

Interpretable Robot Control via Structured Behavior Trees and Large Language Models

Ingrid Maéva Chekam¹, Ines Pastor-Martinez¹, Ali Tourani^{1,2}, Jose Andres Millan-Romera¹, Laura Ribeiro¹, Pedro Miguel Bastos Soares¹, Holger Voos^{1,3}, and Jose Luis Sanchez-Lopez¹ ✉

October 9, 2025

Abstract

As intelligent robots become more integrated into human environments, there is a growing need for intuitive and reliable Human-Robot Interaction (HRI) interfaces that are adaptable and more natural to interact with. Traditional robot control methods often require users to adapt to interfaces or memorize predefined commands, limiting usability in dynamic, unstructured environments. This paper presents a novel framework that bridges natural language understanding and robotic execution by combining Large Language Models (LLMs) with Behavior Trees. This integration enables robots to interpret natural language instructions given by users and translate them into executable actions by activating domain-specific plugins. The system supports scalable and modular integration, with a primary focus on perception-based functionalities, such as person tracking and hand gesture recognition. To evaluate the system, a series of real-world experiments was conducted across diverse environments. Experimental results demonstrate that the proposed approach is practical in real-world scenarios, with an average cognition-to-execution accuracy of approximately 94%, making a significant contribution to HRI systems and robots.

The complete source code of the framework is publicly available at https://github.com/snt-arg/robot_suite.

1 Introduction

With the increasing presence of intelligent robots in everyday life, the demand for reliable and straightforward Human-Robot Interaction (HRI) interfaces is rapidly rising. Traditional robot control paradigms require users to learn particular commands [1] or interact with the robots through rigid user interfaces, especially in unstructured environments [2]. However, recent works target more flexible and adaptive communication strategies, unlocking the full potential of autonomous agents in human-centered environments.

Accordingly, advances in generative AI and Large Language Models (LLMs) reveal new opportunities for enabling seamless communication between humans and robots, where natural language is the primary means of communication [3]. Such models are powerful enough to comprehend given instructions and even “*reason*” about the demanded tasks, intentions, and environmental context [4]. When paired with robotic perception and control systems, LLMs enable users to intuitively instruct the robot to perform complex tasks such as following multiple objects [5], navigating through dynamic scenes [6], or interacting with specific items [7], all using natural dialogue. Furthermore, integrating multimodal capabilities, including vision and speech, enhances HRI by enabling more natural, context-aware communication and improving adaptability across tasks and environments [8].

^{*1} Automation and Robotics Research Group (ARG), Interdisciplinary Centre for Security, Reliability, and Trust (SnT), University of Luxembourg, Luxembourg. (e-mail: {ingrid.chekam.001, ines.pastor.001}@student.uni.lu, pedro.bastos@ext.uni.lu, {ali.tourani, jose.millan, laura.ribeiro, joseluis.sanchezlopez, holger.voos}@uni.lu)

^{†2} Institute for Advanced Studies (IAS), University of Luxembourg, Luxembourg.

^{‡3} Faculty of Science, Technology, and Medicine, University of Luxembourg, Luxembourg.

[§] This research was funded, in part, by the Luxembourg National Research Fund (FNR), DEUS Project (Ref. C22/IS/17387634/DEUS), RoboSAUR Project (Ref. 17097684/RoboSAUR), and MR-Cobot Project (Ref. 18883697/MR-Cobot). It was also partially funded by the Institute of Advanced Studies (IAS) of the University of Luxembourg through an “Audacity” grant (project TRANSCEND - 2021).

[¶] For the purpose of open access, and in fulfillment of the obligations arising from the grant agreement, the author has applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.



Figure 1: High-level overview of the proposed LLM-driven robotic control method, where a user interacts with the system through natural language, interpreted by an LLM to guide robot behavior via a modular control structure.

Yet, transforming these high-level instructions into interpretable and reactive robot behaviors requires an underlying control framework. Various solutions, such as Finite-State Machines (FSMs) [9], hierarchical planners [10], and learning-based controllers [11], can aid in structuring robot behavior and decision-making. However, they often face restrictions in scalability and interpretability when deployed in active, real-world scenarios. In contrast, Behavior Trees (BTs) offer a reactive solution for seamlessly incorporating domain-specific modules while preserving a transparent and extensible execution flow [12]. While BTs are conceptually similar to FSMs, they offer superior flexibility, scalability, and clarity, especially for handling complex tasks involving sequencing, fallbacks, or concurrency. However, their direct manipulation often requires technical expertise, which limits accessibility and hinders intuitive HRI for non-expert users.

Bringing these elements together, this paper proposes a novel end-to-end framework that unifies the interpretative power of LLMs alongside the structured execution of BTs to enable robust robot behavior in response to natural language instructions. Our goal, as illustrated in Fig. 1, is to present an end-to-end framework that supports a dynamic “command-to-execution” flow, real-time decision-making, and seamless integration of new behaviors. In contrast to existing state-of-the-art methods, our approach emphasizes modularity, interpretability, and scalability, by autonomously translating natural language commands into executable actions. As depicted in Fig. 2, the LLM bridges human objective and actionable robot behaviors by triggering domain-specific behavior modules through the behavior tree. The system is developed to support multiple behavior modules with effortless integration, primarily within the computer vision domain, including object tracking [13] and hand gesture recognition [14].

With this, the paper offers the following contributions:

- A modular and model-agnostic robotic framework that integrates LLMs with customized BTs to interpret natural language commands and autonomously execute defined actions,
- An open-source implementation of the complete framework, promoting reproducibility and further research; and
- Novel systematic evaluation methodologies for measuring success, latency, and robustness of the framework.

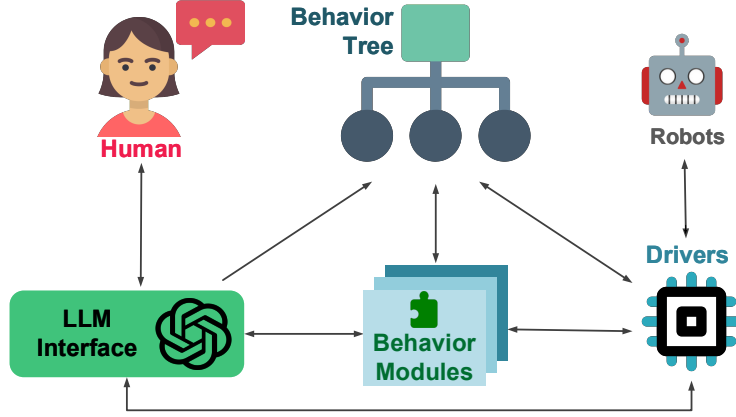


Figure 2: The outline of the proposed system architecture. An LLM interprets natural language instructions from the human, interfacing with a behavior tree to coordinate modular plugins that control the robot’s actions.

The remainder of the paper is structured as follows: Section 2 reviews similar frameworks that integrate LLMs and BTs. Section 3 details the proposed system and its interior modules. Experimental results in real-world scenarios are presented in Section 4. Finally, the paper concludes and discusses future directions in Section 5.

