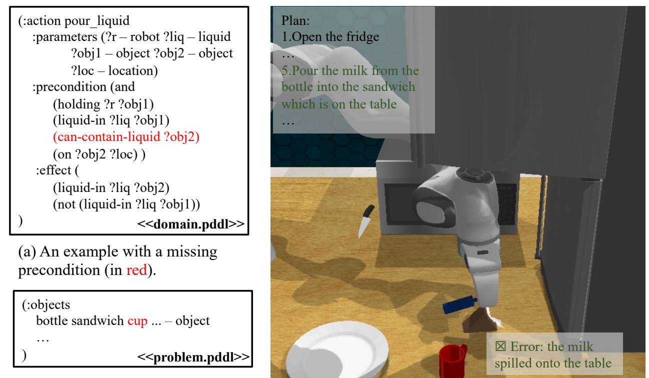
Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

Language-Augmented Symbolic Planner for Open-World Task Planning

Guanqi Chen*, Lei Yang*, Ruixing Jia*, Zhe Hu†, Yizhou Chen‡, Wei Zhang§, Wenping Wang¶, and Jia Pan*‡
*The University of Hong Kong
†Chongqing Institute of Microelectronics Industry Technology, University of Electronic Science and Technology of China
‡Centre for Transformative Garment Production (TransGP)
§Southern University of Science and Technology
¶Texas A&M University

***Abstract*—Enabling robotic agents to perform complex long-horizon tasks has been a long-standing goal in robotics and artificial intelligence (AI). Despite the potential shown by large language models (LLMs), their planning capabilities remain limited to short-horizon tasks and they are unable to replace the symbolic planning approach. Symbolic planners, on the other hand, may encounter execution errors due to their common assumption of complete domain knowledge which is hard to manually prepare for an open-world setting. In this paper, we introduce a Language-Augmented Symbolic Planner (LASP) that integrates pre-trained LLMs to enable conventional symbolic planners to operate in an open-world environment where only incomplete knowledge of action preconditions, objects, and properties is initially available. In case of execution errors, LASP can utilize the LLM to diagnose the cause of the error based on the observation and interact with the environment to incrementally build up its knowledge base necessary for accomplishing the given tasks. Experiments demonstrate that LASP is proficient in solving planning problems in the open-world setting, performing well even in situations where there are multiple gaps in the knowledge.**



(a) An example with a missing precondition (in red).

(b) An example with a missing object (in red).

(c) Visualization of execution failure.

Fig. 1: An illustrative example of the challenge in open-world task planning is that the knowledge (i.e., domain and problem files shown in (a) and (b)) of the symbolic planner can be incomplete. This causes the execution failure of the planned action as depicted in (c).

I. Introduction

Enabling robotic agents to perform complex long-horizon tasks has been a long-standing goal in robotics and artificial intelligence (AI). To accomplish long-horizon tasks, it is crucial to decompose such a task into a sequence of proper actions with preconditions to ensure that the actions can be correctly executed by the robotic agent, and with effects to ensure that each of the subgoals and eventually the final goal can be reached.

The emergence of Large Language Models (LLMs) in various natural language understanding and instruction following tasks [19, 17, 21] motivates the use of LLMs in robotic task planning, aiming to exploit the commonsense reasoning capability of LLMs. Recent studies have demonstrated the capabilities of LLMs in task planning [8, 2, 9, 12, 22]. For example, [2, 9] proposed to prompt LLMs to directly synthesize robotic actions based on robot capabilities and adapt to the objects available in the environment. On the other hand, some works [12, 22] provide LLMs with code-style prompts that contain task information, hints, and few-shot examples to synthesize executable codes for robots. Despite these endeavors in task planning, growing evidence [24] suggests that achieving long-horizon tasks still poses challenges to LLMs. A recent work, Demo2Code [25], presents textual demonstrations as supplementary information to generate code for controlling robots in long-horizon tasks. However, the textual demonstrations must be detailed and complete to accomplish the task.

