{s_\pi \in s_\Pi} C$ 
9:    $n = n + 1$ 
10: end while
Output:  $S$ : skill list

```

---

language description  $s_\pi$  given language instruction  $i$ . In lines 4–5, for each skill  $s_\pi \in s_\Pi$ ,  $p(s_\pi|i)$  is calculated, and skill  $s_n$  with the highest probability is selected in step  $n$  from  $C$ , which represents the set of predicted probabilities for each skill (shown in lines 3, 6–8). Subsequently, skill  $s_n$  is appended to the end of language instruction  $i$ , and the next skill is selected in lines 1, 8–9. This process is repeated until completion command is selected by  $p(s_\pi|i, s_{n-1}, \dots, s_0)$  (lines 2–10). Using the predicted probabilities of skills, instead of directly generating text, we can prevent hallucinations where undefined skills are output.

The prompt configuration used in this step is shown below.

- 1) *Purpose of inference and brief description:* Describe the number of robots, information about the position and name of objects in the environment, and the objectives of the inference.
- 2) *Rules:* Describe basic rules, including explanations of what each generated skill represents.
- 3) *Considerations for Inference:* Provide information to consider during inference, such as the necessary conditions for object transfer.
- 4) *Few-shot Examples [31]:* The Few-shot method is used to improve the prediction performance. For this method, five examples are provided.

#### B. Dependency Graph Generation

Considering the skill list  $S$  and language instruction generated in the previous steps as inputs, a dependency graph is generated using LLMs. The dependency graph  $G_S(V_S, E_S)$  is a Directed Acyclic Graph (DAG) with skills as nodes and the dependencies between skills as edges. In this graph:

- Each node  $v_i \in V$  represents a skill, where  $v_i$  corresponds to each element in the sequence of skills from previous steps.
- Each edge  $e_{ij} \in E$  represents the dependencies between skills, indicating that skill  $v_j$  depends on skill  $v_i$ .

This process involves leveraging LLMs to infer dependencies and constructing a dependency graph that captures the sequential relationships among skills, thereby enabling parallel skill selection. In this study, a “dependency” is defined as the relationship between two tasks where task B cannot be executed unless

task A was completed. This dependency ensures that tasks are performed in the correct order.

We utilized a step-wise inference method known as the Chain of Thought [32] to enhance the inference performance of the language model during dependency graph generation. First, we generated dependency relationships between skills in textual form. Subsequently, we generated  $T_S$ , that represents the edges between nodes in the text format “ $v_i \rightarrow v_j$ ” in line 2 of Algorithm 2. In line 3, this output was converted into edge  $e_{ij} \in E_S$  to create graph  $G_S(V_S, E_S)$ , consisting of nodes  $V_S$  representing each element of the skill list, and edges  $E_S$ . In line 1-4, we detect whether the generated graph contains cycles, and if a cycle is found, the dependencies are regenerated. As described in the previous section, the prompt includes information about the environment surrounding the robot to provide a comprehensive context.

### C. Task Allocation

This allocation is achieved using an optimal assignment problem approach, which is commonly employed in task allocation for multi-robot systems. Algorithm 2 presents the algorithm for Dependency Graph Generation and Task Allocation.

#### 1) Election of skills with no dependencies

A dependency graph is a directed graph with skills as nodes and the dependencies between skills as edges. We searched for root nodes  $V_r$  within the graph and selected these as executable skills (lines 6–11). Because the selected nodes do not have parent nodes, no dependencies existed between them, enabling parallel execution.

