University of Stuttgart
Institute of Industrial Automation and
Software Engineering

A Comparative Study of LLM-Based and PDDL-Based Methods for Automatic Behavior Tree Generation

Research Project

Submitted at the University of Stuttgart by
Chuang Yan

Electromobility – Master of Science

Examiner: Prof. Dr.-Ing. Dr. h.c. Michael Weyrich

Supervisor: Yuchen Xia

Co-Supervisor: Ruichao Wu (Fraunhofer IPA)

2024-9-14

Table of Contents

Table of Figures

Table of Tables

Table of Abbreviations

NLP	**N**eural **L**anguage **P**rocessing
LLM	**L**arge **L**anguage **M**odel
GPT	**G**enerative **P**re-trained **T**ransformer
XML	**E**xtensible **M**arkup **L**anguage
NN	**N**eural **N**etworks
API	**A**pplication **P**rogramming **I**nterface
PDDL	**P**lanning **D**omain **D**efinition **L**anguage
RL	**R**einforcement **L**earning
BT	**B**ehavior **T**ree
ROS2	**R**obot **O**perating **S**ystem 2
FSM	**F**inite **S**tate **M**achine
PDDL2.1	**P**lanning **D**omain **D**efinition **L**anguage **V**ersion 2.1
PDDLStream	**P**lanning **D**omain **D**efinition **L**anguage with **S**tream **I**ntegration
TAMP	**T**ask and **M**otion **P**lanning
POPF	**P**artial-**O**rder **P**lanning with **F**orward Search

Glossary

XML	A flexible text format used to create structured documents by defining a set of rules for encoding documents in a format that is both human-readable and machine-readable.
NLP	A process or technique by applying computer techniques to analyze and synthesize natural language and speech
API	A set of rules and definitions that allows different software applications to communicate with each other.
PDDL	A formal language used in artificial intelligence (AI) for defining planning problems and actions. It provides a structured framework for specifying the environment (domain) and the tasks (problem) that a planner needs to solve.

Abstract

BTs are a powerful framework for behavior control in robotics, offering modularity and adaptability in decision-making processes. This paper presented a comparative study of automatic BT generation using three approaches: PDDL-based planning, LLMs, and hybrid methods combining both. While PDDL planners excelled in structured task environments, they struggled with high-level goal abstraction and dynamic planning. Conversely, LLM-based methods showed promise in handling flexible, complex tasks but fell short in achieving optimal plans. Hybrid methods leveraged the strengths of both, achieving higher accuracy but at the cost of increased planning time. Through simulations in Isaac Sim, this paper evaluated BT accuracy rate and BT generation time across scenarios such as warehouse operations and object sorting. Results showed that PDDL-based planners performed well in simple tasks, while LLMs offered better adaptability in more complex tasks. Hybrid methods struck a balance between accuracy and flexibility, providing a robust solution for dynamic task environments. This paper concluded by highlighting the strengths and limitations of each method and proposed future directions, including reinforcement learning-based BT generation and further exploration of advanced PDDL versions.

Key Words: BTs, PDDL, LLMs, Isaac Sim, AutoGen, ROS2,TAMP,POPF.

1 Introduction

BTs are a structured methodology initially crafted for modular artificial intelligence in video games, which has progressively been adopted within the realm of robotics due to their capacity to handle increasing AI complexities [1]. In their essence, BTs facilitate a hierarchical organization of decision-making processes, where a multitude of simple tasks are structured in a tree-like formation to dictate the behavior of an agent—be it a robotic entity or a virtual character [2]. This configuration not only enhances modularity but also simplifies both the development and analytical evaluation by human operators and automated systems, paving the way for more dynamic and adaptive robotic applications.

The automation of BT generation has become a focal point in contemporary research due to the traditional manual crafting of BTs requiring extensive expertise and often falling short in dynamic adaptability in complex scenarios. The first major approach involves the use of Planning Domain Definition Language (PDDL) with classical planners, exemplified by systems like PlanSys2. This method integrates BT generation and execution modules that interface with ROS2 to tackle industrial tasks, necessitating a predefined model of the world. However, it often struggles with continuous tasks that require dynamic effect analysis, highlighting a critical research gap. Concurrently, the potential of LLMs has been explored for generating BTs using natural language inputs, which specify roles, scenarios, and tasks. While LLMs offer innovative text generation capabilities, their performance varies across different task complexities, particularly in more demanding scenarios where they may underperform relative to classical planners. A third emerging research direction seeks to amalgamate classical planners and LLMs to leverage the benefits of both approaches in creating more versatile and effective planning solutions.

This paper focused on assessing the comparative advantages, limitations, and applicability of these three methodologies in generating BTs for robots across varying levels of task complexity. Through methodical experiments conducted in simulation environments like Isaac Sim, this study evaluated scenarios including warehouse operations and cube sorting tasks. It contrasted the effectiveness of PDDL-based classical planners, such as POPF, against LLM-driven approaches and their hybrid implementations. Preliminary results indicated that while LLMs performed comparably to classical planners in simpler scenarios, they exhibited longer computation times. For moderately complex tasks, PDDL design presented considerable challenges, and LLM accuracy diminished. However, in highly complex, continuous task settings that required real-time adaptive planning, LLMs—particularly those augmented with reinforcement learning and Markov decision processes—demonstrated superior problem-solving efficiency. This comprehensive analysis aimed to illuminate the pathways towards more autonomous and flexible robotic systems through advanced BT generation techniques.

2 Background

2.1 Introduction to BT

A BT is a dynamic and hierarchical model used to structure the decision-making process in an autonomous agent, such as mobile robot or a virtual entity in a computer game [1,2,3]. This method is distinctly different from Finite State Machines (FSM), which have traditionally been used to manage simpler state transitions without the inherent modularity or flexibility offered by BTs [5]. Moreover, in large scale tasks design, BTs can benefit in programming design and dynamic interaction with Environment.

The structure of BT consists of several distinct node types, including the root, control nodes, and leaf nodes. The leaf nodes, also known as execution nodes, are responsible for executing specific actions or conditions, while the non-leaf nodes, referred to as control flow nodes, manage the decision-making process and direct the flow of execution. Fig.2-1 illustrates a typical BT structure, demonstrating how these nodes interact hierarchically. A detailed breakdown of the node types and their respective functions within the BT framework is provided in Table 2-1, offering further insight into their operational roles.

Table 2-1 The five nodes of a BT

Node type	Symbol	Succeeds	Fails	Running
Sequence	→	If all children succeed	If one child fails	If one child returns running
Fallback	?	If one child succeeds	If all children fail	If one child returns running
Parallel	⇒	If ≥ M children succeed	If > N – M children fail	Else
Action	Shaded box	Upon completion	When impossible to complete	During completion
Condition	White oval	If true	If else	Never

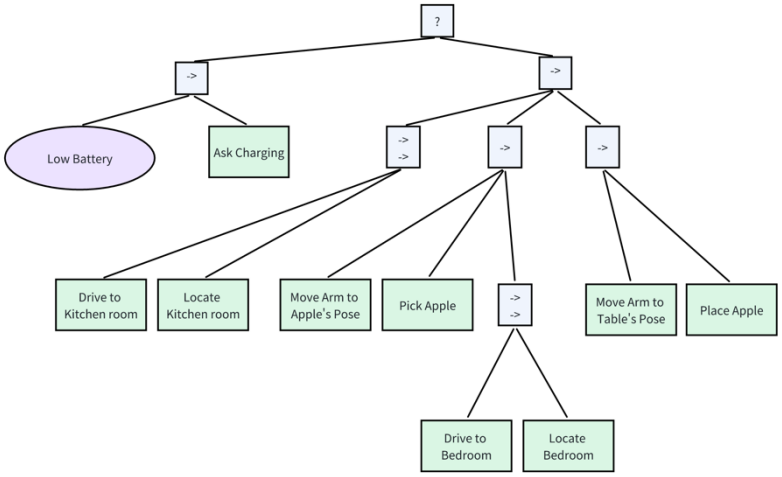


Fig. 2-1. A BT Example perfoming pick and place task.

The execution of a BT begins at the root node, which emits activation signals known as "Ticks." These Ticks propagate down the hierarchy, activating child nodes to execute their designated tasks. Each node, upon receiving a Tick, performs its function and reports its status back to the parent node: 'Running' if the task is ongoing, 'Success' if the goal is achieved, or 'Failure' if it is not. This continual feedback allows the BT to dynamically adapt its actions based on real-time assessments of each node's performance.

2.2 Application Fields of BTs

In this section, typical applications of BTs are discussed. In general, BTs can help agent behavior execution and robotics area.

2.2.1 Game AI

In [6] BTs were first been using in field of NPC Game, where the NPC's behavior is predefined in BT structure. Over the past decade, BTs have achieved remarkable success in various types of video games, particularly in real-time strategy (RTS) games, first-person shooters (FPS), platform games, and dialogue-based games. BTs have become the preferred architecture for implementing complex, intelligent behavior in game characters due to their modularity and flexibility.

In real-time strategy games (RTS), BTs are used to control units that need to make quick decisions based on dynamic, ever-changing environments [7]. The ability of BTs to react fluidly to these changes allows game developers to create more lifelike, strategic behaviors in units that can engage in combat, gather resources, or defend territories.

In first-person shooters (FPS), BTs are employed to govern the behavior of non-playable characters (NPCs), providing them with the capacity to adapt to player actions, such as seeking cover, returning fire, or engaging in tactical maneuvers [8]. This adaptability leads to more challenging and engaging gameplay, as NPCs can exhibit complex, human-like decision-making.

In platform games, BTs are frequently used to manage the behavior of enemies and other in-game characters that follow predefined patterns while reacting to player movements [9]. For example, an enemy might patrol an area until spotting the player, at which point the BT activates an attack sequence or a chase, making the game feel more responsive and interactive.

In dialogue-based games [1,2], BTs are highly effective in managing branching conversations and interactions between characters. By structuring dialogue choices and responses within a BT framework, game developers can create more dynamic and context-sensitive dialogue systems, where characters' reactions are determined not only by the player's choices but also by the ongoing narrative and previous interactions.

The widespread adoption of BTs in these genres illustrates their versatility in enhancing the complexity and intelligence of game AI, leading to richer and more immersive player experiences.

2.2.2 Robotics

In 2012, J. Andrew Bagnell etc. [10]. developed an integrated system for autonomous robotics manipulation with BT. They propose to use BTs for UAV control, the possibilities of BTs were first brought to robotics. Nowadays BTs are mainly applied to two categories: manipulators, and mobile robots.

In robotic manipulation, tasks such as moving the arm from an initial pose to a target pose, combined with controlling the gripper state, are fundamental for performing actions like pick-and-place. BTs play a critical role in this context by providing a modular and flexible framework for managing task execution. Through goal state definitions and condition checks, BTs ensure that complex actions are executed in a structured and adaptive manner, allowing for efficient control over the robot's action lifecycle and real-time decision-making in dynamic environments. An industry collaborative robots utilizing BTs include ABB YUMI were introduced in [11, 12].

Navigation2 [13] is a great successful example utilizing BT in mobile robots. The framework of it is shown in Fig 2-2. BTs are employed to orchestrate key tasks such as global planning, local path control, and recovery actions. Each task is represented as a node within the BT, which allows the system to dynamically adjust its navigation strategy based on real-time conditions. For instance, when navigating through an environment, a BT might sequence the following steps: activate the global planner to determine a route, invoke the local planner to follow the path while avoiding obstacles, and, if a failure occurs (e.g., a blocked path), trigger a recovery behavior to handle the situation. This hierarchical structure enables the robot to navigate autonomously in

dynamic environments by adapting its actions based on conditions observed at runtime. In multi robot navigation, framework combined with BTs can complete complex mission by parallel node [14].

