MATDS : Multi-Agent Task Decomposition System based on LLMs

Joonyoung Jung, Dong-oh Kang
Visual intelligence Research Section
Electronics and Telecommunications Research Institute
Deajeon, Korea
jyjung21@etri.re.kr

***Abstract*— In multi-agent systems, task decomposition must consider physical constraints, temporal dependencies among tasks, and interactions between agents. Decomposing high-level natural language instructions into executable multi-agent actions is a core component of multi-agent planning systems. In this paper, we propose the multi-agent task decomposition system (MATDS), which leverages the natural language understanding, reasoning, and evaluation capabilities of large language models (LLMs) to perform multi-agent task decomposition. The MATDS takes high-level task instructions provided in natural language as input and decomposes them into executable forms that can be carried out by multiple agents. The MATDS employs LLMs to iteratively decompose high-level tasks through Chain-of-Thought (CoT) reasoning and evaluates the decomposition results through a critic module to ensure that the generated task plans satisfy feasibility and logical consistency. This enables effective task decomposition that considers multi-agent execution capabilities and parallelism. We conducted experiments on task decomposition using the AI2-THOR simulator and demonstrated that the MATDS can mitigate errors that may occur in few-shot prompting approaches. Overall, MATDS provides a flexible and scalable solution for generating and refining task decompositions for multi-agent systems.**

Keywords—multi-agent, task decomposition, chain of thought, reasoning.

I. INTRODUCTION

In recent years, multi-agent systems have been rapidly expanding across various domains such as logistics automation, household service robots, and rescue operations. However, for these systems to perform complex tasks in a human-like manner, they require high-level natural language-based task decomposition capabilities that go beyond simple command execution. Task decomposition involves determining the appropriate sequence of actions while considering the abilities of each agent, thereby breaking down high-level task instructions into sub-tasks executable by multiple agents.

Early approaches to task planning primarily relied on symbolic planning or behavior tree-based structures, which are grounded in explicit rule-based policies and state transition models. However, these methods require domain model modifications whenever new tasks are introduced, and they struggle to comprehensively account for task ordering, object dependencies, and physical constraints. To address these limitations, recent research has focused on task decomposition based on large language models (LLMs) [1–4]. In particular, the Chain-of-Thought (CoT) approach has shown effectiveness in solving complex problems through step-by-step reasoning [5–7].

The objective of this paper is to propose a multi-agent task decomposition system (MATDS) that can understand complex, natural language-based high-level task instructions and decompose them into executable plans, enabling multiple agents to effectively accomplish task objectives.

The remainder of this paper is organized as follows. Section II introduces MATDS, Section III presents the experiments, and Section IV concludes the paper with final remarks.

II. MULTI-AGENT TASK DECOMPOSITION SYSTEM

As shown in Fig. 1, MATDS decomposes high-level task instructions step by step into detailed execution plans that can be carried out by multiple agents, and maintains logical consistency by evaluating and refining the generated task decompositions.

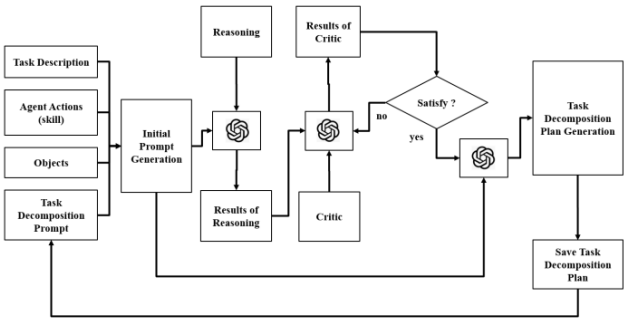

Fig. 1. Overview of the MATDS Architecture.

Fig. 1 illustrates the architecture of MATDS, while Fig. 2 presents the pseudo-code for the operational procedure of MATDS. The procedure can be described as follows. First, information about agents and objects is extracted from the environment, allowing the system to recognize details such as the action capabilities of agents and the weights of objects. Using the collected environmental information, together with a task decomposition prompt for few-shot prompting and the high-level task description, an initial prompt is constructed for generating multi-agent task decomposition.

LLMs are then employed to decompose the high-level task description into executable tasks that can be carried out by multiple agents. To mitigate potential hallucinations in the decomposition results generated by the LLMs, reasoning based on CoT is applied. The resulting multi-agent task decomposition is subsequently evaluated to verify whether it satisfies logical consistency and supports parallel execution. If the evaluation results are unsatisfactory, the decomposition is revised to meet the evaluation criteria, and the evaluation process is repeated. Once a satisfactory result is achieved, the

system produces a multi-agent task decomposition for the given high-level task description.

```
      # Initial prompt
 1 :  recognize environment
 2 :    agent actions (skill)
 3 :    objects

 4 :  receive high-level task description
 5 :  read task decomposition prompt
 6 :  generate initial prompt

      # Reasoning
 7 :  read initial prompt
 8 :  for step in CoT-step
 9 :    read n-th CoT prompt
10 :    execute n-th CoT

11 :  reasoning task decomposition

      # Critic
12 :  while critic-satisfy
13 :    read critic prompt
14 :    execute critic
15 :    if not satisfy the critic
16 :      update task decomposition

      # Task decomposition plans
17 :  generate task decomposition plan
18 :  save task decomposition plan
19 :  update task decomposition prompt
```

Fig. 2. MATDS Pseudo-code.

