

DART-LLM: Dependency-Aware Multi-Robot Task Decomposition and Execution using Large Language Models

Yongdong Wang¹, Runze Xiao¹, Jun Younes Louhi Kasahara¹,
Ryosuke Yajima¹, Keiji Nagatani², Atsushi Yamashita³, and Hajime Asama⁴

Abstract—Large Language Models (LLMs) have demonstrated promising reasoning capabilities in robotics; however, their application in multi-robot systems remains limited, particularly in handling task dependencies. This paper introduces DART-LLM, a novel framework that employs Directed Acyclic Graphs (DAGs) to model task dependencies, enabling the decomposition of natural language instructions into well-coordinated subtasks for multi-robot execution. DART-LLM comprises four key components: a Question-Answering (QA) LLM module for dependency-aware task decomposition, a Breakdown Function module for robot assignment, an Actuation module for execution, and a Vision-Language Model (VLM)-based object detector for environmental perception, achieving end-to-end task execution. Experimental results across three task complexity levels demonstrate that DART-LLM achieves state-of-the-art performance, significantly outperforming the baseline across all evaluation metrics. Among the tested models, DeepSeek-r1-671B achieves the highest success rate, whereas Llama-3.1-8B exhibits superior response time reliability. Ablation studies further confirm that explicit dependency modeling notably enhances the performance of smaller models, facilitating efficient deployment on resource-constrained platforms. Please refer to the project website <https://wyd0817.github.io/project-dart-llm/> for videos and code.

I. INTRODUCTION

Large Language Models (LLMs) have demonstrated significant reasoning capabilities that can be applied to robot systems. Chen et al. introduced the Decision Transformer, a reinforcement learning method that redefines decision-making processes as sequence modeling problems [1]. Reed et al. developed Gato, a generalist artificial intelligence model capable of learning and adapting across multiple tasks and modalities [2]. Ahn et al.’s FLAN-SayCan and PaLM-SayCan leverage the capabilities of large pre-trained language models (LLMs) to interpret and execute tasks based on natural language instructions, enabling robots to understand and perform tasks driven by complex natural language commands [3]. Huang et al.’s Inner Monologue method integrates language models into planning and reasoning processes, enhancing robots’ decision-making capa-

¹Yongdong Wang, Runze Xiao, Jun Younes Louhi Kasahara and Ryosuke Yajima are with Graduate School of Engineering, The University of Tokyo, Tokyo 113-8656, Japan

²Keiji Nagatani is with Faculty of Systems and Information Engineering, The University of Tsukuba, Ibaraki 305-0006, Japan

³Atsushi Yamashita is with Graduate School of Frontier Sciences, The University of Tokyo, Tokyo 277-8563, Japan

⁴Hajime Asama is with the Tokyo College, The University of Tokyo, Tokyo 113-0033, Japan

Project Webpage: [project-dart-llm](https://wyd0817.github.io/project-dart-llm/)

Correspondence to wangyongdong@robot.t.u-tokyo.ac.jp

bilities [4]. Brohan et al. proposed RT-1, a model for real-time control and decision-making in complex environments, applying transformer architecture directly to robotics technology to ensure unprecedented precision and adaptability in understanding and navigating the physical world [5]. Brohan et al. further introduced the RT-2 model, which integrates vision, language, and action capabilities, enabling robots to autonomously learn and control actual actions from web knowledge [6]. Driess et al.’s Palm-E model enhances the perception and action capabilities of language models, offering a new way for artificial intelligence systems to interact more naturally with the external world through enhanced multimodal understanding [7]. Liang et al. launched Code as Policies, enabling large language models (LLMs) to convert natural language commands into robot strategy code with minimal prompts [8]. Huang et al. introduced VoxPoser, which extracts language-conditioned affordances and constraints from LLMs and applies them to the perceptual space through VLMs. VoxPoser uses a code interface without additional training for any component [9].

Despite significant advancements in LLM-powered robotics research, most studies have primarily focused on single-robot systems, while applications in multi-robot systems remain at an early stage of exploration. Multi-robot systems demonstrate significant potential in complex task scenarios such as construction and disaster rescue, improving task execution efficiency and enabling collaborative problem-solving for tasks that are difficult for a single robot to accomplish independently [10]. However, multi-robot task planning still faces numerous challenges, particularly concerning the complexity of mobile robots and real-time execution. The RoCo framework proposed by Zhao et al. employs LLMs for high-level task communication and integrates multi-arm motion planning to accelerate trajectory generation. It also incorporates environmental feedback (e.g., collision detection), allowing the LLM to dynamically adjust task plans based on context. However, RoCo is designed for fixed-position robotic arm systems, making its task planning approach less adaptable to mobile robots or broader task scenarios [11]. Kannan et al.’s SMART-LLM has made progress in multi-robot task planning. However, it generates outputs using Python and does not explicitly handle dependencies among complex tasks. Due to the inherent complexity and rigidity of programming languages, practical testing revealed a high failure rate. Furthermore, SMART-LLM requires generating Python code that must be manually executed, lacking real-time LLM inference

TABLE I
COMPARISON OF RELATED WORK IN MULTI-ROBOT SYSTEMS AND LARGE LANGUAGE MODELS (LLMs)

Related Work	Multi-Robot System	Robot Mobility	Real-time Capability	Dependency-Aware
Chen et al., Decision Transformer [1]		✓	✓	
Reed et al., Gato [2]		✓	✓	
Ahn et al., FLAN-SayCan, PaLM-SayCan [3]		✓	✓	
Huang et al., Inner Monologue [4]		✓	✓	
Brohan et al., RT-1 [5]		✓	✓	
Brohan et al., RT-2 [6]		✓	✓	
Driess et al., Palm-E [7]		✓	✓	
Liang et al., Code as Policies [8]		✓		
Huang et al., VoxPoser [9]			✓	
Zhao et al., RoCo [11]	✓		✓	
Kannan et al., SMART-LLM [12]	✓	✓		
Proposed DART-LLM	✓	✓	✓	✓

capabilities [12].