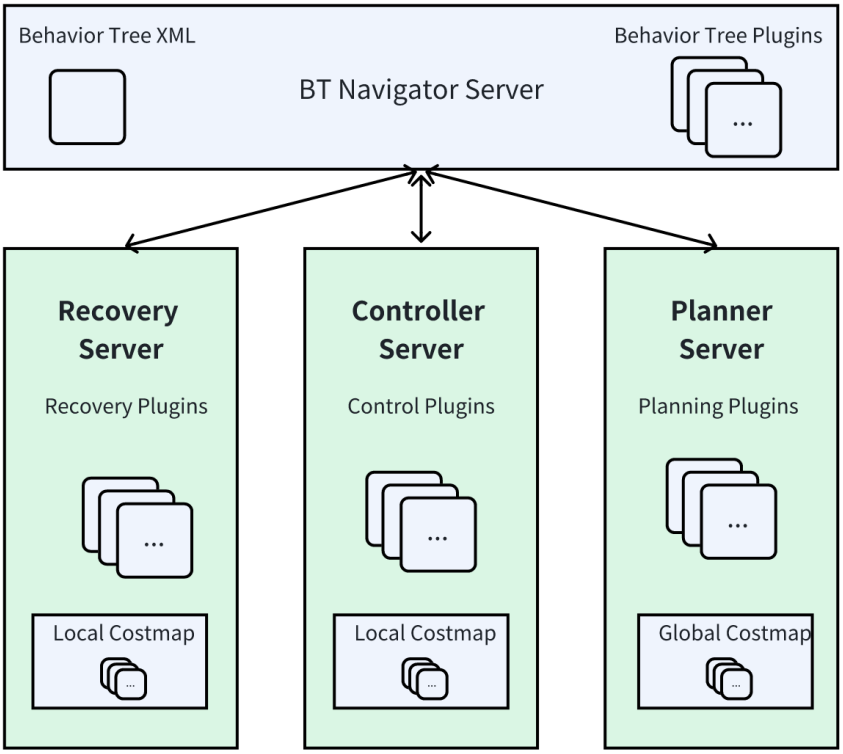


Fig. 2-2 Overview of Navigation2 Design

3 Related Work

This section summarizes relevant research efforts in automatic BT generation using both PDDL-based methods, LLM-based approaches, and hybrid systems that combine LLMs with classical planners such as PDDL. The discussion highlights key advancements, limitations, and contributions from these research areas, focusing on how they address the challenges in autonomous task planning for robotics.

3.1 PDDL-based Planning

PDDL is a formal language used to describe planning problems and actions in artificial intelligence (AI) and robotics. Originally developed for the AI planning community, PDDL provides a structured way to define the domain (i.e., the possible actions and objects in an environment) and the problem (i.e., the initial state, goal state, and constraints) [15]. It allows robotic systems to reason about the sequence of actions required to achieve a specific goal, enabling autonomous systems to plan complex tasks.

One significant work in this area is **PlanSys2**, introduced in [16], which integrates PDDL-based planning with BTs. In this system, a classic PDDL planner is used as a plugin to solve the task described in PDDL, and the resulting plan is converted into a BT, which is then executed by the BT.CPP library. This framework ensures a flexible execution of tasks where plans can be dynamically adjusted as conditions change in the environment.

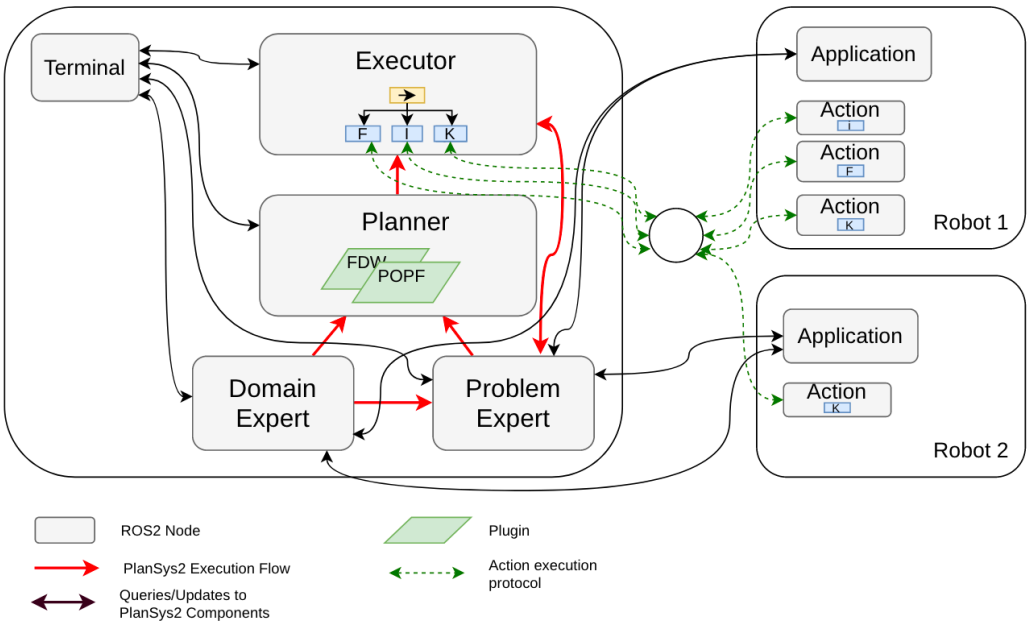


Fig. 3-1 PlanSys2 Architecture

Another related work is the ACROBA project, which aims to create a flexible robotic platform for agile production environments [17]. This project integrates a PDDL solver within a BT-based control system to allow automatic rewriting of tasks at runtime based on factory or sensor inputs [17]. The task planner is designed to provide a user-friendly interface that allows for quick reconfiguration of robotic cells, making the system adaptable to different industrial scenarios. PDDL is used to optimize or automate task and process designs, and the resulting plans are executed through a modular BT system that interacts with various ROS2 components.

SkiROS2 further extends the use of PDDL-based planning in robot control, providing a skill-based architecture that combines PDDL and BTs to execute complex tasks autonomously [16]. In SkiROS2, task-level planning is done using PDDL, and the plans are translated into extended BTs (eBTs) to enhance modularity and allow fast adaptation to change in the environment. This hybrid control structure is particularly useful for skill-based tasks in industrial robotics, where actions need to be dynamically adjusted based on sensor input or changes in the workspace.

Furthermore, hierarchical planning approaches have emerged to address the challenge of scaling PDDL in complex tasks. One example is the work by Lu et al. [18], which focuses on hierarchical extraction, planning, and BT process control using historical trajectory data. The proposed system extracts high-level operators from past planning trajectories, optimizing the planning process by reducing the search space and time. This method leverages PDDL to define the planning problem and uses BTs for task execution, ensuring that the system can handle real-time changes and task failures by replanning at different hierarchical levels. The integration of causal analysis in the PDDL planning process ensures that contradictions in operator ordering are minimized, enhancing the system's robustness.

3.2 LLM-Based Planning

While PDDL-based planning provides a structured and formalized approach to task generation and execution, its limitations become apparent in dynamic environments where continuous re-planning and adaptability are required. This challenge has led to the exploration of LLMs as an alternative or complementary approach for BT generation. Unlike PDDL, which relies on predefined domain models and explicit action descriptions, LLMs leverage vast amounts of data and NLP capabilities to generate task plans in a more flexible and adaptive manner [19].

Recent studies have explored the potential of LLMs for planning and BT generation in robotics and AI applications. One such area of research involves using LLMs to convert human-provided task descriptions into executable BTs. This approach allows non-expert users to interact with robots through natural language, specifying tasks and goals without the need for technical knowledge of PDDL or formal planning languages. The LLM interprets these inputs and generates a corresponding BT structure, which the robot can then execute.

One prominent example is the LLM-BT system [20], where LLMs such as GPT are used to generate BTs by converting task descriptions into executable sequences. The system uses an LLM to translate natural language commands into BTs, which can then be executed by the robot. This approach has been demonstrated in complex scenarios such as household robotics, where a robot can receive instructions like "clean the living room," and the LLM generates a BT with subtasks such as "vacuum the floor" and "dust the furniture." The adaptability of LLMs allows for dynamic task restructuring if the environment changes during execution.

Another significant advancement in LLM-based planning is the LLM-MARS framework [21], which integrates the Falcon 7B model to generate BTs and support human-robot dialogue. This system was successfully demonstrated in the Eurobot 2023 competition, where multi-agent systems of robots used LLM-generated BTs to perform collaborative tasks, such as navigating and collecting objects in dynamic environments. In this framework, the LLM not only generates BTs but also allows robots to engage in real-time dialogue with human operators, explaining their actions and providing feedback. The question-answering feature enhances human-robot interaction, making the system more intuitive and adaptable to user input.

Another important development in LLM-based planning is BTGenBot, which uses lightweight LLMs such as LLaMA and GPT-3.5 for generating BTs. By fine-tuning the models with specific datasets related to robotic tasks, BTGenBot enables efficient and scalable BT generation for applications like robotic manipulation and navigation. This research explores the deployment of compact LLMs directly on robots, making the system more practical for real-world usage where hardware limitations are a concern [22].

In addition, SayCan [23], a hybrid system developed by Google Robotics, utilizes LLMs to interpret natural language instructions and combine them with value functions that guide the robot’s actions based on environmental feedback. This approach enables robots to handle tasks with high-level goals, such as "fetch me a drink," while adapting to real-time environmental conditions, ensuring efficient task completion. SayCan bridges the gap between LLM-driven task interpretation and traditional robotic control systems.

3.3 Hybrid Approaches Combining PDDL and LLMs Planning

LLMs, while proficient in interpreting natural language and generating high-level task plans, struggle with the functional competence required for complex robotic task planning. LLMs often fail to account for long-term dependencies, task constraints, and physical feasibility [24], leading to incomplete or non-executable plans. In robotic systems, particularly in Task and Motion Planning (TAMP), these limitations are problematic, as the robots must deal with real-world constraints such as collision

avoidance and dynamic environmental changes. To address these challenges, hybrid approaches that combine LLMs with classical planners like PDDL have emerged. These hybrid systems allow LLMs to handle the flexibility and natural language interface while PDDL planners ensure precise, executable, and feasible task plans.

NL2Plan [27] is a hybrid framework where LLMs convert high-level, natural language task descriptions into PDDL specifications. The PDDL planner then refines these specifications into structured and feasible task plans, ensuring that dependencies and constraints are respected. By combining LLMs for task interpretation and PDDL for execution precision, NL2Plan addresses LLMs' limitations in handling complex, long-term tasks, improving planning accuracy and task feasibility.

LLM+P [24] takes a similar hybrid approach, using LLMs to translate user-provided natural language commands into PDDL specifications. The PDDL planner refines the task plan and ensures that it aligns with the robot's operational constraints. This hybrid system enhances the user interface, making task specification easier and more flexible, while ensuring that the plans are executable in real-world environments. LLM+P successfully bridges the gap between natural language task input and structured task execution.

LLM3 [26] extends the hybrid approach by integrating LLMs into the Task and Motion Planning (TAMP) framework. Unlike traditional TAMP methods, LLM3 uses LLMs to propose both symbolic actions and continuous motion parameters. The unique contribution of LLM3 lies in its ability to incorporate feedback from motion planning failures, refining task proposals iteratively based on this feedback. This allows the system to dynamically adjust plans to avoid motion failures, such as collisions or unreachable positions. In simulations, LLM3 demonstrated significant improvements in task success rates and planning efficiency, particularly in complex tasks like box-packing, where the system needed to navigate dynamic obstacles and refine its approach based on real-time motion constraints.

PDDLEGO [28] introduces a hybrid approach designed for partially-observed environments, where the robot does not initially have complete information about its surroundings. In PDDLEGO, LLMs are used to iteratively generate a PDDL problem file as the environment is explored, gradually refining the task plan as more information is gathered. This iterative planning approach ensures that the robot can adapt its actions based on new observations, significantly improving planning efficiency in dynamic, partially known environments.

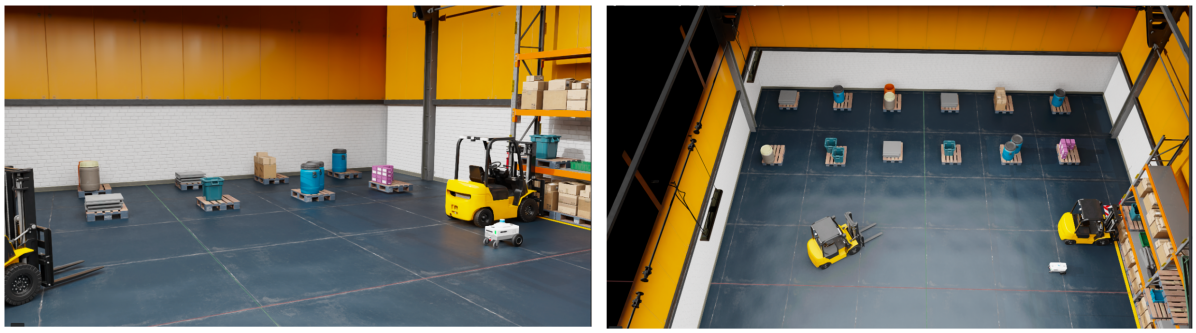
4 Experiments

4.1 Simulation Scenarios

In prior research, task planning performance has often been evaluated using goal sequences at mid-level complexity. However, the performance of classical planners and LLM-based models on higher-level, more abstract goals remains underexplored [24, 26, 27]. Moreover, there has been limited investigation into how these models cope with noisy environments or NL inputs. To bridge these gaps, this paper proposed two baseline tasks designed to test three methods, including classical (PDDL-based) planners, LLM-based and hybrid planners' methods across varying levels of task complexity and under noisy conditions. These experiments offered insight into how well each approach handles high-level planning challenges and environmental uncertainty. Both Simulations are setting with Isaac Sim, which provide well communication with ROS2.

