2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

# LLM-HBT: Dynamic Behavior Tree Construction for Adaptive Coordination in Heterogeneous Robots

Chao-ran Wang[1], Jingyuan Sun*, Yan-hui Zhang, Mingyu Zhang, Chang-ju Wu*

***Abstract*— We introduce a novel framework for automatic behavior tree (BT) construction in heterogeneous multirobot systems, designed to address the challenges of adaptability and robustness in dynamic environments. Traditional robots are limited by fixed functional attributes and cannot efficiently reconfigure their strategies in response to task failures or environmental changes. To overcome this limitation, we leverage large language models (LLMs) to generate and extend BTs dynamically, combining the reasoning and generalization power of LLMs with the modularity and recovery capability of BTs. The proposed framework consists of four interconnected modules—task initialization, task assignment, BT update, and failure node detection—which operate in a closed loop. Robots tick their BTs during execution, and upon encountering a failure node, they can either extend the tree locally or invoke a centralized virtual coordinator (Alex) to reassign subtasks and synchronize BTs across peers. This design enables long-term cooperative execution in heterogeneous teams. We validate the framework on 60 tasks across three simulated scenarios and in a real-world café environment with a robotic arm and a wheeled-legged robot. Results show that our method consistently outperforms baseline approaches in task success rate, robustness, and scalability, demonstrating its effectiveness for multirobot collaboration in complex scenarios.**

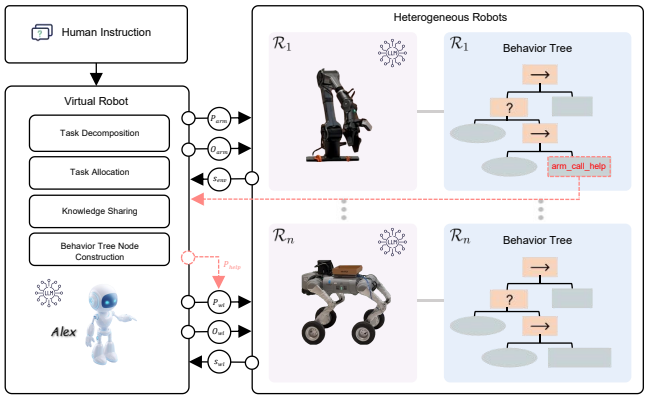


Fig. 1. The framework consists of three main components: (i) Human instruction, where natural language commands are provided; (ii) Virtual robot (*Alex*), which decomposes tasks, allocates subtasks, and shares knowledge; and (iii) Heterogeneous robots, including drones, robotic arms, and quadrupeds, which execute their own behavior trees collaboratively.

## I. Introduction

multirobot systems have demonstrated significant potential in improving operational efficiency by distributing tasks across heterogeneous robots. However, existing learning-based methods often struggle to generalize to new environments, as they rely heavily on predefined priors and limited training data [1]. This limitation becomes critical in dynamic and unstructured settings [2], where robots must adapt to failures, environmental changes, and unforeseen conditions. Traditional decision-making frameworks are either too rigid to support online reconfiguration or too brittle to ensure long-term robustness [3], [4]. Addressing these challenges requires a methodology that enables heterogeneous robots to dynamically reconfigure their task strategies while maintaining scalable and reliable execution.

Despite these advances, existing approaches still face fundamental limitations when applied to long-term coordination in heterogeneous multirobot systems [2]. LLM-based methods, while demonstrating strong reasoning capabilities, often generate task plans in a single-shot manner and lack mechanisms for online correction once execution begins [5]. This makes them fragile in dynamic or partially observable environments, where unexpected failures or environmental changes are inevitable. Conversely, behavior-tree-based methods offer modularity and recovery mechanisms through structured execution policies [6], but they rely heavily on manually designed action nodes and predefined task structures [7]. Such reliance restricts their scalability to unseen tasks and their adaptability to heterogeneous robots with diverse capabilities. More critically, current research has not fully addressed how heterogeneous robots can collaboratively reconstruct and share behavior tree (BT) at runtime to ensure persistent task execution [3]. These gaps highlight the need for a unified framework that combines the semantic reasoning strength of LLMs with the structural robustness of BT to enable dynamic, distributed, and long-term multirobot coordination [1].

To bridge these limitations, we propose a novel framework that integrates large language models with BT to enable dynamic task planning and long-term coordination in heterogeneous multirobot systems [5]. At the core of our approach is a centralized allocator that monitors execution progress across robots and detects failures during runtime. When a behavior tree condition node fails, the allocator leverages LLM-based reasoning to infer the most appropriate robot and corresponding actions for recovery [8]. The selected robot then extends its behavior tree by inserting a new subtree that encodes the updated task logic [6]. If the failure requires collaboration, the extended subtree is propagated to additional robots, ensuring distributed consistency while preserving the modularity of BT [7]. Through this closed-

Corresponding author: Chang-ju Wu is with the School of Aeronautic and Astronautics, Zhejiang University, Hangzhou 310027, China. Jingyuan Sun is with Shanghai Huawei Technologies Co., Ltd., Shanghai 201799, China (e-mail: sunjingyuan1@huawei.com,

loop cycle of failure detection, reasoning, and tree adaptation, our framework allows heterogeneous robots to continuously refine their execution strategies, complement one another's functional limitations, and maintain robustness in dynamic and uncertain environments [3].

To validate the proposed framework, we design a new benchmark that incorporates diverse simulated scenarios and heterogeneous robot teams, including drones, quadruped robot and robotic arms, tasked with solving complex, long-horizon objectives. Experimental results demonstrate that our method significantly improves task success rates and adaptability compared to existing approaches. The main contributions of this work are summarized as follows:

- A dynamic framework that integrates large language model (LLM) reasoning with behavior trees (BTs) for heterogeneous multirobot coordination.
- A hybrid centralized–distributed mechanism that enables runtime adaptation through local BT extension and centralized task reallocation.
- A new benchmark covering diverse simulated tasks and a real-world environment, demonstrating the robustness and scalability of the proposed approach.

