PLANTOR: LLM-Aided Knowledge Base Generation for Temporal Planning of Robotic Tasks

Enrico Saccon, Edoardo Lamon, Luigi Palopoli, Marco Roveri

***Abstract*— We introduce PLANTOR, a novel framework that utilises Large Language Models (LLMs) to aid in knowledge-base (KB) management and Prolog-based planning for multi-robot systems. This system employs a two-phase KB generation for reusable reasoning and a three-step planning process addressing temporal dependencies, resource constraints, and parallel execution, ultimately producing Behavior Trees. Experiments in block world and arch-building scenarios demonstrate that LLMs can generate accurate KBs with limited human input, while Prolog ensures formal correctness and explainability, highlighting the potential of LLM integration for advanced, human-understandable robotic planning.**

I. Introduction

With the incredible progress made in the field of artificial intelligence (AI) every day, one of the greatest challenges is how to effectively manage the knowledge about the environment that humans and robots share. In this abstract, we are going to focus on: I) **Knowledge-base generation**: using Large Language Models (LLMs) [1], [2] to create a Knowledge Base (KB) written in Prolog [3] to exploit its inference abilities. II) **Explainable plan generation**: the framework should provide plans that are explainable. III) **Probabilistic extension**: the framework must be able to work under uncertainty. IV) **Learning from experience**: the framework should be able to change KB learning from the execution of the plan. State-of-the-art knowledge management systems heavily rely on ontologies to ground information, especially common knowledge. However, ontologies are usually task-specific, and their creation is complex and time-consuming. Another important challenge for our work comes from the need for the system to adapt to real-time changes in the environment. The objectives of our framework are I) *robustness*: the plan generated using the existing frameworks usually does not consider any of the theories behind planning and in particular probabilistic temporal planning; II) *handling partial knowledge*: at the best of our knowledge, no framework considers uncertainty in the KB; III) *multi-agent collaboration*: the existing frameworks do not directly address multi-agent systems. IV) *input query*: none of the above frameworks considers the possibility of taking as input a series of images depicting agent actions, similar to the instructions to build a piece of furniture.

We acknowledge the support of the MUR PNRR project FAIR - Future AI Research (PE00000013), the project INVERSE (Grant Agreement n. 101136067), the project MUR PRIN 2020 (RIPER - Resilient AI-Based Self-Programming and Strategic Reasoning - CUP E63C22000400001) and the contribution of VRT foundation.

Department of Information Engineering and Computer Science, University of Trento, Trento, Italy. edoardo.lamon@unitn.it

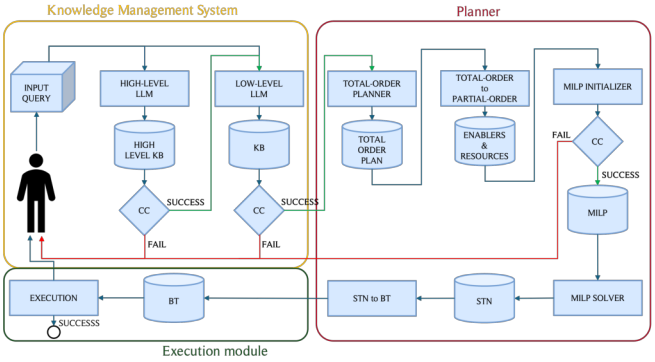

Fig. 1: The structure of the PLANTOR framework.

II. PLANTOR Framework

In Figure 1, the general structure of the framework is shown. The framework is composed of many blocks that can be organised in the following macro blocks: i) **Knowledge management system**: it takes care of creating the KB and maintaining it. ii) **Planner**: it takes the KB and produces a feasible plan. iii) **Execution module**: it produces a BT that can then be executed calling the due APIs.

Knowledge Management System. This module takes a natural language description of the environment, the actions and the task we want to accomplish and queries an LLM to create a Prolog KB. We use a few-shot prompting by passing on a series of examples describing how we want the output to be organized. After calling the APIs and getting back a result, we enter a feedback loop in which we parse the output to check for some common mistakes, both semantic and syntactic. If the output is correct, then we incorporate it into the KB, otherwise we ask the LLM to regenerate the output minding to correct the detected errors. Similarly to the work of [4], we employ two LLMs, one for the generation of a high-level KB in which we incorporate actions and predicates that generically describe the environment and the agents' capabilities, while a second LLM is provided with the APIs that the agents expose, and it is responsible for mapping high-level actions to low-level ones. In Figure 1, a human is depicted, who has a twofold goal: I) provide the initial description, and II) update the KB based on the feedback loops. The idea is to automate the second step by checking for some common mistakes and automatically updating the description without the human-in-the-loop.

Planner We exploit Prolog inference mechanics to find a series of actions in the KB, which let the system reach the final state. Prolog search is depth-first, meaning that it will find solutions that are probably sub-optimal, not to

mention the complexity. To mitigate such a problem, we introduce a cap to the depth of the recursion. The list of actions that Prolog extracts represents the *total order plan*. Then we look for the achievers of the different actions, which means that we look for the causality relationships. This allows us to build a *partial-order plan* where some actions may be executed in parallel. Next, we add the temporal constraints over the durative actions, meaning that an end action must occur within a certain amount of time from the start action . By doing this, we extract a *STN*, the consistency of which can be checked by asserting the absence of negative cycles. At this point, as shown in Figure 1, we set up a Mixed Integer Linear Programming (MILP) problem, which allows for allocating the resources, e.g., the agents, to shrink the makespan of the plan and parallelize possible actions. This is done by considering the enabler of the different actions from the STN. Using a combination of partial-order planning and temporal planning, we can construct plans for multiple agents working together. Moreover, given the Prolog KB, it is possible to use inference to coordinate multiple heterogeneous agents.

