거대 언어 모델을 이용한 뉴로-심볼릭 작업 재계획법

Neuro-Symbolic Task Replanning using Large Language Models

권 민 서[1] · 김 영 준[†]

Minseo Kwon[1], Young J. Kim[†]

Abstract We introduce a novel task replanning algorithm that combines a symbolic task planner with a multimodal Large Language Model (LLM). Our algorithm starts to describe the scene by extracting the semantic and spatial relationships of objects in the environment through a multimodal LLM and an open-vocabulary object detection model. Then, the LLM formulates a planning problem in symbolic form based on the scene description and the user's goal descriptions, which are then processed by the symbolic planner to create task plans. These plans are converted into low-level executable codes for the robot, with the LLM performing syntax and semantic checks to ensure validity and facilitate replanning if necessary. We demonstrate the application of our replanning pipeline using dual UR5e manipulators in various benchmark tasks, including pick-and-place operations, block-stacking, and block rearrangement.

Keywords: Task Planning, Large Language Models, Replanning

1. Introduction

Integrating Large Language Models (LLMs) into robotic task planning presents promising results due to their enhanced commonsense reasoning and generalization abilities. LLMs can interpret unstructured input, providing a detailed understanding of tasks based on natural language and enabling more intuitive interactions between robots and humans.

Previous research leveraging LLMs in robotics has largely focused on extensive prompting strategies to obtain action sequences from models directly. However, due to the frequent generation of hallucinatory outputs, LLMs excel in aiding in solving planning tasks, such as using an LLM output to guide a search-based planner rather than solving tasks on their own [1-3]. Consequently, significant research has focused on integrating LLMs with symbolic planning formulations like Planning Domain Definition Language (PDDL) [4] to improve reliability. Some approaches leverage the commonsense knowledge of LLMs to aid in formulating planning problems [5-8], while others use LLMs to supplement the results of symbolic planners [9].

Furthermore, methods such as iteratively reprompting LLMs with feedback from external evaluators have also been employed to refine outputs, improving both task relevance and execution accuracy [10, 11]. This approach helps resolve one of the main challenges in LLM-based task planning—errors during the planning process, such as syntax or semantic issues in the problem specification [7], which can result in a failed plan. Without replanning, these errors would cause the system to halt, requiring manual intervention. In real-world scenarios, where continuous adaptation by the robot is critical, replanning is essential for automatically detecting and solving these issues.

In this paper, we propose a novel task replanning pipeline that takes advantage of LLMs — their capacity to process natural language queries and commonsense reasoning— while mitigating their limitations by combining a symbolic planner with a multimodal LLM and employing a replanning method when planner failures occurred. Our approach uses a multimodal LLM to analyze semantic relationships among target objects in the scene and a 2D open-vocabulary object detection model to obtain the geometric information. Based on the scene and user-provided goal descriptions, the LLM encodes the planning problem using PDDL formulation. Given this problem PDDL and pre-defined domain PDDL, a symbolic PDDL planner finds a plan for the task, which is later translated into a low-level code such as Python code with action parameter selection to invoke motion planning for robot execution.

Throughout this whole process, replanning occurs when a failure is detected. When a symbolic PDDL planner fails to find a valid plan, LLM validates syntax and semantic errors in the problem PDDL, and we replan the task by reprompting the error messages to LLM as error feedback. If an exception occurs when executing the Python code, we replan the task by reprompting the exception messages to LLM.

We also conducted experiments using a dual robot manipulator setup and an LLM across three task planning domains to demonstrate the effectiveness of our pipeline on diverse and complex robotic tasks.

In summary, the main contributions of our work are:

※ This work was supported in part by the ITRC/IITP program (IITP-2024-RS-2020-II201460) and NRF (2022R1A2B5B03001385) in Korea.

