Fast and Accurate Task Planning using Neuro-Symbolic Language Models and Multi-level Goal Decomposition

Minseo Kwon, Yaesol Kim and Young J. Kim

Abstract— In robotic task planning, symbolic planners using rule-based representations like PDDL are effective but struggle with long-sequential tasks in complicated planning environments due to exponentially increasing search space. Recently, Large Language Models (LLMs) based on artificial neural networks have emerged as promising alternatives for autonomous robot task planning, offering faster inference and leveraging commonsense knowledge. However, they typically suffer from lower success rates. In this paper, to address the limitations of the current symbolic (slow speed) or LLM-based approaches (low accuracy), we propose a novel neuro-symbolic task planner that decomposes complex tasks into subgoals using LLM and carries out task planning for each subgoal using either symbolic or MCTS-based LLM planners, depending on the subgoal complexity. Generating subgoals helps reduce planning time and improve success rates by narrowing the overall search space and enabling LLMs to focus on smaller, more manageable tasks. Our method significantly reduces planning time while maintaining a competitive success rate, as demonstrated through experiments in different public task planning domains, as well as real-world and simulated robotics environments.

I. INTRODUCTION

In the field of AI planning, symbolic language-based planning using logic formulations such as Planning Domain Definition Language (PDDL) [1] has been effective in generating valid plans across various domains. Such use of symbolic language in robotic task planning is traced back to the Shakey robot project in the early 1970s using STRIPS [2]. However, since the time complexity of these symbolic planners is known to be PSPACE-hard [3], solving long-sequential tasks in domains with extensive search spaces using these symbolic planners is intractable, making their practical application to robot task planning limited. Recently, Large Language Models (LLMs) have shown advantages as autonomous robot task planners due to the short inference time compared to symbolic planners and their ability to leverage commonsense knowledge and generalization capabilities [4].

At a high level, the use of LLMs for task planning is divided into treating LLMs as a policy model (known as *L-Policy*) or as a world model (known as *L-Model*) [4]. L-Policy exploits the commonsense knowledge of LLMs to directly query proper policy for a given state, while L-Model utilizes LLMs as a simulation model of the world to query the state of the world as a result of executing an action or a policy. However, despite their impressive ability, LLMs still struggle with issues like token inefficiency and correction

The authors are with the Department of Computer Science and Engineering at Ewha Womans University in Korea {*minseo.kwon*|*kimy*}@*ewha.ac.kr*, *kimyaesol@gmail.com*.

inefficiency [5], generating hallucinatory action sequences and failing to identify accurate plans as task complexity increases [6]. To address the limitations of current LLM-based task planners, we propose a novel neuro-symbolic task planner that leverages LLMs as both L-Policy and L-Model to solve a long-sequential robotic task. Our planner is significantly faster than symbolic planners and more accurate than LLM-based planners.

An immediate issue in handling long-sequential tasks using LLMs is LLM's token inefficiency since the planning descriptions involve a long and repetitive sequence of world and robotic states as well as a history of policies and their results. To circumvent this issue, we utilize LLMs as L-Model to generate a sequence of subgoals for a long-horizon task, effectively decomposing it into smaller and manageable sub-tasks. This goal decomposition also provides a useful side-effect to reduce the overall search space, yielding an accurate subgoal planner based on LLMs. Indeed, we use the Monte Carlo Tree Search (MCTS) algorithm while using LLMs as L-Policy to accurately solve each subgoal, reducing the correction inefficiency common in LLM-based planners. Furthermore, if the original task is moderately complex, requiring a smaller minimum description length (MDL) [4] to encode the given problem, one can rely on a symbolic planner to solve the subgoals exactly while effectively avoiding the exponential growth of planning time.

Overall, our planning pipeline consists of three major steps:

1) **Planning formulation:** Given a planning goal in natural language description and domain knowledge, our task planner relies on PDDL to encode the problem descriptions. We also obtain the semantic and spatial relationships of target objects in the environment using a multi-modal LLM, translated and encoded in problem PDDL.
2) **Subgoal generation:** We utilize the L-Model to generate a sequence of subgoals by decomposing the given goal.
3) **Task planning:** If the MDL is moderate, we rely on a symbolic planner to solve each subgoal; otherwise, we generate and expand a search tree and use the MCTS algorithm with L-Policy as a roll-out policy to solve the subgoal. This subgoal planning is repeated for each sub-task, and the plans are combined to form the overall plan.