Planning with symbolic representations is a systematic approach to synthesizing verifiable plans against a set of specifications of the environment for robotic agents. However, hand-crafted symbolic representations may not be able to capture all constraints of a planning problem, resulting in unexpected failures during execution. A few changes in the environment may require a major rewrite of the symbolic plans, making it a tedious process for controlling a robot. For example, in Fig. 1, consider a scenario where the robot is assigned the task of *heating milk*. The milk is initially in a bottle that is not microwave-safe, therefore it needs to be transferred to another microwave-safe container for heating. But the action *pour_liquid* may lack a precondition "*the target object should be able to contain liquid*", resulting in the robot pouring the milk from the bottle to any object in the scene, or a sandwich in this example as shown in Fig. 1. This action led to an unexpected effect "*the milk spilled onto the table*", which is recognized as an execution failure. This example showcases that the risk of incomplete knowledge about the deployment environment hinders applications of symbolic planning in *open-world* environments where the objects of interest may change and novel preconditions may be necessary to accomplish the same goal.

In this paper, we propose a Language-Augmented Symbolic Planner or LASP. This framework integrates pre-trained large language models to enable conventional symbolic planners to operate in an open-world environment where only incomplete knowledge of action preconditions, objects, and properties is initially available. Like humans, LASP will interact with the environment to collect information and incrementally build up its knowledge base necessary for accomplishing the given tasks. When an execution plan encounters an error alongside the observation of this error, LASP can utilize the LLM to diagnose the cause of the error based on the observation and identify the missing element in the current plan. Recall the example in Fig. 1, LASP can reason the target of the pouring action must be a container based on the observation that the milk spills onto the table. Based on this identified precondition and the goal of the task, LASP can select suitable objects in the environment to expand its knowledge base.

This brings a few clear advantages to LASP. First, LASP keeps the advantages of interpretability, verifiability, and composability that symbolic planning possesses. In addition, the observation-based reasoning mechanism enables LASP to correct errors encountered during plan execution. Unlike COWP [3] that relies on the provided factors that may cause errors, our LASP is designed to automatically identify the potential causes of the errors based on the observations.

In summary, the contributions of this work are as follows:

1) We introduce **LASP**, a novel framework that integrates pre-trained large language models to enable conventional symbolic planners to operate in an open-world environment where only incomplete knowledge of action preconditions, objects, and properties is initially available.
2) Experiments demonstrate that LASP is proficient in solving planning problems in the open-world setting, performing well even in situations where there are multiple gaps in the knowledge.

II. Related Work

A. Task Planning

Classical Task Planning. Task planning aims to determine the sequence of actions that a robot should perform to achieve a specific goal. Conventional methods transfer this planning problem to a search problem via symbolic planners [6, 7]. They define the planning problem in a declarative language, such as planning domain definition language (PDDL) [1, 4], specifying the initial state, goal, and a set of actions that the robot can take to transition between states. The algorithms then search through the space of possible action sequences to find a plan that starts from the initial state and reaches the given goal. However, the symbolic planner relies on complete domain knowledge which encompasses all the constraints of the environment to search for a valid solution. This is hard and tedious for a non-expert to tailor. To address this challenge, Lin et al. [13, 14] proposed an approach to repair flawed action preconditions and effects in the planning models to make the plans viable, focusing on scenarios where the planning models contradict the provided valid plans. Gragera et al. [5] proposed an automated planning approach to repair planning models with incomplete action effects, allowing symbolic planners to generate solutions for tasks that were originally unsolvable. Besides, Sreedharan et al. [23] introduced a method using hierarchical abstractions to generate explanations for unsolvable planning problems. These explanations can be further utilized for the planning model repair to make the planner find a solution. In our work, we focus on enabling symbolic planners to operate in the open-world environment where only incomplete knowledge of action preconditions, objects, and properties is initially available. When planning with such incomplete knowledge, the planner may generate an invalid plan that leads to execution errors. Our proposed LASP can leverage LLMs to repair the planning model based on the textual observation of the execution error, enabling the planner to find a valid solution. Compared to the works by Liu et al. [13, 14], which repair the planning model to make the provided valid plan viable, our proposed LASP starts with an incorrect plan, refining the planning model to find a valid plan. In contrast to the work by Gragera et al. [5] which can repair missing action effects, LASP is able to repair missing action preconditions, properties, and objects. Unlike the work by Sreedharan et al. [23], our work mainly focuses on repairing the incomplete planning model and enabling the symbolic planner to find solutions rather than explanations. Besides, our method does not conflict with these works and can be used in conjunction with them.