1. M.S. Student, Dept. of Computer Science and Engineering, Ewha Womans University, Seoul, Korea (minseo.kwon@ewha.ac.kr)

† Professor, Corresponding author: Dept. of Computer Science and Engineering, Ewha Womans University, Seoul, Korea (kimy@ewha.ac.kr)

- We propose a novel neuro-symbolic task replanning algorithm for executing various robotics tasks by combining symbolic planner and multimodal LLM and enabling LLM as a syntax and semantic checker.
- We demonstrated the utility of our algorithmic pipeline on real-world robotic tasks. We also highlighted cases where planning failures occurred and showed how the LLM refined these failures.
- We also demonstrated the effectiveness of replanning, showing that success rates increased by up to 31.92% compared to cases without replanning, while observing how the occurrence of each failure cause decreases.

The rest of the paper is structured as follows. In Section 2, we overview relevant work to robot task planning using traditional methods and LLM-based methods. In Section 3, we outline the overall task replanning pipeline in four steps. In Section 4, we present the task planning results in real-world scenarios. Lastly, we conclude the paper and discuss the future work in Section 5.

2. Related Works

2.1 Symbolic Task Planning

Symbolic task planning is based on a classical approach in AI planning, where tasks are represented using symbolic languages, such as PDDL [12]. This approach defines the world as a set of logical states and actions, where each action has preconditions and effects that change the state. Symbolic planners search these states, often using heuristics, to generate a sequence of actions that achieves a given goal.

Additionally, advancements in hierarchical planning and the integration of symbolic task planning with motion planning (TAMP) have further improved performance [13]. A key factor for generalizing TAMP approaches lies between discrete *high-level* task planning and continuous *low-level* motion planning. This involves selecting hybrid action parameters, such as how to grasp or where to place an object, that satisfy constraints and control the system's allowable continuous motions. The downward refinement property assumes that every solution to the high-level task plan has a corresponding low-level motion solution. However, this assumption often does not hold in real-world scenarios, limiting the practical application of TAMP in dynamic, real-world environments [12].

2.2 LLM-based Robot Task Planning

LLM-based robot task planning utilizes LLMs to enhance understanding of the real world and generate action sequences that lead to the goal. For instance, [14] demonstrated that LLMs can effectively combine language understanding with action grounding, allowing robots to execute tasks based on their capabilities and real-world affordances. [15] also employed LLMs, prompting them with environmental state feedback to generate task plans in Python code incorporating robot action primitives. Additionally, research has focused on frameworks that utilize LLMs to generate spatial relationships between objects and provide motion planning feedback, addressing TAMP problems [16, 17]. However, despite these advantages, LLMs often encounter difficulties with large-scale tasks and may yield unreliable outputs in complex scenarios [2].

Integrating LLM with symbolic planners has also been a significant research area. [6] and [18] utilized LLMs as translators between the user's natural language and PDDL, converting natural language problem descriptions into PDDL problems through few-shot prompting [19]. However, these methods treat LLMs as translators and do not deal with situations when LLM-generated problem descriptions are incorrect. It also struggles with real-world scenarios where planning problems are not provided in natural language. [7] addressed TAMP problems by translating natural language problem descriptions into STL problem specifications and correcting their syntax and semantic errors through reprompting. However, this method is not directly applicable to robot manipulation tasks in real-world scenes.

Building on this, [5] combined LLMs with object detection and image captioning models to generate a problem specification, then solved by a symbolic PDDL planner. [9] presented a framework leveraging LLM's commonsense knowledge in household environments to reduce plan length generated by a symbolic PDDL planner. However, these approaches are limited as they primarily address task planning and do not extend to the low-level details required for robot execution, such as action parameter selection.

2.3 Corrective Reprompting with LLM

The concept of corrective reprompting [20] with LLMs has been explored to address planning errors caused by the LLM's hallucinatory outputs. [21] focused on detecting unmet action precondition errors in LLM-generated plans and reprompting the LLM to adjust actions accordingly. [11] incorporated the PDDL plan validator VAL[22] to identify errors in LLM-generated plans and refine them through interactive debugging. Additionally, [7] employed a rule-based STL syntax checker for syntax errors alongside an LLM correction module for semantic errors.

In contrast, our approach utilizes a symbolic planner to ensure plan success without relying on external syntax checkers or verifiers. Moreover, we address both syntax and semantic errors in the planning problem formulation using a zero-shot approach with our automatic LLM reprompting.

