Neuro-Symbolic Task Planning and Replanning using Large Language Models*

Minseo Kwon and Young J. Kim

***Abstract*— In robotic task planning, symbolic planners are robust but struggle with long, complex tasks due to their exponential growth of the search space. By contrast, LLM-based planners reason faster and incorporate commonsense knowledge, but show lower success rates and lack failure recovery. We present a novel neuro-symbolic task planning framework with subgoal decomposition to overcome the drawbacks of symbolic planners (slow speed) and LLM-based methods (low accuracy). It breaks down complex tasks into subgoals using a multimodal LLM, then selects either a symbolic planner or an MCTS-based LLM planner to handle each subgoal according to its complexity. Furthermore, we propose a neuro-symbolic task replanning algorithm for task planning failure recovery. During task planning and low-level code generation, the LLM acts as a multimodal error detector, ensuring the validity of the planning process and triggering replanning when necessary. We demonstrate that both task planning and replanning improve high success rates across diverse PDDL domains, as well as in real and simulated robotics environments. More details are available at http://graphics.ewha.ac.kr/LLMTAMP/.**

I. Introduction

Robotic task planning has traditionally relied on symbolic methods such as PDDL [3], which generate action sequences from structured problem specifications. However, their exhaustive search over large state spaces often leads to computational inefficiency [4]. In contrast, Large Language Model (LLM)-based methods leverage commonsense reasoning for faster inference [5], but tend to suffer from lower success rates and limited failure recovery when faced with complex tasks or real-world uncertainties. LLMs can serve as policy models (*i.e.,* L-Policy) that directly generate action sequences, or as world models (*i.e.,* L-Model) that predict state transitions [5] in task planning. Despite improved adaptability, these methods still struggle with accuracy and robustness in long-horizon planning tasks.

To address these challenges, we introduce a neuro-symbolic task planning framework that leverages a multimodal LLM as an L-Model to divide complex tasks into subgoals. By extracting semantic and spatial information through a multimodal LLM, we encode the task into a PDDL problem, allowing the L-Model to generate subgoals that constrain the overall search space. If the given subgoal is complex, we use the Monte Carlo Tree Search (MCTS) algorithm while using LLMs as L-Policy to solve each subgoal, reducing the correction inefficiency in LLM-based planners. If the given subgoal is moderately complex, requiring a smaller minimum description length (MDL) [5], we use a symbolic planner instead. Finally, the generated plan is translated into low-level Python code, where each high-level action is grounded by selecting continuous parameters for real-world manipulation.

Additionally, we propose a neuro-symbolic task replanning framework that enhances failure recovery by integrating a symbolic planner with a multimodal LLM. When failures occur during planning formulation or low-level code execution, the LLM acts as a syntax and semantic checker grounded in both language and vision, triggering replanning. If the symbolic planner fails to generate a valid plan, we reprompt the LLM with the planner's error message, the scene image and description, and the initial problem PDDL to enable visually grounded correction. For Python execution errors, the exception message is similarly reprompted to generate a revised plan.

Experiments with dual robot manipulators and a robotic simulator confirm that both frameworks increase success rates across diverse manipulation tasks. Moreover, the first framework significantly reduces planning time. For detailed algorithmic descriptions and experimental results, please refer to the original publications [1], [2].

II. Related Works

Recent studies have integrated LLMs with symbolic planning methods. [6] translated natural language problem descriptions into PDDL initial states and goals using few-shot prompting. [7] addressed task and motion planning (TAMP) by translating text-based tasks into STL specifications and correcting syntax and semantic errors via reprompting. However, these approaches do not directly apply to real-world robotic manipulation tasks, where structured language input is unavailable. [8] extended this by combining LLMs with vision models to generate planning problems from real-world scenes, but primarily focused on task-level planning without addressing low-level execution details like action parameter selection.

[9] proposed a hierarchical planning framework leveraging VLM-generated subgoals for a TAMP planner, evaluating performance against a VLM-based action generator regarding task completion and success rates. Our approach not only demonstrates the effectiveness of subgoal decomposition but also integrates a multi-level subgoal generator with both symbolic and MCTS LLM planners, analyzing their complementary strengths based on task complexity by evaluating success rates and planning times.