Finally, the satisfactory multi-agent task decomposition result is stored, and the task decomposition prompt is updated so that it can be utilized as additional prompt data for future multi-agent task decomposition processes.

III. EXPERIMENTS

Experiments on task decomposition for high-level task instructions using MATDS were conducted in the AI2-THOR simulator. For the reasoning experiment, the task decomposition of "slice the tomato" was performed. Without reasoning, the decomposition result appears as shown in Fig. 3, whereas when reasoning with CoT is applied in MATDS, the decomposition result is obtained as shown in Fig. 4

```
# Task : Slice the Tomato.
  (Skills Required: GoToObject, PickupObject, SliceObject, PutObject)

# SubTask
def slice_tomato():
   GoToObject('Knife')
   PickupObject('Knife')
   GoToObject('Tomato')
   SliceObject('Tomato')
   GoToObject('CounterTop')
   PutObject('Knife', 'CounterTop')

# Execute SubTask
slice_tomato()

# Task slice the tomato is done
```

Fig. 3. Task decomposition result without applying reasoning.

When task decomposition for "slice the tomato" is performed without reasoning, as shown in Fig. 5, an error may occur during the assignment of decomposed tasks to multiple agents. Specifically, the 'PickupObject('Knife')' and the 'SliceObject('Tomato')' may be assigned to different agents. In other words, it is possible that 'PickupObject('Knife')' is assigned to agent 1 while 'SliceObject('Tomato')' is assigned to agent 2, leading to a failure in completing the high-level task instruction.

```
**Reasoning:**
Only one agent is needed to perform this task most efficiently,
as each step must be performed in sequence and there is no opportunity for parallelization.

The steps are:

1. Go to the Knife.
2. Pick up the Knife.
3. Go to the Tomato.
4. Slice the Tomato.
5. Go to the CounterTop.
6. Put the Knife back on the CounterTop.

Each action depends on the previous one, and only one agent can hold and use the knife at a
time. Thus, the most efficient number of agents is one.
```

Fig. 4. Reasoning Results for Task Decomposition.

Therefore, as illustrated in Fig. 4, if the number of agents required to efficiently execute the high-level task instruction is explicitly specified through reasoning after task decomposition, such erroneous assignments of sub-tasks across multiple agents can be avoided.

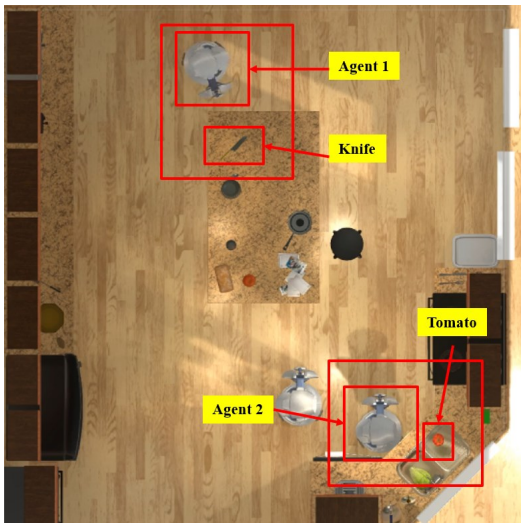

Fig. 5. Inability to execute a high-level task when single-agent instructions are assigned across multiple agents.

For the critic experiment, task decomposition was performed for the instruction "Slice the lettuce, trash the mug, and switch off the light". Without applying the critic, the task decomposition result appears as shown in Fig. 6. Even if the sub-tasks are allocated to multiple agents according to the efficient number of agents through reasoning, as shown in Fig. 7, an error may occur if the task 'switch off light' is executed first. In such a case, the remaining sub-tasks ('slice lettuce' and 'trash mug') would have to be carried out in darkness, leading to incorrect execution.

By applying the critic to evaluate the execution order, the task decomposition was corrected so that the sub-task 'switch off light' is performed only after the other sub-tasks have been completed, as shown in Fig. 8. Specifically, the decomposition was modified such that the sub-tasks 'slice lettuce' and 'trash mug' are executed in parallel first, followed by the execution of the sub-task 'switch off light'.

```
# Task : Slice the lettuce, trash the mug and switch off the light

# SubTask
def slice_lettuce():
    GoToObject('Knife')
    PickupObject('Knife')
    GoToObject('Lettuce')
    SliceObject('Lettuce')
    GoToObject('CounterTop')
    PutObject('Knife', 'CounterTop')

def trash_mug():
    GoToObject('Mug')
    PickupObject('Mug')
    GoToObject('GarbageCan')
    PutObject('Mug', 'GarbageCan')

def switch_off_light():
    GoToObject('LightSwitch')
    SwitchOff('LightSwitch')

# Parallelize SubTask 1, SubTask 2, and SubTask 3
task1_thread = threading.Thread(target=slice_lettuce)
task2_thread = threading.Thread(target=trash_mug)
task3_thread = threading.Thread(target=switch_off_light)

# Start executing all subtasks in parallel
task1_thread.start()
task2_thread.start()
task3_thread.start()

# Wait for all subtasks to finish
task1_thread.join()
task2_thread.join()
task3_thread.join()

# Task slice the lettuce, trash the mug and switch off the light is done
```

Fig. 6. Task decomposition result without applying the critic.