3. Task Replanning Pipeline

Our task planning algorithm consists of four parts: planning formulation, task planning with symbolic planner, low-level code generation, and replanning with syntax and semantic checking. The overall pipeline is shown in [Fig. 1]. We will go through each step in the following subsections.

3.1 Planning Formulation

Our objective is to generate a problem PDDL, which can be formulated as below:

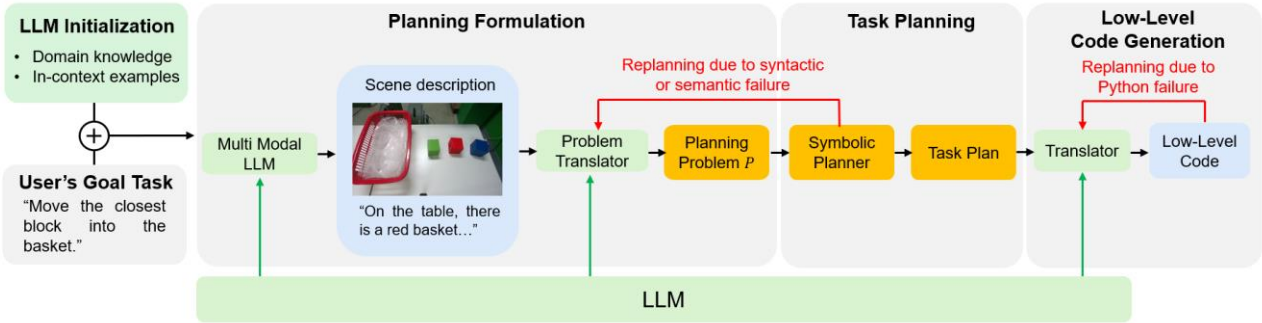

Fig. 1. Our neuro-symbolic task replanning pipeline. The green blocks represent the use of LLM, and the orange blocks represent symbolic planning using symbolic languages. Red arrows show two cases of replanning, where the first arrow indicates syntax/semantic errors in problem formulation, and the second one contains Python exceptions in low-level codes.

$$P \equiv < S, O, A, T, s_0, S^* > \quad (1)$$

where S is a finite set of all possible fully observable states, O is environment objects, A is a finite set of possible actions, $T: S \times A \rightarrow S$ is a deterministic state transition function, $s_0 \in S$ is an initial state, and $S^* \subset S$ is a set of goal states.

To enable the robot to interpret the initial scene and encode it into PDDL, we need to gather information about the types of objects in the environment and their spatial relationships. We use GPT-4-Vision as multimodal LLM to simultaneously understand image and text prompts. By providing a color image from the robot's perspective together with the prompt, "*What objects are on the table? Tell me each of their appearance and spatial relationships.*", the LLM can generate a scene description about the objects on the table, including their relative position, and spatial relationships.

Using this scene description, along with the user-provided goal task, domain PDDL, and a one-shot example, the LLM formulates the planning problem P. Moreover, instead of relying on multiple in-context examples, we use one-shot prompting [19] to improve the LLM's output [3]. Furthermore, LLM can obtain information about PDDL predicates from the domain PDDL.

The objects identified in the scene description become the set O (e.g., *(:objects red_block green_block blue_block)*), later serving as parameters for PDDL actions and predicates. From the spatial relationships between objects along with the positions of the objects (e.g., "*the red block is on top of the blue block and green block is on top of red block*"), we translate it into predicates (e.g., *(on red_block blue_block) (on green_block red_block)*) and this set of predicates form the initial state s_0. Additionally, based on the user-provided goal task, the LLM translates the goal into a PDDL goal description (), forming S^*. The remaining components S, A and T are derived from the domain PDDL.

3.2 Task Planning with Symbolic Planner

Once the planning problem P is formulated, the objective is to use a symbolic task planner to find a policy $\pi = \{a_1, \cdots, a_n | \forall a_i \in A\}$ for P. The generated problem PDDL and domain PDDL are then put into a search-based symbolic planner to produce a plan PDDL. We utilize the Fast Downward planner [23], specifically employing the "*seq-opt-fdss-1*" configuration.