Planning with LLMs. Recently, after training on massive data, LLMs have acquired rich commonsense knowledge and demonstrated powerful in-context learning ability. Consequently, numerous researchers are exploring applications of these models in task planning. Huang et al. [8] attempted to query LLMs to predict actions for task completion with several demonstrations and the task description in natural language. Ahn et al. [2] proposed SayCan to ground the free-form output of LLMs in robotic affordances. Some works [12, 22] provided code-style prompts to LLMs and required LLMs to generate codes that the robot can execute directly. However, a significant drawback of existing LLMs is that they exhibit limited proficiency in undertaking long-horizon planning endeavors for complex tasks [24]. Wang et al. [25] introduced Demo2Code to leverage textual demonstrations as supplementary information to generate code for controlling robots in long-horizon tasks. However, Demo2Code relies on detailed and complete textual descriptions of completing the task. Therefore, some researchers [15, 26] explored using LLMs to translate the task description from natural language to PDDL and leveraged the symbolic planner to solve the planning problem. In this work, we leverage LLMs to enable symbolic planners to operate in an open-world environment where only incomplete knowledge of action preconditions, objects, and properties is initially available. When the robot encounters an error during executing the plan, we utilize the commonsense reasoning capability of LLMs to identify the cause of the error, and further refine the planning model. Additionally, in instances where the planner cannot find a solution due to the incomplete object set, we harness the commonsense reasoning ability of LLMs to supplement missing objects, thereby facilitating the planner in reaching a solution.

B. Robot Error Correction

Recovering from failure or correcting an error is a significant capability for robots, especially in scenarios where interactions occur with non-expert users or in the absence of human assistance. Liu et al. [16] proposed REFLECT, a framework that utilizes LLMs to summarize robot experiences, explain the error, and predict actions to correct the error directly. Huang et al. [9] introduced Inner Monologue that can leverage a variety of sources of feedback from the environment and humans for replanning to predict new actions to continue completing tasks. Compared to the works by Liu et al. [16] and Huang et al. [9], our proposed LASP can avoid repeating the same errors after correction. Besides, Ding et al. [3] designed COWP to identify possible errors in a plan based on the provided situation and utilize LLMs to augment the robot's action knowledge to find a new plan to correct the possible erroneous actions. In contrast to COWP [3], our proposed approach can eliminate the dependence on provided error factors and identify potential sources of errors.

III. Background and Problem Statement

In this work, we employ a PDDL-based symbolic planner to perform task planning.

a) Planning Task: A planning task is defined as a tuple $P = (O, R, s_0, g, A)$, consisting of the following components:

- O is the set of objects in the planner's internal world;
- R is the set of properties of each object in O. The property can be a binary-valued predicate or a numeric-valued function. We define F as the set of propositions instantiated from R with respect to O, and 2^F refers to the state space;
- $s_0 \in 2^F$ is the initial state. For each time step t, we denote $s_t \in 2^F$ as the state of this step, i.e., the set of its true propositions;
- $g \subseteq F$ is the goal description composed of a set of propositions;
- A is the set of actions that can change the current state by adding or deleting some propositions. Each action can be specified by a triplet $a = (\text{Pre}(a), \text{Add}(a), \text{Del}(a))$, where $\text{Pre}(a), \text{Add}(a), \text{Del}(a) \subseteq F$ represent the set of preconditions, the set of added propositions, and the set of removed propositions that modify the current state. If an action a_t is applicable in a state s_{t-1}, it implies $\text{Pre}(a) \subseteq s_{t-1}$. And the resulting state s_t should become $(s_{t-1} \setminus \text{Del}(a)) \cup \text{Add}(a)$.

The solution to P is a plan consisting of a sequence of actions $\pi = (a_1, a_2, ..., a_n)$ that can achieve the goal g starting from s_0.

b) Types of Incomplete Knowledge: In our open-world setting, we assume that O, R, and $\text{Pre}(a)$ may be incomplete. Here, we denote $\bar{O}$, $\bar{R}$, $\bar{A}$ as complete object, property, and action sets.