#### 2) Weight Calculation

The weight of each robot skill was calculated. The experiment involves two types robot: an arm robot and a mobile robot. For the arm robots, weights were assigned based on skill feasibility and adjusted by distance because operating on distant objects increases the risk of motion failure or collision with other robots. The weights were calculated based on the distance from the robot to the target object, with the observation  $o$  representing environmental data from the simulation (line 12). Weight  $w_{jk}$  for each skill  $k$  is calculated using the following normalized value:  $d'_{jb_k}$  of the distance  $d_{jb_k}$  between each robot  $j$  and the grasping object  $b_k$ , using (1).  $\alpha$  represents a constant ranging from 0 to 1. In this case,  $\alpha$  was empirically set to 0.3. For mobile robots, a value of 1 or 0 is calculated as the weight, depending on the skill executability (2).

$$w_{jk} = 1 - \alpha d'_{jb_k} \quad (1)$$

$$w_{jk} = \begin{cases} 1 & \text{if executable} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

#### 3) Optimal Assignment

To allocate tasks to multiple robots, we used a linear programming assignment problem based on skill weights. For each skill selected in Section III-A, we created an allocation problem as described in (3), where the sum of the skill weights for each robot is the objective function.

---

### Algorithm 2: Dependency Graph Generation and Task Allocation.

---

**Input:** Language instruction  $i$ , observation  $o$ , skill list  $S$

1: **while** CycleDetection( $G_S$ ) = *True* **do**

2:  $T_S \leftarrow \text{LLMs}(\text{prompt} + i + S)$

3:  $G_S(V_S, E_S) \leftarrow \text{CreateGraph}(S, T_S)$

4: **end while**

5: **while**  $V_S \neq \emptyset$  **do**

6:  $V_r = \emptyset$

7: **for**  $v \in V_S$  **do**

8: **if**  $v$  is rootnode **then**

9:  $V_r = V_r \cup v$

10: **end if**

11: **end for**

12:  $W_{jk} \leftarrow \text{CalculateWeights}(o)$

13:  $V_e \leftarrow \text{OptimizationSolver}(V_r, W_{jk})$

14: Execute  $V_e$

15:  $G_S \leftarrow \text{DeleteNode}(G_S, V_e)$

16: **end while**

---

Let  $N$  and  $M$  represent the number of robots and skills, respectively, and  $x_{jk}$  be a variable indicating whether robot  $j$  executes skill  $k$ .

$$\begin{aligned} &\text{maximize} && z = \sum_j^N \sum_k^M w_{jk} x_{jk} \\ &\text{subject to} && \sum_j^N x_{jk} \leq 1, \sum_k^M x_{jk} \leq 1, \\ &&& x_{jk} \in \{0, 1\}, \forall j \in N, \forall k \in M \end{aligned} \quad (3)$$

A robot can simultaneously perform at most one skill, and at most one robot can be assigned to each skill. Therefore, we imposed the constraint that each robot was assigned no more than one skill, and that each skill was assigned to no more than one robot.

In lines 13–14, the solution to this allocation problem assigns skill  $V_e$  to each robot. Any robot that is not assigned a skill is provided the waiting skill “stay()”. When the robot completes its skills and provides feedback corresponding node  $V_e$  and its associated edges are removed from dependency graph  $G_S$  (line 15). These steps were repeated until all the nodes in the graph were removed or until no more skills could be performed (lines 5–16).

## IV. EXPERIMENT

Two types of experiments were conducted to evaluate the performance of the proposed LiP-LLM. Experiment 1 evaluates the versatility of a planner to examine whether it could appropriately plan various instructions. In Experiment 2, the experiments were conducted in a more complex environment with numerous of robots than in Experiment 1, to evaluate the limitations of the planner’s performance and scalability.