Planning is successful if the Fast Downward planner generates a PDDL plan starting from the initial state s_0 and reaches one of the possible goal states $s_g \in S^*$ within the given search time limit. The planning attempt is considered unsuccessful if the planner fails to generate such a plan within the time limit.

3.3 Low-Level Code Generation

As mentioned in Section 2.1, to execute the high-level plan obtained from task planning, it is necessary to search for hybrid action parameters that satisfy constraints and then call the motion planner [13]. Similarly, in our pipeline, the plan PDDL generated by the symbolic planner is converted into Python code. We prompt the LLM to translate each action a_i from the plan PDDL into predefined Python action primitives, such as '*pick_up_object*' and '*place_object*' [24]. By calling these primitives, the motion planner [25] is invoked, and the robot will execute the path. While the PDDL actions are high-level, including discrete and semantic parameters such as object names, these action primitives are low-level, requiring continuous and real-valued parameters, such as the grasp or place poses for each object. Therefore, the intermediate process of selecting action parameters for Python action primitives is necessary.

In most TAMP systems where the downward refinement property is not satisfied, these parameters must satisfy constraints like collision avoidance and robot joint limits. If any constraints are violated, the system backtracks and tries alternative high-level plans [13]. But for simplicity, we assume that the downward refinement property holds, which means no such constraints exist in our real-world scene.

The process for determining action parameters is as follows. We use a 2D open-vocabulary object detection model to compute the bounding boxes of target objects. Leveraging the scene description provided by the multimodal LLM, the LLM assigns names to objects, and these names, along with the detection model, help generate 2D bounding boxes. These 2D bounding boxes are then expanded into 3D by integrating depth information from segmented object masks captured by an RGB-D camera from the robot's perspective. We employ Grounded-Segment-Anything [26] as the object detection model, which combines Grounding DINO [27] and

Fig. 2. Demonstration of a pick-and-place task using a physical robot and our planning pipeline. Initially, three RGB-colored blocks are placed in a row next to the basket (leftmost image). The goal is to identify the closest block to the basket and drop it into the basket (rightmost image).

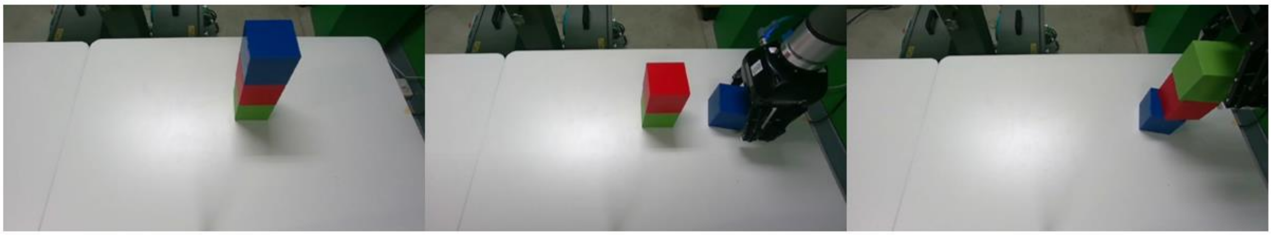

Fig. 3. Demonstration of a block stacking. In this case, the blocks are stacked in blue, red, and green from top to bottom (leftmost image). The goal is to restack the blocks in the order of green, red, and blue from top to bottom on the right side of the table (rightmost image).

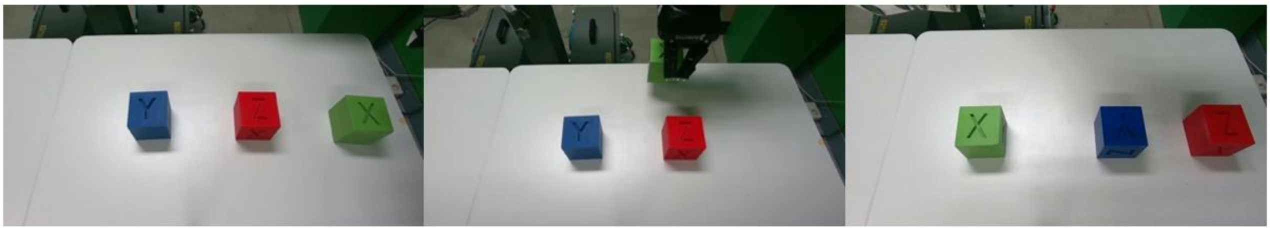

Fig. 4. Demonstration of a block rearrangement. Initially, blue, red, and green blocks with the letters Y, Z, and X are arranged from left to right (leftmost image). The goal is to rearrange the blocks in alphabetical order from left to right: X, Y, and Z (rightmost image).