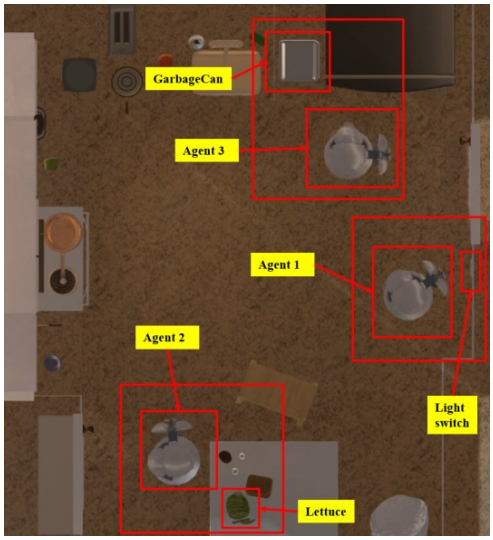

Fig. 7. 'Switch off light' executed first, causing 'slice lettuce' and 'trash mug' to be performed in darkness.

```
# Parallelize SubTask 1 and SubTask 2
task1_thread = threading.Thread(target=slice_lettuce)
task2_thread = threading.Thread(target=trash_mug)

# Start executing SubTask 1 and SubTask 2 in parallel
task1_thread.start()
task2_thread.start()

# Wait for both SubTask 1 and SubTask 2 to finish
task1_thread.join()
task2_thread.join()

# Execute SubTask 3 after SubTask 1 and SubTask 2 are complete
switch_off_light()

# Task slice the lettuce, trash the mug and switch off the light is done
```

Fig. 8. Task decomposition result after applying the critic to revise the execution order of sub-tasks.

As observed in the above experiments, the use of few-shot prompting with LLMs can lead to potential errors. However, such errors can be mitigated through CoT reasoning and the application of a critic to the generated task decomposition results.

IV. Conclusions

In this paper, we proposed MATDS, a system that leverages the reasoning and evaluation capabilities of LLMs to generate task decompositions from natural language instructions that can be executed in parallel by multiple agents. MATDS employs reasoning to decompose natural language task instructions into detailed execution plans suitable for multi-agent execution, and utilizes a critic to evaluate and refine the generated decompositions. Experiments conducted in the AI2-THOR simulator demonstrated that MATDS can correct errors that may arise in few-shot prompting approaches. Overall, MATDS provides a flexible and scalable system for generating and refining task decompositions for multi-agent systems, making it applicable to various collaborative multi-agent applications.

Acknowledgment

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) [No.RS-2022-II220907, Development of AI Bots Collaboration Platform and Self-organizing AI]

References

[1] J. Wu et al., "TidyBot: Personalized Robot Assistance with Large Language Models," *International Conference on Intelligent Robots and Systems*, Detroit, USA, pp. 3546-3553, Oct. 2023.

[2] I. Singh et al., "PROGPROMPT: Generating Situated Robot Task Plans using Large Language Models," *International Conference on Robotics and Automation*, London, UK, pp. 11523-11530, May 2023.

[3] J. Liang et al., "Code as Policies: LanguageModel Programs for Embodied Control," *International Conference on Robotics and Automation*, London, UK, pp. 9493-9500, May 2023.

[4] S. S. Kannan, V. L. N. Venkatesh, and B. Min, "SMART-LLM: Smart Multi-Agent Robot Task Planning using Large Language Models, " *International Conference on Intelligent Robots and Systems*, Abu Dhabi, UAE, pp. 12140-12147, Oct. 2024.

[5] Z. Chu et al., "A Survey of Chain of Thought Reasoning: Advances, Frontiers and Future," *Annual Meeting of the Association for Computational Linguistics*, Bangkok, Thailand, pp. 1173‑1203, Aug. 2024

[6] C. Mitra, B. Huang, T. Darrell, and R. Herzig, "Compositional Chain-of-Thought Prompting for Large Multimodal Models," *Conference on Computer Vision and Pattern Recognition*, Seattle, USA, pp. 14420‑14431, Jun. 2024

[7] X. Wang and D. Zhou, "Chain-of-Thought Reasoning without Prompting," *Conference on Neural Information Processing Systems*, Vancouver, Canada, pp. 66383-66409, Dec. 2024.