- **Incomplete** $\text{Pre}(a)$. In this case, the state s_{t-1} before executing action a_t may not satisfy the actual preconditions $\text{Pre}(\overline{a_t})$, where $\bar{a}_t \in \bar{A}$. This can lead to the agent's failure to execute the action a_t to obtain the desired result. For example, when the precondition "*the target object should be able to contain liquid*" is missing in the precondition set of action *pour_liquid*, the planner may instantiate an action of pouring milk from the bottle to the sandwich, resulting in an execution error.
- **Incomplete** R. Without a complete set of properties, the planning task may lack some propositions, for example, "*the target object is a liquid container*". This may lead to incomplete $\text{Pre}(a)$, eventually leading to the previous error type.
- **Incomplete** O. Lastly, the lack of a proper object prevents finding a solution to P. For example, when no object in O is simultaneously *microwave-safe* and *able to contain liquid*, the planner will not find a solution to the planning task shown in the *heating milk* example.

In this work, we focus on refining the aforementioned incomplete knowledge within the planning task P to find a plan that can achieve the given goal g.

We assume that there exists a valid solution for the planning problem with complete domain knowledge. To verify whether an action can be successfully executed in the environment, we introduce an oracle that contains the complete domain knowledge for this purpose. When the state of the environment aligns with the preconditions specified in the oracle PDDL model for a particular action, it can be successfully executed. Conversely, if the preconditions are not met, the agent will be given a few sentences describing the observation $d(e)$ of the error e during execution failure. In the practical applications, vision-and-language models[20, 11, 18] can be used to generate a natural language paragraph describing images that capture the scene during execution failure, serving as $d(e)$. Given $d(e)$, the agent needs to infer the cause of the error through interactions with the environment to refine the planning task, updating the precondition set for the actions it takes, expanding its object set O with elements in $\overline{O}$, and acquiring novel properties of an object to enlarge its property set R from $\overline{R}$.

IV. Methodology

LASP enables symbolic planners to solve planning problems in an open-world setting with the support of LLMs. The overview of LASP is shown in Fig. 2. While the symbolic planner can find a solution to a given task if the problem P is complete, in the case of an incomplete problem, our LLM is tasked to refine and complete the problem based on the error observations through interactions with the environment. In the following section, we first introduce the proposed algorithm and then elaborate on the functionalities of our LLM.