## II. Related works

### A. Behavior Tree Automatic Design

Behavior Trees (BTs) have been widely recognized for their modularity, reusability, and interpretability in robotic decision-making [9]. Recent works have begun to explore automatic BT generation. For instance, [10] proposed an LLM-based approach for robotic arm control, enabling dynamic addition of operations to handle environmental changes. Similarly, [11] developed a fine-tuned Llama2 model to generate standard C++ BT strategies. Other studies have integrated optimization-based techniques, such as hierarchical auction algorithms, into BT construction to improve multi-agent task allocation [12], [13]. However, these approaches typically require manual cost function design or operate under simplified assumptions, limiting their adaptability in unstructured environments. In contrast, our method eliminates the need for handcrafted cost functions by leveraging LLM reasoning to interpret environmental observations and dynamically extend BT structures across heterogeneous robots.

### B. multirobot Planning with LLM

The integration of large language models (LLMs) into collaborative robotics has recently gained increasing attention. [14] introduced a language-based task allocation method for multi-arm manipulation, while [15] leveraged pre-trained LLMs for high-level communication and low-level path planning. To improve efficiency in complex scenarios, [16] proposed a "prompt-failure-feedback" mechanism that enhances task reasoning through historical context. Although effective, such methods may suffer from ambiguous interpretations when facing dynamic environmental changes. Parallel efforts have explored multi-round dialogue mechanisms for task planning [17]–[19], but these studies mainly target homogeneous robot systems. Zhao et al. [20] combined LLMs with Monte Carlo tree search (MCTS) to improve planning efficiency by generating world models as priors. While these methods demonstrate the promise of LLMs for multirobot collaboration, they lack a formalized and adaptable execution framework. More importantly, the coordination of heterogeneous robots—each with distinct capabilities—remains underexplored.

In summary, prior research on automatic BT design has advanced modular decision-making but often relies on handcrafted cost functions or limited adaptation mechanisms. Meanwhile, recent LLM-based multirobot planning methods primarily focus on homogeneous systems and lack a structured execution framework. To the best of our knowledge, no existing work has integrated LLM reasoning with dynamic BT construction for heterogeneous multirobot systems. This study fills that gap by proposing a unified framework that combines the reasoning power of LLMs with the modular adaptability of BTs. Our approach enables online task reconfiguration, distributed execution, and long-term robustness in complex environments.

## III. Preliminaries

In this section, we formalize the core concepts and notations employed in this work, covering the representation of heterogeneous robot teams, the behavior-tree (BT) formalism, and the specification of action preconditions and effects. These definitions establish the theoretical basis for the LLM-HBT framework developed in the subsequent sections.

### A. Heterogeneous multirobot Systems

We consider a heterogeneous multirobot system (HMRS) $\mathcal{R} = \{r_1, \ldots, r_N\}$, where $r_i$ is the $i$-th robot and $N$ is the total number of robots. Each robot has an action space

$$\mathcal{A}_i = \{a_i^1, \ldots, a_i^{m_i}\}, \tag{1}$$

where $m_i$ is the number of primitive actions $a_i^j$ available to $r_i$ (e.g., *MoveTo*, *Grab*, *TakeOff*). Heterogeneity arises from $\mathcal{A}_i \neq \mathcal{A}_j$ for $i \neq j$, reflecting differences in morphology and capabilities.

A task $\tau$ is represented by a set or sequence of required actions $\mathcal{A}_\tau \subseteq \bigcup_i \mathcal{A}_i$. Assignment of $\tau$ to $r_i$ is feasible if $\mathcal{A}_\tau \subseteq \mathcal{A}_i$. For complex tasks, multiple robots may cooperate by combining complementary actions from their respective action spaces.

### B. Behavior Trees

A Behavior Tree (BT) is a rooted directed tree $T = (V, E, r)$, where $V$ is the set of nodes, $E$ defines parent–child edges, and $r$ is the root. Each node $v \in V$ encodes control logic or an action and returns *Success*, *Failure*, or *Running*. Execution proceeds in discrete *ticks* from root to leaves.

Nodes are:

- **Control nodes** (e.g., *Sequence*, *Fallback*, *Parallel*) govern flow; *Sequence* fails on the first failing child, *Fallback* succeeds on the first succeeding child.