4.1.1 Warehouse Task

In this experiment, a robot carter and 18 waypoints are placed within a simulated warehouse environment. Various objects are located at each waypoint. The carter must perform patrol and shot action with expected waypoints sequence. The tasks are categorized into low-level, mid-level, and high-level settings, with detailed description in Table 4-1 below. The scenario simulation is shown in Fig. 4-1.



(a) (b)

Fig. 4-1 Simulation of Warehouse in Isaac Sim (a): side view (b): top view

- **Low-level** tasks: in these tasks, the waypoints are unconnected, meaning the carter can move directly from starting point to the target waypoint. The planning challenge at the level is minimal as no route optimization is required, but only give the expected action sequence in BT.
- **Mid-level** tasks: at this level, a single bidirectional connection is predefined between waypoints. The carter must traverse these connections to patrol and perform actions at the expected waypoints. The planning challenge at the level is performing a path in predefined connection.

- **High-level** tasks: in this case, the carter must complete the patrol and shot action sequence across multiple waypoints with the minimum cost. Here, the planner must optimize the path, solving for the shortest route while completing the necessary actions in the correct sequence.
- **Domain with action variant:** in this task, different action variant is considered: types, predicates and also the surrogate actions with different effects. The goal of this experiment is to evaluate which planning approach can maintain optimal performance despite the noise. The expectation is that some planners, particularly LLM-based models, may be more susceptible to environmental noise due to their reliance on natural language interpretation, while classical planners (PDDL-based) or hybrid models could potentially handle the variant action more effectively by leveraging structured reasoning. The example of Skills variant is listed in table 4-3.

Table 4-1 Overview for Warehouse Task

Warehouse	Task Complexity	Tests	Test Description	Noise/Changes
Low level tasks:	Simple	Test1	Partial waypoints are patrolled.	None
		Test2	Partial waypoints are patrolled and shot	None
		Test3	All waypoints are not patrolled and shot	None
Middle level tasks	Moderate	Test1	Partial waypoints are patrolled.	None
		Test2	Partial waypoints are patrolled and shot	None
		Test3	All waypoints are not patrolled and shot	None
High level tasks	Complex	Test1	Partial waypoints are patrolled.	None
		Test2	Partial waypoints are patrolled and shot	None
		Test3	All waypoints are not patrolled and shot	None
Domain with noise tasks	Varies	Test1(Type Noise)	Variant are types (Add 2 irrelevant waypoints)	2 extra waypoints
		Test2(Action Noise)	Variant are actions (Add 2 surrogate actions with different effects)	2 extra actions
		Test3(Predicate Noise)	Variant are predicates (Add 2 irrelevant predicates)	Predicate mismatch

The Illustration of different scenario setting are shown as Fig.4-2.

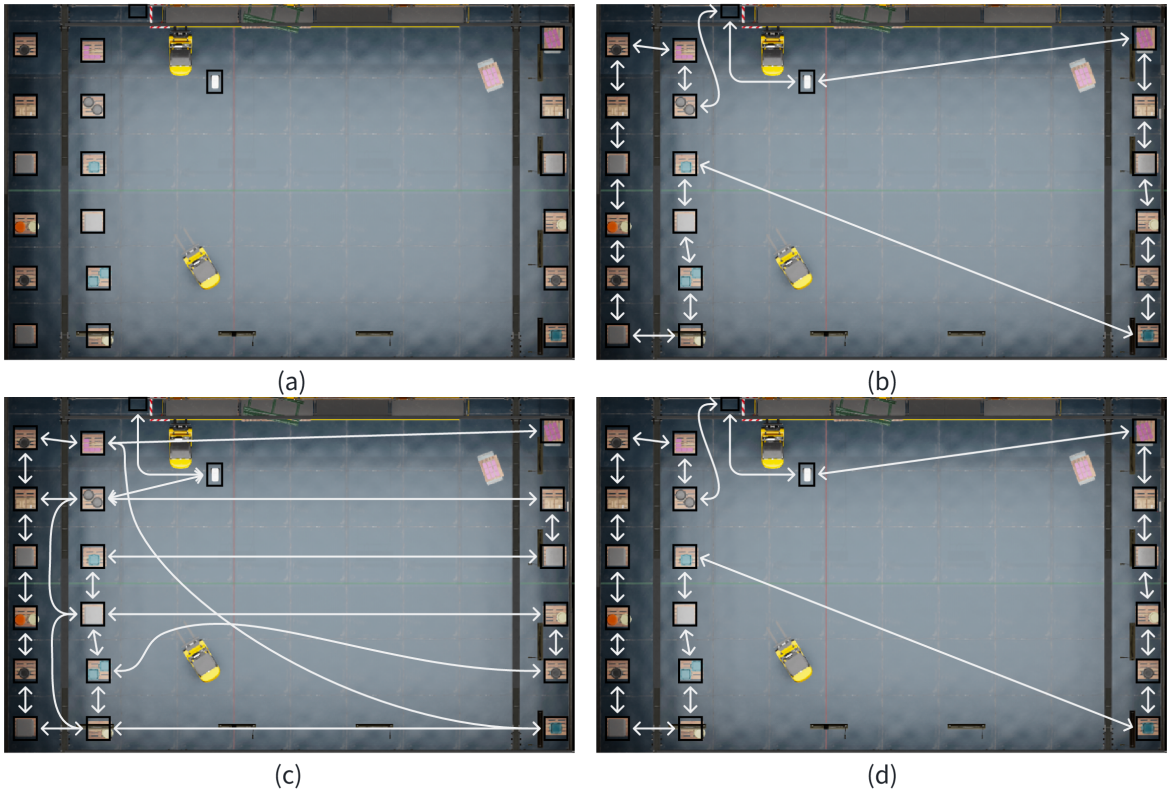


Fig. 4-2 Scenario Setting for 4 Tasks in Isaac Sim. (a) 18Waypoints have no connection. (b) Every two waypoints are bidirectional connected. (c) Multi connections between waypoints. (d) Keep connection setting as (b) but with variant inputs.

The available skills in ros2 for scenario warehouse are shown in Table 4-2.

Table 4-2 Skill set in Warehouse

Skill's name	Skilll Description	Properties	
askCharge	Carter will check battery state. It's been executed when state is low.	input	None
		output	Wp
charging	Carter will be charged at waypoint: wp_charge	input	Wp
		output	None
move	Carter can move from original waypoint to target waypoint.	input	Wp1; Wp2
		output	None
patrol	Carter can do patrol at waypoint.	input	Wp
		output	None

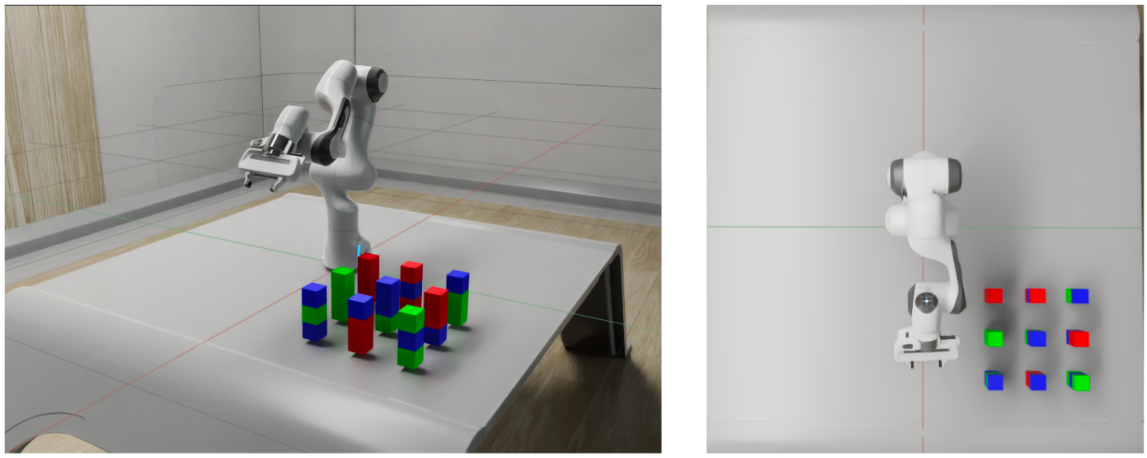
shot	Carter can shot the objects at waypoints, the pictures will be analyzed later.	input	Wp
		output	None

Table 4-3 Skills example variant in Warehouse

Skill's name	Skilll Description	Properties	
move	Carter can move from original waypoint to target waypoint.	input	Wp1; Wp2
		output	None
move_variant	Carter can move from original waypoint to target waypoint without limitation to waypoint connection	input	Wp
		output	None
patrol	Carter can do patrol at waypoint	input	Wp
		output	None
patrol_variant	Carter can do patrol at waypoint in specific behavior	input	Wp
		output	None

4.1.2 Objects Sorting Task

The Object Sorting Task is a comprehensive evaluation scenario designed to test both path planning and task planning across varying levels of complexity. Object sorting, a common task in industrial robotics [29], requires precise handling of both object manipulation and spatial arrangement. This task provides an ideal benchmark for comparing the performance of PDDL-based, LLM-based, and hybrid planning methods under increasingly complex conditions. In this experiment, a 7-DOF Panda robot was been choosen to sort cubes [30], each represented by different colors to simulate distinct objects. Initially, the cubes are randomly scattered in the environment, and the robot's task is to sort them into designated target regions based on their color. The simulation scenario is shown in Fig.4-3.



(a) (b)

Fig. 4-3 Simulation of Objects Sorting in Isaac Sim (a): side view, (b): top-down view

Four task configurations are defined, each simulating different levels of complexity and behavior. The purpose of this task is to evaluate the planners' ability to manage increasingly complex goals, from simple cube sorting to more challenging configurations that require dynamic re-planning. Notably, this experiment does not incorporate noise (as in the warehouse task), focusing instead on assessing how well each method handles varying goal levels. One key aspect of the task setup is the inclusion of both 2D and 3D sorting, where cubes may be in either an **accessible** or **inaccessible** state. The latter adds complexity, as cubes stacked underneath others require reasoning about spatial constraints and action dependencies.

- **Low-level** tasks: In these tasks, the planners are tested on simple goal execution. The planner is given a predefined sequence in which to order the cubes, and it must generate the appropriate action sequence. The challenge in this task is minimal, focusing on basic object ordering and pick-and-place operations.
- **Middle-level** tasks: These tasks increase in complexity by requiring the planner to stack cubes. Instead of a simple positioning goal (object_at target_position), the task now involves sorting three cubes into a stack. The planner must generate an entire sequence of actions to correctly stack the cubes in the desired order.
- **High-level** tasks: In this level, the task becomes significantly more complex by introducing **color priority**. The planner is required to sort cubes based on color precedence, dynamically adjusting its plan when cubes of the highest priority are inaccessible. For instance, if red cubes are initially blocked by other cubes, the planner must re-prioritize and sort blue or green cubes, returning to red once they become accessible again. This scenario demands continuous task re-planning and effective interaction with a dynamic environment, presenting a significant challenge for classical planners, particularly those based on PDDL

2.1, which may struggle with real-time re-planning and handling of non-static conditions.

Across these task levels, this paper aims to identify the strengths and limitations of each planning approach in handling increasingly complex and dynamic goals. The overview of task setting is shown at Table 4-3. For example, in Fig.4-4. The planner should plan a BT to execute the sorting behavior as sorting stacks (from stack1 to stack9. Panda needs sort all 3 cubes (from top cube to bottom cube) in one stack, then move to next stack, until all stacks are sorted.) As the main goal, The stacks sequence are defined, but the sub goal sequence is the part the planner needs to solve.

Table 4-4 Overview of Objects Sorting Tasks

Objects Sorting	Task Complexity	Tests	Test Description
Low level tasks:	Simple	Test1	Cube order without specific connections
		Test2	Cube order with specific connection(mode 1)
Middle level tasks	Moderate	Test1	s1*,s2,s3,s4,s5,s6,s7,s8,s9
		Test2	s9,s6,s3,s2,s5,s8,s7,s4,s1
		Test3	s1,s4,s7,s8,s9,s6,s3,s2,s5
High level tasks	Complex	Test1	Color priority: Red, Blue, Green
		Test2	Color priority: Blue, Green, Red
		Test3	Color priority: Green, Red, Blue

The available skill set is shown at Table 4-4.