Segment Anything [28]. Then, a grasp pose selection algorithm [29] is applied within the 3D bounding boxes, and the resulting grasp pose serves as the continuous parameter for the 'pick' action. For the 'place' action, continuous parameters are set based on predefined positions for different table sections.

3.4 Replanning with Syntax and Semantic Checking

Given that the LLM may produce erroneous outputs, we have integrated an automatic replanning module to prevent program interruptions caused by failures. This module detects planning failures and reprompts the LLM to resolve the issues. Failures within the pipeline typically stem from two main sources: errors in problem PDDL generation and low-level code generation.

Errors in problem PDDL generally fall into two categories: syntax and semantic errors. Syntax errors, such as misplaced parentheses or incorrect object names in the set O, cause the planner to terminate during the parsing stage due to invalid PDDL inputs. Semantic errors occur when the initial state s_0 does not match the actual scene or when the goal description S^* misaligns with the user's intended goal. As a result, the planner is unable to find a valid path from state s_0 to any goal state $s_g \in S^*$ leading to a dead end. In both cases, LLM is reprompted using the planner output and a zero-shot Chain of Thought (CoT) [30] prompt, guiding it to analyze the planner error message. If a syntax error is detected, the LLM corrects the problem by fixing the incorrect syntax. When a semantic error is identified, the LLM adjusts s_0 or S^* or both, and refines the planning problem P.

While Python code errors are less frequent than problem PDDL errors, they typically involve simple runtime issues, such as LLM using incorrect action names or neglecting to define action primitive parameters before using it. In such cases, the LLM is similarly reprompted with the Python exception message and a zero-shot CoT prompt to correct the code.

4. Experiments

4.1 Experimental Setup

The experiments used an Intel Core i9 CPU and NVIDIA RTX Ada 6000 GPU. For the physical robot setup, we used UR5e dual manipulators, each equipped with Robotiq 3F grippers and an Intel RealSense D455 RGBD camera mounted above the table for a top-down view. We utilized GPT-4-Turbo [31] as the multimodal LLM and Fast Downward [23] as the symbolic PDDL planner.

All experiments were based on a PDDL domain inspired by the well-known Blocksworld domain. Additionally, the table was divided into six sections, with the position of each section specified in the prompts.

4.2 Robot Demonstration

4.2.1 Pick and Place

The first experiment involved a simple pick-and-place task. As in [Fig. 2], three colored blocks—red, green, and blue—and a basket were placed on the table. Given the user's goal task, "*Move the closest block into the basket*," the system identified the block nearest to the basket and then planned the corresponding action sequence.

4.2.2 Block Stacking

The second experiment focused on stacking blocks in a specified order. As in [Fig. 3], three blocks were initially stacked at the center of the table. The user's goal was to "*Stack the blocks in a new order at the right side of the table. Red at the bottom, green in the middle, and blue at the top.*", the LLM identified the current stack order and position (e.g., near the center, left, right) and generated the problem specification. The robot then unstacked the blocks from top to bottom and stacked the blocks in the specified order.

4.2.3 Block Rearrangement

The third experiment was rearranging blocks in alphabetical order based on letters marked on them. Given the user command "*Rearrange the blocks in alphabetic order from left to right, based on the letters on the top.*", the LLM recognized the letters on each block and their initial positions. Using its alphabet knowledge, the LLM generated a problem description to rearrange the blocks in the order X, Y, and Z from left to right, as shown in [Fig. 4].

This task demonstrated the capability of the LLM to perceive detailed attributes of the blocks, such as letters and abstract positions.