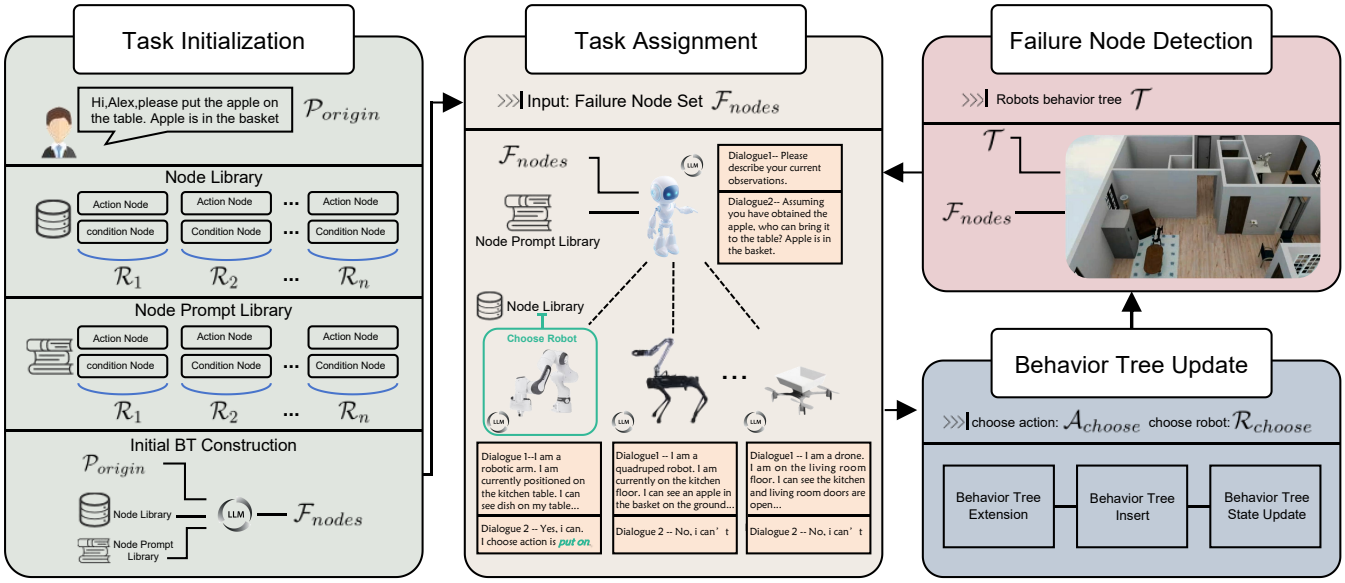


Fig. 2. Overview of the proposed framework for automatic behavior tree (BT) construction and adaptation in heterogeneous multirobot systems. The framework consists of four interconnected modules: (1) **Task Initialization**, where human instructions are translated into an initial BT using an LLM and predefined node templates; (2) **Task Assignment**, where failure nodes trigger the central allocator (Alex) to reassign tasks based on robot capabilities; (3) **Behavior Tree Update**, where new subtrees are inserted or synchronized across robots to extend task execution strategies; and (4) **Failure Node Detection**, where robots continuously tick their BTs to monitor execution and identify bottlenecks. The modules form a closed loop for robust long-horizon collaboration.

- **Leaf nodes** represent actions or conditions, interfacing with perception and actuation.

A *Sequence* node with children $\{c_1, \ldots, c_n\}$ executes as:

$$\text{Seq}(c_1, \ldots, c_n) = \begin{cases} \text{Failure}, & \exists i : c_i = \textit{Failure}, \\ \text{Running}, & \exists i : c_i = \textit{Running}, \\ \text{Success}, & \forall i : c_i = \textit{Success}. \end{cases} \quad (2)$$

Fallback and Parallel follow analogous semantics.

Each action node $a$ is associated with a set of prerequisite conditions, denoted $\text{Pre}(a) = \{c_1^{\text{pre}}, \ldots, c_m^{\text{pre}}\}$, which must hold before execution, and a set of resulting conditions, denoted $\text{Post}(a) = \{c_1^{\text{post}}, \ldots, c_m^{\text{post}}\}$, which are enforced upon successful execution. These conditions link BT execution with symbolic reasoning, allowing task states to be explicitly updated and enabling formal analysis of behavior correctness.

## IV. Method

### A. Framework Overview

The framework (Fig. 2) enables automatic BT construction and adaptation in heterogeneous multirobot systems. Human instructions are first translated into an initial BT using an LLM and a library of node templates (**Task Initialization**). Robots tick their BTs to monitor progress, reporting unsatisfied conditions as failure nodes (**Failure Node Detection**). A centralized allocator, *Alex*, collects observations and queries the LLM to assign suitable robots and actions to resolve failures (**Task Assignment**). The selected actions are integrated into BTs either locally or as synchronized subtrees for collaboration (**Behavior Tree Update**). Iteratively cycling through these modules allows robots to adapt BTs dynamically to environmental feedback, enabling robust and scalable task execution. LLM queries are triggered only when a robot encounters a locally-unresolvable failure node, returning a robot-action suggestion that is compiled into a BT extension.

### B. Task Initialization

Before execution, each robot must construct an initial BT based on the human-provided task description. The execution policy of robot $\mathcal{R}_i$ is represented by a BT $\mathcal{T}_i$, which is updated over time during task execution. Each BT is executed using the $Tick(\cdot)$ function, which propagates control from the root node and returns one of three statuses: *Success*, *Failure*, or *Running*.

To initialize the BTs, we first define a library of action nodes $\mathcal{A}_i$ for each robot $\mathcal{R}_i$, along with their associated preconditions and postconditions. A precondition specifies the environmental requirements that must hold before an action can be executed, while a postcondition describes the state transition resulting from the action. For example, the action *grasp(object)* executed by a robotic arm requires the precondition that the object is within reachable distance, and its postcondition is that the object is held by the gripper.

Given a high-level natural language instruction $\mathcal{P}_{origin}$ from a human researcher, we employ an LLM to decompose the task into condition nodes $\mathcal{C}$ that represent the required states of objects and environments. These condition nodes are assembled into an initial BT $\mathcal{T}_0$ using predefined templates (e.g., *Sequence* nodes to enforce ordering constraints). The resulting tree is assigned a unique identifier (initialized as $id = 1$) and its unsatisfied condition nodes are inserted into the failure node set $\mathcal{F}_{nodes}$. This initialization provides a

structured representation of the task, serving as the foundation for subsequent execution and online adaptation.

The prompts used for LLM reasoning follow a structured template that maps symbolic task representations into natural language descriptions. Each prompt includes the task specification, robot capability descriptions encoded in the behavior tree node library, the current environmental observations, and the failed condition node that requires resolution. Based on this information, the LLM performs reasoning to determine an appropriate robot and a candidate action node whose execution can satisfy the failed condition. The generated outputs are subsequently translated into behavior tree nodes, enabling automatic construction or extension of behavior trees during task execution.