We conducted experiments using a commercial LLM model across three task planning domains while varying

the problem complexity. Compared to the state-of-the-art symbolic task planner, such as the fast-downward planner [3], our approach significantly reduced planning time while maintaining an acceptable success rate. Additionally, we conducted experiments using dual robot manipulators as well as using a robotic simulator to demonstrate the practical utility of our planner.

In summary, the main contributions of our work are:

- We propose a novel neuro-symbolic task planning pipeline for executing complex robotic tasks on physical robots utilizing LLMs as both L-Model and L-Policy.
- L-Model is used to decompose the given goal into multi-level subgoals to reduce the planning time while increasing the planning success rates. L-Policy is exploited to plan subgoals combined with MCTS. For a moderately complex planning task, a symbolic planner is alternatively used to guarantee more accurate planning results.
- Experimentally, we have shown that our new planner achieves an average success rate of $88.2\% \sim 100\%$ while the planning time is only $3.3\times \sim 10.2\times$ slower than the baseline LLM planner approaching zero success rate, depending on the problem complexity.
- We demonstrate the applicability of our new planner on both real and simulated robot task planning scenarios. We also perform an ablation study to demonstrate the effectiveness of our goal decomposition strategy.

The rest of this paper is organized as follows. In Sec. II, we review relevant work to task planning. In Sec. III, we outline the overall pipeline, and in Sec. IV, we explain the algorithms of both the symbolic subgoal planner and the LLM-based subgoal planner. In Sec. V, we present the task planning results and experiments in real and simulation robotics environments, and conclude the paper and discuss future work in Sec. VI.

II. RELATED WORK

A. Symbolic Robot Task Planning

Symbolic or rule-based robot task planning is rooted in classical AI planning using symbolic languages and has been extensively studied for over four decades [2]. We refer the readers to recent surveys on this topic, such as [7]. The current trend in symbolic task planning is to use hierarchical planning to solve a complex problem or to integrate it with geometric motion planning, known as Task and Motion Planning (TAMP) [8]. However, the intrinsically high time complexity of symbolic planning hinders its scalability to adapt to the physical world [2].

B. LLM-based Robot Task Planning

Several recent studies have explored using LLMs for robot task planning, leveraging their understanding of the real world. [9] combines language understanding with action grounding in real-world affordances, enabling robots to execute tasks based on their capabilities. Similarly, [10] introduced a prompting scheme that enables LLM to generate

Python codes composed of robot action primitives, incorporating environmental state feedback. TAMP has also been addressed using LLM by [11] and [12], enabling LLM as spatial relationship generators between environment objects. However, a common limitation of these approaches is low success rates in solving long sequential tasks.

C. Hybird Task Planning

Recently, studies have been conducted on integrating LLMs with symbolic planning methods. [13] and [14] used LLMs as translators between natural languages and symbolic languages like PDDL, converting natural language problem descriptions into PDDL initial states and goals through few-shot prompting. However, these approaches struggle in real-world applications where problems are not presented in natural language. [15] combined LLMs with vision models to generate planning problem specifications based on real-world scenes, using re-prompting to correct specification errors. Several studies have also explored using LLMs to solve PDDL problems. [16] showed that while LLMs can solve some non-trivial PDDL problems, they often fail on more complex tasks, though their outputs can guide heuristic planners. Building on this, [17] proposed a method where LLMs generate Python functions to create PDDL plans with automated debugging. [18] introduced a framework that iteratively refines PDDL plans using validator feedback. While these methods have improved success rates compared to LLM-only methods, they have been tested mostly on small-size problems.

D. Integrating LLMs with Tree Search

Combining tree structures with LLM-generated actions has been explored. [19] samples possible next actions from the current state using an LLM and selects the best action via an LLM-evaluator, repeating this process with DFS or BFS. To address this method's token and runtime inefficiencies, [5] proposes sampling multiple plans at once rather than repeatedly calling the LLM to generate an action tree, and proposes selecting actions from the tree based on observations and histories. [20] integrates MCTS with LLMs by iteratively calculating state transitions in MDPs, and [4] also uses LLMs as both L-Model and L-Policy combined with MCTS to solve large-scale POMDPs.

Our method also uses an LLM to sample multiple plans, and exploit MCTS to find a plan. Still, it differs from [5] in that ours relies on LLM-induced goal decomposition to generate multiple subplans and solve a deterministic problem, unlike [4]. Our MCTS is performed on a fixed tree for a sub-problem, not the entire planning problem.