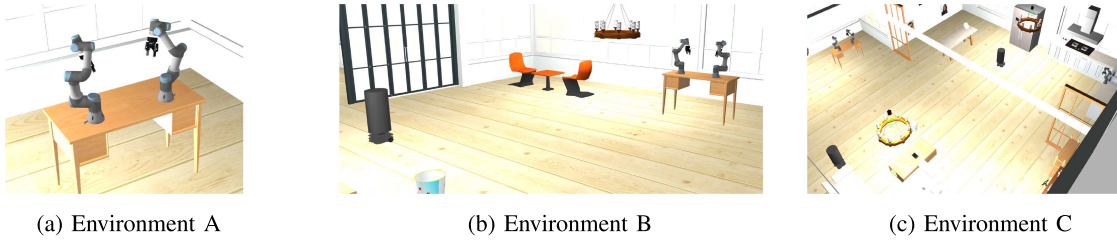


Fig. 2. Experiment environments: (a) Environment A includes two arm robots. (b) Environment B includes two arm robots and one mobile robot. (c) Environment C includes three arm robots (two in the living room and one in the kitchen) and two mobile robots.

TABLE I  
LANGUAGE INSTRUCTION ‘GROUP NAME’ REPRESENTS THE INSTRUCTION GROUP

Group name	Detail	Example	Trial
Group1: Basic	Instruction composed of a stacking task and a placement task, which are the basic operations among the tasks assumed in this study.	Stack the blocks in the order of red, yellow on the middle, and put green block and blue block on the different corner of the table.	15
Group2: Name Change	Instruction refers to a name that differs from that of the object or location defined in the skill, such as “block” being described as “box”.	Put green cube on the lower right edge, and stack cubes in order of red, blue on the upper right edge.	15
Group3: Ambiguity	Name and location of the block are euphemistic and ambiguous, such as place the block clockwise.	Put the blocks in the order of red, blue, yellow, and green clockwise starting from the top right corner.	15
Group4: Add Bowl	Instruction consisting of an environment with an additional bowl in the environment.	Put bowl to the middle, then put red, green, and blue block in the bowl.	18

\*Detail\* provides details about each instruction group. \*Example\* shows example of the instructions. \*Trial\* is the number of instructions in each group.

In this experiment, the AWS RoboMaker Small House World ROS package<sup>1</sup> was used to simulate the home environment. The furniture arrangement was partially modified for this experiment. Four colored blocks and bowls (blue, red, yellow, and green) were placed in the experimental environment. The robot was assumed to know the locations of the objects. Three experimental environments were used with variation in the type and number of robots. LiP-LLM use OpenAI’s GPT model “text-davinci-003” as the LLM.

- *Environment A*: This setup featured a living room with two arm robots (Fig. 2(a)). These robots could interact with object placement and stacking tasks on a table.
- *Environment B*: This environment includes a living room with two arm robots and a mobile robot (Fig. 2(b)). The robots performed object placement and stacking tasks on a table, with some objects requiring transportation by the mobile robot.
- *Environment C*: This configuration comprises two arm robots in the living room and one arm robot in the kitchen. As well as two mobile robots (Fig. 2(c)). The robots performed tasks either at the table or in the kitchen and might have needed to move objects between these areas.

#### A. Task Setting

1) *Experiment 1*: In Experiment 1, the generality of LiP-LLM was evaluated by changing two variables: language instructions and constraints provided to the robot in Environments A and B.

- 1) *Instruction*: Four groups of instructions were used to evaluate generality, resulting in 63 different types of languages instruction. The instructions are shown in Table I.

2) *Constraints on the robot*: Each robot in the environment has different characteristics. Two arm robots in living rooms have constraints regarding the objects that they can grasp.

- a) Condition 1: There were no restrictions on the robots, allowing them to grasp any object in the environment.
- b) Condition 2: Robot 1 can grasp red, blue, and yellow objects (blocks and bowls), whereas Robot 2 can grasp only red and green objects.
- c) Condition 3: Robot 1 can grasp red and blue objects, and Robot 2 can grasp yellow and green objects.

2) *Experiment 2*: We evaluated the utility of LiP-LLM by varying three variables: language instructions, constraints imposed on robots, and object placement positions in Environment C.

- 1) *Instruction*: The same 15 language instructions were provided to the instruction group (basic instructions used in the Experiment 1) were applied, with the same structure but different installation positions.