Table 4-5 Skill set in Objects Sorting Scenario

Skill's name	Skilll Description	Properties	
setScene	Add all dynamic objects to moveit2, which helps generate trajectories to skill: move	input	case
		output	None
removeScene	After one workflow finishes, the scene should be removed	input	case
		output	None
move	Manipulator can move end effector from original pose to target pose	input	Pose
		output	None
pick	Manipulator will close the gripper, touch the object, then hold it.	input	object
		output	None
isGripperOpen	The Manipulator will check the state of the gripper.	input	None
		output	None

place	Manipulator will loose the gripper.	input	object
		output	None

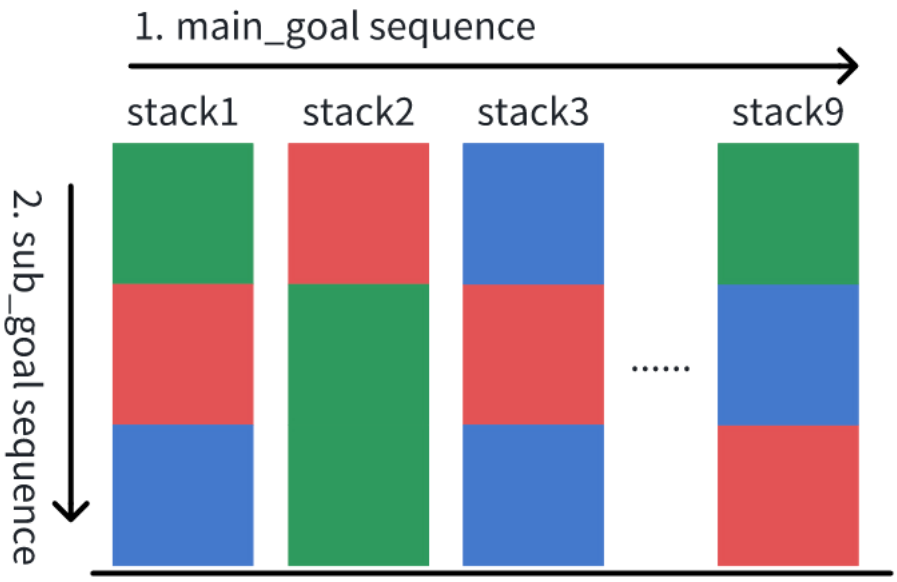


Fig. 4-4 The illustration of one example for middle level task

4.2 Experiment with PDDL-based planner

For the execution of the testing scenarios, PlanSys2 was chosen over other frameworks like SkiROS2 [31, 32]. Several key factors motivated this decision, particularly in the context of our experimental objectives. PlanSys2 directly translates PDDL-generated plans from the POPF planner into executable BTs via the BT_executor, enabling efficient, automated testing without manual intervention. This seamless workflow contrasts with SkiROS2, which requires more configuration for its extended BTs (eBTs) and modular skills, adding unnecessary complexity for our task-oriented study. While SkiROS2 excels in industrial applications with its skill-based control, PlanSys2 is better suited for task planning, especially in experiments where automatic BT generation and execution are essential. PlanSys2 also simplifies the developer workflow by automating the planning-to-execution pipeline; developers only need to define PDDL domain and problem files, and the system handles the rest. In contrast, SkiROS2 demands more setup, making it less practical for our needs. Given our focus on testing the performance of classical and LLM-based planners in automatic BT generation, PlanSys2's robust PDDL integration and simplified execution process made it the optimal choice for our experiments.

The following Fig.4-5 demonstrate the architecture of PlanSys2. In this framework, the input consists of pairs of PDDL files, including domain.pddl and problem.pddl files.

These files are processed by the PlanSys2 system's POPF planner to generate a plan for each application, where the plan represents a sequence of actions. The executor_client then automatically translates these generated plans into BT. Next, the BT_executor executes the BTs by calling each atomic action. The actions can communicate with either a simulation platform, such as Isaac Sim, or a real-world environment to carry out the necessary tasks. During execution, various skills are invoked, such as warehouse skills (e.g., setScene, move, place, pick) and object sorting skills (e.g., askCharging, move, patrol). These operations are logged in real-time by the Log Monitor, which records all execution data into a database. Finally, in the evaluation phase, the results are collected and assessed based on two primary metrics: BT Accuracy Rate (how accurately the tasks are performed) and BT Generation Time (how quickly the BTs are created). BT Generation Time consists of two parts: the time taken to generate the plan and the time taken to translate the plan into a BT. The evaluation process is only been discussed here once, because all the frameworks following share same structure. This entire workflow is highly automated, streamlining the planning-to-execution pipeline for developers and eliminating the need for manual intervention, thereby improving efficiency.

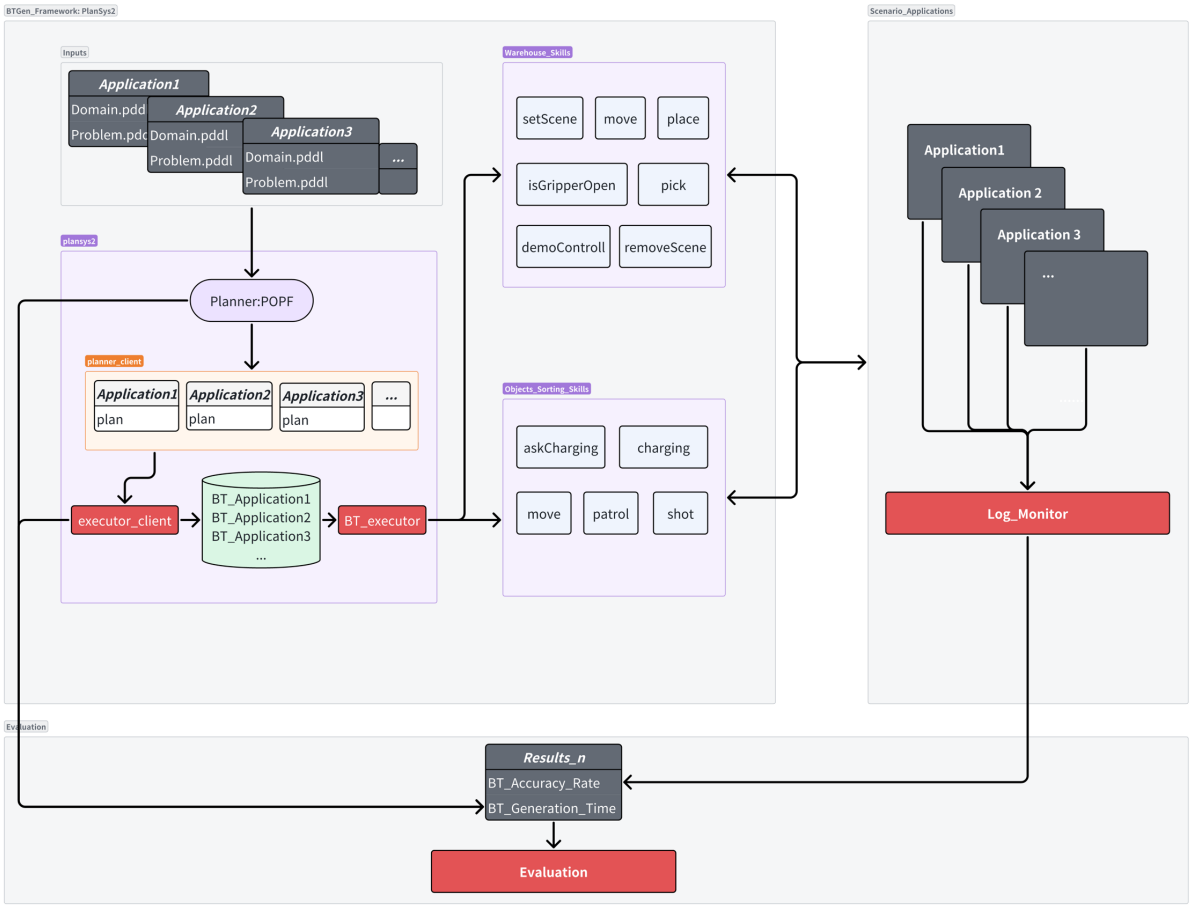


Fig. 4-5 Architecture of PlanSys2 for experiments scenarios

4.3 Experiment with LLMs

Based on the analysis in Section 3.2, numerous frameworks have been proposed by researchers that leverage LLMs as planners for automatic BT generation. Through comparative experiments, it has been observed that models such as GPT-4 and Claude 3.5 exhibit high accuracy in generating BTs from natural language descriptions. Fine-tuning smaller models with specialized datasets has proven effective for translation tasks; however, their performance in reasoning tasks remains limited. Given the higher reasoning capability required for complex planning, GPT-4o was choosen and Claude 3.5 as the primary LLM-based planners for our experiments. Other supplementary techniques, such as graph search algorithms, are beyond the scope of this study and are not included in our evaluation.

In this paper the AutoGen framework was chosen [33], which particularly in multi-agent environments, where a complex task is decomposed into smaller sub-tasks that can be handled by individual agents. This approach was been named as LLM2BT, it ensures that each agent is responsible for a specific task within the overall system, increasing the scalability and efficiency of task execution. The input to the planner consists of a scenario description provided in natural language, detailing the required tasks and conditions. The planner then generates a BT that satisfies the given constraints.

To ensure the correctness and feasibility of the generated BT, the BT_executor performs semantic validation before task execution. The success rate of the generated BT is evaluated through simulations in Isaac Sim, which allows for rigorous testing of the planner’s performance under various task complexities and environmental conditions. The overall architecture is illustrated in the figure below, which outlines the flow from NL input to BT generation, semantic validation, and execution in the simulation environment. This structured approach enables us to systematically assess the performance of LLM-based planners in generating and executing BTs, focusing on both task accuracy and adaptability in dynamic environments.

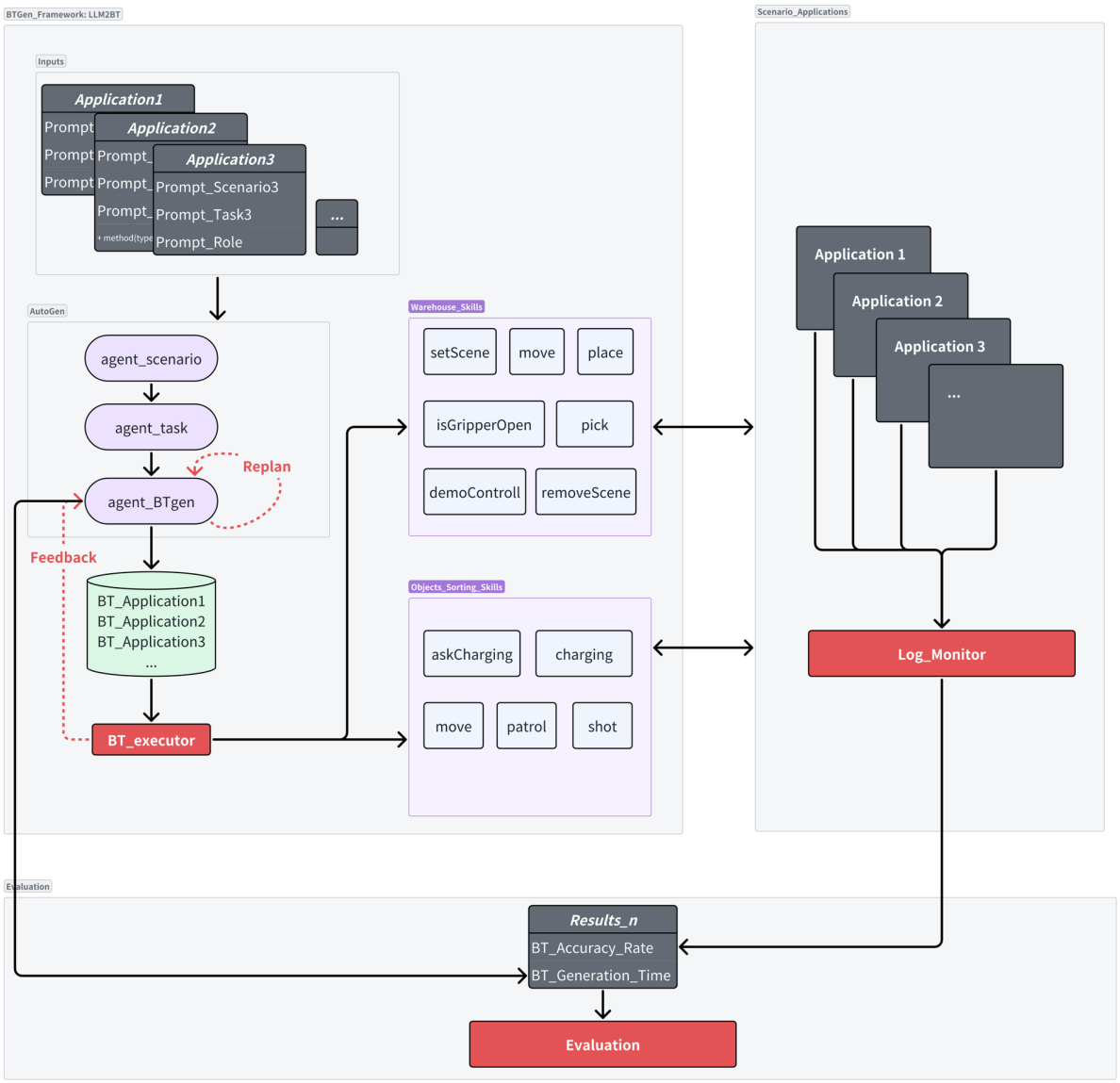


Fig. 4-6 Architecture of LLMBT

The framework depicted in the Fig. 4-6 operates using three agents to handle the various stages of BT generation. Each agent is assigned a specific role in the workflow, from scenario identification to task execution and BT generation.