*This paper is an extended abstract version of the original papers [1], [2].

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

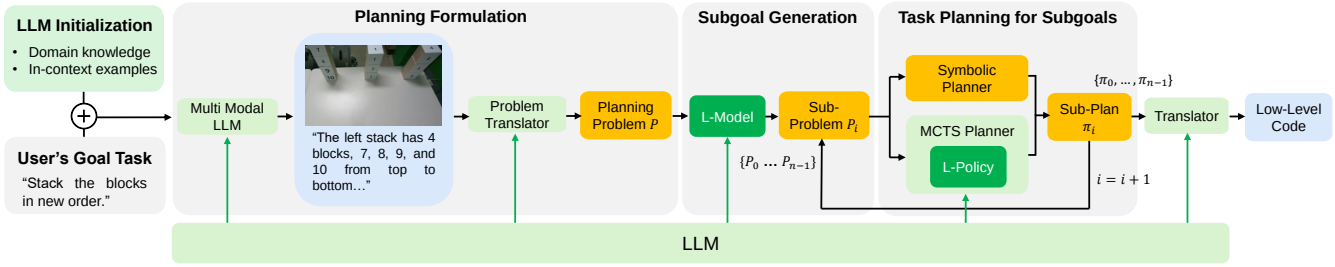


Fig. 1: Neuro-symbolic task planning framework with subgoal decomposition. LLM (the green blocks) and symbolic languages (the orange blocks) are used for various steps in the pipeline.

III. Problem Formulation

We formulate our task planning problem as below in both frameworks:

$$P \equiv \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, s_0, S^\star \rangle, \tag{1}$$

where $\mathcal{S}$ is a finite set of fully observable states, $\mathcal{O}$ is environment objects, $\mathcal{A}$ is a finite set of possible actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a deterministic state 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.

IV. Task Planning with Subgoal Decomposition

This section introduces our first framework: task planning with subgoal decomposition. An overview of the pipeline is in Fig. 1.

A. Planning Formulation

For the robot to interpret the initial scene and encode it into a PDDL problem, we first extract information about object types and their spatial relationships. A multimodal LLM such as GPT-4o is used to process visual and textual prompts simultaneously. Given a color image from the robot's viewpoint and the prompt, "*What objects are on the table? Tell me each of their appearance and spatial relationships.*", the LLM returns a structured scene description capturing object types and their spatial relations. Based on this scene description, together with the user-provided goal task in natural languages, the domain PDDL, and an in-context example, the LLM generates the full planning problem P.

The objects detected in the scene form the object set $\mathcal{O}$ (*e.g.,* `(:objects red-block green-block blue-block)`), which is later used as arguments for PDDL actions and predicates. The spatial relationships (e.g., "*with the blue block on top, followed by the red block, and the green block at the bottom*") are converted into grounded predicates (*e.g.,* `(on red green)(on blue red)`), forming the initial state s_0. Similarly, the user-provided goal is mapped to PDDL goal states (*e.g.,* `(on-table red t6)(on green red)(on blue green)`), defining $S^\star$. The rest of the planning components $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{T}$ are obtained directly from the domain PDDL.

B. Subgoal Generation

We leverage the commonsense reasoning of LLMs, *i.e.,* the L-Model, to decompose a given goal into multiple subgoals, reducing the complexity of task planning. Specifically, we define an ordered set of subgoals $\mathcal{G} = \{S_0^\star, S_1^\star, \cdots, S_n^\star\}$, where each subgoal $S_i^\star$ must be reachable from the previous state $S_{i-1}^\star$ through a finite sequence of transitions, with $S_0^\star = \{s_0\}$ as the initial state and $S_n^\star = S^\star$ as the final goal.

Our objective is to decompose the original problem P into smaller sub-problems P_i, each defined as:

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

To generate $\mathcal{G}$, we prompt the LLM with domain knowledge, a one-shot example, and step-by-step problem-solving explanations. For instance, in the Blocksworld-new domain, if blocks are initially stacked as `(on b1 b2)(on b2 b3)(on-table b3 t1)`, reversing the order requires sequentially unstacking each block to achieve `(clear b1)(clear b2)(clear b3)(clear-table t1)`, enabling proper rearrangement.