**Algorithm 1** Automatic Design of Behavior Trees for Heterogeneous Multirobots

**Require:** Researcher Prompt $\mathcal{P}_{origin}$
**Ensure:** Multirobot behavior tree $\mathcal{T} = \{\mathcal{T}_1, \cdots, \mathcal{T}_k\}$
Constructing Pre-defined Behavior Tree Nodes $\mathcal{N}$
Text Generation: $f_{nodes} = llm(\mathcal{P}_{origin}, \mathcal{N}\{condition\})$
Initialize Failure nodes set: $\mathcal{F}_{nodes}.add(f_{nodes})$
**while** $\mathcal{F}_{nodes} \neq \emptyset$ **do**
  $f_{exe} = \mathcal{F}_{nodes}[0]$
  $\mathcal{P}_{Alex} = Q(f_{exe}, \mathcal{N}\{query\}, \mathcal{O})$
  $\mathcal{R}_{choose}, \mathcal{P}_{robot} = llm(\mathcal{P}_{Alex})$
  $\mathcal{A}_{choose} = llm(\mathcal{P}_{robot}, \mathcal{N}\{\mathcal{R}_{choose}\}.Con)$
  **if** $\mathcal{A}_{choose} \neq None$ **then**
    $\mathcal{T}_{ext} = BT\_Extension(\mathcal{A}_{choose}, \mathcal{N}\{\mathcal{R}_{choose}\})$
    $\mathcal{T}_{\mathcal{R}_{choose}} = BT\_Melt(\mathcal{T}_{\mathcal{R}_{choose}}, \mathcal{T}_{ext})$
  **else**
    **Warning: This Task Cannot be Completed.**
  **end if**
  **for** $i \rightarrow N$ **do**
    $f_i, \mathcal{S}_i$=*Tick*($\mathcal{T}_i$)
    **if** $f_i \neq None$ **then**
      $\mathcal{F}_{nodes}.add(f_i)$
    **end if**
    **if** $i \equiv \mathcal{R}.index\{\mathcal{R}_{choose}\}$ **then**
      **if** $Tick(f_{exe}) \equiv success$ **then**
        $\mathcal{F}_{nodes}.delete(f_{exe})$
      **end if**
    **end if**
  **end for**
**end while**

### C. Task Assignment

After initialization, the system enters an iterative execution loop in which each robot evaluates its behavior tree using $Tick(\mathcal{T}_i)$. This process yields both the execution status $\mathcal{S}_i$ and, when applicable, a failed condition node $f_i$. Not all failures require external intervention: a robot first attempts to resolve its own failure by extending its BT using its internal action set $\mathcal{A}_i$. When the robot determines that its capabilities are insufficient to satisfy the failed condition, it triggers a dedicated *call_help* action node.

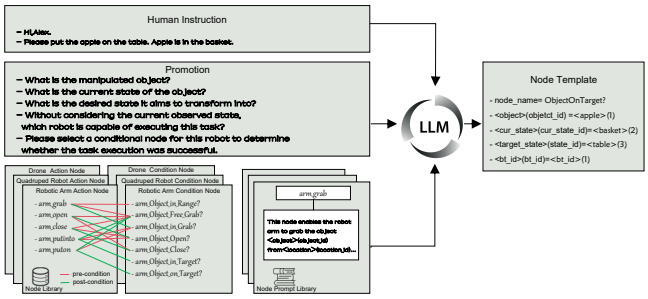


Fig. 3. Illustration of the prompt-based behavior tree construction pipeline. Human natural language instructions are first converted into condition/action node prompts, which are then assembled into a behavior tree for robotic execution.

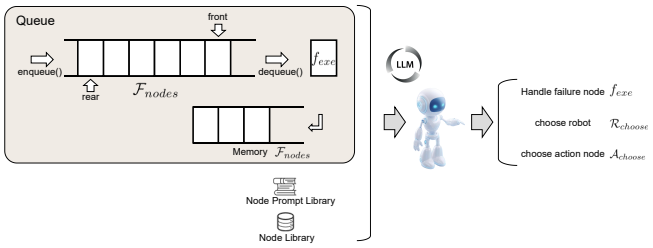


Fig. 4. Process of failure node detection, storage, and reuse in behavior tree execution.

To handle such requests, we introduce a virtual centralized allocator, denoted as *Alex*. The role of *Alex* is to mediate information sharing among robots and to allocate tasks dynamically when local resolution is not feasible. Each call_help request is formulated as a prompt containing three elements: (1) the failed condition node $f_i$, (2) the preconditions and postconditions associated with the node, and (3) the robot's partially observable state $\mathcal{O}_i$, which includes both environmental observations and relational information between objects.

Given this structured prompt, the LLM reasons about which robot $\mathcal{R}_j$ is best suited to resolve the failure, considering the action sets $\mathcal{A}_j$ available across the team. *Alex* then returns a task assignment $\mathcal{P}_{robot}$ and the corresponding candidate action node $\mathcal{A}_{choose}$. The selected robot incorporates this node into its own BT by extending it with a new subtree, while the requesting robot updates its BT to synchronize with the assignment. This mechanism resolves locally-infeasible failures by assigning a capable robot and synchronizing the corresponding BT updates.

### D. Behavior Tree Update

Each action node is associated with preconditions and postconditions to enforce symbolic consistency during BT execution and update.

The update of the BT structure is primarily realized through two operations: *extension* and *insertion*. Let $\mathcal{A}_i$ denote the set of executable action nodes of robot $\mathcal{R}_i$, and $\mathcal{C}_i$ denote its condition nodes. During task execution, each robot executes its BT $\mathcal{T}_i$ through $Tick(\mathcal{T}_i)$, which returns an execution status $\mathcal{S}_i$ and, if applicable, a failure node $f_i$.