1. **agent_scenario**: This agent is responsible for retrieving an appropriate scenario description from a predefined database, based on the task at hand. It compiles necessary information such as executable skill sets, skill attributes, and requirements specific to the task. This data forms the foundation for the next step and is passed forward to the subsequent agent.

2. **agent_task**: This agent receives input from either the user or a pre-defined task set. It processes this input and aligns it with the task requirements obtained from the scenario. The output of **agent_scenario** and **agent_task** is combined to generate a cohesive task description that includes the needed actions and skills, which are then passed to the **agent_BTGen** for BT generation.

3.**agent_BTGen**: This agent is guided by a prompt-based role definition (Prompt_Role) and is tasked with generating the actual BT. The role prompt specifies the format of the BT, including details such as whether to use sub-trees and other structural requirements. The textual information from the previous agents is processed by **agent_BTGen**, which translates it into a structured BT ready for execution.

4. **BT_executor**: Before the BT is executed, the **BT_executor** module acts as a validator, checking the generated BT for semantic accuracy. If any errors are identified during this validation process, feedback is provided to **agent_BTGen**. If necessary, a replan process is initiated to correct any issues with the BT, ensuring that the final BT passes the initial validation.

5. **Feedback and Replanning**: The feedback loop ensures that any inaccuracies in the generated BT are addressed by **agent_BTGen.** If the BT does not meet the required standards, **agent_BTGen** revises the tree until it passes the semantic check conducted by **BT_executor**. Once validated, the BT is then ready for execution across the series of predefined test cases (Application1, Application2, etc.).

The framework not only ensures that BT generation is iterative, robust, and capable of adapting to dynamic inputs and changes in task requirements, but also presents several advantages over traditional methods like PlanSys2. One of the key benefits of this approach is the simplification of scenario description. Unlike PlanSys2, which relies heavily on PDDL-based domain and problem modeling, this method allows users to directly model the world and scenario using NL. This greatly reduces the complexity of the input and makes the process more accessible, as users no longer need deep expertise in PDDL to define tasks and environments. As a result, this approach offers both simplicity and adaptability, making it a more versatile solution for automatic BT generation compared to PlanSys2.

4.4 Experiment with Hybrid Planners

This section investigates two types of hybrid planners that combine the strengths of LLMs and classical planners, with distinct goals for each method.

4.4.1 Hybrid_LLM2PDDL

The first hybrid approach uses LLMs primarily for translating high-level natural language task descriptions into problem.pddl files, which are then solved by a classical planner such as POPF [34]. The aim here is to ensure that the generated BTs are accurate and executable, as the classical planner handles the core task planning and constraint resolution. By offloading the reasoning and task-solving to a traditional planner, this method leverages the LLM's ability to simplify task descriptions while relying on proven planning algorithms to execute complex tasks with precision. This approach prioritizes correctness in the generated BTs by ensuring that the final plan is structurally sound and adheres to task constraints.

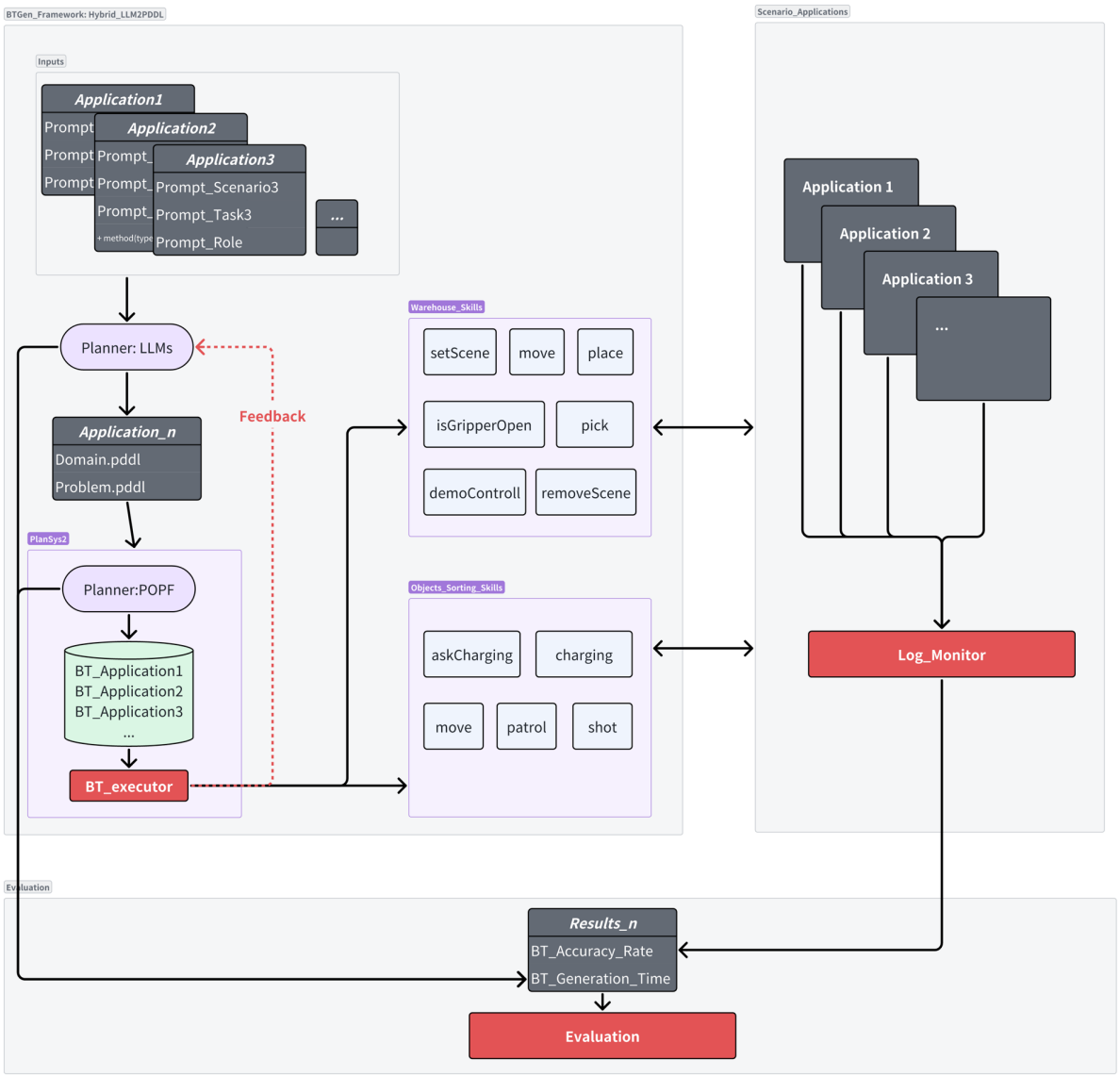


Fig. 4-7 Architecture of BT Generation Framework using LLM with POPF

4.4.2 Hybrid_PDDL2LLM

The second hybrid approach evaluates the LLM's ability to handle task solving independently. In this method, the LLM is responsible for both task planning and execution by directly generating BTs from PDDL inputs without the involvement of a classical planner. This approach tests whether the reasoning capabilities of the LLM can be enhanced by giving it more control over the problem-solving process. The goal is to assess how the quality of the generated BTs is influenced by different input formats (NL vs. PDDL) and to determine if the LLM can handle task dependencies and constraints effectively. The architecture of this framework is shown as Fig.4-8.

Through these two experiments, this paper aim to evaluate (1) the impact of input formats on the accuracy and feasibility of BT generation, and (2) the potential for LLMs to enhance reasoning and adaptivity in task planning when directly responsible for task execution. The first approach prioritizes correctness and reliability by incorporating

classical planners, while the second aims to explore the LLM's full reasoning potential in dynamic and complex environments.

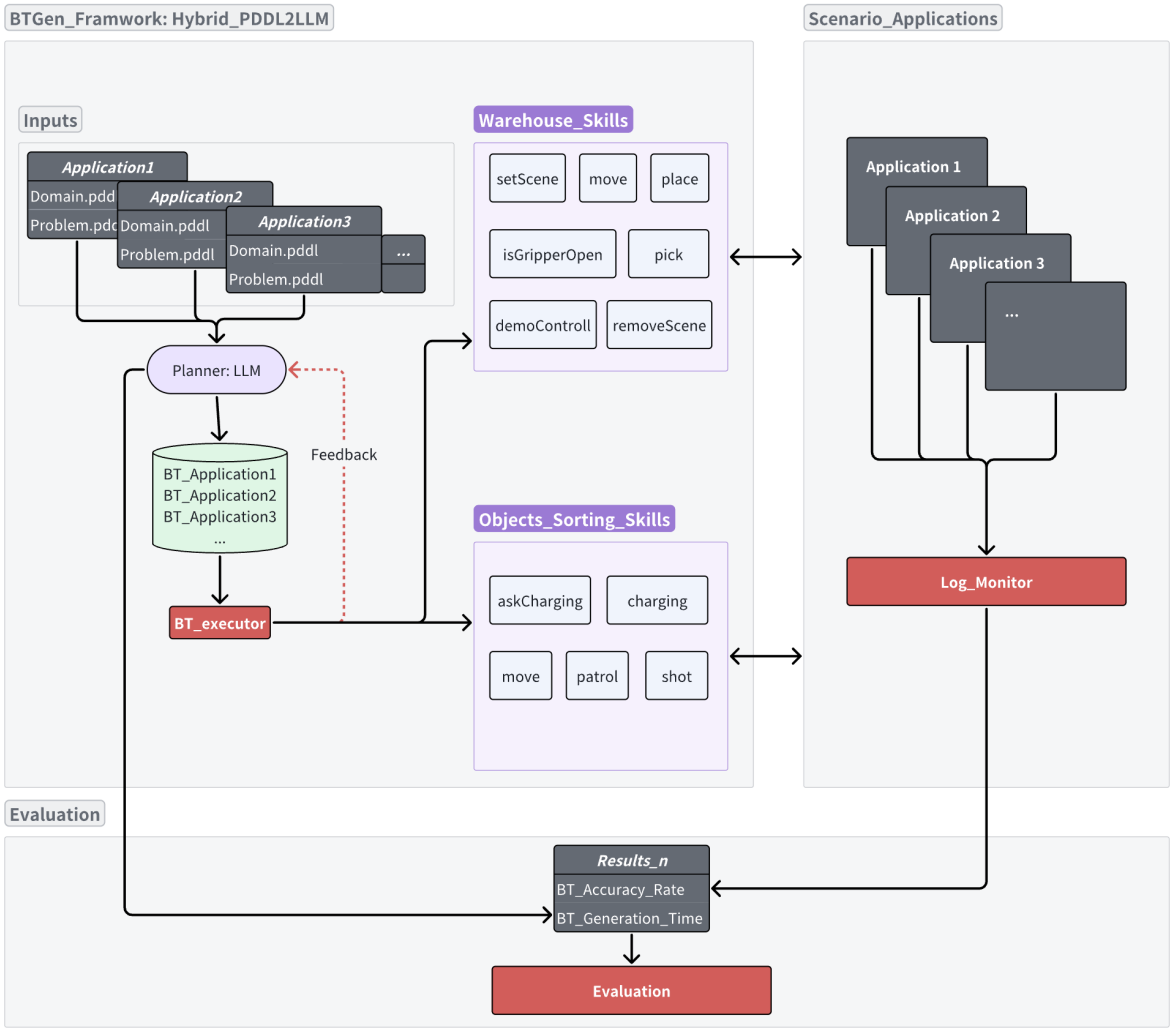


Fig. 4-8 Architecture of BT Generation Framework of second hybrid approach

5 Evaluation

5.1 Overview of Evaluation Criteria

To effectively compare the performance of various planners, two key metrics are selected: BT Accuracy Rate and BT Generation Time. These metrics evaluate the planners' abilities in terms of accuracy and efficiency across tasks of varying complexity [33]. It is important to note that BT Accuracy Rate carries more weight than BT Generation Time, emphasizing the prioritization of accuracy in task execution, particularly in complex or critical scenarios.