C. Task Planning

After generating subgoals, we find a policy $\pi_i \subset \pi$ for each sub-problem P_i. If applying π_i to s_i results in $s_{i+1} \in S_{i+1}^\star$, we proceed to the next sub-problem P_{i+1}, setting s_{i+1} as its initial state. The final policy is $\pi = \bigcup_i \pi_i$, represented as a PDDL plan. For each P_i, we select:

1) Symbolic LLM Planner: If the size $|P_i|$ is moderate, we use the Fast Downward [4] planner. While symbolic planners plan slowly for complex tasks, they guarantee an exact solution to P_i if one exists.

2) MCTS LLM Planner: For large $|P_i|$, using a symbolic planner to solve P_i is impractical due to the high combinatorial search space. In this case, we use an MCTS algorithm combined with the LLM. Our MCTS LLM planner first samples n_s plans for a sub-problem P_i using an LLM (*i.e.,* , L-Policy), then builds 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 that leads to a state satisfying the subgoal $S_{i+1}^\star$.

D. Low-Level Code Generation

To execute the high-level task plan, each PDDL action is translated into low-level Python action primitives

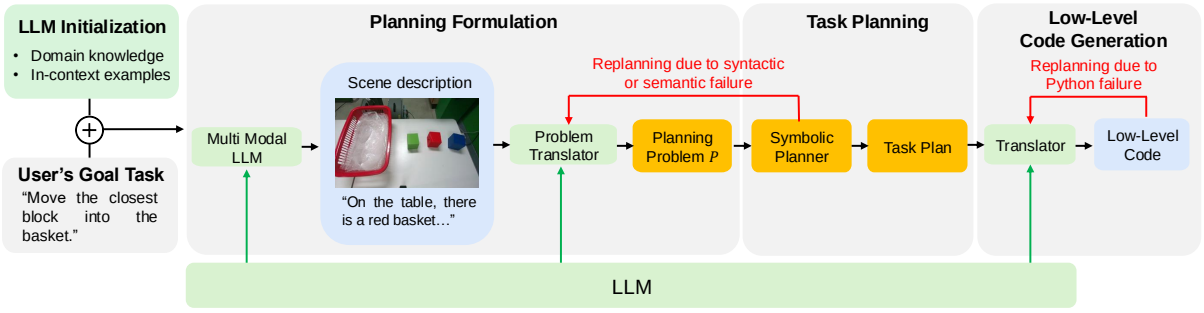


Fig. 2: Neuro-symbolic task replanning framework. The green blocks represent the use of LLM, and the orange blocks represent symbolic planning using symbolic languages. Red arrows show two cases of replanning, where the first arrow indicates syntax/semantic errors in problem formulation, and the second one contains Python exceptions in low-level codes.

(*e.g.,* `pick_up_object`, `place_object`). While symbolic actions only specify discrete parameters like object names, the robot requires continuous parameters (*e.g.,* grasp or place poses). Hence, we compute 2D bounding boxes of target objects using an open-vocabulary object detection model. Then, we expand them to 3D by integrating depth information and applying a pointcloud-based grasp pose estimation algorithm for picking. For placing, we enable the LLM to select the appropriate target from a set of predefined table positions based on the scene context.

V. Task Replanning

We present a task replanning framework to address the lack of failure recovery in LLM-based planning pipelines, as illustrated in Fig. 2. While the planning formulation and low-level code generation stages remain unchanged from the previous framework, this version differs in two key aspects: it only handles symbolic planning without subgoal decomposition and incorporates LLM-based replanning grounded in both language and vision.

We integrate an automatic replanning module that detects failures and reprompts the LLM for correction. Failures typically arise from two sources: errors in PDDL generation (*e.g.,* syntax issues or semantic mismatches between the initial state and goal) and low-level Python execution errors (*e.g.,* invalid function names or missing arguments). Unlike [7], which compares the planner's output trajectory to the original text instruction for semantic checking, we check not only whether the PDDL goal state aligns with the user's goal, but also whether the PDDL initial state matches the actual scene image.