2 Related Works

2.1 LLM-driven HRI Frameworks

Using LLMs for interacting with robots has absorbed significant attention in recent years, enabling more flexible communication between users and robots. ROSGPT [15] integrates ChatGPT with robotic systems using prompt engineering and ontologies guidance, though it is limited to single-task execution without autonomous decision-making. ROS-LLM¹ [16] generates action plans through “chain-of-thought” reasoning and few-shot prompting. While these plans adopt a tree-like structure for decomposing tasks into sub-actions, they lack core Behavior Tree features, such as hierarchical structuring, fallback mechanisms, and conditional branching. HELPER [17] is an embodied agent that uses retrieval-augmented prompting to parse free-form dialogues into action programs, yet remains constrained to simulated settings and abstract tasks despite its memory-based learning. In contrast, ROSA [18] offers a scalable solution by integrating a “reasoning–action–observation” loop in a flexible agent, where a GPT model autonomously selects tools for both low-level control and high-level reasoning.

Incorporating multimodality, ROSGPT_Vision² [19] extends ROSGPT by integrating visual language models and perception modules to ground language in a visual context. However, its end-to-end prompt-based design, relying on a single vision module and fixed prompt logic, limits its adaptability to dynamic environments and tasks that require context-sensitive perception. LaMI [8] enhances multimodal understanding by integrating speech (audio), vision, text, and gestures to infer high-level human intent. However, its main limitation lies in its substantial computational overhead, stemming from its intricate multimodal fusion mechanisms.

2.2 LLMs for Behavior Tree Control

Other works integrate BTs with LLMs to facilitate its use, enabling intuitive behavior control via natural language [20]. Specifically, LLM-BRAIn [21] and BTGenBot [22] utilized LLMs to “generate” Behavior Trees from natural language instructions. LLM-BRAIn primarily focuses on model optimization for deployment on lightweight mobile robotic platforms, aiming to minimize computational overhead. On the other hand, BTGenBot focuses on fine-tuning language models to generate behavior trees with enhanced structural quality. While both approaches contribute to the automated

¹<https://github.com/Auromix/ROS-LLM>

²https://github.com/bilel-bj/ROSGPT_Vision

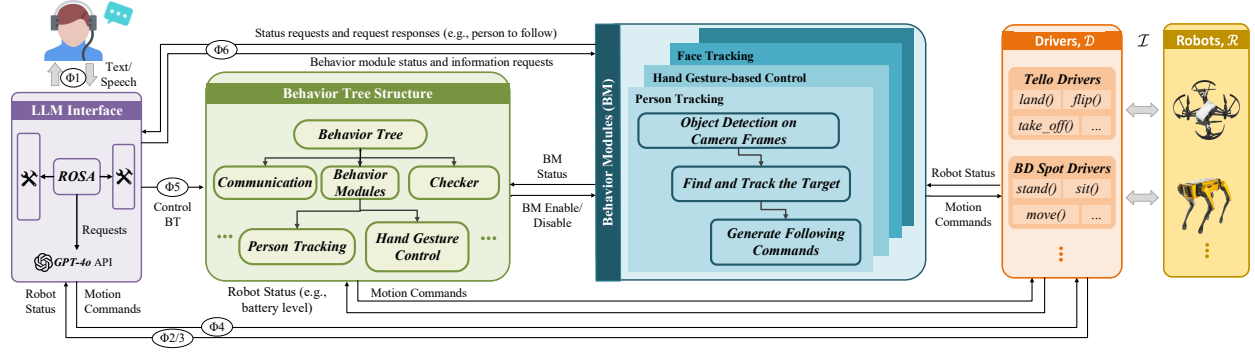


Figure 3: A detailed overview of the proposed system architecture, depicting the integration of LLM-based language understanding, behavior tree core, plugin, and driver modules. Arrow labels indicate the interaction category ($\Phi 1$ to $\Phi 6$) corresponding to evaluation scenarios described in §4.1.

generation of BTs, they lack support for interactive modification or real-time interaction during execution, which is critical for adaptive HRI.

Apart from generating BTs using LLMs, some approaches focus on modifying predefined BT structures that contain robot behaviors. In [23], GPT language models are employed to indirectly manipulate XML-driven BTs, including querying, generating, and modifying its functions. The primary weakness of this approach is the introduction of an extra layer of complexity and potential integration latency. BETR-XP-LLM [24] and LLM-BT [25] enable real-time BT adaptation and modification, minimizing human intervention. In this regard, BETR-XP-LLM combines a long-horizon planner with an LLM to handle unexpected failures, providing reasoning feedback to operators. LLM-BT utilizes ChatGPT for reasoning and a BERT-based model to extract keywords; however, its keyword parsing relies on rigid if/else rules, which limit task generality. Similarly, DiaGBT [26] uses an LLM to define mapping rules on sub-BTs for constructing complex BTs and incorporates *memory* to learn reusable skills. Though it requires imperative commands from the user and relies on Q&As templates for disambiguation, which can hinder the HRI.

It can be seen that the existing works lack a direct, dynamic command-to-execution flow that enables real-time modification of Behavior Trees using an LLM. The proposed framework aims to address these gaps by introducing a generalizable architecture for seamless integration of LLM-driven interpretation with dynamic behavior tree execution.

3 Proposed Framework

The proposed system is a modular and open-source framework publicly accessible via https://github.com/snt-arg/robot_suite, featuring an LLM interface built on top of ROSA’s [18] architecture. It is designed to be robot-agnostic and adaptable to a wide range of robotic tasks and hardware configurations. As shown in Fig. 3, the system operates as an end-to-end application with components that can evolve independently, enabling plug-and-play integration of new plugins (*i.e.*, behaviors) without complexity or the need for overall system modifications.

According to the figure, when the user issues natural language commands, they serve as the primary inputs for task interpretation and planning. These commands reflect the user’s intention to initiate specific robot actions, which in turn demand triggering the appropriate control *drivers* (§3.1) tailored for the robot. The unstructured inputs are then processed by the *LLM interface* module (§3.2), which employs GPT-4 backbone to translate them into structured action commands. At the core of the system, a *behavior tree* (§3.3) handles high-level decision-making procedures based on the input interpreted by the LLM, also offering task execution alongside transparent reasoning. To support various capabilities, a set of *behavior modules* (§3.4) (implemented as callable plugins) is integrated into the system. These behavior modules (*e.g.*, person-following) are independently developed and can be triggered by the BT, operating in conjunction with hardware-specific drivers. In this regard, the *LLM interface* interacts with both the *behavior tree* (by activating relevant nodes of the tree) and the *behavior modules*, either by querying their status or through bidirectional communication (as in the person-following task, where it requires specifying the target and receiving on-demand updates). It should be noted that although we present a sample set of executable plugins and the behavior tree structure, the system supports a diverse range of behaviors, making it extensible to employ additional robot capabilities.