III. TASK PLANNING PIPELINE

We formulate our task planning problem as a multi-valued planning task (MPT) [3] using a tuple:

$$P \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, s_0, S^\star \rangle, \qquad (1)$$

where $\mathcal{S}$ is a finite set of fully observable states, $\mathcal{A}$ is a finite set of possible actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a deterministic state
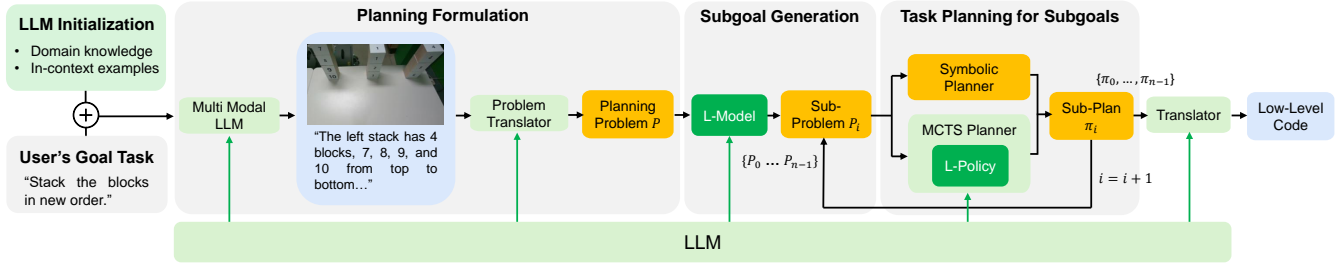
Fig. 1: Neuro-symbolic task planning pipeline. LLM (the green blocks) and symbolic languages (the orange blocks) are used for various steps in the pipeline.

transition function, $s_0 \in \mathcal{S}$ is an initial state, and $S^\star \subset \mathcal{S}$ is a set of goal states. Our planning objective is to find a policy $\pi = \{a_1, \cdots, a_n | \forall a_i \in \mathcal{A}\}$ for P in Eq. 1 to transit from s_0 to $\exists s_n \in S^\star$ in finite steps. Now, we explain each step in our planning pipeline to find a valid π for P and provide a more detailed explanation of the subgoal planner in the next section. An overview of our pipeline is also illustrated in Fig. 1.

A. Planning Formulation

For the robot to fully understand and interact with its environment, both semantics and geometry about the objects in the environment are required. We use a multi-modal LLM such as GPT-4o[1] to simultaneously process both image and text prompts. By providing a color image captured by an RGBD camera along with the prompt, *e.g., "What objects are on the table? Tell me each of their appearance and spatial relationships."*, the LLM can describe the objects on the table, including their spatial relationships, positions, and appearance. Given the scene description, user-provided goal task, the domain PDDL, and an in-context example, the LLM generates a problem PDDL consisting of environment objects $\mathcal{O}$, the initial state s_0, and the goal state $S^\star$ to specify the planning problem P. We utilize one-shot prompting [21] by providing an example of problem PDDL generation to enhance the LLM's responses.

We also employ a 2D open-vocabulary object detection model [22] to estimate the geometric information, specifically the bounding box of the target objects identified by the multi-modal LLM. These bounding boxes are essential for a robot manipulator to motion-plan their grasp poses.

B. Subgoal Generation

Solving a complex task by breaking it down into smaller, easier tasks is often effective [23]. In our case, while LLMs can directly generate relatively accurate plans for smaller tasks, their performance significantly decreases as the task complexity increases and the plan grows beyond a certain size [6]. To address this problem, we leverage the common-sense knowledge of LLMs, *i.e.,* the L-Model, to decompose a given goal into multiple subgoals, simplifying the planning process.

Let us call an ordered set of $\mathcal{G} = \{S_0^\star, S_1^\star, \cdots, S_n^\star\}$ a *sequence of subgoals* or simply *subgoals* of P in Eq. 1 iff

$S_i^\star$ is *reachable* from $S_{i-1}^\star$ for $1 \leq \forall i \leq n$ via a finite number of state transitions from $\exists s_{i-1} \in S_{i-1}^\star$ to $\exists s_i \in S_i^\star$ and $S_0^\star = \{s_0\}, S_n^\star = S^\star$. Our objective is to decompose the original task problem P into n smaller sub-problems P_i's, $0 \leq \forall i \leq n-1$ as

$$P_i \equiv \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, s_i, S_{i+1}^\star \rangle. \qquad (2)$$

We prompt the LLM with domain knowledge and a one-shot planning example along with the explanation of the steps for solving the problem and then ask the LLM to generate $\mathcal{G}$ by observing how the example problem is solved. For instance, in the Blocksworld-new domain, if the blocks are stacked in the order `(on b1 b2)(on b2 b3)`, the reverse order stacking requires each of the three blocks to be unstacked with no objects on each block— `(clear b1)(clear b2)(clear b3)` —to rearrange them appropriately.