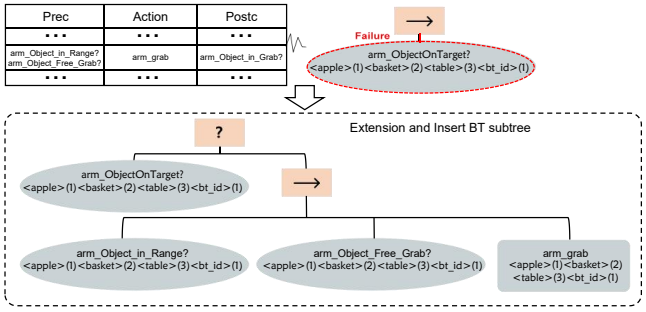


Fig. 5. Behavior tree extension and insertion process. When a failure node is encountered, the system generates an extended subtree through LLM-based reasoning. This subtree is then inserted into the existing behavior tree, enabling the robot to recover from failures and continue task execution.

When a failure node is encountered, the system constructs a candidate extended subtree $\mathcal{T}_{ext}$, which is then inserted into $\mathcal{T}_i$ in place of $f_i$ (Algorithm 1, Line 10–11). The purpose of this process is to transform the outcome of the failed condition into a successful state by either exploiting the robot's intrinsic capabilities or by delegating the task to other robots.

In our framework, delegation is managed by the centralized virtual allocator *Alex*, which maintains a shared view of the robots and their environmental states. When a failure node $f_i$ is reported, *Alex* collects contextual information including the label, object, location, and target state of the failed condition, along with the reporting robot's partially observable state. Based on this information, *Alex* identifies whether the failure can be addressed locally by robot $\mathcal{R}_i$ or should be resolved by another robot $\mathcal{R}_j$ with the appropriate a posteriori effect. If the latter holds, *Alex* instructs $\mathcal{R}_j$ to construct an extended subtree $\mathcal{T}_{ext}^j$, which is integrated into $\mathcal{T}_j$ with high execution priority to ensure timely recovery.

Two collaborative update cases arise from this mechanism. (1) **Independent extension**: If the robot can resolve the failed condition using its own action set, the subtree $\mathcal{T}_{ext}$ is directly merged into $\mathcal{T}_i$ by replacing $f_i$ (Fig. 5a). This allows the robot to recover autonomously from execution failures. (2) **Delegated extension**: If the failure node cannot be resolved locally, *Alex* assigns the task to a peer robot $\mathcal{R}_j$. In this case, $\mathcal{R}_i$ temporarily suspends execution at $f_i$ while monitoring its status, whereas $\mathcal{R}_j$ incorporates the delegated subtree $\mathcal{T}_{ext}^j$ at the root of its behavior.

### E. Failure Node Detection

During task execution, each robot $\mathcal{R}_i$ continuously evaluates its behavior tree $\mathcal{T}_i$ through the standard $Tick(\cdot)$ operation. A *Failure* status indicates that the corresponding condition node cannot be satisfied under the current environmental state. The failed node is then recorded as a failure node $f_i$ and inserted into a dedicated failure queue $\mathcal{F}_{nodes}$ for further processing (Algorithm 1, Lines 15-25).

Unlike conventional BT implementations where failures simply propagate upward and potentially terminate execution, our framework explicitly monitors and stores these nodes. This mechanism provides two key advantages: (1) it allows systematic identification of execution bottlenecks that require intervention, and (2) it serves as the trigger for invoking the extension process to recover from the failure.

Once a failure node is detected, the information is forwarded to the centralized allocator *Alex*, which maintains a shared view of the environment and robot states. By leveraging this global knowledge, *Alex* determines whether the failure can be resolved by the same robot through a local action extension, or whether the task requires delegation to another robot with the appropriate capabilities. In the latter case, *Alex* coordinates the integration of an extended subtree into the corresponding robots' behavior trees, ensuring that the overall mission continues to progress.

Through this mechanism, failure detection does not represent the termination of execution but instead acts as an adaptive decision point that drives the dynamic reconstruction of behavior trees. By systematically recording and processing failure nodes, the system achieves resilience against unexpected disturbances and maintains long-term task execution in dynamic environments.

## V. Experiments

### A. Simulation Experiments

We first evaluate the proposed framework in simulation using the Behavior-1K dataset, which provides diverse task descriptions spanning navigation, object manipulation, and cooperative missions. To construct a representative yet tractable benchmark, we sample 20 tasks per group and report the detailed execution results in Table III. Each entry indicates the number of behavior-tree ticks (execution steps) required to complete the task, while $\times$ denotes failure. The sampled tasks span action horizons ranging from 2 to 20 steps. To assess scalability, we designed three scenarios with increasing heterogeneity: (i) a single quadruped robot, (ii) a quadruped robot paired with a drone, and (iii) a quadruped robot, a drone, and a robotic arm. For each scenario, the 20 tasks were executed over five independent trials with randomized initial positions of both robots and objects, and the averaged results are reported.

We further evaluated the proposed BT-based execution framework on a heterogeneous multirobot team comprising a robotic arm, a quadruped robot, and a drone. Each robot is modeled with a library of primitive actions, where every action is specified by explicit preconditions and postconditions to maintain logical consistency (see Table I). The overall number of available actions and conditions for each robot is summarized in Table II. At the beginning of each trial, all robots were initialized in nominal configurations. Actions were executed only when their preconditions were satisfied, and postconditions were continuously monitored to verify success or detect failures. This design ensures that the BT framework strictly enforces logical dependencies across heterogeneous robots while enabling both sequential and coordinated execution.