- 2) *Constraints on the robot*: The arm robot to be placed in the living room was subjected to the same constraints as in Experiment 1. However, no constraints are imposed on the arm robot placed in the kitchen.
- 3) *Location of objects*: In Environment C, the task planning results can vary significantly based on the object location, even when using the same language instructions. Therefore, three difficulty levels were set based on the necessity of object transportation by the mobile robot.
  - a) *Level 1*: Objects are placed close to the target positions, thus requiring no object transportation for the initial placement. This is the least challenging setting because a mobile robot does not need to transport objects.
  - b) *Level 2*: Objects may be held by the mobile robot instead of leaving them at the target position, which

<sup>1</sup><https://github.com/aws-robotics/aws-robomaker-small-house-world>

requires initial transportation by the mobile robot. This setting has a medium difficulty level because the mobile robot must transport objects.

- c) *Level 3*: Objects are placed in a room that differs from the target location, which necessitate object transportation between rooms. This is the most challenging setting, requiring coordination between the arm and mobile robots to transport objects.

### B. Comparative Methods

- *RoCo* [5]: A method in which multiple robots interactively make skilled decisions using LLMs. The LLMs uses OpenAI's GPT-4-0613.
- *SMART-LLM* [8]: Task planning method in which a series of operations, including task decomposition, team formation and task allocation were performed using the LLMs. The LLMs also used the OpenAI's GPT-4-0613.

### C. Evaluation Index

- 1) *Success Rate*: The success rate was recorded as 1 if a plan that fulfilled language instruction was generated and 0 if it failed. Only trials with a success rate of one were considered in the computation time metrics.
- 2) *Success weighted by Path Length(SPL)* [33]: This metric evaluates the plan efficiency by comparing the ideal shortest number of steps, determined by a human, with the steps in the generated plan. In this study, the SPL reflects the success rates weighted by the step count rather than path length. Equation (4) was used in this experiment.

$$\frac{1}{N} \sum_{i=1}^N S_i \frac{l_i}{\max(p_i, l_i)} \quad (4)$$

where  $N$  is the number of trials,  $S_i$  is one for success and zero for failure,  $l_i$  is the minimum number of steps in trial:  $i$ , and  $p_i$  is the number of steps in trial  $i$ .

- 3) *Process Time*: The time taken by the planner to process the allocation and planning from the time it receives instructions to the time it sends commands to the robots.

### D. Result

#### 1) Experiment 1:

- 1) *Success Rate*: As shown in Tables II and III, LiP-LLM achieved the highest success rate in Environments A and B, with a maximum difference of 0.82. Table II indicates that LiP-LLM maintains robust task planning even when language commands aim to induce failures. Table III shows no significant changes across methods under varying constraints.