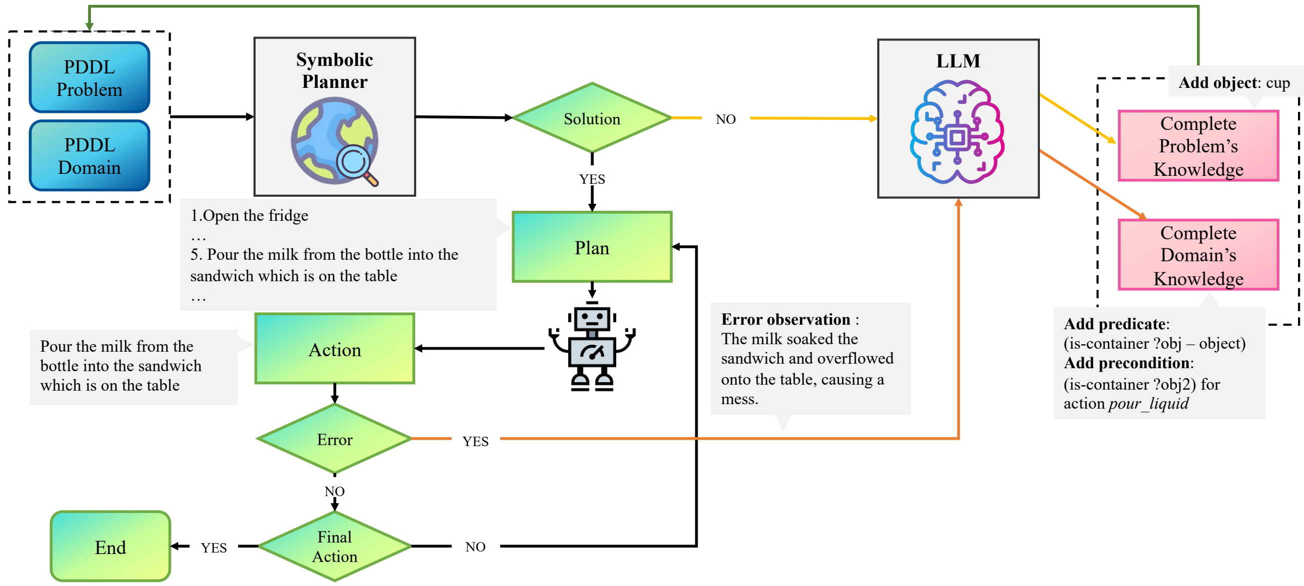


Fig. 2: An overview of our proposed LASP, a task planning framework in the open world. The symbolic planner is responsible for finding a plan to accomplish the given task, and subsequently, the robot executes this plan. Given an initial, incorrect plan, the robot will encounter some errors during action execution. Our proposed method LASP will invoke the LLM to recursively refine the task planning problem by supplementing action knowledge to the planner. Moreover, if the planner is unable to find a plan afterward, the LLM can augment the planner's task-specific knowledge with necessary objects to assist the planner in finding an error-free solution.

A. *LASP Algorithm*

As shown in Algorithm 1, the symbolic planner initially generates a plan π for the given planning problem P (Line 37). The robot then interacts with the environment by sequentially executing the actions in the plan. In cases of an execution error, LASP is provided with a description of the observation regarding the error $(d(e))$ and then it invokes a recursive function REFINE_AND_REPLAN to refine the planning problem according to the error observation $d(e)$.

In the step of refining the planning problem, LASP first utilizes the LLM to infer potential reasons causing the error (Line 19). Based on the list of potential reasons, LASP employs the LLM to progressively generate a precondition regarding the action, i.e., REFINE_PREC_AND_PROP_SET, and add it to the planner. If the existing properties are insufficient to express the precondition, LASP also will add corresponding new properties to the planner. However, adding a precondition does not guarantee a solution due to the lack of a suitable object. Hence, LASP then tries to search for a suitable object from $\overline{O}$ using the LLM to augment the planner's knowledge by using the function REFINE_OBJECT_SET.

After each refining step, the symbolic planner undergoes a replanning step. If a new plan π' is found, it will be executed to check whether the goal is reached. There are three possible outcomes: (1) the robot accomplishes the task; (2) the robot encounters a new error observation (i.e., $d(e') \neq d(e)$, where e' and e refer to the new error and the previous one, respectively, that are latent and not observable); (3) the error is not corrected. For the second situation, LASP will recursively continue the refining and replanning step. For the last, LASP will iterate over all potential reasons to refine the problem.

B. *Functionalities of LLM*

In our proposed algorithm, the LLM plays five roles: (1) cause analyzer, (2) action precondition generator, (3) property completeness evaluator, (4) NL-to-PDDL translator, and (5) object expander. The first four modules (or functions) are designed to correct errors during action execution, while the last function is intended for scenarios where the planner cannot find a solution due to the lack of objects.

In the following, we give an example to walk the reader through these LLM-empowered functions. We denote the variables in a template prompt as [$VARIABLES$] and we provide examples of actual textual descriptions for some variables.

Cause Analyzer. The cause analyzer is designed to generate potential causes of the execution error. We provide a rich context to the LLM, allowing it to have a clear understanding of the robot's objective, what it has done, the incorrect action it has taken, and the corresponding observation. This enables the LLM to make reasonable speculations about the causes of errors.

Prompt: You are expected to provide possible reasons for errors in robot actions, where errors are essentially unexpected outcomes. You are given the robot's task, the historical actions, the action that caused the error, and the observation of the error. [$OUTPUT REQUIREMENTS$]
Given [$ROBOT TASK$]: The robot's task is to heat the milk and place it on the table.

Algorithm 1 LASP: Language Augmented Symbolic Planning

Require: Planning Problem $P = (O, R, s_0, g, A)$; Symbolic Planner SP; Large Language Model LLM.
Ensure: Plan $\boldsymbol{\pi}$.