For a fair comparison, both MCTS and LLM-MCTS were configured with an identical rollout budget of 500 simulations per decision step and a maximum search depth of

TABLE I

BEHAVIOR TREE (BT) NODES FOR ROBOT ARM, QUADRUPED ROBOT, AND DRONE

| Precondition | Action | Postcondition |
|---|---|---|
| **Robot Arm** | | |
| ArmObjectFreeGrab?<br>ArmContainClose? | Open | ArmContainOpen? |
| ArmObjectInRange?<br>ArmContainOpen? | Grab | ArmObjectInGrab? |
| ArmObjectFreeGrab?<br>ArmContainOpen? | Close | ArmContainClose? |
| ArmObjectInGrab?<br>ArmContainOpen? | PutInto | ArmObjectInTarget?<br>ArmObjectFreeGrab? |
| ArmObjectInGrab? | PutOn | ArmObjectOnTarget?<br>ArmObjectFreeGrab? |
| **Quadruped Robot** | | |
| QuadFreePath? | MoveToNoObject | QuadInRangeNoObject? |
| QuadInRangeNoObject?<br>QuadObjectFreeGrab? | MoveToWithObject | QuadInRangeNoObject? |
| QuadObjectFreeGrab?<br>ContainClose? | Open | QuadContainOpen? |
| QuadCanGetObject?<br>QuadInRangeNoObject?<br>QuadObjectFreeGrab?<br>QuadContainOpen? | Grab | QuadObjectInGrab? |
| QuadObjectFreeGrab?<br>QuadContainOpen? | Close | QuadContainClose? |
| QuadInRangeWithObject?<br>QuadContainOpen?<br>QuadObjectInGrab? | PutInto | QuadObjectInTarget?<br>QuadObjectFreeGrab? |
| QuadInRangeWithObject?<br>QuadObjectInGrab? | PutOn | QuadObjectOnTarget?<br>QuadObjectFreeGrab? |
| **Drone** | | |
| DroneObjectInBasket?<br>DroneOnGround? | TakeOffWithObject | DroneInAirWithObject? |
| DroneObjectInBasket?<br>DroneInAirWithObject?<br>DroneInRangeWithObject? | LandOnWithObject | DroneAtTargetWithObject?<br>DroneOnGround? |
| DroneObjectInBasket?<br>DroneInAirWithObject?<br>DronePathFree? | MoveToWithObject | DroneInRangeWithObject? |
| DroneOnGround? | TakeOffNoObject | DroneInAirNoObject? |
| DroneInRangeNoObject? | LandOnNoObject | DroneAtTargetNoObject?<br>DroneOnGround? |
| DronePathFree?<br>DroneInAirNoObject?<br>DronePathFree? | MoveToNoObject | DroneInRangeNoObject? |

TABLE II

SUMMARY OF ACTION NODES AND CONDITIONS.

| Robot | Number of Actions | Number of Conditions |
|---|---|---|
| Robotic Arm | 5 | 6 |
| Quadruped | 7 | 8 |
| Drone | 6 | 7 |

20. These hyperparameters were selected to balance computational feasibility and planning performance, and were kept consistent across all scenarios. We compare the proposed framework (**LLM-HBT**) with two representative planning baselines. MCTS performs task planning using Monte Carlo Tree Search over the available primitive actions. At each decision step, the planner conducts stochastic rollouts to estimate the expected value of candidate actions and selects the action with the highest estimated return. LLM-MCTS extends MCTS by incorporating LLM-generated high-level action proposals that guide the search toward promising regions of the action space. To ensure a fair comparison, both baselines operate on the same action library and environment state representation as LLM-HBT.

For quantitative evaluation, we adopt two widely used metrics: **Success Rate (SR)** and **Average Steps (AS)**. Let $N$ denote the total number of tasks, and let $s_i \in \{0, 1\}$ indicate whether task $i$ is successfully completed. The success rate is defined as

$$SR = \frac{1}{N} \sum_{i=1}^{N} s_i. \tag{3}$$

To measure efficiency, let $k_i$ denote the number of behavior-tree ticks required to complete task $i$. The average steps is then defined as

$$AS = \frac{1}{N} \sum_{i=1}^{N} k_i. \tag{4}$$

Together, these two metrics characterize both the *effectiveness* of different methods (task completion capability) and their *efficiency* (execution cost in terms of steps).

Table IV summarizes the results across all scenarios. LLM-HBT achieves the highest success rate across all scenarios, while the baselines exhibit substantial degradation as heterogeneity and task complexity increase. In Scenario 1, all methods maintain relatively high success rates (MCTS: $95\%$, LLM-MCTS: $90\%$, LLM-HBT: $100\%$). However, in Scenario 2, both baselines drop to $55\%$, whereas LLM-HBT continues to achieve perfect success. In the most challenging Scenario 3, where three heterogeneous robots must coordinate, the baselines succeed in only $40\%$ of tasks, whereas LLM-HBT still achieves $100\%$.

The strong performance of LLM-HBT stems from its failure-driven execution mechanism: when a precondition is violated, the system explicitly surfaces the violated condition as a failure node and triggers behavior-tree expansion, enabling recovery via subtree insertion or subtask reallocation. By contrast, the baselines do not explicitly represent execution failures, making recovery difficult once an infeasible action sequence is committed. LLM-HBT also improves efficiency; for example, in Scenario 3 it achieves fewer average steps ($8.4$) than MCTS ($8.80$) and LLM-MCTS ($9.00$).

Baseline failures mainly occur in tasks with cross-robot temporal dependencies (e.g., handover or staged manipulation), where MCTS-based planners may commit to action sequences that violate execution preconditions at deployment time. In contrast, LLM-HBT exposes such violations as failure nodes and recovers via behavior-tree expansion and/or subtask reallocation.

Table III reports per-task results for the three methods, where each entry is the number of execution steps and $\times$ denotes failure in all five trials.

Overall, the proposed LLM-HBT consistently outperforms both baselines across all three groups. In Group 1, LLM-HBT completed all tasks with only minor deviations in execution steps, achieving lower or comparable step counts relative to MCTS and LLM-MCTS. Notably, tasks T12 and T16 that failed under MCTS or LLM-MCTS were successfully executed by LLM-HBT, demonstrating improved robustness.