The comparison of the above-mentioned studies is shown in Table I. Table I highlights two key challenges that persist in multi-robot systems: (1) existing systems lack support for end-to-end real-time execution across multiple types of mobile robots; (2) current methods rely solely on the models intrinsic reasoning ability without explicitly representing task dependencies, which results in suboptimal performance, particularly for smaller models in handling complex tasks. For instance, RoCo [11] lacks support for mobile robots, being designed for fixed-position robotic arms, while SMART-LLM [12] does not support end-to-end real-time execution and fails to explicitly model task dependencies. To address these limitations, we propose DART-LLM, A Dependency-Aware Task Decomposition and Execution System for Multi-Robot Coordination, as illustrated in Fig. 1. This study makes the following contributions:

1. Dependency-Aware Task Decomposition Mechanism: A novel task decomposition mechanism that utilizes a Directed Acyclic Graph (DAG) to model subtask dependencies. This method significantly enhances system reasoning capabilities, particularly improving the ability of smaller models to handle complex task dependencies. It ensures proper task execution order while maximizing overall efficiency.

2. End-to-End Real-Time Execution Framework: A real-time execution framework includes the QA LLM module, Breakdown Function module, Actuation module, and a Vision-Language Model (VLM)-based object detection module. These modules perform Instruction Parsing and Task Decomposition, Parsing of Decomposed Tasks, Grounding in Embodiments, and Updating the Object Map Database to accomplish task decomposition and execution from natural language instructions to robotic actions.

3. Benchmark Dataset for Construction Robot Evaluation: A dataset of 102 high-level natural language instructions for construction robot tasks, designed with strict execution order constraints and spanning three complexity levels.

II. PROBLEM FORMULATION

This section presents the foundational mathematical concepts of the DART-LLM system. We formalize the definitions

of robot sets and their associated skills, team formation with collaborative skill enhancement, task decomposition into sub-tasks with temporal and dependency relationships, and the skill library essential for task execution.

A. Robot Set and Skills

We define the set of all robots as:

$$\mathbb{R} = \{R^1, R^2, \dots, R^N\} \quad (1)$$

where N is the number of robots. The set of skills for all robots are:

$$\mathbb{S} = \{S^1, S^2, \dots, S^N\} \quad (2)$$

B. Team of Robots and Team Skills

A team of robots is defined as:

$$\mathbb{A} = \{A^1, A^2, \dots, A^Q\} \quad (3)$$

where Q denotes the number of robots in the team. The skills of a team of robots are expressed as:

$$\mathbb{S}_A = \{S_A^1, S_A^2, \dots, S_A^\Omega\} \quad (4)$$

where $\Omega \geq Q$. Here, Ω represents the total number of skills, including both the individual skills of each robot and the new skills generated through robot collaboration. The set of all skills across all teams formed by N robots is:

$$\Delta = \bigcup \mathbb{S}_A \quad (5)$$

It should be noted that Δ includes \mathbb{S} , the set of all individual robot skills.

C. Task and Function Requirements

The high-level language instruction is represented as \mathbb{I} , and the environment is denoted by \mathbb{E} . The instruction \mathbb{I} is decomposed using the *QA LLM* module into a set of sub-tasks. A set of sub-tasks is defined as:

$$\mathbb{T} = \{T_{t_1}^1, T_{t_2}^1, \dots, T_{t_j}^K\} \quad (6)$$

where K is the number of sub-tasks, and t_j denotes the temporal order of a sub-task. Since sub-tasks may be executed simultaneously, we have $j \leq K$. Each task $T_{t_j}^k$ should be executable using the skills from the skill library within Δ .