Algorithm 1 LLM Interface Module.

```
1: procedure INIT
2:   Initialize ROS2 node, publishers, and subscribers.
3:   Initialize and configure ROSA.
4: end procedure

5: procedure COMMANDEXECUTOR
6:    $query \leftarrow$  user input.
7:   Submit  $query$  to LLM for interpretation and tool selection.
8:   if LLM maps  $query$  to a valid tool then
9:     Run corresponding tool ▷ like takeoff(), land(), move(), etc..
10:    Publish messages to ROS2 topics ▷ robot command, BT update, or plugin response
11:   else
12:     Inform the user that the command could not be understood/executed.
13:   end if
14: end procedure

15: procedure MAIN
16:   Run INIT()
17:   if  $cmd \leftarrow$  natural language input. then
18:     Run COMMANDEXECUTOR()
19:   end if
20: end procedure
```

3.1 Robots and Drivers

Interfacing between the high-level decision logic and the physical robots, the *Drivers* module plays a crucial role in executing commands generated by the system (either the LLM interface or the plugins). These drivers implement atomic, hardware-specific operations (such as `take_off`, `land`, or `stand`), tailored to the capabilities of different robot platforms. The driver layer abstracts and encapsulates these low-level controls, enabling straightforward integration across robots. The current version of the framework supports both **legged robots**, such as the Boston Dynamics Spot[®], and **aerial platforms**, including drones like the DJI Tello. However, it is designed to be easily extensible to additional robot platforms with minimal effort required for integration. As shown in Fig. 3, robot-specific drivers provide the interface layer between the physical robot hardware and the rest of the framework, handling bidirectional communications by executing low-level control commands (e.g., motion and velocity) and transmitting robot status feedback (e.g., battery or connectivity).

Consequently, for the supported robots $\mathbf{R} = \{r_i \mid i \in \mathbb{N}\}$, there exists a mapping $\mathcal{D} : \mathbf{R} \rightarrow \mathcal{D}_R$ such that each robot $r_i \in \mathbf{R}$ is associated with a specific driver $d_{r_i} = \mathcal{D}(r_i)$. Accordingly, task execution on robot r_i is delegated to the corresponding driver d_{r_i} via an interface function $\mathcal{I} : \mathcal{C} \times \mathbf{R}$, where $\mathcal{C} = \{c_j \mid j \in \mathbb{N}\}$ is a set of low-level robot commands (e.g., `land`). Thus, the appropriate low-level action on r_i is triggered using:

$$\mathcal{I}(r_i, c_j) = d_{r_i}(c_j) \quad (1)$$

3.2 LLM Interface

The framework builds upon and extends ROSA [18] to better support our proposed BT-based control structure. In this regard, Algorithm 1 outlines the core logic of the LLM Interface module within the framework. Upon receiving a natural language input from the user, the module submits it as a prompt to the LLM, interprets the generated result, and dispatches the appropriate behavior by invoking the corresponding robot control tool. It should be noted that the LLM interface operates as **one-shot prompting**, where the initial prompt includes contextual instructions, accessible tool descriptions, and behavioral constraints, enabling the LLM to generate outputs without iterative dialogue.

The framework introduces four key extensions to the original ROSA’s architecture: (i) autonomous behavior selection, (ii) multimodal HRI support, (iii) failure reasoning, and (iv) structured control integration through BT. A detailed description of each of these new features is presented below:

(i) Autonomous behavior selection: Unlike frameworks like ROSGPT [15] and ROSA [18], which rely solely on the LLM back-end to interpret given user commands and directly invoke the required tools, our framework introduces an explicit *decision-making layer* for behavior selection. This layer provides an internal mechanism to autonomously determine which behavior modules (implemented as framework plugins) to trigger based on both the interpreted user intent and additional contextual signals. Suppose $\mathcal{B} = \{b_1, b_2, \dots, b_n\}$ contains the set of available behavior modules and $\mathcal{Q} = \{q_1, q_2, \dots, q_m\}$ represents the set of semantically parsed queries derived from the LLM interface. The

autonomous selection mechanism chooses the most appropriate behavior candidate for $q_i \in Q$ using:

$$\Gamma : Q \rightarrow \mathcal{B} \quad (2)$$

which maps each interpreted query q_i to a suitable behavior $b_i \in \mathcal{B}$ through $b_{q_i} = \Gamma(q_i)$. This query-to-behavior mapping is not purely based on simple keyword matching or static prompts. Instead, $\Gamma(\dots)$ considers the task context, robot capabilities, and relevant sensor feedback. For instance, upon receiving the command “move forward for five seconds,” the behavior selection function evaluates runtime conditions, such as battery level or robot operational status, to ensure feasibility before dispatching the appropriate behavior module. This layered abstraction provides decoupling of *semantic understanding* from *low-level behavior invocation*, offering flexibility and robustness in decision-making. Finally, once the behavior b_{q_i} is selected, it delegates low-level command execution to the corresponding robot driver module d_{r_i} to invoke the specific robot action c_j through the described interface function $\mathcal{I}(r_i, c_j)$ in Equation 1.

(ii) Multimodal HRI support: While ROSA supports only text-based interaction with the user, the proposed framework extends the HRI channel by integrating a multimodal input interface $\mathcal{U} = \{\text{text, voice}\}$, with each $u_i \in \mathcal{U}$ serves as a source of natural language commands to the LLM interface. It should be noted that the current work limits the interface modality to text (using the keyboard input) to ensure controlled execution and reproducibility. Nonetheless, the framework’s architecture is designed to be readily extensible to other modalities, which requires only adding an appropriate perception module to \mathcal{B} and extending the mapping Γ to include the corresponding behaviors. The multimodal HRI interface functions as an additional pre-processing layer to the LLM interface (§3.2), converting inputs such as voice into standard text for the LLM, which is then forwarded for semantic parsing and behavior selection. Accordingly, it ensures that all input modalities follow a unified *interpretation–decision–execution* pipeline, preserving consistency across heterogeneous HRI channels.

(iii) Failure reasoning: The proposed framework extends its baseline capabilities by integrating a failure reasoning and explanation mechanism inspired by [20]. With this, the system has a more advanced task failure management strategy, which invokes an explanation function when a given query q_i fails to be executed. When an interpreted query $q_i \in Q$ maps to a behavior module b_{q_i} but fails to achieve a successful execution, the failure analysis stage is triggered to invoke the explanation function, as defined below:

$$\Psi : \mathcal{F} \rightarrow \mathcal{E} \quad (3)$$

where \mathcal{F} is a set of identifiable failure modes and \mathcal{E} represents the set of natural language explanations generated for the user. The failure reasoning mechanism uses the LLM back-end to transform low-level failure reports of the driver d_{r_j} or the behavior module b_{q_i} into semantically-aligned and context-aware reasoning. Hence, such explanations and structured feedback contribute to the framework’s transparency and simplify the HRI procedure.