- 2) *SPL*: The SPL results followed a similar trend to the success rate, indicating differences in efficiency. SMART-LLM, LiP-LLM, and RoCo scored higher in efficiency, with a maximum difference of 0.77 compared with the baseline. Although SMART-LLM had a lower success

TABLE II  
EXPERIMENT 1: RESULTS FOR EACH INSTRUCTION GROUP

	methods	Environment A			Environment B		
		Success Rate↑	SPL↑	Time [s]↓	Success Rate↑	SPL↑	Time [s]↓
G1	LiP-LLM	<u>0.93</u>	<u>0.89</u>	<u>21.0</u>	<u>0.82</u>	<u>0.72</u>	<u>33.2</u>
	RoCo	0.71	0.60	26.3	0.58	0.44	86.9
	SMART-LLM	0.62	0.59	33.4	0.49	0.45	49.4
G2	LiP-LLM	<u>0.93</u>	<u>0.88</u>	<u>21.9</u>	<u>0.91</u>	<u>0.81</u>	<u>35.5</u>
	RoCo	0.69	0.45	47.9	0.36	0.22	117
	SMART-LLM	0.11	0.11	46.0	0.22	0.19	66.0
G3	LiP-LLM	<u>0.44</u>	<u>0.40</u>	<u>25.0</u>	<u>0.44</u>	<u>0.43</u>	<u>37.2</u>
	RoCo	<u>0.44</u>	0.37	46.6	0.18	0.12	105
	SMART-LLM	0.36	0.34	45.3	0.29	0.17	60.9
G4	LiP-LLM	<u>0.50</u>	<u>0.46</u>	<u>25.4</u>	<u>0.17</u>	<u>0.15</u>	<u>40.6</u>
	RoCo	0.22	0.18	66.0	0.09	0.06	117
	SMART-LLM	0.37	0.37	48.8	0.13	0.11	52.6

TABLE III  
EXPERIMENT 1: RESULT FOR EACH CONSTRAINT

	methods	Environment A			Environment B		
		Success Rate↑	SPL↑	Time [s]↓	Success Rate↑	SPL↑	Time [s]↓
C1	LiP-LLM	<u>0.70</u>	<u>0.67</u>	<u>22.8</u>	<u>0.56</u>	<u>0.51</u>	<u>34.3</u>
	RoCo	0.55	0.43	37.5	0.25	0.17	97.7
	SMART-LLM	0.48	0.46	40.9	0.33	0.29	58.1
C2	LiP-LLM	<u>0.67</u>	<u>0.60</u>	<u>22.7</u>	<u>0.60</u>	<u>0.56</u>	<u>36.2</u>
	RoCo	0.49	0.36	53.2	0.33	0.23	96.9
	SMART-LLM	0.30	0.29	42.3	0.24	0.18	57.1
C3	LiP-LLM	<u>0.71</u>	<u>0.67</u>	<u>22.8</u>	<u>0.52</u>	<u>0.47</u>	<u>34.2</u>
	RoCo	0.49	0.39	53.2	0.29	0.20	109
	SMART-LLM	0.32	0.31	41.8	0.25	0.21	51.9

rate, its SPL was not high because it generated near-optimal plans in the successful trials. LiP-LLM and RoCo were slightly less efficient because of redundant tasks.

- 3) *Process Time*: LiP-LLM outperformed SMART-LLM and RoCo in terms of process time. This showed a difference of up to approximately 80 s from the baseline method. In the proposed method, the waiting time for the LLM invocation accounted for the majority of the processing time. Therefore, the processing time increased with the number of inference calls for both the proposed and comparative methods.
- #### 2) Experiment 2:
- 1) *Success Rate*: As shown in Tables IV and V, LiP-LLM had the highest success rate, with a maximum difference of 0.34, similar to that in Experiment 1. Although the success rates were lower for all methods in the more complex long-term tasks of Experiment 2, LiP-LLM produced robust plans. As Table IV indicates, the success rate of the comparative method declined under stricter constraints, whereas LiP-LLM was less affected. The comparative method often ignores robot constraints, leading to errors, whereas LiP-LLM makes no such errors.