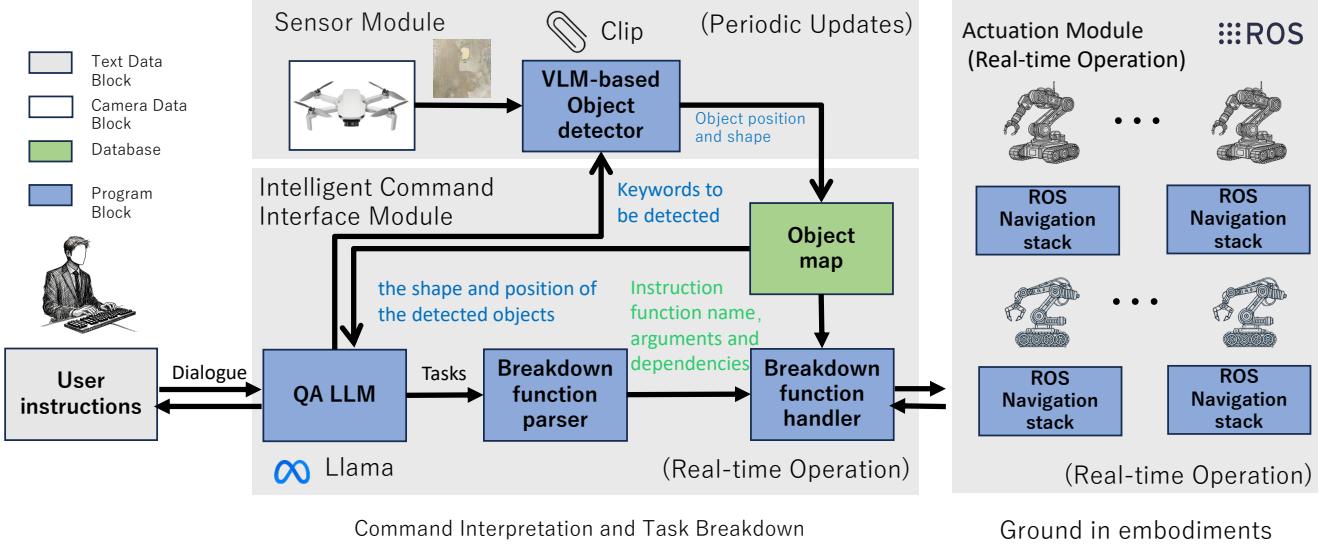


Fig. 1. An overview of the DART-LLM system architecture. The system is divided into three main modules: the Sensor Module, the Intelligent Command Interface Module, and the Actuation Module. The Sensor Module captures and processes data using a Vision-Language Model (VLM)-based object detector, updating the object map with detected items. The Intelligent Command Interface Module interprets user instructions via a Question-Answering Large Language Model (QA LLM), decomposing tasks into subtasks with dependencies through the Breakdown Function Parser and Handler. This allows the establishment of complex task dependencies and coordination between multiple robots. Finally, the Actuation Module executes real-time operations using the ROS Navigation stack, guiding each robot according to the parsed and dependency-aware instructions.

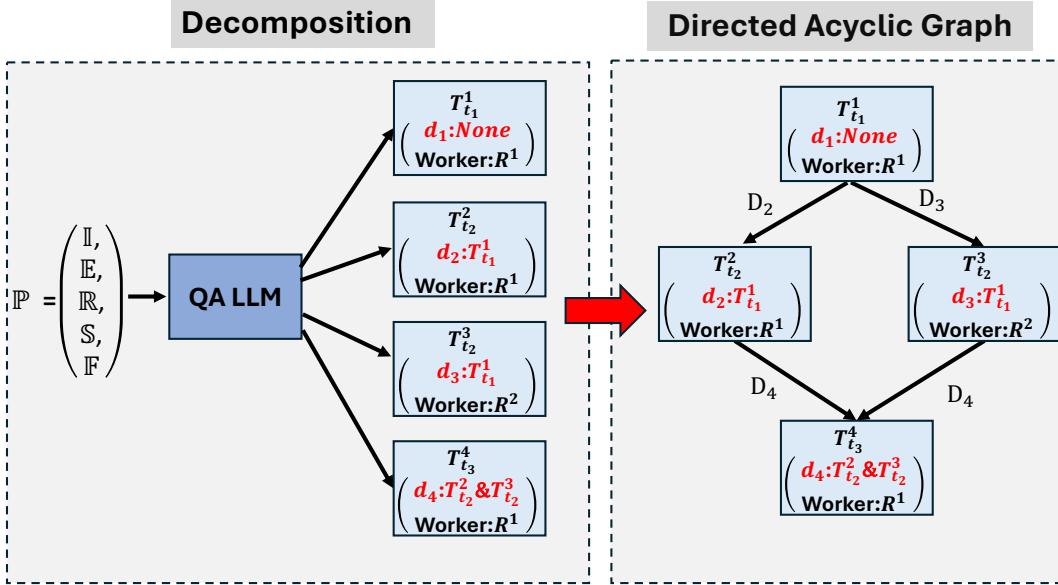


Fig. 2. Dependency-aware task decomposition in DART-LLM. Left: QA LLM decomposes high-level instruction into subtasks with explicit dependency lists (d_k) and assigned workers. Right: Construction of the Directed Acyclic Graph (DAG) based on dependency relationships, establishing the execution order where $T_{t_1}^1$ is executed first, followed by parallel execution of $T_{t_2}^2$ and $T_{t_2}^3$, and finally $T_{t_3}^4$ after its dependencies are satisfied.

D. Dependency Relationships Between Tasks

Complex tasks in multi-robot systems often involve multiple subtasks with interdependencies. These dependencies are managed through task decomposition using a Directed Acyclic Graph (DAG). The DAG is defined as $G = (\mathbb{T}, \mathbb{D})$, where:

$$\mathbb{D} \subseteq \mathbb{T} \times \mathbb{T} \quad (7)$$

represents the set of directed edges indicating dependencies between subtasks. An edge $\mathbb{D}_k = (T_{t_i}^{k-1}, T_{t_j}^k) \in \mathbb{D}$ indicates that subtask $T_{t_j}^k$ depends on the successful completion of subtask $T_{t_i}^{k-1}$.

III. THE DART-LLM FRAMEWORK

This section provides an overview of DART-LLM. The overall control architecture of DART-LLM is illustrated in Fig. 1. Communication between all modules is based on

Robot Operating System 2 (ROS2) topics [13].

To illustrate the workflow of DART-LLM, Algorithm 1 presents the pseudocode for the system’s dependency-aware multi-robot task decomposition and execution process. This algorithm outlines the system’s processing of natural language instructions through multiple stages, including instruction parsing, dependency-aware multi-robot task decomposition, dependency-aware task parsing and robot assignment, and task execution with actuation. Following this high-level overview, we detail each module shown in Fig. 1 and explain how they implement the different stages of Algorithm 1 in the subsequent subsections.