BT Accuracy Rate measures the planner's ability to execute tasks accurately and consistently at different levels of complexity. It reflects how well the planner handles goal definitions, environmental challenges, and dynamic conditions. In real-world applications such as autonomous driving or robotics, where accuracy is critical, a high BT Accuracy Rate indicates the planner's robustness in managing intricate and unpredictable scenarios. Planners that excel in this metric demonstrate their ability to consistently generate BTs that lead to correct task execution, even in dynamic or complex environments

BT Generation Time assesses the time taken by the planner to generate a valid BT from the given task description. This metric is particularly crucial for real-time applications where delays in planning can significantly impact task execution and recovery from unexpected changes. Although speed is important, especially in time-sensitive environments, accuracy often takes precedence. A longer BT Generation Time may be acceptable if it ensures correct task execution. However, in dynamic systems such as autonomous vehicles or real-time robotic control, minimizing BT Generation Time is also critical for maintaining operational efficiency and responsiveness.

5.2 Evaluation of PDDL-based Planners

The performance of PDDL-based planners, in this paper PlanSys2 was chosen, is evaluated across two scenarios: Warehouse and Object Sorting. These scenarios encompass a range of task complexities, from simple to high-level, allowing for a comprehensive evaluation of the planners' strengths and limitations. BT Accuracy Rate and BT Generation Time are the key metrics considered for this evaluation.

BT Accuracy Rate:

In the Warehouse scenario, PlanSys2 consistently achieves a 100% BT accuracy rate across all levels of task complexity—simple, moderate, and complex, as long as the PDDL domain and problem definitions are correctly set up(The results are shown in Fig.5-2). This highlights the strength of structured, logic-based planning systems in tasks where the environment and task constraints are clearly defined. For example, in simple tasks where the carter must follow a predefined sequence of waypoints, the success rate remains perfect, as PlanSys2 reliably executes the plan without deviation. Similarly, in mid-level tasks where the waypoints are connected bidirectionally, the planner accurately traverses the expected path. However, PlanSys2's limitations are exposed when the environment contains multiple actions or skills with the same name but different effects. In these cases, PlanSys2 tends to select the action that results in the least state variation, even if it is not the optimal choice. For instance, when multiple move actions (in Fig.5-1) with identical function but differing effects are present, the planner may incorrectly prioritize the action that minimally affects the system state, rather than the one that is required to meet the task objectives. This behavior is particularly problematic in tasks where the planner must select actions based on nuanced conditions, such as differentiating between similar skills that produce different outcomes.

```
(:durative-action variant_move
  :parameters (?r - robot ?wp1 ?wp2 - waypoint)
  :duration ( = ?duration 10)
  :condition (and
    (at start(robot_at ?r ?wp1))
  )
  :effect (and
    (at start(not(robot_at ?r ?wp1)))
    (at end(robot_at ?r ?wp2))
  )
)
```

(a)

```
(:durative-action move
  :parameters (?r - robot ?wp1 ?wp2 - waypoint)
  :duration ( = ?duration 20)
  :condition (and
    (at start(connected ?wp1 ?wp2))
    (at start(robot_at ?r ?wp1))
    (over all(battery_full ?r))
    )
  :effect (and
    (at start(not(robot_at ?r ?wp1)))
    (at end(robot_at ?r ?wp2))
  )
)
```

(b)

Fig. 5-1 Example of multiple move actions (a) variant_move, (b)standard move

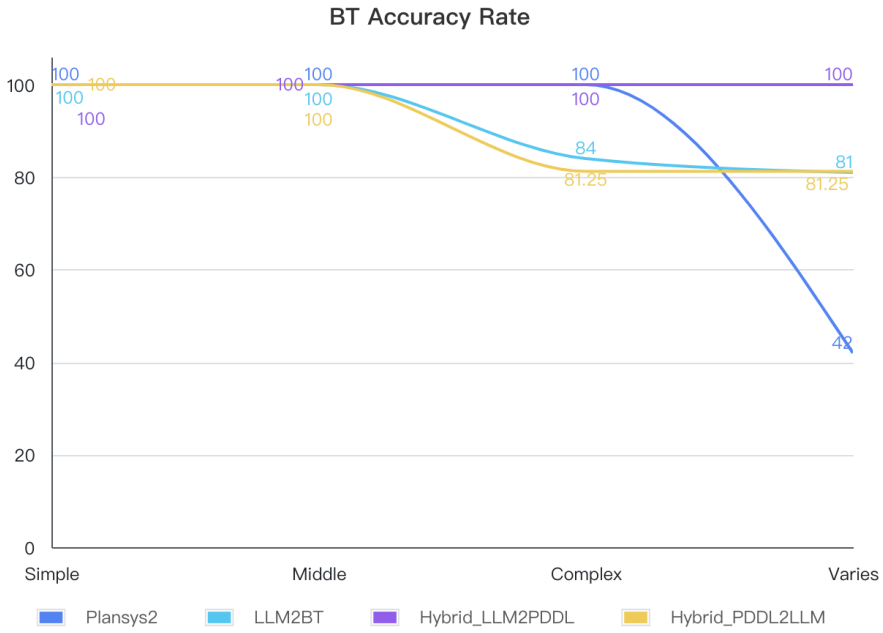


Fig. 5-2 BT Accuracy Rate in warehouse experiment

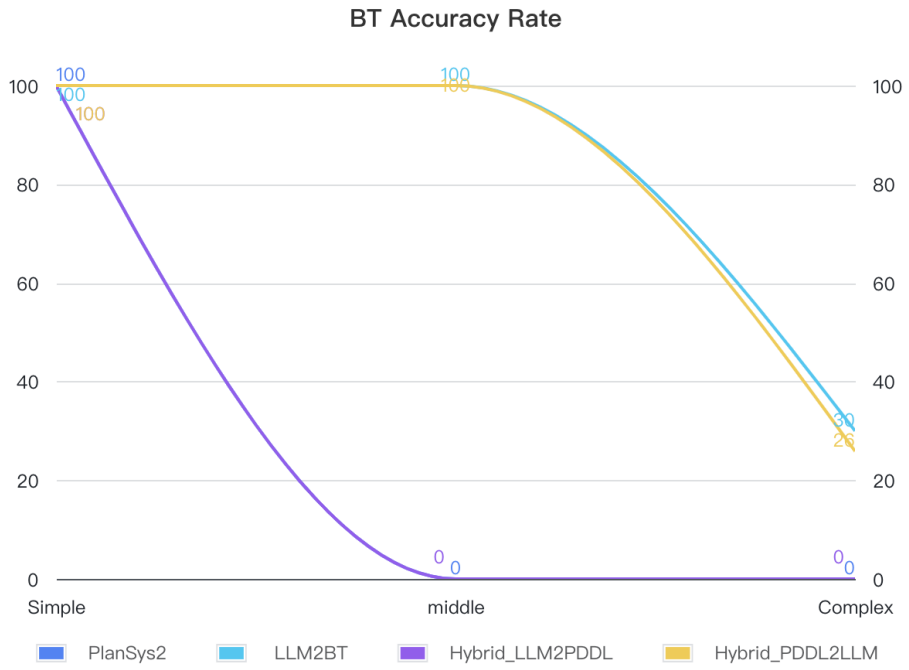


Fig. 5-3 BT Accuracy Rate in Objects Sorting experiment

In the Objects Sorting scenario, which tests the planners' ability to handle more abstract and dynamic goals, PlanSys2 excels at simple tasks where the objectives and action sequences are explicitly defined. For example, when the task involves sorting cubes according to a predefined sequence (giving cube's ID as goal), PlanSys2 successfully completes the task with a 100% success rate, given that the task is fully specified in the problem file(The results are shown in Fig.5-3).

However, as the complexity of the task increases, PlanSys2 struggles to maintain its performance. In mid-level tasks that require the planner to solve for stack configurations without specific goal details (e.g., the task simply specifies that the cubes must be stacked in a certain order), PlanSys2 begins to show limitations. The most significant challenge arises in high-level goal in complex tasks, where the sorting must follow color priority (e.g., sort red cubes first, followed by blue, then green). These tasks require dynamic re-planning and goal adjustment based on the current state of the environment, which traditional PDDL-based planners are not designed to handle. Without additional skills or explicit action definitions, PlanSys2 is unable to solve these high-level tasks, leading to failure in such cases.

BT Generation Time:

In terms of Planning Time, PlanSys2 performs efficiently in simple and mid-level tasks, especially when the problem is well-structured, and the goal is clearly defined. In these cases, the planner can quickly generate a valid plan without significant computational overhead. For example, in the Warehouse scenario, when the task is straightforward (e.g., following a predefined path with minimal waypoints), PlanSys2 generates BTs quickly and accurately, making it highly suitable for static environments where goals and constraints are known in advance.