C. Task Planning

Once the subgoals $\mathcal{G}$ are generated, we attempt to find a policy $\pi_i \subset \pi$ for each sub-planning problem P_i. The role of the subgoal planner is explained in detail in Sec. IV. By sequentially applying actions from the policy π_i to the initial state s_i, we determine the resulting state s_{i+1}. If $s_{i+1} \in S_{i+1}^\star$, π_i is called a *valid* policy for P_i, and s_{i+1} becomes the initial state for the next sub-problem P_{i+1}. Finally, by aggregating each valid policy $\pi_0, \pi_1, \ldots, \pi_{n-1}$ for each sub-problem, we can obtain the final policy, $\pi = \bigcup_i \pi_i$, which is symbolically represented as a plan PDDL. LLM then translates the plan PDDL into robot-executable low-level code. The robot then automatically executes the corresponding actions by invoking predefined high-level robot action primitives, *e.g.,* such as `pick`, `place` [24].

IV. SUBGOAL PLANNER

We employ two approaches to solve each sub-problem P_i in Eq. 2 using a symbolic planner or MCTS LLM planner depending on the size of P_i. We explain each of these planners.

A. Symbolic LLM Planner

When the size $|P_i|$ of P_i is moderate, it is possible to use a symbolic planner to solve P_i precisely. However, estimating $|P_i|$ is not easy. In theory, one can use a problem measure like MDL [4] to estimate it, but in practice, deriving the MDL for a challenging task is quite hard. Instead, one may estimate
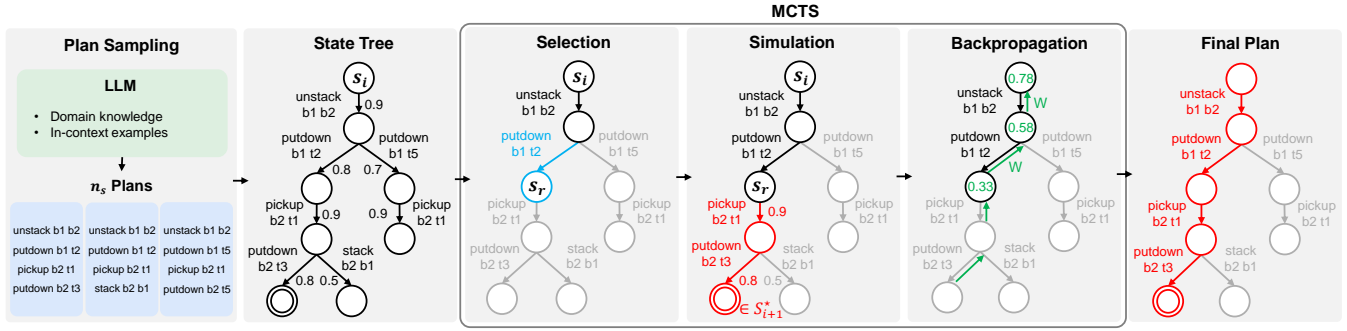
Fig. 2: An overview of the MCTS LLM Planner. First, the L-Policy samples n_s plans for a sub-problem P_i. For instance, the initial state s_i of P_i is (on b1 b2)(on-table b2 t1), etc., and the goal state $S_{i+1}^\star$ should satisfy (clear b1)(clear b2)(clear-table t1). A state tree T_i is then generated, and our MCTS algorithm uses T_i to search for a plan that reaches $S_{i+1}^\star$.

an MDL-like metric for P_i by empirically measuring the planning time spent by running the MCTS LLM planner or a symbolic planner for a sampled P_i. If such an estimate is sufficiently high, we assume that P_i is complex and resort to the MCTS LLM planner in the next section; otherwise, we use a symbolic planner.

To solve P_i symbolically, one can use any symbolic planner, but we opted for the fast downward planner [3], one of the fastest symbolic planners. This guarantees an exact solution to P_i if one exists.