Algorithm 1 Dependency-Aware Multi-Robot Task Decomposition and Execution

Require: Natural language instruction \mathbb{I} , environment information \mathbb{E} , robot set \mathbb{R} with skill sets \mathbb{S} and a few-shot example set \mathbb{F}

- 1: **Instruction Parsing**
- 2: Provide QA LLM module with \mathbb{I} for parsing
- 3: Generate prompt $\mathbb{P} = (\mathbb{I}, \mathbb{E}, \mathbb{R}, \mathbb{S}, \mathbb{F})$ with explicit instruction that robot skills can combine through teamwork
- 4: Obtain parsed command structure in JSON format, including subtasks, robot assignments, dependencies, and object keywords
- 5: **Dependency-Aware Multi-Robot Task Decomposition**
- 6: Decompose \mathbb{I} into subtasks $\mathbb{T} = \{T_{t_1}^1, T_{t_2}^1, \dots, T_{t_j}^K\}$ with QA LLM, where each $T_{t_j}^k$ corresponds to an atomic skill in \mathbb{S}
- 7: Define dependency relationships $\mathbb{D} \subseteq \mathbb{T} \times \mathbb{T}$ during decomposition
- 8: Represent dependencies using Directed Acyclic Graph $G = (\mathbb{T}, \mathbb{D})$, where \mathbb{D} enables task coordination and implicit team formation
- 9: **Dependency-Aware Task Parsing and Robot Assignment**
- 10: Initialize object map database with object locations and attributes using VLM-based object detector
- 11: **for** each subtask $T_{t_j}^k$ in topological order of DAG G **do**
- 12: **if** $T_{t_j}^k$ has no dependencies in \mathbb{D} **then**
- 13: Execute $T_{t_j}^k$ in parallel
- 14: **else**
- 15: Wait for all dependencies of $T_{t_j}^k$ in \mathbb{D} to complete
- 16: Execute $T_{t_j}^k$
- 17: **end if**
- 18: **end for**
- 19: **for** each subtask $T_{t_j}^k \in \mathbb{T}$ **do**
- 20: **if** QA LLM specified a particular robot for $T_{t_j}^k$ **then**
- 21: Assign $T_{t_j}^k$ to the specified robot from \mathbb{R} with corresponding skill in \mathbb{S}
- 22: **else**
- 23: Use Breakdown function parser to find available robot of specified type from \mathbb{R} with matching skill
- 24: **end if**
- 25: **end for**
- 26: **Task Execution using Actuation Module**
- 27: **for** each parsed subtask $T_{t_j}^k \in \mathbb{T}$ **do**
- 28: Execute assigned robot-specific atomic skill based on \mathbb{S}
- 29: **end for**
- 30: **Update Object Map Database**
- 31: Continuously update object map based on real-time VLM object detector feedback

A. Instruction Parsing and Dependency-Aware Multi-Robot Task Decomposition Using the QA LLM Module

This subsection details steps 1-8 of Algorithm 1, focusing on how high-level natural language instructions \mathbb{I} are processed by the QA LLM module and decomposed into executable subtasks with dependency relationships.

High-level natural language instructions \mathbb{I} serve as the input for DART-LLM. The input is first processed by the

QA LLM for Instruction Parsing and Dependency-Aware Task Decomposition. The QA LLM parses the high-level instructions \mathbb{I} while considering the environment \mathbb{E} and the skills \mathbb{S} possessed by all robots \mathbb{R} .

In this module, we employ a DAG-based task decomposition system that decomposes a given instruction \mathbb{I} into subtasks while also generating a dependency list for each subtask to ensure correct execution sequencing. As shown in Fig. 2, when instruction \mathbb{I} is decomposed into a set of subtasks \mathbb{T} by QA LLM, each subtask $T_{t_j}^k$ is assigned a dependency list d_k , which specifies all prerequisite subtasks that must be completed before executing $T_{t_j}^k$. The system then determines the execution order t_i automatically based on these dependencies and constructs a directed edge set D_k for each d_k , thereby forming a subtask execution graph as illustrated. Finally, the system executes subtasks strictly in the specified order until the overall task objective is achieved.

During Task Decomposition, DART-LLM uses awareness of dependency relationships \mathbb{D} to decompose the task into multiple subtasks $\mathbb{T} = \{T_{t_1}^1, T_{t_2}^1, \dots, T_{t_j}^K\}$, where each subtask corresponds to an atomic skill in \mathbb{S} . This facilitates parallel execution of decomposed subtasks without mutual dependencies and clearly expresses dependencies between subtasks.

Therefore, the prompt \mathbb{P} for the QA LLM is defined as $\mathbb{P} = (\mathbb{I}, \mathbb{E}, \mathbb{R}, \mathbb{S}, \mathbb{F})$, where \mathbb{F} represents the few-shot example, with explicit instructions that robot skills can combine through teamwork.

The output of the QA LLM uses a structured standard format to handle the parsed commands and their dependencies. The standardized structure follows JSON syntax, as shown below:

```
{
  "instruction_function": {
    "name": "<breakdown function 1>",
    "dependencies": ["<dep 1>", "<dep 2>", "<dep n>"]
  },
  "object_keywords": ["<key 1>", "<key 2>", "<key n>"],

  "instruction_function": {
    "name": "<breakdown function 2>",
    "dependencies": ["<dep 1>", "<dep 2>", "<dep n>"]
  },
  "object_keywords": ["<key 1>", "<key 2>", "<key n>"],

  ...
  "instruction_function": {
    "name": "<breakdown function m>",
    "dependencies": ["<dep 1>", "<dep 2>", "<dep n>"]
  },
  "object_keywords": ["<key 1>", "<key 2>", "<key n>"]
}
```

The structured standard format message contains the

names of the breakdown functions (i.e., atomic action skills) along with their parameters, object keywords, and dependencies. These atomic action skills belong to \mathbb{S} . For specific atomic action skills, please refer to subsection III-C. The parameters of the atomic action skills are searched within the *object map* database. For details, please refer to subsection III-D.