4.3 Experiment Results

Table 1. Success and failure rates (%) without replanning

Domain	Problem failure		Python failure	Success rates
	syntax	semantic		
Stack	6.6	16.7	3.3	73.3
Rearrange	3.3	3.3	0	93.3

Table 2. Success and failure rates (%) with replanning

Domain	Problem failure		Python failure	Success rates
	syntax	semantic		
Stack	0	3.3	0	96.7
Rearrange	0	0	0	100

[Table 1] and [Table 2] summarize the success rates and failure causes for two domains, Block Stacking and Block Rearrangement, comparing cases with and without replanning. For each domain, 30 initial scenes and goal descriptions were randomly generated, and we observed whether problem PDDL generation, task planning, and low-level code execution succeeded.

Without replanning, the Block Stacking domain had a success rate of 73.3%. Among the failures, the most common cause was problem PDDL semantic errors, followed by problem PDDL syntax errors and failures caused by Python exceptions. In the Block Rearrangement domain, the success rate was higher at 93.3%, with problem PDDL semantic and syntax errors each accounting for 3.3% of failures, and no Python failures observed. The lower success rate in Block Stacking is likely due to difficulties the multimodal LLM had in accurately capturing the spatial relationships between stacked blocks for the PDDL initial state.

In both domains, problem PDDL semantic errors were primarily caused by confusion between the *on-table* predicate (indicating a block on the table) and the *on* predicate (representing the relationship between blocks). Problem PDDL syntax errors were typically due to incorrect domain names or improper PDDL formatting. Python failures arose when the pose variable was used in the action primitive without being retrieved by the *get_grasp_pose* function first.

Limiting the number of replanning attempts to four, we observed a significant increase in success rates, approaching almost 100% in both domains. Notably, the rate of problem PDDL syntax errors and Python failures dropped to zero with replanning, demonstrating the effectiveness of our LLM-based replanning method in ensuring plan correctness.

5. Conclusion and Future Work

In this paper, we proposed a neuro-symbolic task replanning pipeline that integrates multimodal LLMs and symbolic planners to address challenges in robot task planning. By leveraging LLMs' commonsense and reasoning abilities, our system generates problem specifications, uses a symbolic planner to find a plan, and converts it into low-level code with action parameter selection. We introduced an automatic replanning module to resolve failures during planning and demonstrated the system’s effectiveness in real-world tasks with dual robot manipulators, showing improved success rates with replanning.

While our approach assumes the downward refinement property for simple scenarios, it does not fully account for real-world complexities. Future work will focus on developing a complete TAMP algorithm that handles cases where downward refinement does not hold, addressing these real-world constraints.

References

[1] S. Kambhampati, K. Valmeekam, L. Guan, K. Stechly, M. Verma, S. Bhambri, L. Saldyt, and A. Murthy, "LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks," *arXiv preprint arXiv:2402.01817,* 2024, DOI: 10.48550/arXiv.2402.01817.

[2] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati, "On the planning abilities of large language models-a critical investigation," in *Advances in Neural Information Processing Systems*, pp. 75993-76005, 2023.

[3] 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*, New Orleans, USA, 2022.

[4] 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, December 2003, DOI: 10.1613/jair.1129.

[5] 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)*, Yokohama, Japan, pp. 2051-2058, 2024, DOI: 10.1109/ICRA57147.2024.10611112.

[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, DOI: 10.48550/arXiv.2304.11477.

[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)*, Yokohama, Japan, pp. 6695-6702, 2024, DOI: 10.1109/ICRA57147.2024.10611163.

[8] M. S. Sakib and Y. Sun, "Consolidating Trees of Robotic Plans Generated Using Large Language Models to Improve Reliability," *arXiv preprint arXiv:2401.07868,* 2024, DOI: 10.48550/arXiv.2401.07868.

[9] R. Arora, S. Singh, K. Swaminathan, A. Datta, S. Banerjee, B. Bhowmick, K. M. Jatavallabhula, M. Sridharan, and M. Krishna, "Anticipate & Act: Integrating LLMs and Classical Planning for Efficient Task Execution in Household Environments," in *International Conference on Robotics and Automation*, Yokohama, Japan, 2024, DOI: 10.1109/ICRA57147.2024.10611164.