B. MCTS LLM Planner

When $|P_i|$ is high, using a symbolic planner to solve P_i is impractical due to the high combinatorial search space. In this case, we use an MCTS planner combined with the LLM. As illustrated in Fig. 2, our MCTS LLM planner first samples n_s plans for a sub-problem P_i using an LLM (*i.e.,* L-Policy), followed by building a state tree with the LLM-sampled plans, which serves as the reduced search space. The MCTS algorithm then searches this tree to identify an action sequence (*i.e.,* a policy) that leads to a state satisfying the subgoal $S_{i+1}^\star$.

1) Plan Sampling: Given the domain PDDL in Sec. III-A and a couple of in-context planning examples, the LLM generates n_s best plans, $\{\pi_i^1, \pi_i^2, \cdots, \pi_i^{n_s}\}$, to achieve the subgoal in P_i. This is reminiscent of [5], but unlike [5], where they sample the entire problem P, we sample only for a sub-problem P_i, which is presumably more accurate. Also, the *action weight* is computed by summing the token log probabilities corresponding to the action generated by the LLM during plan sampling, reflecting the LLM's confidence when generating the action [25]. Since a token's log probability represents the token's conditional probability given the preceding tokens, the action weight can be viewed as the conditional probability of the current action occurring, given the history of previous actions. This action weight will guide the rollout process in the MCTS.

2) State Tree Generation: We generate a state tree T_i for P_i by coalescing the sampled n_s plans where a node in T_i is a state $s \in \mathcal{S}$ and an edge is an action $a \in \mathcal{A}$ connecting s, s' when $s' = \mathcal{T}(s, a)$. T_i bounds the search space for the MCTS to explore later, ensuring the search is constrained

within the valid actions generated by the LLM. Moreover, we check if the preconditions of a, which is included in the domain PDDL, match the state s, then a is valid and thus added to T_i; otherwise, subsequent actions are removed from T_i. This post-validity test is performed for each action in all the sampled plans.

3) Monte Carlo Tree Search: We search the state tree T_i using the MCTS to find a policy π_i for P_i. Our MCTS is quite different from conventional MCTS like [20] in that: 1) we already expanded the tree T_i that is fixed and constrains the overall search space, so the expansion step is not needed during the search; 2) our rollout policy searches only within T_i. The goal of our MCTS is to estimate the reward for tree nodes and find a valid π_i from the initial state s_i to a goal state $s^\star \in S_{i+1}^\star$, guided by the rewards. The following selection, simulation, and backpropagation processes are repeated to find π_i.

1. Selection: Starting from the root node s_i, we recursively traverse T_i by selecting the child node with the highest UCB1 score [26] from the set of visited nodes until we arrive at a node whose all child nodes are visited for the first time. Then, one of the child nodes is randomly selected, say s_r. If s_r is included in the goal states $s_r \in S_{i+1}^\star$, the MCTS stops immediately and s_r is traced back to s_i, thereby constructing a plan π_i for P_i.

2. Simulation: The simulation step is rolled out and estimates the reward of s_r passed from the selection process. Our rollout policy works as follows: among the possible next nodes (states) that can be transited from the current node (state), the node with the highest action weight (on the red edges in Fig.2), already computed during the plan sampling step, is selected for the next node to visit. This process is repeated on the tree T_i until a leaf node $s^\star$ is reached. If $s^\star \in S_{i+1}^\star$, the returned reward is $\frac{1}{1+d}$ where d is the nodal distance from s_r to $s^\star$; otherwise, zero reward is returned. If $s_r \in S_{i+1}^\star$, the reward is 1.