(iv) Structured control integration with BT: A significant limitation of ROSA is the absence of an explicit structured control logic, which can result in inconsistent execution when handling multi-step or conditional commands. The proposed framework addresses this by embedding a BT-based architecture at the core of the pipeline, enabling hierarchical and reactive control flow. Accordingly, each user task $t_i \in \mathcal{T}$ derived from an interpreted query q_i and its mapped behavior b_{q_i} is encapsulated as a behavior tree node n_i as below:

$$n_i \in \mathcal{N} \rightarrow \text{status} \in \{\text{success, failure, running}\} \quad (4)$$

where \mathcal{N} is the set of all active tree nodes. Accordingly, the behavior tree imposes an execution policy to govern the order and conditional flow between nodes, while ensuring the coherent coordination of behaviors. This integration allows the BT to dynamically invoke nodes based on LLM interpretations, runtime conditions from the driver interface $\mathcal{I}(r_i, c_j)$, and sensor feedback, maintaining a continuous alignment between high-level user intent and low-level robotic execution. The detailed behavior tree composition and node taxonomy are presented in Section 3.3.

3.3 Behavior Tree Structure

The behavior tree represents an executable policy constrained by both the robot’s capabilities and the environmental context. The strength of the behavior tree within the proposed framework lies in its *configurable hierarchical design* and *tick-based execution mechanism*, which enable fine-grained control, flexible task composition, and real-time

responsiveness to dynamic system states. In this regard, the BT operates through discrete execution cycles (or “ticks”), during which the tree is traversed from the root node, evaluating and executing child nodes sequentially from left to right. Each node returns a status (as shown in Algorithm 4), which determines the subsequent flow of execution within the tree. This tick-based mechanism enables the behavior tree to continuously update node statuses and adapt the overall behavior in real-time to evolving tasks and conditions.

Let \mathcal{S} denote the overall task space and Π represent the employed behavior tree in the proposed framework. Each node $n_i \in \mathcal{N}$ in the behavior tree is associated with a specific behavior $b_i \in \mathcal{B}$ implemented as a behavior module, encapsulating its execution logic and up-to-date status. In this regard, the behavior mapping function is defined as:

$$\Lambda : \mathcal{S} \times \mathcal{N} \rightarrow \mathcal{B} \quad (5)$$

where a given task $s_j \in \mathcal{S}$, in combination with the active node n_i , determines the selected behavior b_i . Thanks to the autonomous behavior selection function $\Gamma(\dots)$ in the framework, the mapped behavior b_{q_k} inside Π for an interpreted query q_k is obtained as:

$$b_{q_k} = \Gamma(q_k) = \Lambda(s_j, n_i) \quad (6)$$

3.4 Behavior Modules

When integrated with the behavior tree Π , each behavior module b_{q_i} of the framework operates as a separate ROS2 node that remains inactive until being explicitly triggered by the LLM-interpreted query q_i . Complex behaviors are implemented by coordinating multiple low-level control actions across different subsystems and by feeding their outcomes back into the BT as updated node statuses. Thanks to the $\Gamma(\cdot)$ and Λ functions, the system ensures that behaviors are mapped and executed only when the current task logic determines them to be relevant. The interaction and tick-based triggering between Π and the behavior modules happens through ROS2 services. Once enabled, the tree “ticks” its corresponding behavior module, which typically loads some inputs and produces outputs as part of its execution cycle.

Although behavior modules can continuously receive data through their dedicated communication channels, they can only broadcast system-driven data (such as signals and commands) when triggered by the BT.

Triggering a behavior function can be viewed as an event-driven execution loop with a frequency governed externally by the behavior tree. This design enforces strict control flow, ensuring that execution aligns with the current task logic. In the *interpretation–decision–execution* pipeline of the framework, this operation corresponds to the three stages $\rho \in \{\text{cog}, \text{disp}, \text{exec}\}$, representing LLM-based cognition of the user command, its dispatching to the appropriate behavior module, and the subsequent execution of the selected behavior, respectively.

4 Experimental Results

A series of real-world experiments in diverse scenarios was conducted to evaluate the proposed system’s capabilities and potential. In this regard, we designed realistic scenarios, collected our in-house dataset, and evaluated the system’s performance in various tasks. It is worth noting that the proposed system incorporates a diverse range of behaviors; however, for the experiments, we employed a limited but representative behavior tree structure along with a curated set of sample behavior modules.
















4.1 Evaluation Criteria

Robot Platform. The introduced system is designed to be platform-agnostic, independent of a specific robot type. This paper demonstrates the system’s functionality on distinct robotic platforms: a lightweight aerial drone (DJI Tello) utilizing its onboard camera, and a legged robot (Boston Dynamics Spot[®]) equipped with an external Intel RealSense D435 camera for visual perception.

Software Stack. The system is implemented in Python using ROS2 Humble for inter-module communication. Based on ROS2, it employs the OpenAI API with the gpt-4o model as the LLM backend, enabling advanced natural language understanding and reasoning capabilities for interpreting user commands and interacting with behavior modules. It is worth noting that during the experiments, we relied exclusively on textual input for querying the LLM interface to ensure consistency and facilitate reproducible evaluation.

Table 1: Evaluation scenarios designed to assess the system’s performance across a range of HRI tasks. The expected LLM responses in the rightmost column should remain semantically aligned with the given examples, despite possible variations in wording.

A battery level below 20% (🔋) is considered low, while levels above are treated as sufficient (🔋).

Category	#	Platform	User Instruction	Environment Status	Expected System Behavior & Response
<i>Unsupported (Φ1)</i>	Φ1.1		“Jump”	landed/stand/flying	Not triggered, response: “I cannot perform this action.”
	Φ2.1		“Do a Flip”	flying, 🔋	Flip maneuver executed.
	Φ2.2		“Do a Flip”	landed, 🔋	Blocked due to status, response: “I cannot do it as the drone is on the ground.”
<i>Context-Aware Response (Φ2)</i>	Φ2.3		“Do a Flip”	landed, 🔋	Blocked due to battery level and status, response: “I cannot do it due to low battery and robot status.”
	Φ3.1		“What is the battery level?”	landed/flying, 🔋	Response: “The battery level is X%”.
	Φ3.2		“Can I do a flip with the drone?”	flying, 🔋	Response: “Yes, since the drone is flying and the battery level is 26%”.
<i>System Inquiry Response (Φ3)</i>	Φ3.3		“Which actions can I perform with the drone?”	landed/flying, 🔋	Response: “You can [a list of supported actions]”.
	Φ3.4		“What is the status of the drone?”	landed/flying, 🔋	Response: “The drone is on the ground with a battery of 26%”.
	Φ3.5		“What are the common causes that I get unknown status?”	landed, disconnected	Response: “The common causes for the robot state being unknown could be [a list of common causes]”.
<i>Motion Command (Φ4)</i>	Φ4.1		“Turn left/right for X sec.”	stand/flying, 🔋	Motion action triggered with duration control.
	Φ4.2		“Move forward/backward for X sec (with velocity Y).”	stand/flying, 🔋	Motion action triggered with duration control.
<i>Plugin Switching (Φ5)</i>	Φ5.1		“Change the control to hand gesture.”	stand/flying, 🔋	Control mode switched, response: “You can now control the robot using hand gestures.”
	Φ5.2		“Change the control to keyboard.”	stand/flying, 🔋	Control mode switched, response: “You can now control the robot using the keyboard.”
<i>Vision-based Interaction (Φ6)</i>	Φ6.1		“Track the person with a phone”	flying, 🔋	Person identified and approached, response: “Now tracking the person with a phone.”
	Φ6.2		“Track the person with a phone”	flying, 🔋	Blocked due to identification failure; the drone waits for 5 sec. to find the person and then halts, response: “No person with a phone detected”