// Refine object set O
function REFINE_OBJECT_SET(P', $d(e)$)
  objects = GET_CANDIDATE_OBJECTS_FROM_ENV()
  new_object = LLM(P', objects, $d(e)$)
  P' = ADD_KNOWLEDGE(P', new_object)
  return P'
// Refine precondition set $Pre(a)$ and property set R
function REFINE_PREC_AND_PROP_SET(P, reason, d(e))
  new_prec, new_prop = LLM(P, reason, $d(e)$)
  P' = ADD_KNOWLEDGE(P, new_prec, new_prop)
  return P'
// The refining step
function REFINING_STEP(P, reason, $d(e)$)
  P' = REFINE_PREC_AND_PROP_SET(P, reason, $d(e)$)
  π' = SP(P')
  if π' is empty **then**
    P' = REFINE_OBJECT_SET(P', $d(e)$)
  return P'
// Refine and replan recursively
function REFINE_AND_REPLAN(P, π, done, $d(e)$)
  if done **then**
    return P, π, done, $d(e)$ // Stop the recursion
  reasons = LLM($d(e)$, π)
  for reason in reasons **do**
    P' = REFINING_STEP(P, reason, $d(e)$)
    π' = SP(P')
    if π' is empty **then**
      continue
    done$'$, $d(e')$ = EXECUTE(π')
    if done$'$ **then**
    // 1) Robot accomplishes the task
      return P', π', done$'$, $d(e')$
    else
      if $d(e') \neq d(e)$ **then**
      // 2) New error encountered
        P'', π'', done$''$, $d(e'')$
        = REFINE_AND_REPLAN(P', π', done$'$, $d(e')$)
        if done$''$ **then**
          return P'', π'', done$''$, $d(e'')$
      else
      // 3) Error not corrected
        continue
  // Error not corrected after all the attempts
  return P, π, done, $d(e)$
function MAIN( )
  π = SP(P)
  done, $d(e)$ = EXECUTE(π)
  if not done **then**
    $*, \pi, *, *$ = REFINE_AND_REPLAN(P, π, done, $d(e)$)
  return π

[$HISTORICAL ACTIONS$]: The historical actions are:
1. Open the fridge.
2. Pick up the sandwich from the fridge.
3. Put the sandwich on the table.
4. Pick up the bottle from the fridge.
[$ERRONEOUS ACTION$]: The action caused the error is: Pour the milk from the bottle into the sandwich which is on the table.
[$ERROR OBSERVATION$]: The observation of the error is: The milk soaked the sandwich and overflowed onto the table, causing a mess.
What are possible reasons that could lead to this error?

LLM output: Possible reasons:
[$REASON$] 1: Misinterpretation of the task instructions: The robot was supposed to pour the milk into a container for heating, not onto the sandwich.
[$REASON$] 2: Lack of spatial awareness: The robot may not have recognized the presence of the sandwich on the table.
...

Action Precondition Generator. With a potential cause, the action precondition generator is tailored to produce a few sentences to describe the precondition of the action to prevent the error.

Prompt: You are given a robotic action that caused an error and the reason why this error occurred. You are expected to provide a precondition for a robotic action to prevent a recurrence of the same error. [$OUTPUT REQUIREMENTS$]
Given [$ERRONEOUS ACTION$] and [$REASON$].
What precondition should be satisfied to prevent a recurrence of the same error?

LLM output: [$NL-PRECONDITION$] Precondition: The target of the pouring action must be a container.

Property Completeness Evaluator. Due to the possibility of incomplete properties in the planner, the property completeness evaluator is developed to determine whether properties are missing. Specifically, the LLM will assess whether the existing properties adequately represent the precondition obtained in the action precondition generator.

Prompt: You are expected to judge whether the given natural language described precondition can be expressed by the candidate predicates and functions, or their respective antonyms. You are given the natural language described precondition, candidate predicates, and functions. [$OUTPUT REQUIREMENTS$]
Given [$NL-PRECONDITION$]
[$CANDIDATE PREDICATES$]: (holding ?r - robot ?obj - object), (is-empty-handed ?r - robot), (is-open ?rec - receptacle), (in ?obj - object ?rec - receptacle), (on ?obj - object ?loc - location), (is-microwave ?rec - receptacle), (is-heat-insulation ?obj - object), (liquid-in ?liq - liquid ?obj - object), (is-microwave-safe ?obj - object), (can-support ?loc - location).
[$CANDIDATE FUNCTIONS$]: (temperature ?obj - object), (tolerance-temperature ?r - robot), (liquid-temperature ?liq - liquid).
Can the precondition be expressed by the candidate predicates and functions, or their respective antonyms?

LLM output: No

NL-to-PDDL Translator. Depending on the output of the property completeness evaluator, the NL-to-PDDL translator has two ways to translate the natural language described precondition into PDDL expression. If the property completeness evaluator determines that the existing properties cannot express the precondition, the NL-to-PDDL translator can convert the precondition without requiring the translated PDDL precondition related to any existing properties. Otherwise, a constraint is imposed to translate the precondition based on the existing properties (included in the output of the property completeness evaluator).