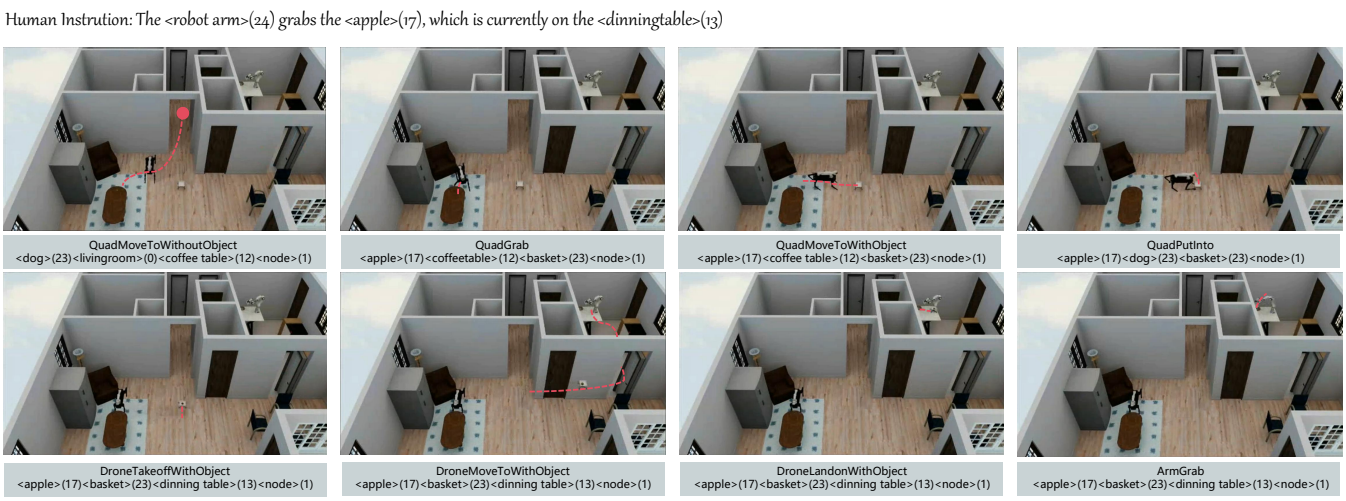


Fig. 6. Illustration of Scenario 3 with three robots collaboratively executing a task. The quadruped robot navigates to the table to grasp an apple, while the final goal is for the robotic arm to take over and grasp the apple.

TABLE III

SIMULATION RESULTS FOR 20 TASKS PER GROUP (HORIZONTAL: TASKS, VERTICAL: METHODS, ×: TASK FAILURE).

| **Group 1** | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MCTS | 2 | 2 | 2 | 4 | 4 | 4 | 6 | 4 | 4 | 4 | 4 | × | 4 | 6 | 3 | 3 | 6 | 3 | 7 | 3 |
| LLM-MCTS | 2 | 2 | 2 | 4 | 4 | 4 | 6 | 4 | 4 | 4 | 4 | 7 | 4 | × | 3 | × | 6 | 4 | 7 | 3 |
| Ours | 2 | 2 | 2 | 3 | 4 | 4 | 6 | 4 | 5 | 5 | 4 | 7 | 4 | 6 | 3 | 1 | 6 | 3 | 7 | 3 |
| **Group 2** | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 |
| MCTS | 2 | 3 | × | × | × | 2 | 4 | × | 7 | 9 | × | × | 4 | 4 | × | × | 5 | 6 | × | 8 |
| LLM-MCTS | 2 | 3 | × | × | × | 2 | 4 | 6 | 6 | 9 | × | × | 5 | 5 | × | × | 6 | × | × | 9 |
| Ours | 2 | 3 | 3 | 3 | 4 | 2 | 4 | 4 | 2 | 7 | 7 | 7 | 3 | 2 | 7 | 7 | 3 | 4 | 10 | 8 |
| **Group 3** | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 |
| MCTS | 6 | 3 | 5 | × | × | 2 | × | 6 | 7 | 7 | 5 | × | × | × | × | × | × | × | × | × |
| LLM-MCTS | 6 | 3 | 5 | × | × | 2 | × | 6 | 7 | 7 | 5 | × | × | × | × | × | × | × | × | × |
| Ours | 4 | 3 | 7 | 20 | 3 | 2 | 15 | 10 | 7 | 7 | 5 | 19 | 12 | 6 | 11 | 2 | 6 | 5 | 7 | 13 |

In Group 2, our method successfully completed several tasks that either baseline failed (e.g., T3, T4, and T20), while maintaining efficient execution in tasks completed by all methods. Group 3 further highlights the advantage of LLM-HBT in complex heterogeneous settings: while MCTS and LLM-MCTS failed the majority of tasks (marked as ×), our method achieved successful execution in nearly all tasks, often with lower or comparable step counts (e.g., T4, T7, T12, and T20).

These results confirm that embedding LLM reasoning within the BT framework enables more reliable and adaptive task execution. LLM-HBT not only improves robustness in the face of task failures but also facilitates coordinated, sequential, and parallel execution across heterogeneous robots, even in the most challenging scenarios where baselines frequently fail.

### B. Real-World Deployment

To further verify the stability and effectiveness of our approach in physical environments, we conduct real-world experiments in a café-like setting. The environment includes two heterogeneous robots: (i) a fixed robotic arm mounted on the bar counter and (ii) a wheeled–legged robot equipped with a tray for transporting objects. The task is initialized with a bottle held in a human hand. The objective is for the robotic arm and the wheeled–legged robot to cooperate in placing the bottle on the counter.