Dataset and Scenarios. We designed diverse scenarios to evaluate the system as an end-to-end application, where natural language commands from the user are interpreted and translated into actionable behaviors executed by the robot. To assess the system’s performance in these scenarios, we collected a multi-modal dataset comprising:

- Natural language instructions paired with ground-truth behavior annotations (expected responses),
- RGB image streams from the robot’s onboard cameras for vision-based perception tasks; and
- Execution logs including “timestamps” and “active nodes” of the behavior tree for control flow analysis.

The mentioned scenarios are summarized in Table 1, providing an overview of the task types for evaluation, with their inter-module interactions tagged in Fig. 3. Accordingly, all scenarios are designed to evaluate the performance and robustness of the proposed system across diverse real-world HRI tasks, covering:

- Φ1 – *Unsupported Actions*: testing the system’s ability to reject infeasible commands;
- Φ2 – *Context-Aware Response*: considering the impact of environmental or system state in decision-making;
- Φ3 – *System Inquiry Response*: assessing the system’s capacity to answer questions regarding its status, active tasks, or capabilities;
- Φ4 – *Motion Commands*: focusing on low-level action chaining with conditional execution;
- Φ5 – *Plugin Switching*: evaluating dynamic control modality changes of the system; and
- Φ6 – *Vision-based Interactions*: investigating the system’s perception-driven behavior modules based on visual attributes.

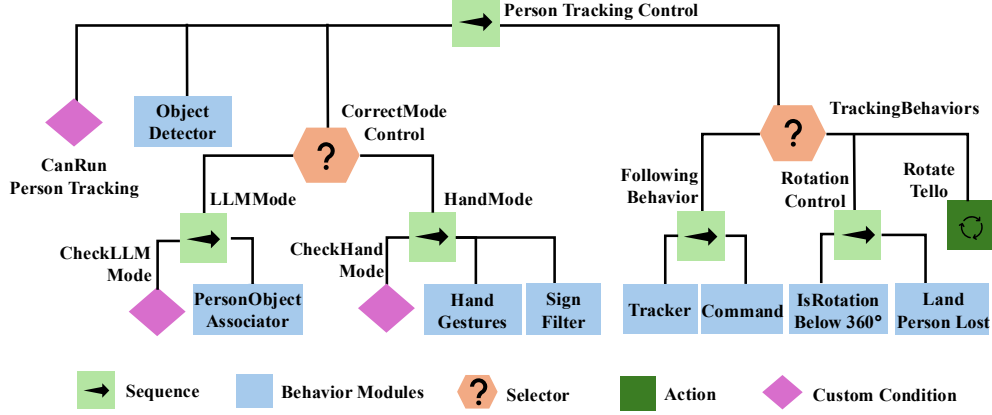


Figure 4: Structure of the sample behavior tree employed in the paper for system evaluation, containing the hierarchical arrangement of execution nodes to manage robotic behaviors.

Evaluation Metrics. Since each evaluation scenario embodies distinct objectives and interaction modalities, we adopt both “qualitative” and “quantitative” metrics tailored to the specific characteristics of each task. These metrics are designed to assess the system’s performance in terms of response clarity and correctness, perceptual accuracy, and successful execution of intended behaviors. The evaluation criteria employed in this study are summarized as follows:

- *Success Rate:* the degree to which the system achieves the intended task outcome across scenarios $\Phi 1$ to $\Phi 6$. Given the variability in LLM responses, we adopt a weighted success strategy, where each scenario case is evaluated over $k = 10$ independent runs and across the introduced stages ρ . Each demanded behavior is scored using a binary metric $\eta_{i,j}^\rho \in \{0, 1\}$, where i indexes the scenario case and j the iteration, with score 1 denoting success and 0 denoting failure. Thus, success rate is calculated as:

$$\sigma_i = \frac{1}{k} \sum_{j=1}^k \left(\eta_{i,j}^{\text{cog}} + \eta_{i,j}^{\text{disp}} + \eta_{i,j}^{\text{exec}} \right)$$

- *Response Latency:* the time delay between the start and end timestamps of stage ρ , i.e., t_s^ρ and t_e^ρ . For each scenario case, latency is measured as $\mathcal{L}_\rho = t_e^\rho - t_s^\rho$, and the total latency will be $\mathcal{L}_{\text{total}} = \sum_{i=1}^\rho \mathcal{L}_i$.

It should be noted that explicitly benchmarking the performance of underlying LLM and computer vision models is beyond the current scope of this paper. The primary objective of the proposed framework is to remain **model-agnostic** and to support any standard LLM or vision model that provides an accessible API for request–response communication. In this design, the LLM and perception components act as interchangeable *back-end modules* responsible for language understanding and environmental perception, respectively. This modular architecture enables the integration of other models without requiring modifications to the overall system structure. While the selection of a particular LLM (such as OpenAI GPT in our implementation) may influence factors such as reasoning quality or response latency, our focus in this work is on demonstrating the end-to-end functionality, scalability, and generalizability of the framework. A systematic comparison of alternative LLM back-ends and their influence on overall system performance will be explored in future work.

4.2 Benchmarking Setup

Custom behavior tree structure. The behavior tree shown in Fig. 4 serves as a representative implementation used for evaluation. However, the framework is agnostic to the specific BT structure and can be extended to any compatible design. The behavior tree employed in our benchmarking setup comprises multiple node types, which can be categorized into *control nodes* and *custom nodes*. While the control nodes are responsible for governing the execution flow, the custom nodes handle task-specific robot behaviors that are aligned with the framework’s architecture.

Custom behavior modules. Similar to the behavior tree structure, the framework can support a diverse set of behavior modules, each encapsulating a specific perceptual or control capability. As depicted in Fig. 4, the current paper

Algorithm 2 Person Tracking Behavior Module.

```

procedure INIT
  Initialize drivers, plugins, LLM, and BT
3:   person  $\leftarrow$  null
end procedure

procedure TRACK
6:   Send bounding_boxes to LLM Interface
   tracking_signal  $\leftarrow$  LLM Interface
   if tracking_signal equals "tracking" then
9:     person  $\leftarrow$  bounding_boxes overlapping tracking_signal
     vel_cmd  $\leftarrow$  Build velocity command from person
     Publish vel_cmd topic to robot
12:   end if
end procedure

procedure LOST
15:   direction  $\leftarrow$  last known person position
   vel_cmd  $\leftarrow$  rotate in axis in direction
   Publish vel_cmd topic to robot
18: end procedure
▷ Rotate to identify the person

procedure MAIN
  Run INIT()
21:   tracking_signal  $\leftarrow$  null
   for all frame of camera frames do
     bounding_boxes  $\leftarrow$  detected person objects in frame using YOLO
24:     if bounding_boxes is not null then
       Run TRACK()
     else
       Run LOST()
     end if
   end for
30: end procedure
```

benchmarks the framework’s capabilities using two representative behavior modules as a proof of concept: *person tracking* and *hand gesture-based control*.





