Prompt: Given the action description, the candidate parameters of the action, the corresponding between instances and parameters, and a natural utterance, you are expected to translate the natural utterance into a PDDL precondition expression for the given action. [$PDDL GRAMMARS$] [$OUTPUT REQUIREMENTS$] and [$TWO EXAMPLES$]
Given [$ERRONEOUS ACTION$]
[$ACTION PARAMETERS$]: Candidate parameters: the robot, denoted as ?r; the liquid, denoted as ?liq; the object, denoted as ?obj1; the object, denoted as ?obj2; the location, denoted as ?loc.
[$CORRESPONDENCE$]: Corresponding between instances and parameters: the agent is ?r; the milk is ?liq; the bottle is ?obj1; the sandwich is ?obj2; the table is ?loc.
[$NL PRECONDITION$].

LLM output: PDDL expression: (is-container ?obj2)

When a new property is added to the planner, the planner needs to acquire the values of every existing object regarding this newly added property for planning. To accomplish this, we assess the similarity between the newly added property and properties in the environment, except those already present in the planning problem. Subsequently, we assign the value of the most semantically similar property to this new property. This enables the planner to incorporate values for every existing object concerning this newly added property into the initial state.

Object Expander. When a new precondition is introduced into the symbolic planner, the existing objects in the symbolic planner may no longer be sufficient to meet the new constraints. Hence, the planner may fail to find a solution. To address this problem, the object expander is introduced to add one or more new objects to the planning problem, thereby allowing the planner to find a solution. Here, we query the object expander to select a new object to assist the robot complete the task.

Prompt: You are expected to select the most suitable object from the candidate objects to prevent a recurrence of the error and assist the robot complete the task. You are given the robot task, the action that causes the error, the precondition that the action does not satisfy, and the candidate objects. [$OUTPUT REQUIREMENTS$]
Given [$ROBOT TASK$] [$ERRONEOUS ACTION$] [$NL PRECONDITION$]
[$CANDIDATE OBJECTS FROM ENV$]: cup, plate, knife.
Which object is most suitable to prevent a recurrence of the error and assist the robot complete the task?

LLM output: Suitable object: cup.

V. Experiments

A. Experimental Setup

a) Evaluation tasks: In our experiments, we designed 4 tasks in the kitchen domain to evaluate the proposed method LASP: *serving fruit*, *storing fruit*, *heating sandwich*, and *heating milk*. For each task, we constructed at least a pair of incomplete PDDL domain and problem files to test LASP. We constructed a total of 9 pairs of incomplete PDDL domain and problem files. Among the 9 test cases, a crucial precondition is missing in seven domain files and two crucial preconditions are missing in each of the other two domain files. More information about the missing knowledge of test cases can be found in Tab. I.

b) Baselines: To demonstrate the advantages of our method, we compare with the following baselines:

- Language Models as Zero-Shot Planners (LMZSP) [8] is a baseline that queries an LLM to generate a plan for a given task with few-shot examples. In this way, the output of the LLM is not grounded, so it utilizes a pre-trained language model to translate each output action into an available action.
- Inner Monologue [9] predicts an initial plan based on the prompt that contains the task description, the primitive actions, available objects in the environment, and few-shot examples. Subsequently, it replans in response to feedback from the environment whenever an error is encountered during plan execution. Here, we provide information on the action that encounters an error and the observation of the error to the LLM for its replanning. Two rounds of feedback are allowed.
- ProgPrompt [22] leverages a code-style prompt to query an LLM to generate a code to accomplish the given task. Its prompt includes available action functions, available parameters to call the functions, and few-shot examples.

c) Evaluation metric: We evaluate the effectiveness of different approaches regarding the success rate. For each task, we execute each baseline method five times and report the average success rate. As for our method, we perform five runs for each case within a task and report the average success rate for that specific task across all cases within the same task.

d) Implementation details: We utilize a numeric fast downward planner[1] to search plans for given planning tasks. Besides, both the baselines and our approach leverage GPT-4 in the implementation. Additionally, to facilitate providing diverse descriptions of error-related observations, we leverage GPT-4 to generate these descriptions based on erroneous actions, unmet conditions, and reference examples.