B. Dependency-Aware Task Parsing and Robot Assignment Using the Breakdown Function Modules

This subsection elaborates on the implementation details of steps 9-25 of Algorithm 1, particularly how the system processes tasks according to their dependencies and assigns robots to subtasks.

The system first initializes the object map database with object locations and attributes using the VLM-based object detector. This provides the environmental context necessary for task execution and robot coordination.

Following the DAG structure established during task decomposition, the system processes each subtask $T_{t_j}^k$ in topological order of the graph G . This ordering ensures that subtasks are executed according to their dependencies:
- Subtasks without dependencies are executed in parallel, maximizing efficiency
- Subtasks with dependencies wait for their prerequisite tasks to complete before execution

The *Breakdown Function Parser* module then processes each subtask $T_{t_j}^k \in \mathbb{T}$ for robot assignment. If the QA LLM has specified a particular robot for a subtask, that robot is directly assigned to the task. Otherwise, the *Breakdown Function Parser* finds an available robot of the specified type from \mathbb{R} with the matching skill required for the subtask.

This dependency-aware approach to task parsing and robot assignment ensures that complex tasks requiring collaboration between multiple robots are executed efficiently while respecting the necessary order of operations defined by the dependency relationships in \mathbb{D} .

C. Task Execution and Grounding Using the Actuation Module

This subsection corresponds to steps 26-29 of Algorithm 1, following the dependency-aware execution and robot assignment described previously, detailing how the system grounds instructions in robot actions through the actuation module.

The *Actuation* module grounds the instructions in embeddings by executing each parsed subtask $T_{t_j}^k \in \mathbb{T}$ using the corresponding atomic skill from \mathbb{S} . All atomic action skills are asynchronously executed by the *Actuation* module.

These atomic action skills are divided into two categories: navigation skills and robot-specific skills, both contained within \mathbb{S} . The implementation of navigation skills is achieved through the ROS Navigation stack [14], while robot-specific skills are realized by creating specific action sets for each robot.

1) *Navigation Skills*: The navigation skills are responsible for directing the movement and area access permissions of all robots or specific robots within a designated area. These skills include whether to avoid certain areas, return to the

initial position, or move to a specified location. Table II provides detailed descriptions of each navigation skill.

2) *Robot Skills*: The robot-specific skills are tailored to the unique operational capabilities of different robot types, such as excavators and dump trucks. These skills handle tasks such as digging, unloading, and loading. Table III provides detailed descriptions of each robot skill. It should be noted that different types of robots have different atomic action skill definitions.

D. Update Object Map Database Using the VLM-based Object Detector Module

This subsection explains steps 30-31 of Algorithm 1, describing how the system continuously updates its object map database through real-time sensing, which is crucial for maintaining an accurate representation of the environment throughout task execution.

The DART-LLM updates the *object map* database in environment \mathbb{E} using the VLM-based object detector module. The object detector focuses on identifying the object keywords that were extracted during QA LLM's instruction parsing phase, ensuring that all entities relevant to the current task are properly detected and tracked.

This module first converts all images captured by Unmanned Aerial Vehicles (UAVs) into bird's-eye views. These views are processed to detect and recognize objects in the environment \mathbb{E} . Selective search [15] is then used on the bird's-eye views to generate candidate bounding boxes. Following region proposals, a CLIP-based model [16] is employed for identification, updating the *object map* database with the names, locations, and shapes of the objects. The updated information is then used by the Breakdown Function Handler to ensure accurate parameter resolution for the atomic skills during execution.

IV. EXPERIMENTS

A. Experimental Setup

To evaluate the performance of the DART-LLM system across different task sets and conduct a quantitative comparison with baseline methods, this study employs construction robots that are highly sensitive to subtask sequencing. These robots serve as test subjects to evaluate the system's effectiveness. Errors in subtask sequencing can lead to task failure. For instance, if a dump truck departs from the soil pile before the excavator has completed unloading, the loading and unloading operation will be unsuccessful. We constructed a benchmark dataset designed specifically for evaluating natural language-driven task planning in multi-robot construction scenarios. The dataset includes three task levels: L1, L2, and L3. The task numbering reflects their respective levels. L1 tasks are fundamental tasks involving a single robot equipped with all necessary skills, eliminating the need for multi-robot coordination. L2 tasks require collaboration among multiple robots. These tasks consist of subtasks that must be executed in a specific sequence, though each subtask can still be completed independently by a single robot. L3 tasks are complex, requiring multiple robots to work together.

TABLE II
NAVIGATION SKILL DESCRIPTIONS

Number	Skill Name	Description
N1	avoid_areas_for_all_robots	Sets the cost map to make all robots avoid specified areas.
	avoid_areas_for_specific_robots	Sets the cost map for selected robots to avoid specified areas.
N2	target_area_for_all_robots	Guides all robots to target points near the specified area.
	target_area_for_specific_robots	Guides selected robots to target points near the specified area.
N3	allow_areas_for_all_robots	Configures the cost map to allow all robots to access specified areas.
	allow_areas_for_specific_robots	Configures the cost map for selected robots to access specified areas.
N4	return_to_start_for_all_robots	Instructs all robots to return to their initial starting position.
	return_to_start_for_specific_robots	Instructs selected robots to return to their initial starting position.