  - 2) *SPL*: Similar trends were observed in Experiment 1, with LiP-LLM performing the best, followed by SMART-LLM

TABLE IV  
EXPERIMENT 2: RESULTS FOR EACH INSTRUCTION LEVEL

	methods	Environment C		
		Success Rate $\uparrow$	SPL $\uparrow$	Time[s] $\downarrow$
L1	LiP-LLM	<u>0.58</u>	<u>0.36</u>	72.9
	RoCo	0.36	0.19	113
	SMART-LLM	0.24	0.24	<u>40.4</u>
L2	LiP-LLM	<u>0.31</u>	<u>0.16</u>	77.2
	RoCo	0.18	0.13	137
	SMART-LLM	0.16	0.15	<u>35.1</u>
L3	LiP-LLM	<u>0.31</u>	<u>0.19</u>	96.9
	RoCo	0.00	0.00	-
	SMART-LLM	0.04	0.04	<u>40.8</u>

TABLE V  
EXPERIMENT 2: RESULT FOR EACH CONSTRAINT

	methods	Environment C		
		Success Rate $\uparrow$	SPL $\uparrow$	Time[s] $\downarrow$
C1	LiP-LLM	<u>0.40</u>	<u>0.24</u>	81.4
	RoCo	0.24	0.15	122
	SMART-LLM	0.22	0.21	<u>39.0</u>
C2	LiP-LLM	<u>0.38</u>	<u>0.24</u>	81.1
	RoCo	0.18	0.09	141
	SMART-LLM	0.09	0.09	<u>33.4</u>
C3	LiP-LLM	<u>0.42</u>	<u>0.23</u>	78.2
	RoCo	0.11	0.08	85.5
	SMART-LLM	0.13	0.13	<u>41.2</u>

and RoCo. The maximum difference from baseline was 0.19. SPL highlighted the task allocation efficiency, where the proposed method excelled with fewer redundant tasks.

- 3) *Process Time*: As shown in Table IV, unlike in Experiment 1, SMART-LLM had the shortest computation time. LiP-LLM took approximately 56 s longer than the baseline because of the increased number of API calls required for skill generation in long-term task planning.

## V. DISCUSSION

### A. Effectiveness

1) *Success Rate*: As detailed in Section IV, LiP-LLM showed high success rates in task planning across all conditions in Experiments 1 and 2, whereas the comparative methods experienced significant drops in success rates with more robots or varying command types. This result is attributed to the proposed method's ability to mitigate LLM hallucinations by selecting skill lists through likelihood calculations and enforcing order constraints inferred from dependencies. However, the comparative methods failed because of command hallucinations and task failures from ignoring order constraints, such as simultaneous stacking tasks that cause collisions. In addition, robustness to an increased number of robots has contributed to the success. In Experiment 2, with five robots and two rooms (kitchen and living room), accurate location tracking was essential, as indicated by the success rate differences between Levels 2 and 3 in Table IV. Contrarily to the LLM-based method, which struggled with task allocation owing to tracking limitations, LiP-LLM's use of linear programming leads to fewer allocation errors. Regarding the

scaling of the number of robots, theoretically, linear programming can be used to assign an optimal task to any number of robots. In these experiments, LiP-LLM demonstrated almost no failures in task allocation when using linear programming.

2) *Efficiency*: The SPL results indicated that LiP-LLM provided task planning results that were closer to the optimal steps set by humans. The proposed method considers the dependencies among skills in the process of dependency graph generation, thereby enabling more tasks to be executed in parallel and efficiently.

3) *Process Times*: In Experiment 1, LiP-LLM demonstrated a reduced computation time for task planning. The proposed method leverages combinatorial optimization for task allocation, resulting in considerable shorter processing times compared with comparative methods, which rely on time-intensive LLM inference. Reducing frequent LLM calls and using algorithmic approaches effectively shortened the process time, although this was not observed in Experiment 2. The process time of LiP-LLM increases with task plan length, whereas SMART-LLM mitigates this by decomposing tasks into a single inference, thereby improving the process time. Although SMART-LLM exhibits advantages in terms of process time, its performance is insufficient. In addition, the application of linear programming affects the scaling of the number of robots. Compared with RoCo, which is a decentralized system, LiP-LLM requires a shorter planning time. Therefore the proposed method is useful for scaling the number of units.

### B. Limitations

1) *Challenges in Skill List Generation*: Creating a predefined skill set is challenging. The skill list was generated using Say-Can, which calculates the likelihood of each predefined skill from a prompt. A skill is defined by a function name and arguments, such as "pick\_and\_place(red block, blue block)". In environments with multiple objects, the number of predefined skills increases because of the multiple object combinations. Additionally, it is not feasible to generate skills that are not predefined. For example, if a door should be opened to move between rooms, the task would be impossible without a predefined "open door" skill.