The *hand gesture-based control* enables intuitive robot control through hand gestures, allowing users to issue behavioral commands using predefined gesture patterns. Once triggered, it processes real-time gesture inputs to dynamically control the robot’s actions. This modality offers a responsive interface, particularly useful in scenarios where text-driven control is unavailable. The *person tracking* behavior module enables the robot to identify and track a person in real-time by processing the video feed from the robot’s onboard/mounted camera and applying an object detection algorithm to identify individuals in the scene. To implement this behavior module, we have employed YOLO11 [27] for real-time object detection. As summarized in Algorithm 2, detected objects are passed to the LLM, which determines the appropriate target for tracking based on the user intent. Once a target person is selected, the plugin continuously issues movement commands to the robot via the driver interface to maintain focus on and follow the individual.

4.3 Success Rate Analysis

The objective of these experiments is to assess the capability of the LLM Interface to generate semantically appropriate interpretations of user commands, dispatch them to proper modules or executors, and verify the correctness of the corresponding task executions. In this regard, command interpretation accuracy η_i^{cog} for the scenario index i is evaluated through manual analysis to ensure that the system’s understanding of each command aligns with its intended meaning. Dispatch accuracy η_i^{disp} is assessed by verifying whether the appropriate module or execution pathway is correctly activated in response to the interpreted command. Finally, η_i^{exec} is determined by observing whether the robot completes the intended task in the given environment, as confirmed by visual observations and system feedback logs. As illustrated in Table 2, the results validate the viability of the proposed framework across various real-world scenarios. The few cases with partially lower σ values are due to cognitive misunderstanding or practical integration issues.

According to the table, the system achieves an average accuracy of 0.93 for cognition, 0.92 for task dispatching, 0.95 for execution, and an overall end-to-end success rate of 0.96 across different robots and various real-world scenarios. Out of the 20 evaluated scenarios, eleven achieved a perfect success rate ($\sigma = 1.00$) and five obtained $\sigma \geq 0.85$, indicating that the LLM was able to correctly interpret the instruction, trigger the appropriate behavior,

Table 2: Success Rate (σ) of behavior execution across scenarios over $k = 10$ independent runs. η_i^{cog} , η_i^{disp} , and η_i^{exec} refer to LLM cognition, dispatching stage, and task execution success levels of the scenario s_i , respectively.

#	Robot	η^{cog}	η^{disp}	η^{exec}	σ	Details
$\Phi 1.1$		1.0	–	1.0	1.00	
$\Phi 1.1$		1.0	–	1.0	1.00	
$\Phi 2.1$		1.0	0.7	0.8	0.83	Flip direction varies with input
$\Phi 2.2$		1.0	1.0	0.8	0.93	Flip direction varies with input
$\Phi 2.3$		1.0	1.0	1.0	1.00	
$\Phi 3.1$		1.0	1.0	1.0	1.00	
$\Phi 3.2$		0.9	0.8	1.0	0.90	
$\Phi 3.3$		1.0	–	1.0	1.00	
$\Phi 3.4$		1.0	1.0	1.0	1.00	
$\Phi 3.5$		1.0	–	1.0	1.00	
$\Phi 4.1$		0.7	1.0	0.7	0.80	Velocity defaults without input
$\Phi 4.1$		0.7	1.0	0.7	0.80	Velocity defaults without input
$\Phi 4.2$		0.8	1.0	0.8	0.87	Velocity defaults without input
$\Phi 4.2$		0.7	1.0	0.7	0.80	Velocity defaults without input
$\Phi 5.1$		1.0	1.0	1.0	1.00	Highly similar LLM outputs
$\Phi 5.1$		0.8	1.0	0.8	0.87	Highly similar LLM outputs
$\Phi 5.2$		1.0	1.0	1.0	1.00	
$\Phi 5.2$		0.9	1.0	1.0	0.97	
$\Phi 6.1$		1.0	1.0	1.0	1.00	
$\Phi 6.2$		1.0	1.0	1.0	1.00	
Avg.	Any	0.93	0.92	0.95	0.94	





















and observe the expected result. The interpretation accuracy η_i^{cog} remained near-perfect in most cases, except for the *Motion Command* scenarios ($\Phi 4$) where the motion direction and duration need to be appropriately understood by the LLM. The legged robot showed lower accuracy than the drone for *Plugin Switching* ($\Phi 5$), possibly due to its complex locomotion dynamics and the higher precision required for behavior activation. Additionally, in scenario $\Phi 3.2$ under *System Inquiry Response*, the LLM exhibited mild signs of behavior mapping ambiguity, possibly due to context mismanagement. Dispatching accuracy η_i^{disp} are consistently high across supported actions except from $\Phi 2.1$ and $\Phi 3.2$, where incomplete routing to executors were observed. Scenarios marked with a dash in the mentioned column indicate cases where no dispatching occurred and the execution took place directly at the driver’s level, without requiring the tools to be called. Finally, regarding η_i^{exec} , the rates strongly correlate with the system’s state awareness and the level of contextual information required for successful task completion. This explains why the drone exhibits lower η_i^{exec} in ($\Phi 2$) and ($\Phi 4$), which specifically evaluate motion control and contextual understanding.

In summary, the experimental results demonstrate that the proposed framework maintains a high level of reliability and semantic consistency across heterogeneous robotic platforms and diverse interaction scenarios. Scenarios with lower σ confirm that a clear trend is highly observable between cognition and execution success, where lower η_i^{cog} values mainly correspond to higher reasoning ambiguity or context misinterpretation. This suggests that the quality of the underlying language model’s semantic understanding is the primary factor influencing the system’s end-to-end performance.

4.4 Runtime Performance

The elapsed time for each scenario, comprising LLM interpretation (\mathcal{L}_{cog}), command dispatch ($\mathcal{L}_{\text{disp}}$), and task execution ($\mathcal{L}_{\text{exec}}$), is presented in Table 3. As observed, $\mathcal{L}_{\text{disp}}$ incurs the lowest computational cost among the three stages, with negligible latency in most scenarios (13 out of twenty cases with $\mathcal{L}_{\text{disp}} \leq 1$). While such low values correspond to direct decision-making, scenarios with higher dispatch times typically result from command-specified

Table 3: Timing results for selected scenarios, including LLM response time (\mathcal{L}_{cog}), dispatching time ($\mathcal{L}_{\text{disp}}$), and task execution duration ($\mathcal{L}_{\text{exec}}$), in milliseconds (ms).