TABLE III
ROBOT SKILL DESCRIPTIONS

Number	Skill Name	Description
FE1	excavator_digging	Instructs the excavator to dig at the specified target location.
FE2	excavator_unloading	Instructs the excavator to unload at the specified target location.
FD1	dump_loading	Instructs the dump truck to load materials at the specified target location.
FD2	dump_unloading	Instructs the dump truck to unload materials at the specified target location.

TABLE IV
DESCRIPTION OF SPECIFIC TASKS IN EACH TASK LEVEL

Task Level	Task Number	Task Description	Skill Number	Applicable Robots
Level 1	L1-T1-001	Inspect a puddle	N1, N2, N3, N4	Excavator or Dump Truck
	L1-T2-001	Clear an obstacle	N1, N2, N3, N4, FE1, FE2	Excavator
Level 2	L2-T1-001	Excavate soil	N1, N2, N3, N4, FE1, FD1, FE2, FD2	Excavator, Dump Truck
	L2-T2-001	Transport soil to the dump truck's initial position	N1, N2, N3, N4, FE1, FD1, FE2, FD2	Excavator, Dump Truck
Level 3	L3-T1-001	Clear the obstacle, then dig soil	N1, N2, N3, N4, FE1, FD1, FE2, FD2	Excavator, Dump Truck
	L3-T2-001	Clear the obstacle, then inspect the puddle	N1, N2, N3, N4, FE1, FE2, FD2	Excavator, Dump Truck

These tasks are decomposed into multiple interdependent subtasks, which must follow a strict execution order but may allow some parallel execution where feasible. The dataset comprises 102 high-level instructions: 47 L1 tasks, 33 L2 tasks, and 22 L3 tasks. It is available on our project website: [project-dart-llm](#). Task execution can be visually validated in both simulated and physical robot environments. The simulation environment is built on the Unity platform and employs the PhysX physics engine. Both the simulation and physical environments feature two C30R tracked transport robots manufactured by Yanmar (Japan) and one ZX120 excavator manufactured by Hitachi Construction Machinery (Japan). Table IV presents two specific example tasks for each task level, along with the required skills and applicable robot types.

B. Evaluation Metrics

To evaluate the performance of the proposed approach on the dataset, this study employs five evaluation metrics:

- **SR (Success Rate):** Whether the task is completely successful.
- **IPA (Instruction Parsing Accuracy):** The accuracy of instruction parsing, defined as the proportion of correctly parsed instructions mapped to the correct atomic action function names and parameters.
- **DSR (Dependency Satisfaction Rate):** The rate of dependency satisfaction, referring to the proportion of subtasks completed in the correct order of dependencies.
- **SGSR (Semantic Grounding Success Rate):** The rate of successful semantic grounding, defined as the propor-

tion of code generated that can be successfully parsed by the breakdown function parse module.

- **RTR (Response Time Reliability):** Assesses the models response time stability across multiple executions, using average response time and standard deviation to evaluate performance consistency under different conditions.

The evaluation relies on dataset ground truth to measure each metric.

C. Experimental Results

- 1) **Evaluation Against Baselines Using Different LLMs:** This study evaluated DART-LLM using multiple foundation models: Llama-3.1-8B [17], GPT-4o [18], GPT-3.5-turbo [19], Claude-3.5-Haiku [20], and DeepSeek-r1-671B [21]. We also compared our approach with the SMART-LLM baseline implemented with Llama-3.1-8B and DeepSeek-r1-671B. Table V shows the evaluation results across tasks of L1, L2, and L3 complexity levels. Notably, all 102 test cases were unseen data, using few-shot examples solely for prompting and excluding them from the test set. All evaluation metrics range from 0 to 1, with higher values indicating better performance.

The results indicate that for task level L1, all implementations achieved perfect scores across SR, IPA, DSR, and SGSR metrics (all 1.00). However, notable differences appear in the RTR metric, where DART-LLM with Llama3.1 achieved the highest score of 0.96, significantly outperforming other models including SMART-LLM implementations (0.24 and 0.08).

As task complexity increases to L2, performance differences become more pronounced. DART-LLM with DeepSeek-r1 achieved the highest SR of 0.97. GPT-4o followed with 0.96, Claude3.5 with 0.90, GPT-3.5-turbo with 0.87, and Llama3.1 with 0.85. All DART-LLM implementations maintained perfect IPA scores, demonstrating the effectiveness of proposed structured JSON format and dependency-aware task decomposition method. In contrast, the SMART-LLM baseline showed substantial performance degradation, with SR scores of 0.36 and 0.78 for Llama3.1 and DeepSeek-r1 respectively, and reduced IPA scores (0.57 and 0.78). The explicit specification of dependencies in proposed approach significantly improved model logical reasoning capabilities, as evidenced by the high DSR scores across all DART-LLM implementations compared to the SMART-LLM baseline. For RTR, DART-LLM with Llama3.1 again demonstrated superior performance (0.90) compared to other models, while SMART-LLM implementations struggled significantly (0.19 and 0.04).

At task level L3, DART-LLM with DeepSeek-r1 demonstrated the best performance with an SR of 0.94. GPT-4o followed with 0.93, while Claude3.5, Llama3.1, and GPT-3.5-turbo achieved 0.86, 0.84, and 0.80 respectively. The SMART-LLM baseline performance degraded substantially, with SR scores dropping to 0.24 and 0.65 for Llama3.1 and DeepSeek-r1 respectively.