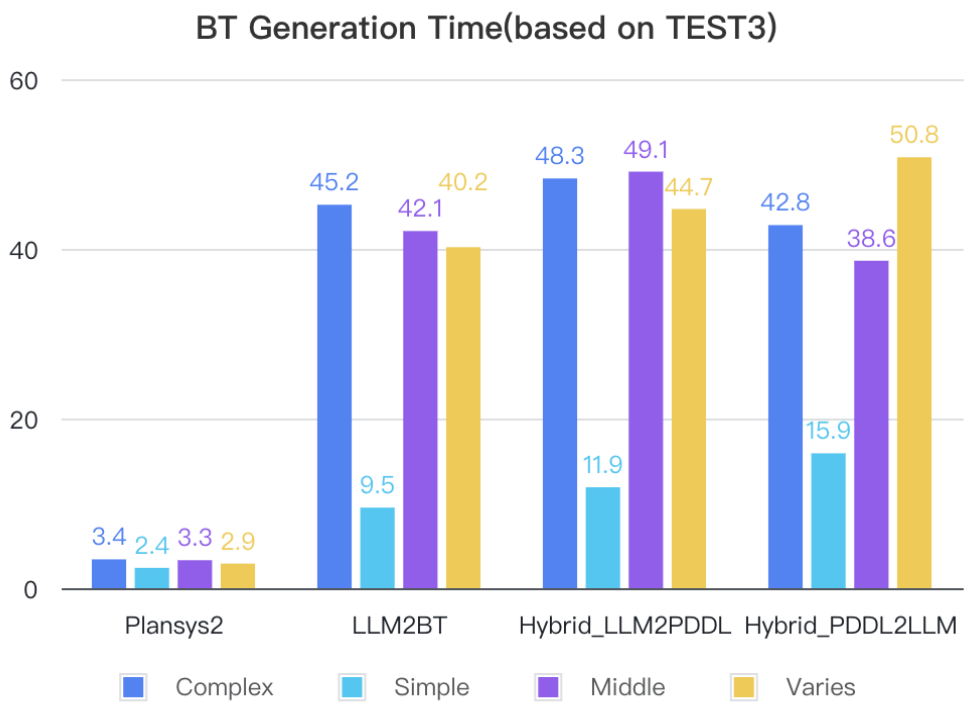


Fig. 5-4 BT Generation Time based on TEST3 in Warehouse experiment

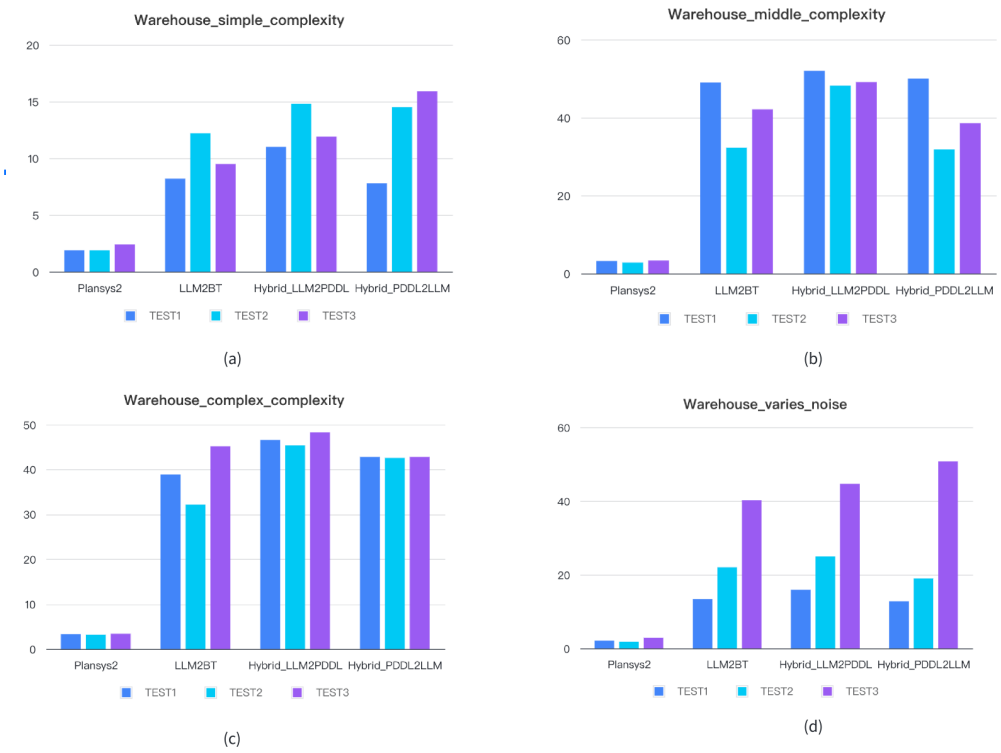


Fig. 5-5 BT Generation Time in Warehouse experiment

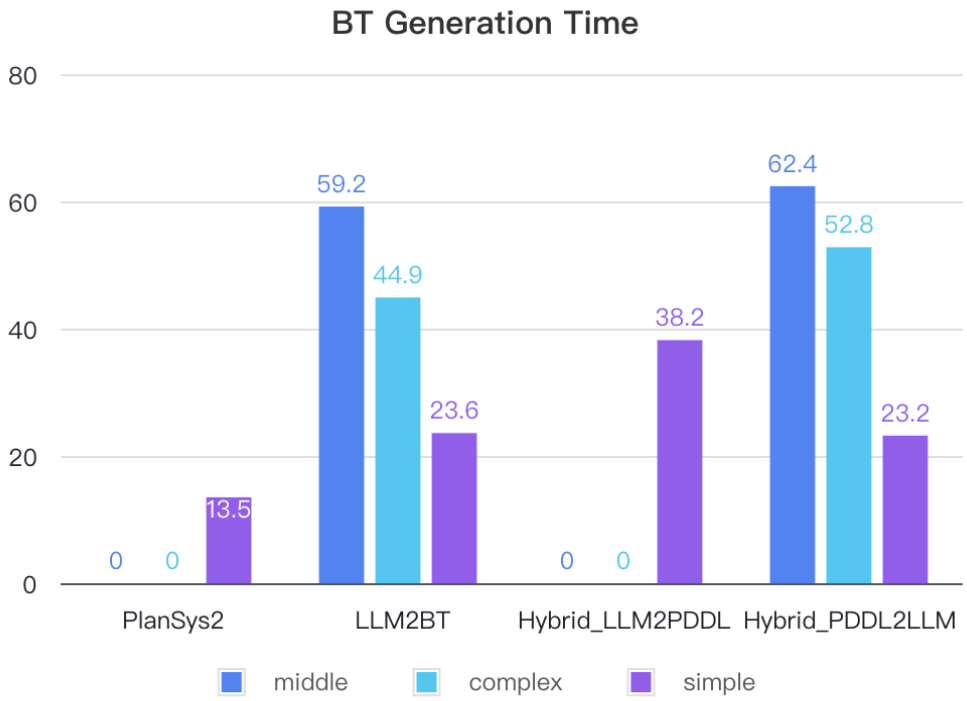


Fig. 5-6 BT Generation Time in objects sorting experiment

However, as task complexity increases, particularly in dynamic environments, BT generation tends to rise due to the need for re-planning. In more complex tasks, such

as those involving dynamic goal re-prioritization or handling multiple action choices, PlanSys2 must repeatedly solve the problem, which significantly increases the planning time. This is especially evident in the Object Sorting scenario during the high-level tasks, where color-priority sorting requires dynamic decision-making based on real-time changes in the environment. PlanSys2's inability to handle these dynamic requirements leads not only to failures in task success but also to increased planning times due to its lack of adaptive capabilities(The results of BT generation Time can be checked in Fig.5-5 and Fig.5-6).

5.3 Evaluation of LLM-based Planners

BT Accuracy Rate:

In the Warehouse scenario, LLM-based planners achieved a 100% BT accuracy rate for both simple and moderate tasks. This indicates that, when the task complexity is limited, LLMs are capable of effectively reasoning through the problem space and generating correct plans. Their ability to translate those plans into BTs for execution by the corresponding system further demonstrates the strength of LLMs in structured tasks. For example, in tasks where the carter robot must follow a predetermined sequence of waypoints, the LLM accurately interprets the instructions and generates the appropriate action sequence.

However, as the task complexity increases, such as in complex tasks, the Task Success Rate for LLM-based planners begins to decline. In high-level tasks, where the carter must optimize routes to minimize the number of waypoints visited, LLMs often fail to identify the optimal solution. For instance(the illustration path is shown at Fig.5-7.), when multiple solution paths are available, the LLM may struggle to consistently select the route that minimizes waypoint traversal, even when the prompt explicitly emphasizes the importance of optimizing the path or provides optimal solutions as examples. This reveals a limitation in the LLM’s ability to perform optimal pathfinding, as its reasoning is not always effective in navigating multiple solution spaces. (In complex task, goal: patrol and shot at wp_16. The path generated by LLM is not minimum cost as the correct path. While patrol and shot action can be correct executed. The initial state of carter is wp_6. The optimum path was wp_6 -> wp_5 -> wp_4 -> wp_1 -> wp_17 ->wp_16. But the path generated from LLM was wp_6 -> wp_7 -> wp_8 -> wp_9 -> wp_11 -> wp_12 -> wp_16. The passing waypoints are more than 4.

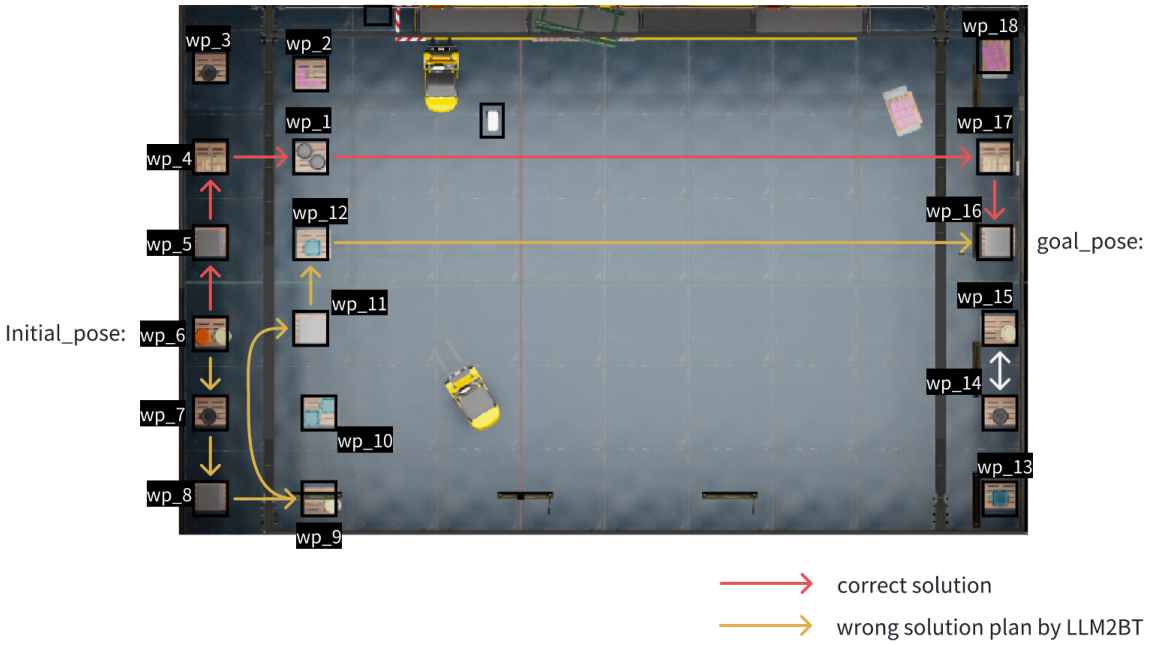


Fig. 5-7 Illustration of plan generated from LLMs

Nevertheless, when variation in skills is introduced, LLMs demonstrate some ability to intelligently avoid incorrect skills by incorporating specific prompt instructions. By augmenting the goal with clear guidance or additional cues, LLMs can circumvent some pitfalls, such as using the wrong action, but they still fall short of producing an entirely accurate plan in all cases. The variability in performance becomes more apparent when more complex paths and decisions need to be made.

In the Object Sorting scenario, the LLM-based planner performed similarly to PDDL-based planners in simple tasks, achieving 100% accuracy. This success can be attributed to the fact that, in simple tasks, the planner only needs to reason through basic state transitions for actions, while the rest of the task involves simple translation and execution. As there are no alternative ways to complete these tasks, LLMs have no difficulty in generating the correct plan.

However, as the tasks become more complex, such as in mid-level and high-level tasks, the LLM begins to struggle. In mid-level tasks, the LLM must reason about the environment, particularly how to stack cubes according to the task goals. This requires a more sophisticated understanding of spatial relationships and sequencing, which poses a challenge for LLMs. For example, in high-level tasks, where cubes must be sorted according to color priority while dealing with dynamic re-planning, the LLM achieves a BT success rate of only 29.6%. The main issue arises from the LLM's difficulty in understanding and tracking the stacking dependencies of the cubes. Specifically, while the LLM can plan for the cubes that are immediately available (such as those on the top of the stack), it fails to account for the inaccessibility of cubes that are blocked by others. This inability to dynamically update the status of cubes based on their availability highlights a key limitation: the need for the LLM to grasp inter-cube

dependencies and continuously update its understanding of the environment, akin to solving a Markov decision process, where the current state of the system depends on its previous state.

BT Generation Time:

In terms of BT Generation Time, LLM-based planners exhibit significantly longer planning times compared to PDDL-based planners like PlanSys2. This is primarily due to the dual processes of reasoning and translation inherent in LLM-based planning. The LLM must first interpret the task in NL, deduce the necessary plan, and then translate that plan into a BT. This adds substantial overhead to the planning process, particularly in more complex tasks where deeper reasoning and environmental understanding are required.

In both the Warehouse and Object Sorting scenarios, the data shows that the overall planning time for LLM-based planners is consistently higher than for PDDL-based planners, especially as task complexity increases. In simple tasks, the additional time spent on natural language interpretation is minimal, but as the tasks grow in complexity (e.g., in high-level tasks that involve multiple decision points or abstract goal definitions), the time required to generate a valid plan increases significantly. For instance, in tasks where multiple waypoints or complex stacking arrangements need to be considered, the LLM takes longer to process the information and produce an executable BT.

5.4 Evaluation of Hybrid Planners

5.4.1 Hybrid_LLM2PDDL

BT Accuracy Rate

In this approach, LLMs are used to translate NL prompts into problem.pddl and domain.pddl files, while the reasoning and task-solving are handled by the PlanSys2 classical planner. Compared to using LLMs solely for task-solving, this hybrid method improves BT success rates, particularly in complex tasks within the Warehouse scenario. The hybrid method maintains 100% accuracy in simple and mid-level tasks, similar to LLM-based planners, but offers notable improvements in complex tasks by leveraging the structured reasoning of the PDDL-based framework.

For tasks involving action variations, this method achieves complete accuracy by excluding irrelevant actions through the use of prompts that emphasize the importance of specific actions. By translating the NL prompt into PDDL files, Hybrid_LLM2PDDL avoids potential pitfalls associated with action selection, ensuring that only the correct and relevant actions are executed. For example, in complex Warehouse tasks, where multiple action variations could lead to incorrect task execution, this hybrid approach

successfully generates valid PDDLs that exclude irrelevant actions and maintain task accuracy.

However, in the Object Sorting scenario, the hybrid method struggles to generate accurate PDDLs for middle and complex task sets. In these cases, the LLM is unable to fully comprehend the spatial and dynamic complexities of cube stacking or color prioritization, leading to failures in generating viable plans. As a result, the BT accuracy in the Object Sorting scenario remains on par with that of PlanSys2, failing to achieve notable improvements in more complex tasks where dynamic re-planning is critical.

BT Generation Time

In terms of BT Generation Time, the hybrid approach in both the Warehouse and Object Sorting scenarios involves the combined time for both LLM-based translation and PlanSys2-based planning. This results in a longer overall planning time compared to using either planner independently, as the translation of NL prompts into PDDL adds an additional step to the process. From the experimental data, it is evident that LLM translation time consistently accounts for the majority of the planning duration, particularly in more complex tasks where the interpretation and translation of the task prompt become more computationally intensive.

5.4.2 Hybrid_PDDL2LLM

BT Accuracy Rate

In this approach, LLMs do not take NL as input; instead, they receive pre-defined PDDL files as input to generate BTs. The purpose of this framework is to determine whether modifying the prompt format—from NL to PDDL—can enhance the LLM's reasoning capabilities and improve task accuracy in complex scenarios.