3. Backpropagation: The reward (the green nodal values in Fig. 2) obtained from the simulation step is backpropagated to update the nodes traversed earlier, incrementing its visit count and adding the reward.
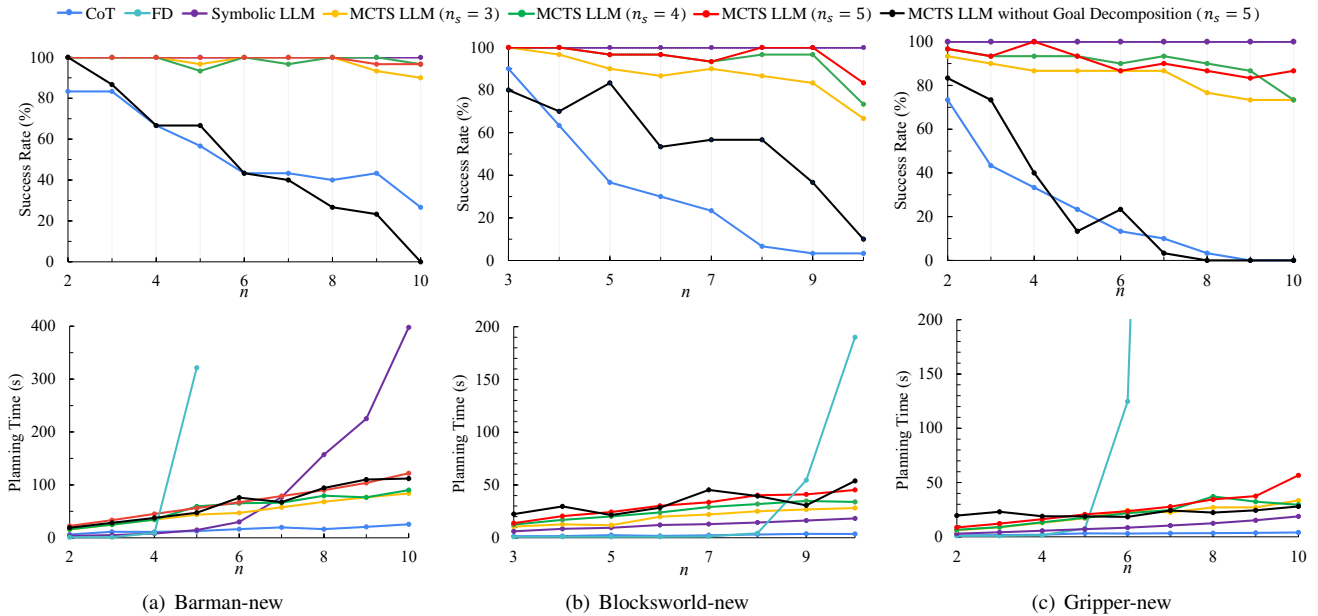
Fig. 3: Success rates (top row) and planning time (bottom row) of CoT, FD, Symbolic LLM, MCTS LLM planners with $3 \leq n_s \leq 5$, and MCTS LLM planner without goal decomposition with $n_s = 5$. The x axis in all the graphs denotes the domain complexity n.

V. EXPERIMENTS

A. Experimental Setup

All task planning experiments were conducted on an Intel Core i9 CPU and NVIDIA RTX 6000 GPUs. We employed GPT-4o for the multi-modal LLM and used the fast downward planner as the symbolic PDDL planner. We conducted PDDL task planning experiments in three well-known IPC domains by modifying their problem complexities [27]: Barman-new, Blocksworld-new, and Gripper-new.

a) Barman-new: This domain involves a dual-arm manipulator making cocktails. The goal is to prepare $2 \leq n \leq 10$ cocktails, and each poured into a different shot glass, similar to examples in [13]. The number of ingredients is three, and the number of shot glasses is $n + 1$.

b) Blocksworld-new: In this domain, a robotic arm stacks $3 \leq n \leq 10$ blocks, randomly divided into one to three stacks arranged on a table. The goal is to rearrange the blocks for each stack. Unlike the original Blocksworld domain, we increase the planning complexity by creating six block placement positions for interim workspace. As a result, the planner must also specify positions for placing the blocks rather than using a single `on-table` predicate as in the original domain.

c) Gripper-new: In the Gripper-new domain, four mobile robots move $2 \leq n \leq 10$ balls to four different rooms from their initial location. We incorporate four multiple robots, making the planning process more complex in a multi-agent scenario, similar to [13]. The positions of the balls and robots in both the initial and goal states are random.

For each n in the above domains, we randomly generated 30 problem PDDL files for the experiments and measured the planning performances.

B. Performance Analysis

The success rate and planning time for each experiment are shown in Figure. 3. The success rate is verified by the PDDL validator VAL [28]. The planning time includes the subgoal generation and planning time. For each task, we compared four methods:

1) **CoT planner**: baseline LLM planner which uses chain-of-thought few-shot prompting [21], [29] with two or three in-context examples to directly generate a plan with LLM [16]–[18].
2) **FD planner**: baseline symbolic planner using the fast downward planner with the "seq-opt-fdss-1" configuration.
3) **Symbolic LLM planner**: our method using the symbolic planner as a subgoal planner, explained in Sec. IV-A.
4) **MCTS LLM planner**: our method using the MCTS planner as a subgoal planner, explained in Sec. IV-B.