TABLE IV

OVERALL PERFORMANCE OF DIFFERENT METHODS ACROSS THREE SCENARIOS (SR: TASK SUCCESS RATE, AVG. STEPS: AVERAGE NUMBER OF ACTION STEPS).

| Method | Scenario1 | | Scenario2 | | Scenario3 | |
|---|---|---|---|---|---|---|
| | SR(%) | AS | SR(%) | AS | SR(%) | AS |
| MCTS | 95 | 3.95 | 55 | 4.91 | 35 | 8.80 |
| LLM-MCTS | 90 | 4.11 | 55 | 5.18 | 35 | 9.00 |
| LLM-HBT | **100** | **4.05** | **100** | **5.05** | **100** | **8.4** |

During execution, the robotic arm first establishes the precondition that the bottle is within its graspable workspace. Since this condition is not initially satisfied, the system extends the behavior tree by allocating the task to the wheeled–legged robot. The robot navigates toward the hu-


Fig. 7. Real-world deployment in a café environment. A wheeled-legged robot equipped with a basket transports a bottle handed over by a human, while a Robotic Arm mounted on the bar counter takes over and places the bottle onto the designated spot.

man, receives the bottle, and delivers it to the counter. The robotic arm then extends its own behavior tree to complete the final grasp-and-place operation. This process demonstrates the ability of the system to dynamically reconfigure behavior trees and allocate subtasks in real time, even under partial observability and physical uncertainties such as motion noise and human variability.

Across ten trials, the proposed method completed the task without failure. Intermediate precondition violations (e.g., the bottle initially outside the arm workspace) were handled by online BT expansion and subtask reallocation to the wheeled–legged robot, demonstrating robust recovery under physical uncertainties. Future work will study more robust recovery under severe sensing noise and communication delays.

## VI. Conclusion

We presented **LLM-HBT**, a novel framework that fuses large language model reasoning with behavior-tree execution to enable long-term, adaptive coordination in heterogeneous multirobot teams. The system employs a closed-loop pipeline—task initialization, failure detection, LLM-guided task assignment (via a virtual allocator *Alex*), and online BT extension—which allows robots to locally recover or reassign subtasks at runtime. Extensive simulation and a real-world café deployment demonstrate that LLM-HBT substantially improves task success rates and execution efficiency compared to strong baselines, while maintaining modularity and interpretability. Limitations include LLM inference latency and the current scope of real-world validation; future work will focus on latency-aware designs, communication-efficient decentralization, and robustness under perceptual uncertainty. Overall, LLM-HBT offers a practical and scalable approach for adaptive multirobot coordination at the intersection of language reasoning and embodied control.

## References

[1] Y. Cai, X. He, H. Guo, W.-Y. Yau, and C. Lv, "Transformer-based multi-agent reinforcement learning for generalization of heterogeneous multi-robot cooperation," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 13 695–13 702.

[2] W. Dai, U. Rai, J. Chiun, C. Yuhong, and G. Sartoretti, "Heterogeneous multi-robot task allocation and scheduling via reinforcement learning," *IEEE Robotics and Automation Letters*, 2025.

[3] H. Lee and D. Panagou, "Maintaining strong $r$-robustness in reconfigurable multi-robot networks using control barrier functions," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 1–7.

[4] L. Zhou and V. Kumar, "Robust multi-robot active target tracking against sensing and communication attacks," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1768–1780, 2023.

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

[6] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on robotics*, vol. 33, no. 2, pp. 372–389, 2016.

[7] S. Chen, T. X. Lin, S. Al-Abri, R. C. Arkin, and F. Zhang, "Hybrid susd-based task allocation for heterogeneous multi-robot teams," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1400–1406.

[8] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.

[9] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule, "Behavior trees and state machines in robotics applications," *IEEE Transactions on Software Engineering*, vol. 49, no. 9, pp. 4243–4267, 2023.

[10] H. Zhou, Y. Lin, L. Yan, J. Zhu, and H. Min, "Llm-bt: Performing robotic adaptive tasks based on large language models and behavior trees," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 16 655–16 661.

[11] R. A. Izzo, G. Bardaro, and M. Matteucci, "Btgenbot: Behavior tree generation for robotic tasks with lightweight llms," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 9684–9690.

[12] G. Heppner, D. Oberacker, A. Roennau, and R. Dillmann, "Behavior tree capabilities for dynamic multi-robot task allocation with heterogeneous robot teams," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 4826–4833.

[13] T. G. Tadewos, L. Shamgah, and A. Karimoddini, "Automatic decentralized behavior tree synthesis and execution for coordination of intelligent vehicles," *Knowledge-Based Systems*, vol. 260, p. 110181, 2023.

[14] R. Gong, X. Gao, Q. Gao, S. Shakiah, G. Thattai, and G. S. Sukhatme, "Lemma: Learning language-conditioned multi-robot manipulation," *IEEE Robotics and Automation Letters*, vol. 8, no. 10, pp. 6835–6842, 2023.

[15] Z. Mandi, S. Jain, and S. Song, "Roco: Dialectic multi-robot collaboration with large language models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 286–299.

[16] K. Liu, Z. Tang, D. Wang, Z. Wang, B. Zhao, and X. Li, "Coherent: Collaboration of heterogeneous multi-robot system with large language models," *arXiv preprint arXiv:2409.15146*, 2024.

[17] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multiagent debate," in *Forty-first International Conference on Machine Learning*, 2023.

[18] Y. Long, X. Li, W. Cai, and H. Dong, "Discuss before moving: Visual language navigation via multi-expert discussions," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 17 380–17 387.

[19] J. Gu, F. Xiang, X. Li, Z. Ling, X. Liu, T. Mu, Y. Tang, S. Tao, X. Wei, Y. Yao *et al.*, "Maniskill2: A unified benchmark for generalizable manipulation skills," *arXiv preprint arXiv:2302.04659*, 2023.

[20] Z. Shi, M. Fang, and L. Chen, "Monte carlo planning with large language model for text-based game agents," *arXiv preprint arXiv:2504.16855*, 2025.