The structured JSON format consistently contributed to high RTR scores for DART-LLM (Llama3.1), maintaining 0.96, 0.90, and 0.86 across L1, L2, and L3 respectively. In contrast, the SMART-LLM baseline implementations showed markedly lower RTR scores (ranging from 0.02 to 0.24), primarily due to its direct generation of more complex Python code, which led to increased planning time for code generation. While the SMART-LLM baseline achieved reasonable SR scores with DeepSeek-r1, its low RTR indicates significant practical limitations in real-world applications where code execution is essential.

Overall, the DART-LLM system showed superior performance when using DeepSeek-r1 for SR metrics and Llama3.1 for RTR metrics. Notably, the Llama3.1 model, with only 8B parameters, performed competitively with much larger models and achieved significantly higher RTR scores than all other approaches. This suggests that using smaller models with proposed JSON-based command generation approach provides an excellent balance between performance and deployability, offering better real-time capabilities while maintaining high success rates.

2) Ablation Study: To evaluate the effectiveness of proposed dependency-aware approach, we conducted an ablation study on L3 complexity tasks using five different LLMs. We compared two conditions: "With Dependency" (i.e., using a DAG to represent task dependencies) and "Without Dependency" (i.e., not using a DAG to specify task relationships).

Fig. 3 presents the SR under both conditions. The experimental results indicate that the "Without Dependency" condition (i.e., without using a DAG) significantly reduces the performance of all models. Among them, Llama3.1

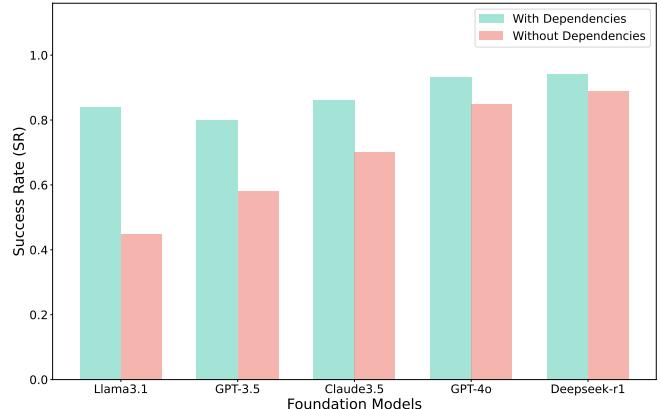


Fig. 3. Success Rate (SR) comparison between "With Dependencies" (using DAG) and "Without Dependencies" (not using DAG) for L3 complexity tasks across different models.

exhibited the largest performance drop (from 0.84 to 0.45). Even state-of-the-art models such as GPT-4o and DeepSeek-r1 experienced notable declines (from 0.93 to 0.85 and from 0.94 to 0.89, respectively). The pronounced performance gap in smaller models suggests that explicit dependency modeling via DAG can effectively compensate for their limited reasoning capabilities, demonstrating that proposed approach enables smaller, more deployable models to achieve competitive performance on complex tasks.

3) Deployment in Real World: To evaluate the deployment capability of DART-LLM, we conducted tests in a real-world environment. Fig. 4 presents the results of the L2-T1-001 task using the Llama3.1 model. The results indicate that DART-LLM demonstrates good applicability on real robots.

For more experiments, please visit the project website [project-dart-llm](#).

V. CONCLUSION AND FUTURE WORK

This paper presented DART-LLM, a novel framework that leverages LLMs and DAG-based dependency modeling for coordinated multi-robot task execution. The proposed system integrates four key modules: a QA LLM module for instruction parsing and dependency-aware task decomposition, a Breakdown Function module for task parsing and robot assignment, an Actuation module for executing robot-specific skills, and a VLM-based object detector module for environmental perception. This architecture demonstrated superior performance compared to existing methods across all evaluation metrics, with DART-LLM (DeepSeek-r1) achieving a 94% success rate on complex tasks while maintaining perfect instruction parsing accuracy. The smaller Llama3.1 model exhibited exceptional response time reliability, making it suitable for resource-constrained deployments. Furthermore, ablation experiments reveal that DAG-based dependency modeling significantly enhances model performance. Notably, it compensates for the limited reasoning capabilities of smaller models, enabling them to execute complex tasks more effectively. Real-world testing validated the system's practical applicability. Future work will focus on

TABLE V
EVALUATION OF DART-LLM AND SMART-LLM BASELINE ACROSS L1, L2, AND L3 TASKS.

Models	L1					L2					L3				
	SR	IPA	DSR	SGSR	RTR	SR	IPA	DSR	SGSR	RTR	SR	IPA	DSR	SGSR	RTR
DART-LLM (Llama3.1)	1.00	1.00	1.00	1.00	0.96	0.85	1.00	0.85	0.90	0.90	0.84	0.93	0.84	0.88	0.86
DART-LLM (GPT-3.5-turbo)	1.00	1.00	1.00	1.00	0.75	0.87	1.00	0.87	0.87	0.68	0.80	0.97	0.85	0.80	0.66
DART-LLM (GPT-4o)	1.00	1.00	1.00	1.00	0.55	0.96	1.00	0.96	0.96	0.50	0.93	1.00	0.93	0.93	0.45
DART-LLM (Claude3.5)	1.00	1.00	1.00	1.00	0.70	0.90	1.00	0.90	0.95	0.63	0.86	1.00	0.86	0.90	0.60
DART-LLM (DeepSeek-r1)	1.00	1.00	1.00	1.00	0.30	0.97	1.00	0.97	0.97	0.25	0.94	1.00	0.94	0.94	0.20
SMART-LLM(Llama3.1) [12]	1.00	1.00	1.00	1.00	0.24	0.36	0.57	0.46	0.36	0.19	0.24	0.37	0.43	0.24	0.16
SMART-LLM(DeepSeek-r1) [12]	1.00	1.00	1.00	1.00	0.08	0.78	0.78	0.91	0.87	0.04	0.65	0.65	0.86	0.78	0.02