Experimental results demonstrate that changing the input from NL to PDDL does not significantly improve the LLM's accuracy or reasoning capabilities. In fact, when fed PDDLs, classical planners like PlanSys2 exhibit higher accuracy in solving tasks compared to LLM-based reasoning. This conclusion is drawn from both the Warehouse and Object Sorting scenarios, where the LLM struggles to handle complex tasks based solely on PDDL inputs, leading to suboptimal task solutions.

However, in the Warehouse action variation tests, adding prompt cues in the Prompt_Role section helps the LLM avoid potential action conflicts and select the correct action. This demonstrates the LLM's ability to function effectively when provided with explicit guidance on action selection, though its performance still lags classical planners in complex, high-level tasks.

In the Object Sorting scenario, the LLM fails to generate viable plans for middle and complex task sets due to its inability to design PDDLs tailored to the required task constraints, resulting in zero accuracy for these task sets.

BT Generation Time

In terms of BT Generation Time, the planning time for this hybrid framework closely mirrors that of LLM2BT, with slight increases as task complexity rises. This is because the process of reasoning over PDDL inputs is more challenging for the LLM than interpreting NL prompts, leading to increased computational overhead as task difficulty increases. However, the LLM's performance in terms of time efficiency remains consistent, albeit slower than classical PDDL-based planners like PlanSys2.

In conclusion, Hybrid_LLM2PDDL provides significant improvements in task accuracy for complex tasks in the Warehouse scenario by combining the strengths of NL-based translation and structured PDDL reasoning, while the Hybrid_PDDL2LLM framework struggles to improve task-solving capabilities when relying solely on PDDL inputs. Despite improvements in handling action variations, the hybrid planners remain slower than PDDL-based approaches and are less effective in high-complexity tasks in the Object Sorting scenario.

6 Conclusion and Future Work

6.1 Conclusion

This paper began by introducing the background of the study, highlighting the potential of BTs for behavior control in robotics. With increasing complexity in robotic tasks, the need for automatically generating BTs has garnered attention in both LLM-based and PDDL-based methods. However, a thorough comparative analysis of the performance of these two approaches remains underexplored. Additionally, the limitations of PDDL-based task solvers, particularly in their ability to abstract complex goals and their sensitivity to noise, provided further motivation for this research [35].

To construct our experiments, the Isaac Sim simulation environment from Nvidia was chosen to build two test scenarios. These scenarios were designed to evaluate the performance of various solvers in terms of their ability to handle tasks with varying levels of complexity and optimal solution strategies. The results showed that:

For simple tasks (where the goal contains explicitly defined steps), both PDDL-based and LLM-based planners achieved 100% accuracy. However, when multiple solutions were possible, the PDDL-based planner consistently found the optimal solution, while LLM-based planners struggled to do so, even when the prompt explicitly emphasized finding the best solution.

For complex tasks (where the goal is a high-level task involving dynamic planning), PDDL 2.1 failed to generate a valid plan or corresponding BTs. In contrast, LLM-based planners could generate solutions with around 30% accuracy, using relatively simple modeling. This reveals the potential of LLMs in solving complex tasks with minimal effort, even though their reasoning capabilities are still limited compared to structured methods.

6.2 Future Work

The primary goal of this paper was to design experiments that compare the capabilities of PDDL-based planners and LLM-based planners across tasks with varying levels of complexity. While the study yielded valuable insights, many areas remain for further exploration. For instance, this work only tested PDDL 2.1, but there is a need to explore the capabilities of PDDL 3 and PDDLStream [37] in solving more complex tasks. Additionally, for LLMs as solvers, the current validation only checks the syntactical correctness of the BTs and their execution effectiveness. Further research is needed to develop validators that can assess the functional correctness of BTs based on task execution performance.

Moreover, reinforcement learning-based BT generation techniques represent a promising avenue for future research [36]. Such methods could enhance the decision-

making abilities of BTs in dynamic environments, further pushing the boundaries of what autonomous robots can achieve.

Bibliography

[1] Iovino, Matteo, et al. "A survey of behavior trees in robotics and ai." Robotics and Autonomous Systems 154 (2022): 104096.

[2] Colledanchise, Michele, and Petter Ögren. Behavior trees in robotics and AI: An introduction. CRC Press, 2018.

[3] Behaviortree.dev. (2024). About | BehaviorTree.CPP. [online] Available at: https://www.behaviortree.dev/docs/intro/ [Accessed 14 Sep. 2024].

[4] Paz, Azaria. Introduction to probabilistic automata. Academic Press, 2014.

[5] Iovino, M., Förster, J., Falco, P., Chung, J.J., Siegwart, R. and Smith, C. (2024). Comparison between Behavior Trees and Finite State Machines. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.2405.16137

[6] M. Mateas and A. Stern, “A behavior language for story-based believable agents,” IEEE Intelligent Systems, vol. 17, no. 4, pp. 39–47, Jul. 2002, doi: https://doi.org/10.1109/mis.2002.1024751.

[7] M. Nicolau, D. Perez-Liebana, M. O’Neill and A. Brabazon, "Evolutionary Behavior Tree Approaches for Navigating Platform Games," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 9, no. 3, pp. 227-238, Sept. 2017, doi: 10.1109/TCIAIG.2016.2543661.

[8] G. Florez-Puga, M. A. Gomez-Martin, P. P. Gomez-Martin, B. Diaz-Agudo and P. A. Gonzalez-Calero, "Query-Enabled Behavior Trees," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 1, no. 4, pp. 298-308, Dec. 2009, doi: 10.1109/TCIAIG.2009.2036369

[9] M. Colledanchise and P. Ögren, "How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees," in IEEE Transactions on Robotics, vol. 33, no. 2, pp. 372-389, April 2017, doi: 10.1109/TRO.2016.2633567

[10] J. A. Bagnell et al., "An integrated system for autonomous robotics manipulation," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 2012, pp. 2955-2962, doi: 10.1109/IROS.2012.6385888.

[11] A. Csiszar, M. Hoppe, S.A. Khader, A. Verl, Behavior trees for tasklevel programming of industrial robots, in: T. Schüppstuhl, J. Franke, K. Tracht (Eds.), Tagungsband Des 2. Kongresses Montage Handhabung Industrieroboter, Springer, Berlin, Heidelberg, 2017, pp. 175–186, http: //dx.doi.org/10.1007/978- 3- 662- 54441- 9_18.

[12] M. Colledanchise, D. Almeida and P. Ögren, "Towards Blended Reactive Planning and Acting using Behavior Trees," 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 8839-8845, doi: 10.1109/ICRA.2019.8794128.

[13] S. Macenski, F. Martín, R. White and J. G. Clavero, "The Marathon 2: A Navigation System," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 2020, pp. 2718-2725, doi: 10.1109/IROS45743.2020.9341207.

[14] M. Colledanchise, A. Marzinotto, D. V. Dimarogonas and P. Oegren, "The Advantages of Using Behavior Trees in Mult-Robot Systems," Proceedings of ISR 2016: 47st International Symposium on Robotics, Munich, Germany, 2016, pp. 1-8.

[15] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld. 1998. PDDL - The Planning Domain Definition Language. (08 1998).

[16] M. Mayr, F. Rovida and V. Krueger, "SkiROS2: A Skill-Based Robot Control Platform for ROS," 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, 2023, pp. 6273-6280, doi: 10.1109/IROS55552.2023.10342216.

[17] T. Ribeaud and C. Z. Sprenger, "Behavior Trees based Flexible Task Planner Built on ROS2 Framework," 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA),

Stuttgart, Germany, 2022, pp. 1-4, doi: 10.1109/ETFA52439.2022.9921683.

[18] X. Lu, H. Fang, Y. Bai and R. Zhang, "Hierarchical Extraction, Planning and Behavior Tree Process Control Based on Historical Trajectory," 2023 42nd Chinese Control Conference (CCC), Tianjin, China, 2023, pp. 4549-4555, doi: 10.23919/CCC58697.2023.10240404.

[19] F. Li, X. Wang, B. Li, Y. Wu, Y. Wang, and X. Yi, “A Study on Training and Developing Large Language Models for Behavior Tree Generation,” arXiv.org, Jan. 15, 2024. https://arxiv.org/abs/2401.08089 (accessed Mar. 31, 2024).

[20] 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,” arXiv (Cornell University), vol. 33, pp. 16655–16661, May 2024, doi: https://doi.org/10.1109/icra57147.2024.10610183.

[21] A. Lykov et al., “LLM-MARS: Large Language Model for Behavior Tree Generation and NLP-enhanced Dialogue in Multi-Agent Robot Systems,” arXiv (Cornell University), Dec. 2023, doi: https://doi.org/10.48550/arxiv.2312.09348.

[22] R. A. Izzo, G. Bardaro, and M. Matteucci, “BTGenBot: Behavior Tree Generation for Robotic Tasks with Lightweight LLMs,” arXiv.org, Mar. 19, 2024. https://arxiv.org/abs/2403.12761 (accessed May 17, 2024).

[23] M. Ahn et al., “Do As I Can, Not As I Say: Grounding Language in Robotic Affordances,” Apr. 2022, doi: https://doi.org/10.48550/arxiv.2204.01691.

[24] B. Liu et al., “LLM+P: Empowering Large Language Models with Optimal Planning Proficiency,” arXiv.org, May 04, 2023. https://arxiv.org/abs/2304.11477 (accessed May 31, 2023).

[25] H. H. Zhuo, X. Chen, and R. Pan, “On the Roles of LLMs in Planning: Embedding LLMs into Planning Graphs,” arXiv (Cornell University), Feb. 2024, doi: https://doi.org/10.48550/arxiv.2403.00783.

[26] S. Wang et al., “LLM^3:Large Language Model-based Task and Motion Planning with Motion Failure Reasoning,” arXiv.org, Mar. 18, 2024. https://arxiv.org/abs/2403.11552 (accessed Mar. 20, 2024).

[27] E. Gestrin, M. Kuhlmann, and J. Seipp, “NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions,” arXiv (Cornell University), May 2024, doi: https://doi.org/10.48550/arxiv.2405.04215

[28] L. Zhang, P. Jansen, T. Zhang, P. Clark, C. Callison-Burch, and N. Tandon, “PDDLEGO: Iterative Planning in Textual Environments,” arXiv (Cornell University), May 2024, doi: https://doi.org/10.48550/arxiv.2405.19793.

[29] Z. Li, Nagadastagiri Challapalle, Akshay Krishna Ramanathan, and V. Narayanan, “IMC-Sort: In-Memory Parallel Sorting Architecture using Hybrid Memory Cube,” Sep. 2020, doi: https://doi.org/10.1145/3386263.3407581.

[30] S. Tittel, "Analytical Solution for the Inverse Kinematics Problem of the Franka Emika Panda Seven-DOF Light-Weight Robot Arm," 2021 20th International Conference on Advanced Robotics (ICAR), Ljubljana, Slovenia, 2021, pp. 1042-1047, doi: 10.1109/ICAR53236.2021.9659393.

[31] F. Martín, J. G. Clavero, V. Matellán and F. J. Rodríguez, "PlanSys2: A Planning System Framework for ROS2," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 2021, pp. 9742-9749, doi: 10.1109/IROS51168.2021.9636544.

[32] M. Mayr, F. Rovida and V. Krueger, "SkiROS2: A Skill-Based Robot Control Platform for ROS," 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, 2023, pp. 6273-6280, doi: 10.1109/IROS55552.2023.10342216.

[33] J. Bernhard, K. Esterle, P. Hart and T. Kessler, "BARK: Open Behavior Benchmarking in Multi-Agent Environments," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 2020, pp. 6201-6208, doi: 10.1109/IROS45743.2020.9341222.

[34] Q. Chen and Y.-J. Pan, “An Optimal Task Planning and Agent-aware Allocation Algorithm in Collaborative Tasks Combining with PDDL and

POPF," arXiv (Cornell University), Jul. 2024, doi: https://doi.org/10.48550/arxiv.2407.08534.

[35] S. Kambhampati et al., "LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks," arXiv.org, Jun. 11, 2024. https://arxiv.org/abs/2402.01817 (accessed Aug. 01, 2024).

[36] Y. Fu, L. Qin, and Q. Yin, "A Reinforcement Learning Behavior Tree Framework for Game AI," Jan. 2016, doi: https://doi.org/10.2991/essaeme-16.2016.120.

[37] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling, "PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning," Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, pp. 440–448, Jun. 2020, doi: https://doi.org/10.1609/icaps.v30i1.6739.

Declaration of Compliance

I hereby declare to have written this work independently and to have respected in its preparation the relevant provisions, in particular those corresponding to the copyright protection of external materials. Whenever external materials (such as images, drawings, text passages) are used in this work, I declare that these materials are referenced accordingly (e.g. quote, source) and, whenever necessary, consent from the author to use such materials in my work has been obtained.

Signature: Chuang Yan

Stuttgart, on the <26.09.2024>