#	Robot	\mathcal{L}_{cog}	$\mathcal{L}_{\text{disp}}$	$\mathcal{L}_{\text{exec}}$	t_i^{total}	Details
$\Phi 1.1$		4200	0	–	4200	
$\Phi 1.1$		4000	0	–	4000	
$\Phi 2.1$		4113	≤ 1	611	4724	LLM may ask for direction
$\Phi 2.2$		2022	551	1195	3768	
$\Phi 2.3$		2681	≤ 1	–	2681	
$\Phi 3.1$		2131	≤ 1	–	2131	Listing actions takes time
$\Phi 3.2$		2540	≤ 1	–	2540	
$\Phi 3.3$		7782	≤ 1	–	7782	
$\Phi 3.4$		1688	≤ 1	–	1688	Listing actions takes time
$\Phi 3.5$		+9999	≤ 1	–	+9999	
$\Phi 4.1$		2559	1010	2321	5890	LLM can ask for more info
$\Phi 4.1$		3675	1920	3154	8749	Re-execution due to LLM
$\Phi 4.2$		5572	542	1647	7761	LLM can ask for more info
$\Phi 4.2$		2424	289	2713	5426	Slower motion execution
$\Phi 5.1$		2391	≤ 1	–	2391	Depends on control change
$\Phi 5.1$		3150	≤ 1	–	3150	Depends on control change
$\Phi 5.2$		3023	≤ 1	–	3023	
$\Phi 5.2$		2981	≤ 1	–	2981	
$\Phi 6.1$		2380	294	+9999	+9999	Varies with user input
$\Phi 6.2$		2030	5000	–	7030	Varies with user input

execution durations embedded in the dispatching process. The most computationally demanding scenario is $\Phi 6.2$, where the user requests person tracking but the camera fails to detect the target. In this case, the dispatching phase lasts approximately five seconds while the robot attempts to locate the person, after which it halts. Other time-intensive cases arise in $\Phi 4.1$ and $\Phi 4.2$ under *Motion Command*, where the system must verify the robot’s status regardless of its type. Thus, an overhead is likely due to the additional sensing, state validation, and decision-making steps required prior to motion commands.

In most scenarios, the majority of the time is spent querying the LLM in \mathcal{L}_{cog} , which can vary depending on the task complexity and the length of the generated text. In other words, \mathcal{L}_{cog} dominates the overall response time due to the time required by the LLM to generate reasonable responses. Notably, $\Phi 3.5$ requires a longer time due to the amount of information that must be displayed to the user. Regarding $\mathcal{L}_{\text{exec}}$, most actions require the standard execution time of the platform. The highest execution time can be seen in $\Phi 6.1$, where the robot remains in an infinite loop while following the person. Moreover, *Motion Command* scenarios in $\Phi 4.X$ exhibit great execution time due to multiple factors, including the magnitude of the requested motion value (e.g., distance or altitude) and hardware-specific capabilities. While the first factor directly affects how long the robot must move before completing the task, the latter is due to the robot’s locomotion variation. For instance, the legged robot is inherently slower than the drone, resulting in longer completion times.

In general, despite the inherent variability in LLM processing, the system consistently exhibits stable timing behavior and minimal dispatch overhead, demonstrating the efficiency of the ROS-based communication layer and the modular system architecture. Additionally, scenarios with higher cognition times are primarily associated with task complexity and context-dependent reasoning, where the LLM needs to process longer or more ambiguous instructions. This is not directly aligned with the $\mathcal{L}_{\text{exec}}$ results, as the execution time is mainly influenced by the robot’s locomotion characteristics and platform-specific speed.

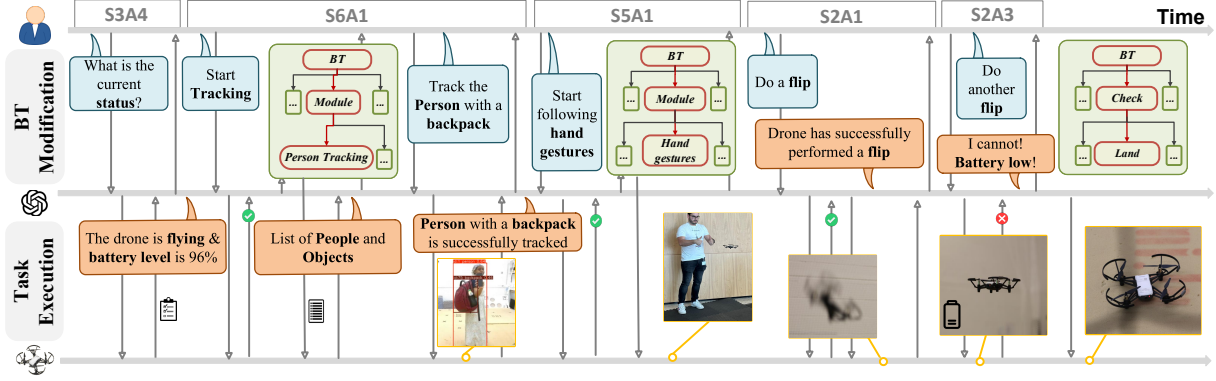


Figure 5: High-level state-flow diagram of the proposed end-to-end HRI system, highlighting command interpretation and execution states.

4.5 Qualitative Results

Fig. 5 presents a high-level state flow of scenario-based HRI evaluations, depicting the transitions and execution states involved in task interpretation and control. Accordingly, based on the commands given by the user, the task execution is triggered, guiding the system through the appropriate states and transitions. The figure illustrates the dynamic nature of multimodal HRI, supported by our proposed framework, showing how natural language inputs activate behavior modules and influence the flow of execution. One dialog is related to the human and the other to the LLM’s response to the corresponding task.

4.6 Discussions

The conducted experiments showcase the capability of the proposed framework as an end-to-end application, integrating an LLM to trigger diverse tasks through its behavior-driven architecture. By integrating an LLM as the cognitive interface, the framework enables high-level task interpretation and coordination of behavior modules, validating its functionality in real-world HRI settings. The system performs reliably in scenarios characterized by low to moderate cognitive and physical complexity. In such cases, the LLM effectively interprets user instructions and triggers the appropriate behavior tree nodes for execution. However, the execution accuracy relies heavily on the clarity of user commands, and ambiguity or under-specified input (*e.g.*, “Do a flip” without indicating direction) can lead to inconsistent or incomplete behavior triggering. In these cases, the LLM may either issue a request for clarification or default to predefined assumptions, potentially deviating from user intent. In more complex scenarios, particularly those requiring multi-step task decomposition, the framework relies on the LLM interface’s ability to reason, formulate a multi-step plan, and activate appropriate behavior modules.