Comparisons: The CoT planner is the fastest among the four planners but has the lowest accuracy, with the success rate approaching nearly zero as n increases; on the other hand, the success rate for the FD planner is always 100%, but its planning time increases exponentially as n grows. This indicates that both baseline methods struggle to solve long-sequential problems in highly complex search spaces. In contrast, our symbolic LLM planner consistently achieved a success rate of 100%, and our MCTS planner obtained on average $98.5\%, 92.6\%, 88.2\%$ success rates for Barman-new, Blocksworld-new and Gripper-new domains, respectively. Compared to the CoT Planner, on average, the planning times of our symbolic and MCTS planners are $6.5\times/3.8\times$ (Barman-new), $4.9\times/10.2\times$ (Blocksworld-new), and $3.36\times/8\times$ (Gripper-new) slower, respectively.

Fig. 4: Physical robotic demonstration of our planner on Blocksworld-new domain. Initially, ten blocks, labeled from 1 to 10, are divided into three stacks and placed on the table (leftmost image). The goal is to restack the blocks at the same position in the following order: 10 on 7, 7 on 9, 9 on 8, 1 on 3, 3 on 2, 6 on 5, and 5 on 4 (rightmost image).
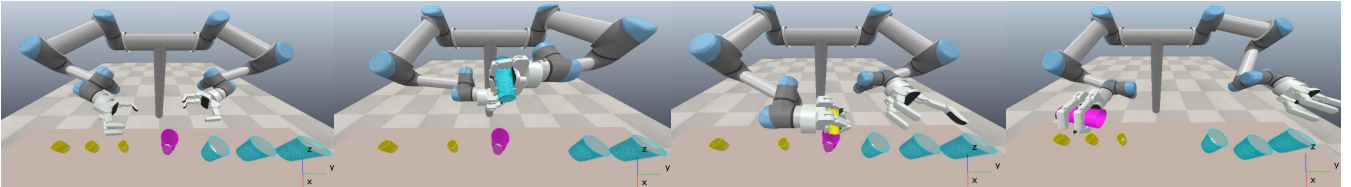

Fig. 5: Simulated robotic demonstration of our planner on Barman-new domain. Initially, three ingredients, three shots, and a shaker are placed on the table (leftmost image). The goal is to make a cocktail and pour it into a shot (rightmost image).

It is generally difficult to compare the performance of our method against other state-of-the-art, LLM-based methods since they use different LLMs or generate non-deterministic results. However, one can estimate comparisons based on the original authors' report. [13] show very low success rates (almost zero) for complex benchmarks like ours, [17] show similar success rates like ours for the single gripper benchmark whereas ours is dual, more complex setup, and [18] show slightly inferior success rates than ours for the Blocksworld when $n \leq 4$, but it is unclear how it would perform for $n > 4$. Thus, even though this comparative study is not purely experimental, one can say that the performance of our methods is substantially better than the state of the art.

Symbolc LLM vs. MCTS LLM: In the Barman-new domain, where the MDL between each subgoal (making each cocktail) is long, and the domain's state space $\mathcal{S}$ is extensive, the planning time for the symbolic LLM Planner increases rapidly as n grows. In contrast, the MCTS LLM planner exhibits an almost linear increase in planning time in terms of n, resulting in faster performance than the symbolic LLM planner. However, in the Blocksworld-new and Gripper-new domains, the planning time for the symbolic LLM planner does not increase as quickly as in the Barman-new domain, and it was faster than the MCTS LLM planner, probably because those domains are less complex than the Barman-new domain, and the MDL between subgoals is moderate.

Sampled Plans: We performed further experiments on the number of sampled plans used by the MCTS LLM planner by varying $3 \leq n_s \leq 5$, and observed a general trend of higher success rates, accompanied by an increase in planning time. As noted in [5], there seems to be a limit to success rate improvement when increasing n_s, as the complexity of the search space has an upper bound.

Ablation Study on Goal Decomposition: We conducted an ablation study on the effectiveness of goal decomposition. We execute our MCTS LLM planner with $n_s = 5$ with and without goal decompositions. As shown in Fig. 3, our planner with goal decomposition achieved a much higher success rate than the one without it, whereas the planner without goal decomposition approached zero success rates for complex problems.