When a planning failure is detected, we invoke a zero-shot CoT prompt that includes the symbolic planner's error message to guide the LLM in refining the problem PDDL. Syntax errors, such as malformed predicates or invalid object names, are corrected by updating the PDDL structure. Semantic errors occur when the initial state does not match the actual scene or when the goal state misaligns with the user's intended goal. This correction process is informed by the symbolic planner's output, the scene image, and the previously generated initial and goal states. For Python-level execution errors, which are less frequent, the LLM is similarly reprompted with the exception message to refine the code.

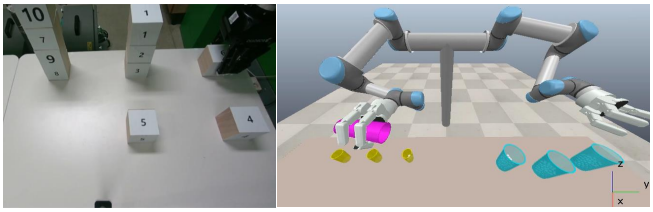


Fig. 3: Physical and simulated robotic demonstration of the first framework, each on Blocksworld-new (left image) and Barman-new domain (right image).

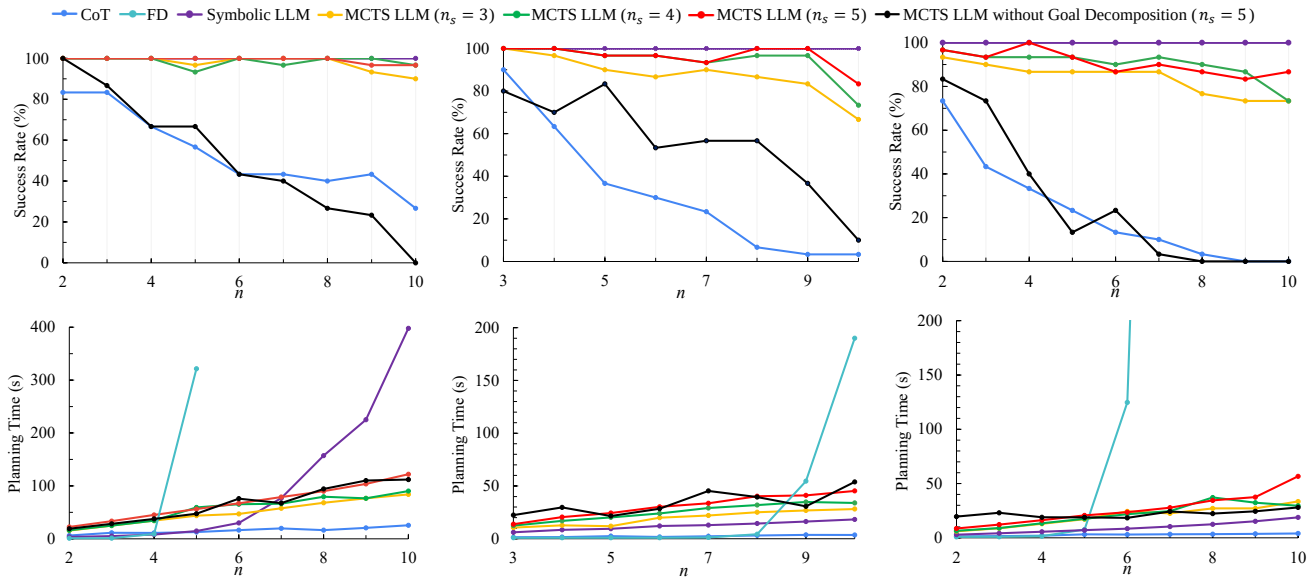
VI. Experiments

A. Subgoal Decomposition

Comparisons: We evaluated our task planning framework across three PDDL domains (Barman-new, Blocksworld-new, Gripper-new) by comparing four methods: (1) CoT planner (baseline LLM planner with few-shot CoT prompting), (2) FD planner (baseline symbolic planner using the Fast Downward planner), (3) Symbolic LLM planner, and (3) MCTS LLM planner ($3 \leq n_s \leq 5$). As in Fig. 4, the CoT planner was the fastest but had near-zero success rates for complex tasks, while the FD planner maintained 100% accuracy but suffered from exponential planning time growth. Our Symbolic LLM planner achieved 100% success, and our MCTS LLM planner reached 98.5%, 92.6%, and 88.2% for the three domains. Both also significantly reduced planning time compared to the FD planner.