[10] 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)*, Yokohama, Japan, pp. 2081-2088, 2024, DOI: 10.1109/ICRA57147.2024.10610065.

[11] 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*, Vancouver, Canada, pp. 20256-20264, 2024, DOI: 10.1609/aaai.v38i18.30006.

[12] S. M. LaValle, *Planning algorithms*, 1 ed. Cambridge university press, 2006.

[13] 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, May 2021, DOI: 10.1146/annurev-control-091420-084139.

[14] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, and K. Hausman, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691,* 2022, DOI: 10.48550/arXiv.2204.01691.

[15] 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)*, London, United Kingdom, pp. 11523-11530, 2023, DOI: 10.1109/ICRA48891.2023.10161317.

[16] 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)*, Detroit, USA, pp. 2086-2092, 2023, DOI: 10.1109/IROS55552.2023.10342169.

[17] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y. N. Wu, S.-C. Zhu, and H. Liu, "LLM^3: Large Language Model-based Task and Motion Planning with Motion Failure Reasoning," *arXiv preprint arXiv:2403.11552,* 2024, DOI: 10.48550/arXiv.2403.11552.

[18] 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, DOI: 10.48550/arXiv.2302.05128.

[19] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165,* 2020, DOI: 10.48550/arXiv.2005.14165.

[20] M. Skreta, N. Yoshikawa, S. Arellano-Rubach, Z. Ji, L. B. Kristensen, K. Darvish, A. Aspuru-Guzik, F. Shkurti, and A. Garg, "Errors are useful prompts: Instruction guided task programming with verifier-assisted iterative prompting," *arXiv preprint arXiv:2303.14100,* 2023, DOI: 10.48550/arXiv.2303.14100.

[21] S. S. Raman, V. Cohen, I. Idrees, E. Rosen, R. Mooney, S. Tellex, and D. Paulius, "CAPE: Corrective Actions from Precondition Errors using Large Language Models," in *2024 IEEE*

International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 2024, DOI: 10.1109/ICRA57147.2024.10611376.

[22] 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*, Boca Raton, USA, pp. 294-301, 2004, DOI: 10.1109/ICTAI.2004.120.

[23] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research,* vol. 26, pp. 191-246, July 2006, DOI: 10.1613/jair.1705.

[24] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *IEEE Access,* vol. 12, pp. 55682-55696, April 2024, DOI: 10.1109/ACCESS.2024.3387941.

[25] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ros topics]," *IEEE robotics & automation magazine,* vol. 19, no. 1, pp. 18-19, March 2012, DOI: 10.1109/MRA.2011.2181749.

[26] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, and F. Yan, "Grounded sam: Assembling open-world models for diverse visual tasks," *arXiv preprint arXiv:2401.14159,* 2024, DOI: 10.48550/arXiv.2401.14159.

[27] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, and J. Zhu, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," *arXiv preprint arXiv:2303.05499,* 2023, DOI: 10.48550/arXiv.2303.05499.

[28] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, and W.-Y. Lo, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, Paris, France, pp. 4015-4026, 2023, DOI: 10.1109/ICCV51070.2023.00371.

[29] A. Ten Pas and R. Platt, "Using geometry to detect grasp poses in 3d point clouds," *Robotics Research: Volume 1,* pp. 307-324, 2018, DOI: 10.1007/978-3-319-51532-8_19.

[30] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Advances in neural information processing systems*, New Orleans, USA, pp. 22199-22213, 2022.

[31] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, and S. Anadkat, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774,* 2023, DOI: 10.48550/arXiv.2303.08774.

권 민 서

2024 이화여자대학교 컴퓨터공학과 학사

2024~ 현재 이화여자대학교 컴퓨터공학과 석사 과정

관심분야: Robot Task Planning

김 영 준

1993 서울대학교 계산통계학과 학사

1996 서울대학교 계산통계학과 석사

2000 Purdue University, Computer Science 박사

2003 University of North Carolina at Chapel Hill 박사후 연구원

2003~ 현재 이화여자대학교 컴퓨터공학과 교수

관심분야: Robot Motion Planning, Haptic Rendering, Physically-based Animation, Geometric Modeling