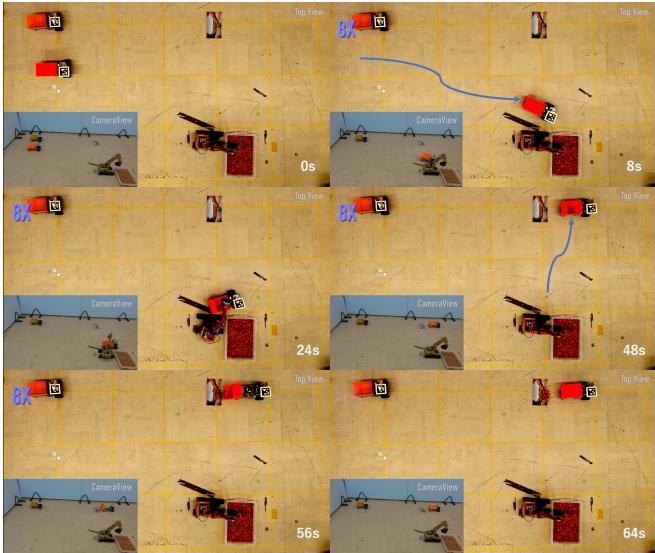


Fig. 4. DART-LLM (Llama3.1) in the L2-T1-001 Task Using Real Robots: The sequence begins at 0 [s] with the initial positioning of the Dump Truck at the starting location. At 8 [s], the Excavator prepares to dig soil. By 24 [s], the Excavator transfers the excavated soil into the Dump Truck, completing the loading operation. At 48 [s], the loaded Dump Truck navigates towards the designated dumping area. At 56 [s], the Dump Truck unloads the soil at the puddle location. Finally, at 64 [s], the Dump Truck completes the task by retracting its vessel.

scaling DART-LLM for larger multi-robot teams to enhance system applicability. Additionally, optimizing the trade-off between model size and performance will be investigated to accommodate diverse deployment scenarios.

ACKNOWLEDGMENT

This work is supported by JST [Moonshot Research and Development], Grant Number [JPMJMS2032]. The work was first submitted to an IEEE conference on September 15, 2024.

REFERENCES

- [1] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, “Decision transformer: Reinforcement learning via sequence modeling,” *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.
- [2] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg *et al.*, “A generalist agent,” *arXiv preprint arXiv:2205.06175*, 2022.
- [3] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, “Do as i can, not as i say: Grounding language in robotic affordances,” *arXiv preprint arXiv:2204.01691*, 2022.
- [4] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, “Inner monologue: Embodied reasoning through planning with language models,” *arXiv preprint arXiv:2207.05608*, 2022.
- [5] A. Brohan, N. Brown, J. Carbalaj, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu *et al.*, “Rt-1: Robotics transformer for real-world control at scale,” *arXiv preprint arXiv:2212.06817*, 2022.
- [6] A. Brohan, N. Brown, J. Carbalaj, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn *et al.*, “Rt-2: Vision-language-action models transfer web knowledge to robotic control,” *arXiv preprint arXiv:2307.15818*, 2023.
- [7] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu *et al.*, “Palm-e: An embodied multimodal language model,” *arXiv preprint arXiv:2303.03378*, 2023.
- [8] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, “Code as policies: Language model programs for embodied control,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.
- [9] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, “Voxposer: Composable 3d value maps for robotic manipulation with language models,” *arXiv preprint arXiv:2307.05973*, 2023.
- [10] K. Nagatani, M. Abe, K. Osuka, P.-j. Chun, T. Okatani, M. Nishio, S. Chikushi, T. Matsubara, Y. Ikemoto, and H. Asama, “Innovative technologies for infrastructure construction and maintenance through collaborative robots based on an open design approach,” *Advanced Robotics*, vol. 35, no. 11, pp. 715–722, 2021.
- [11] Z. Mandi, S. Jain, and S. Song, “Roco: Dialectic multi-robot collaboration with large language models,” *arXiv preprint arXiv:2307.04738*, 2023.
- [12] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, “Smart-llm: Smart multi-agent robot task planning using large language models,” *arXiv preprint arXiv:2309.10062*, 2023.
- [13] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.
- [14] S. Macenski, F. Martin, R. White, and J. Gins Clavero, “The marathon 2: A navigation system,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
- [15] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient graph-based image segmentation,” *International journal of computer vision*, vol. 59, pp. 167–181, 2004.
- [16] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, “Learning transferable

- visual models from natural language supervision,” in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.
- [17] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.
 - [18] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, “Gpt-4o system card,” *arXiv preprint arXiv:2410.21276*, 2024.
 - [19] “OpenAI GPT-3.5 Turbo,” accessed: 2025-03-01. [Online]. Available: https://openai.com/index/gpt-3-5-turbo-fine-tuning-and-api-updates/?utm_source=chatgpt.com
 - [20] “Anthropic Claude 3.5 Haiku,” accessed: 2025-03-01. [Online]. Available: <https://www.anthropic.com/news/3-5-models-and-computer-use>
 - [21] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” *arXiv preprint arXiv:2501.12948*, 2025.