Ablation Study on Goal Decomposition: We also conducted an ablation study by running our MCTS LLM planner ($n_s = 5$) with and without goal decomposition. The version with decomposition achieved much higher success rates, while the version without it converged to zero on complex tasks.

Symbolic LLM vs. MCTS LLM: In the Barman-new domain, where each subgoal (making a cocktail) has a long MDL and the state space $\mathcal{S}$ is large, the planning time of the Symbolic LLM planner grows rapidly with increasing n, as in Fig. 4. In contrast, the MCTS LLM planner exhibits nearly linear growth in planning time, achieving better performance in this domain. On the other hand, in Blocksworld-new and Gripper-new, domains where the MDL between subgoals is shorter and the domain's state space $\mathcal{S}$ is smaller, the Symbolic LLM planner performs faster than the MCTS LLM planner, as its planning time scales more gradually.



(a) Barman-new (b) Blocksworld-new (c) Gripper-new

Fig. 4: 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.

Robot Demonstration: We also validated our framework using a dual UR5e manipulator with Robotiq 3F grippers in the Blocksworld-new domain and using the CoppeliaSim [10] in the Barman-new domain as in Fig. 3.

B. Replanning with Syntax and Semantic Checking

We evaluated our replanning framework on block-stacking and block rearrangement tasks derived from the Blocksworld-new domain. With replanning (limited to four attempts), success rates improved to 96.7% for block-stacking from 73.3%, and 100% for block rearrangement from 93.3%. PDDL syntax errors and Python failures were completely eliminated. The framework was also validated in a real-world setting using a dual UR5e manipulator with Robotiq 3F grippers as in Fig. 5.

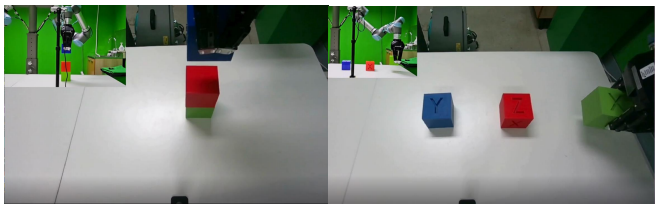


Fig. 5: Physical demonstration of the second framework on block-stacking (left image) and block rearrangement task (right image).

VII. CONCLUSION AND FUTURE WORK

We presented two neuro-symbolic frameworks integrating a symbolic planner and an LLM to address the limitations of symbolic planners (slow speed) and LLM-based planners (low accuracy, lack of failure recovery): task planning with subgoal decomposition and task replanning. Although currently separate, integrating replanning into the subgoal decomposition framework could improve robustness by refining problem formulation and correcting low-level execution errors. Future work will focus on unifying these approaches to enhance overall efficiency and reliability in real-world tasks.

ACKNOWLEDGMENT

This work was supported in part by the ITRC/IITP Program (IITP-2025-RS-2020-II201460), and in part by the NRF (NRF-2022R1A2B5B03001385) in South Korea.

REFERENCES

[1] M. Kwon, Y. Kim, and Y. J. Kim, "Fast and accurate task planning using neuro-symbolic language models and multi-level goal decomposition," in *2025 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2025, accepted.

[2] M. Kwon and Y. J. Kim, "Neuro-symbolic task replanning using large language models," *Journal of Korea Robotics Society*, vol. 20, no. 1, p. 52–60, Feb. 2025. [Online]. Available: http://dx.doi.org/10.7746/jkros.2025.20.1.052

[3] 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.

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

[5] 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.

[6] 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.

[7] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan, "Autotamp: Autoregressive task and motion planning with llms as translators and checkers," in *2024 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2024, pp. 6695–6702.

[8] 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," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 2051–2058.

[9] Z. Yang, C. Garrett, D. Fox, T. Lozano-Pérez, and L. P. Kaelbling, "Guiding long-horizon task and motion planning with vision language models," *arXiv preprint arXiv:2410.02193*, 2024.

[10] 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.