2) *Challenges in Dependency Graph Generation*: In dependency graph generation, the increase in environmental information and diversification of tasks owing to the complexity of the environment is a challenge. In this experiment, the success rate of environment C was lower than those of environments A and B. The lower success rate can be attributed to both increased environmental complexity and skill adjustments within the robot owing to task diversification. As the number of rooms increases, the environmental information input to the LLM increases, complicating tasks such as moving objects between rooms. To address scalability issues, methods that can adequately handle the complexity introduced by environmental changes, including inter-robot skill coordination, must be developed.

## VI. CONCLUSION

In this study, we proposed a multi-robot task planning method LiP-LLM, which integrates dependency graph generation using

LLMs and task allocation via linear programming. By generating a dependency graph between skills, we demonstrated that skills with order constraints can be executed in an appropriate sequence, leading to more accurate and efficient task execution. Additionally, by adopting optimization methods for the task allocation step, we showed that the computation time can be reduced compared with methods that use LLMs for the same process.

Future studies should focus on applying LiP-LLM to real-world environments. The simulation environment used in this study was ideal, with known object coordinates. However, obtaining accurate environmental information in real environments is challenging and the available data may be limited. Therefore, it is necessary to improve the updating of environmental information through sequential environmental recognition and to enhance the methods for calculating the weights used in linear programming. In addition, the actions selected by a robot in a real environment may not always be successful. For example, a robot may fail to pick up or drop a red block. To manage such situations more effectively, it is crucial to develop a robust task planning method that includes sequential environmental recognition and task replanning with feedback, as demonstrated by Huang et al. [26]. This ensures that robots can adapt to dynamic environments and unexpected results, thereby improving overall task execution reliability.

## REFERENCES

- [1] A. Brohan et al., "Do as i can, not as i say: Grounding language in robotic affordances," in *Proc. Conf. Robot Learn.*, 2023, pp. 287–318.
- [2] J. Kiener and O. V. Stryk, "Towards cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots," *Robot. Auton. Syst.*, vol. 58, no. 7, pp. 921–929, 2010.
- [3] Y. Rizk, M. Awad, and E. W. Tunstel, "Cooperative heterogeneous multi-robot systems: A survey," *ACM Comput. Surveys*, vol. 52, no. 2, pp. 1–31, 2019.
- [4] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *Int. J. Adv. Robotic Syst.*, vol. 10, no. 12, 2013, Art. no. 399.
- [5] Z. Mandi, S. Jain, and S. Song, "RoCo: Dialectic multi-robot collaboration with large language models," in *Proc. 2024 IEEE Int. Conf. Robot. Automat.*, 2024, pp. 286–299.
- [6] H. Zhang et al., "Building cooperative embodied agents modularly with large language models," in *Proc. 12th Int. Conf. Learn. Representations*.
- [7] J. Wang, G. He, and Y. Kantaros, "Safe task planning for language-instructed multi-robot systems using conformal prediction," 2024, *arXiv:2402.15368*.
- [8] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, "SMART-LLM: Smart multi-agent robot task planning using large language models," in *Proc. 2024 Int. Conf. Intell. Robots Syst.*, 2024.
- [9] B. Yu, H. Kasaei, and M. Cao, "Co-NavGPT: Multi-robot cooperative visual semantic navigation using large language models," 2023, *arXiv:2310.07937*.
- [10] L. Wang et al., "A survey on large language model based autonomous agents," *Front. Comput. Sci.*, vol. 18, no. 6, Art. no. 186345, 2024.
- [11] E. Nunes, M. Manner, H. Mitiche, and M. Gini, "A taxonomy for task allocation problems with temporal and ordering constraints," *Robot. Auton. Syst.*, vol. 90, pp. 55–70, 2017.
- [12] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, 2004.
- [13] M. McIntire, E. Nunes, and M. Gini, "Iterated multi-robot auctions for precedence-constrained task scheduling," in *Proc. 2016 Int. Conf. Auton. Agents Multiagent Syst.*, 2016, pp. 1078–1086.