C. Robot Demonstration

We successfully conducted planning experiments with a real robot in the Blocksworld-new domain to demonstrate the practicality of our neuro-symbolic robot task planners. For the real robot demonstration, we used dual UR5e manipulators equipped with Robotiq 3F grippers. An Intel RealSense D455 RGBD camera was employed for visual input, fixed above the table to provide a top-down view. Also, for the Barman-new domain, we conducted experiments in the CoppeliaSim environment [30], which was set up similarly to the real robot setup described above. For both experiments, our task planners are integrated into the MoveIt motion planner [31] in ROS via the translated action primitives. Key robot action primitives, such as `pick` and `place`, were predefined using MoveIt, and the task planning results were converted into code composed of these action primitives using the LLM. Once executed, the corresponding robot actions are carried out accordingly.

VI. CONCLUSION AND FUTURE WORK

This paper proposes a novel task-planning method based on neuro-symbolic language models by decomposing a complicated, long-sequential goal into multi-level subgoals. Our planner performs much faster than the baseline symbolic methods, achieving high accuracy. We would like to pursue a couple of future directions to improve our current method. The criterion for choosing the level of goal decompositions and picking either a symbolic or MCTS planner for the subgoal is empirical. One needs more systematic or thorough studies of the decision problem. We also need more investigation into integrating our tasking planning pipeline to motion planning, *i.e.,* the TAMP.

REFERENCES

[1] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[2] S. M. Lavalle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[4] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[5] M. Hu, Y. Mu, X. Yu, M. Ding, S. Wu, W. Shao, Q. Chen, B. Wang, Y. Qiao, and P. Luo, "Tree-planner: Efficient close-loop task planning with large language models," *arXiv preprint arXiv:2310.08582*, 2023.

[6] K. Valmeekam, S. Sreedharan, M. Marquez, A. Olmo, and S. Kambhampati, "On the planning abilities of large language models (a critical investigation with a proposed benchmark)," *arXiv preprint arXiv:2302.06706*, 2023.

[7] H. Guo, F. Wu, Y. Qin, R. Li, K. Li, and K. Li, "Recent trends in task and motion planning for robotics: A survey," *ACM Computing Surveys*, vol. 55, pp. 1 – 36, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:256630415

[8] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, no. 1, pp. 265–293, 2021.

[9] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[10] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530.

[11] X. Zhang, Y. Zhu, Y. Ding, Y. Zhu, P. Stone, and S. Zhang, "Visually grounded task and motion planning for mobile manipulation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1925–1931.

[12] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, "Task and motion planning with large language models for object rearrangement," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 2086–2092.

[13] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "Llm+ p: Empowering large language models with optimal planning proficiency," *arXiv preprint arXiv:2304.11477*, 2023.

[14] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh, "Translating natural language to planning goals with large-language models," *arXiv preprint arXiv:2302.05128*, 2023.

[15] K. Shirai, C. C. Beltran-Hernandez, M. Hamaya, A. Hashimoto, S. Tanaka, K. Kawaharazuka, K. Tanaka, Y. Ushiku, and S. Mori, "Vision-language interpreter for robot task planning," *arXiv preprint arXiv:2311.00967*, 2023.

[16] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling, "Pddl planning with pretrained large language models," in *NeurIPS 2022 foundation models for decision making workshop*, 2022.

[17] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, and M. Katz, "Generalized planning in pddl domains with pretrained large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 18, 2024, pp. 20 256–20 264.

[18] Z. Zhou, J. Song, K. Yao, Z. Shu, and L. Ma, "Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 2081–2088.

[19] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[20] S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, and Z. Hu, "Reasoning with language model is planning with world model," *arXiv preprint arXiv:2305.14992*, 2023.

[21] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[22] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan *et al.*, "Grounded sam: Assembling open-world models for diverse visual tasks," *arXiv preprint arXiv:2401.14159*, 2024.

[23] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[24] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *IEEE Access*, 2024.

[25] M. Xiong, Z. Hu, X. Lu, Y. Li, J. Fu, J. He, and B. Hooi, "Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms," *arXiv preprint arXiv:2306.13063*, 2023.

[26] P. Auer, "Finite-time analysis of the multiarmed bandit problem," 2002.

[27] J. Seipp, Á. Torralba, and J. Hoffmann, "PDDL generators," https://doi.org/10.5281/zenodo.6382173, 2022.

[28] R. Howey, D. Long, and M. Fox, "Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl," in *16th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, 2004, pp. 294–301.

[29] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[30] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2013, pp. 1321–1326.

[31] S. Chitta, I. Sucan, and S. Cousins, "Moveit!" *IEEE Robotics and Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.