Behaviour Trees Once the MILP problem has been resolved, we extract a graph corresponding to an updated STN and proceed with the creation of a Behaviour Tree (BT). We start from the root node and we check the number of children that a node has. If it has more than one, then the children do not have a causal relationship and can be executed in parallel, i.e., we add a parallel control node to the BT. Instead, if the node has only one child, the control node will be sequential. We also check for the number of parents of a node: if there is more than one, then the node needs to wait for all the parents to have terminated before being able to execute. This procedure is repeated recursively until all nodes of the updated STN have been inserted into the BT [5].

III. Example Application and Conclusion

The experimental evaluation of the proposed PLANTOR framework aimed to assess its efficacy in knowledge-base generation and task planning for multi-agent robotic systems [6]. The experimental section comprised evaluations across two distinct robotics scenarios: the classical block world and an arch-building task. These scenarios were used to test both the knowledge base generation capabilities using LLMs and the subsequent planning performance of the framework.

We started evaluating the ability of LLMs (GPT-4 o1 and GPT-4 120K) to generate accurate and consistent Prolog-based knowledge bases from natural language descriptions. The process involved a validation check of the input queries, followed by the generation of high-level and low-level knowledge bases. Few-shot prompting [7] and Chain-of-Thought [8] techniques were utilised to guide the LLMs. The generated KBs were assessed for formal correctness and their ability to enable plan generation. Results indicated that GPT-4o demonstrated a higher accuracy in both query validation and KB generation compared to GPT-4 120K, although some manual corrections were still required. Errors predominantly occurred in the definition of actions and mappings between high-level and low-level operations (see Table I).

	Blocks world					Arch	
	1	2	3	4	5	1	2
120K (HL)	X(2,14)	X(3,17)	X(1,2)	X(1,2)	X(2,2)	X	X
o1 (HL)	✓	✓	✓	✓	✓	✓	X(1,1)
120K (LL)	X	X	X	X	X	X	X
o1 (LL)	✓	X(1,8)	X(2,10)	X(1,8)	✓	X(1,10)	X(4,18)

TABLE I: Results for the generation of the KBs (HL and LL), using the model on the left. A ✓indicates that the model's output was completely correct, while X denotes incorrect output. In cases where a fixable number of errors occurred, the first value inside parentheses represents the number of distinct errors, and the second value indicates the number of changes required to fix the KB.

The performance of the PLANTOR planner was evaluated by measuring the execution times for different stages: high-level total-order (TO) plan generation, low-level TO plan generation, partial-order (PO) and resource extraction, and the MILP-based optimisation and BT generation. Experiments were conducted on the KBs generated and corrected by GPT-4 o1. The results in Table II showed that the high-level TO plan generation was the most computationally intensive phase, attributed to the unguided Prolog search in a potentially large state space. The subsequent steps, including mapping to low-level actions, causal dependency analysis, and MILP optimisation, were comparatively faster. Scalability was also investigated by varying the number of predicates in the knowledge base for a block world example, revealing an expected increase in planning time with growing complexity.

	Blocks world					Arch	
Examples	1	2	3	4	5*	1	2
Plan Steps	2	4	8	6	6	6	6
HL TO Plan	30.81	1640.14	23083.37	81.18	39067.04	2633.67	520.62
LL TO Plan	0.11	0.22	0.37	0.31	0.45	0.27	0.27
PO and Resources Extraction	0.15	0.70	2.83	1.95	1.63	1.45	1.48
Total Prolog	63.81	1731.89	23512.65	102.65	39559.44	2726.42	550.69
MILP – BT	399.63	264.48	292.95	268.50	276.98	262.94	258.29

TABLE II: Execution times (in milliseconds) for planning.

To validate the framework's applicability, a real-world experiment was conducted using two Universal Robots (UR3e and UR5e) to collaboratively build an arch from three blocks. The PLANTOR framework successfully generated a Behavior Tree from a natural language query, which was then executed on the physical robots via ROS2, demonstrating the feasibility of the generated plans in a tangible robotic setting. In conclusion, the experimental evaluation demonstrated the potential of the PLANTOR framework to integrate LLMs for knowledge acquisition and Prolog for formal planning in multi-robot systems. While LLMs showed promise in generating knowledge bases with limited human intervention, the logical planning component ensured correctness and explainability. The optimisation phase further enabled parallel task execution, and the successful real-world deployment highlighted the framework's practical relevance. The study also identified areas for future work, such as enhancing the scalability of the planner and further reducing the need for manual KB validation.

References

[1] M. Shanahan, *Talking about large language models*, 2023. arXiv: 2212.03551 [cs.CL].

[2] W. X. Zhao *et al.*, *A survey of large language models*, 2023. arXiv: 2303.18223 [cs.CL].
[3] T. C. Son *et al.*, "Planning for multiagent using asp-prolog," in *International Workshop on Computational Logic in Multi-Agent Systems*, Springer, 2009, pp. 1–21.
[4] B. Li *et al.*, *Interactive task planning with language models*, 2023. arXiv: 2310.10645 [cs.RO].
[5] J. Zapf *et al.*, "Constructing behaviour trees from pddl temporal plans," Tech. Rep., 2023.
[6] E. Saccon *et al.*, "A temporal planning framework for multi-agent systems via llm-aided knowledge base management," *Robotics and Autonomous Systems*, Submitted, 2025.
[7] X. Huang *et al.*, *Fewer is more: Boosting llm reasoning with reinforced context pruning*, 2024. arXiv: 2312.08901 [cs.CL].
[8] J. Wei *et al.*, "Chain of thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, A. H. Oh *et al.*, Eds., 2022.