- [14] K. Kawaharazuka, T. Matsushima, A. Gambardella, J. Guo, C. Paxton, and A. Zeng, "Real-world robot applications of foundation models: A review," *Adv. Robot.*, vol. 38, pp. 1232–1254, 2024.
- [15] B. Liu et al., "LLM+P: Empowering large language models with optimal planning proficiency," 2023, *arXiv:2304.11477*.
- [16] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 79081–79094.
- [17] Z. Zhou, J. Song, K. Yao, Z. Shu, and L. Ma, "ISR-LLM: Iterative self-refined large language model for long-horizon sequential task planning," in *Proc. 2024 IEEE Int. Conf. Robot. Automat.*, 2024, pp. 2081–2088.
- [18] A. Capitanelli and F. Mastrogianni, "A framework for neurosymbolic robot action planning using large language models," *Front. Neurobotics*, vol. 18, 2023, Art. no. 1342786.
- [19] Y. Cao and C. S. G. Lee, "Robot behavior-tree-based task generation with large language models," 2023.
- [20] F. Lin et al., "Graph-enhanced large language models in asynchronous plan reasoning," in *Proc. 41st Int. Conf. Mach. Learn.*, 2024, pp. 30108–30134. [Online]. Available: [https://proceedings.mlr.press/v235/lin24\\_k.html](https://proceedings.mlr.press/v235/lin24_k.html)
- [21] S. H. Vemprala, R. Bonatti, A. Buckner, and A. Kapoor, "ChatGPT for robotics: Design principles and model abilities," *IEEE Access*, vol. 12, pp. 55682–55696, 2024.
- [22] I. Singh et al., "ProgPrompt: Generating situated robot task plans using large language models," in *Proc. 2023 IEEE Int. Conf. Robot. Automat.*, 2023, pp. 11523–11530.
- [23] J. Liang et al., "Code as policies: Language model programs for embodied control," in *Proc. 2023 IEEE Int. Conf. Robot. Automat.*, 2023, pp. 9493–9500.
- [24] Y. Jiang et al., "VIMA: General robot manipulation with multimodal prompts," in *Proc. 4th Int. Conf. Mach. Learn.*, 2023, pp. 14975–15022.
- [25] K. Kawaharazuka, N. Kanazawa, Y. Obinata, K. Okada, and M. Inaba, "Continuous object state recognition for cooking robots using pre-trained vision-language models and black-box optimization," *IEEE Robot. Automat. Lett.*, vol. 9, no. 5, pp. 4059–4066, 2024.
- [26] W. Huang et al., "Inner monologue: Embodied reasoning through planning with language models," in *Proc. Conf. Robot Learn.*, 2023, pp. 1769–1782.
- [27] B. Zitkovich et al., "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Proc. 7th Conf. Robot Learn.*, 2023, pp. 2165–2183. [Online]. Available: <https://proceedings.mlr.press/v229/zitkovich23a.html>
- [28] Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan, "Scalable multi-robot collaboration with large language models: Centralized or decentralized systems?," in *Proc. 2024 IEEE Int. Conf. Robot. Automat.*, 2024, pp. 4311–4317.
- [29] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan, "Auto-Tamp: Autoregressive task and motion planning with LLMs as translators and checkers," in *Proc. 2024 IEEE Int. Conf. Robot. Automat.*, 2024, pp. 6695–6702.
- [30] T. Guo et al., "Large language model based multi-agents: A survey of progress and challenges," in *Proc. 33rd Int. Joint Conf. Artif. Intell.*, 2024, pp. 8048–8057. [Online]. Available: <https://doi.org/10.24963/ijcai.2024/890>
- [31] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1877–1901.
- [32] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 24824–24837.
- [33] P. Anderson et al., "On evaluation of embodied navigation agents," 2018, *arXiv:1807.06757*.