While the system shows promising results in executing demanded tasks, several performance limitations remain. Performance bottlenecks are primarily linked to the LLM’s nature and its dependency on third-party perception or control plugins (such as YOLO11 for person tracking). These components introduce variability in behavior outcomes, particularly in cases involving sensor noise, uncertain scene dynamics, or subtle linguistic variations in user input.

5 Conclusions

This paper presented an open-source, modular, and extensible framework that combines LLMs with behavior trees, enabling robots to interpret and execute natural language commands in a transparent and scalable manner. The core contributions of this work lie in simplifying human-robot interaction and the interpretability of robot decision-making, enabling more dynamic and adaptable robot behaviors. Real-world experiments across diverse environments have shown that the system demonstrates a high success rate in both the “semantic interpretation” of user instructions and the “execution of corresponding robotic behaviors” stages. These experiments were structured into defined scenarios, encompassing perception-driven tasks and motion commands, to evaluate the system’s robustness and versatility under realistic conditions.

Future work will address system design aspects not covered in this paper. First, we plan to benchmark the impact of various LLM back-ends (e.g., OpenAI, ST, Llama) and perception-driven behavior modules on the system’s decision-making and execution quality. Additionally, extending the framework to multi-robot coordination is another step to evaluate collaborative task execution across heterogeneous robots. Finally, integrating more input modalities and supporting dynamic behavior selection through the BT are additional extensions that can further enhance the framework.

References

- [1] T. Wang, P. Zheng, S. Li, and L. Wang, “Multimodal human-robot interaction for human-centric smart manufacturing: a survey,” Advanced Intelligent Systems, vol. 6, no. 3, p. 2300359, 2024.
- [2] R. A. S. Fernandez, J. L. Sanchez-Lopez, C. Sampedro, H. Bavlé, M. Molina, and P. Campoy, “Natural user interfaces for human-drone multi-modal interaction,” in 2016 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, 2016, pp. 1013–1022.
- [3] B. Obrenovic, X. Gu, G. Wang, D. Godinic, and I. Jakhongirov, “Generative ai and human-robot interaction: implications and future agenda for business, society and ethics,” AI & society, pp. 1–14, 2024.
- [4] C. Y. Kim, C. P. Lee, and B. Mutlu, “Understanding large-language model (llm)-powered human-robot interaction,” in Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction, 2024, pp. 371–380.
- [5] Y. Wu, Y. Tao, P. Li, G. Shi, G. S. Sukhatmem, V. Kumar, and L. Zhou, “Hierarchical llms in-the-loop optimization for real-time multi-robot target tracking under unknown hazards,” arXiv preprint arXiv:2409.12274, 2024.
- [6] W. Zu, W. Song, R. Chen, Z. Guo, F. Sun, Z. Tian, W. Pan, and J. Wang, “Language and sketching: An llm-driven interactive multimodal multitask robot navigation framework,” in 2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2024, pp. 1019–1025.
- [7] W. Zhao, L. Li, H. Zhan, Y. Wang, and Y. Fu, “Applying large language model to a control system for multi-robot task assignment,” Drones, vol. 8, no. 12, p. 728, 2024.
- [8] C. Wang, S. Hasler, D. Tanneberg, F. Ocker, F. Joubin, A. Ceravola, J. Deigmoeller, and M. Gienger, “Lami: Large language models for multi-modal human-robot interaction,” in Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–10.
- [9] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegwart, and C. Smith, “On the programming effort required to generate behavior trees and finite state machines for robotic applications,” in 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023, pp. 5807–5813.
- [10] Y. Cheng, L. Sun, and M. Tomizuka, “Human-aware robot task planning based on a hierarchical task model,” IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 1136–1143, 2021.
- [11] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, “Safe learning in robotics: From learning-based control to safe reinforcement learning,” Annual Review of Control, Robotics, and Autonomous Systems, vol. 5, no. 1, pp. 411–444, 2022.
- [12] M. Colledanchise and P. Ögren, Behavior trees in robotics and AI: An introduction. CRC Press, 2018.
- [13] A. A. Kareem, D. A. Hammood, R. A. Khamees, and N. B. H. Ismail, “Object tracking with the drone: Systems analysis,” Journal of Techniques, vol. 5, no. 2, pp. 89–94, 2023.
- [14] J. Qi, L. Ma, Z. Cui, and Y. Yu, “Computer vision-based hand gesture recognition for human-robot interaction: a review,” Complex & Intelligent Systems, vol. 10, no. 1, pp. 1581–1606, 2024.
- [15] A. Koubaa, A. Ammar, and W. Boulila, “Next-generation human-robot interaction with chatgpt and robot operating system,” Software: Practice and Experience, vol. 55, no. 2, pp. 355–382, 2025.

- [16] C. E. Mower, Y. Wan, H. Yu, A. Grosnit, J. Gonzalez-Billandon, M. Zimmer, J. Wang, X. Zhang, Y. Zhao, A. Zhai et al., “Ros-llm: A ros framework for embodied ai with task feedback and structured reasoning,” arXiv preprint arXiv:2406.19741, 2024.
- [17] G. Sarch, Y. Wu, M. J. Tarr, and K. Fragkiadaki, “Open-ended instructable embodied agents with memory-augmented large language models,” in Findings of the Association for Computational Linguistics: EMNLP 2023, 2023, pp. 3468–3500.
- [18] N. J. P. Laboratory, “ROSA: Reasoning-observation-action framework,” <https://github.com/nasa-jpl/rosa>, 2023, accessed: June 2025.
- [19] B. Benjdira, A. Koubaa, and A. M. Ali, “Prompting robotic modalities (prm): A structured architecture for centralizing language models in complex systems,” Future Generation Computer Systems, vol. 166, p. 107723, 2025.
- [20] C. Tagliamonte, D. Maccaline, G. LeMasurier, and H. A. Yanco, “A generalizable architecture for explaining robot failures using behavior trees and large language models,” in Companion of the 2024 ACM/IEEE International Conference on Human-Robot Interaction, 2024, pp. 1038–1042.
- [21] A. Lykov and D. Tsetserukou, “Llm-brain: Ai-driven fast generation of robot behaviour tree based on large language model,” in 2024 2nd International Conference on Foundation and Large Language Models (FLLM). IEEE, 2024, pp. 392–397.
- [22] 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.
- [23] Z. Yang and Z. Jia, “Robot behavior tree manipulation using language models,” in 2023 IEEE 11th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), vol. 11, 2023, pp. 1342–1345.
- [24] J. Styrud, M. Iovino, M. Norrlöf, M. Björkman, and C. Smith, “Automatic behavior tree expansion with llms for robotic manipulation,” in 2025 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2025, pp. 1225–1232.
- [25] 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.
- [26] J. Liang, Y. Chang, Q. Wang, Y. Wang, and X. Yi, “Diagbt: An explainable and evolvable robot control framework using dialogue generative behavior trees,” in 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2024, pp. 7056–7062.
- [27] G. Jocher and J. Qiu, “Ultralytics yolo11,” 2024. [Online]. Available: <https://github.com/ultralytics/ultralytics>