[1]https://github.com/ipc2023-numeric/team-1

Test case	Task	Missing preconditions	Missing properties	Missing objects
1	serve fruit	Picking up an object requires its weight to be below the robot's lifting capacity.	weight and lift-capacity	apple
2	store fruit	Placing an object in the fridge is conditioned upon the door being open.	-	-
3	heat sandwich	Picking up an object requires its temperature to be below the robot's tolerance temperature.	temperature and tolerance-temperature	glove
4	heat sandwich	Using a microwave to heat objects is conditioned upon the door being closed.	-	-
5	heat milk	Pouring liquid from object1 to object2 requires object2 can contain liquid.	able to contain liquid	cup
6	heat milk	Wearing the heat-insulator requires the robot not holding anything.	-	-
7	heat milk	Picking up an object requires its temperature to be below the robot's tolerance temperature.	temperature and tolerance-temperature	glove
8	heat sandwich	Combination of Case 3 and 4		
9	heat milk	Combination of Case 5 and 7		

TABLE I: Test cases and their missing knowledge for evaluating LASP.

Task	Optimal Plan Steps	LMZSP [8]	Inner Monologue [9]	Progprompt [22]	LASP (ours)
serve fruit	2	20%	100%	80%	100%
store fruit	3	20%	100%	100%	100%
heat sandwich	10	0%	20%	0%	100%
heat milk	13	0%	20%	0%	85%

TABLE II: Quantitative results of LASP and the language-driven methods in open-world task planning. Our method can achieve superior results as compared to the language-driven planning methods

Test case	Task	LASP (ours)
1	serve fruit	100%
2	store fruit	100%
3	heat sandwich	100%
4	heat sandwich	100%
5	heat milk	80%
6	heat milk	100%
7	heat milk	100%
8	heat sandwich	100%
9	heat milk	60%

TABLE III: Success rate of our approach on test cases.

B. Results

The experimental results are shown in Table II. Among them, our proposed LASP achieves the highest success rate for all the tasks. We could notice that LASP performs well in handling both short-horizon tasks (i.e., *serving fruit* and *storing fruit*) and long-horizon tasks (i.e., *heating sandwich* and *heating milk*). Tab. III presents the detailed performance of LASP on test cases. LASP achieves 100% success rates in 7 cases with multiple tests. Even in instances where there are two missing preconditions, as constructed in cases 8 and 9, LASP demonstrates superior performance, achieving an average success rate of 80%. For long-horizon tasks such as *heating sandwich* and *heating milk*, LASP attains an average success rate of 91.4% across all test cases related to these long-horizon tasks. In comparison, other purely LLM-driven methods exhibit poor performance in handling long-horizon planning tasks, highlighting the advantage of symbolic planning, which incorporates logical reasoning mechanisms to explicitly reason about the constraints and conditions.

For *LMZSP*, it is difficult to ground the commonsense knowledge to a specific domain via the few-shot examples, which leads to its poor performance on all the tasks. For example, when the robot was asked to finish the task of storing an apple in a closed fridge, *LMZSP* instructed the robot to first grasp an apple and then attempt to open the fridge. However, it failed to open the fridge because the robot's hand was occupied with holding the apple, preventing it from manipulating the fridge door handle. *Inner Monologu*e lacks a clear understanding of the constraints on actions and its own state after each action, leading to frequent omissions of crucial actions or disruptions in the correct sequencing of actions. Therefore, it is difficult to generate a long plan for complex tasks. For example, it instructed the robot to place the cup on the table when the robot did not hold the cup. For *Progprompt*, there are some noise and unforeseen conditions in its generated codes, which results in its poor performance for long-horizon planning tasks. For example, it considers the temperature of the sandwich being higher than the tolerance temperature of the robot as a precondition for picking up the sandwich; however, this precondition is incorrect.

The qualitative result is shown in Fig. 3. Starting with the initial planning task which involves incomplete knowledge of the action precondition set, the object set, and the property set, LASP is able to refine the planning task based on the error observation through interactions with the environment, which enables the planner to find a plan to accomplish the task.

C. Discussions and Limitations

Firstly, the performance of our approach largely depends on the capability of the LLM to read PDDL expressions and

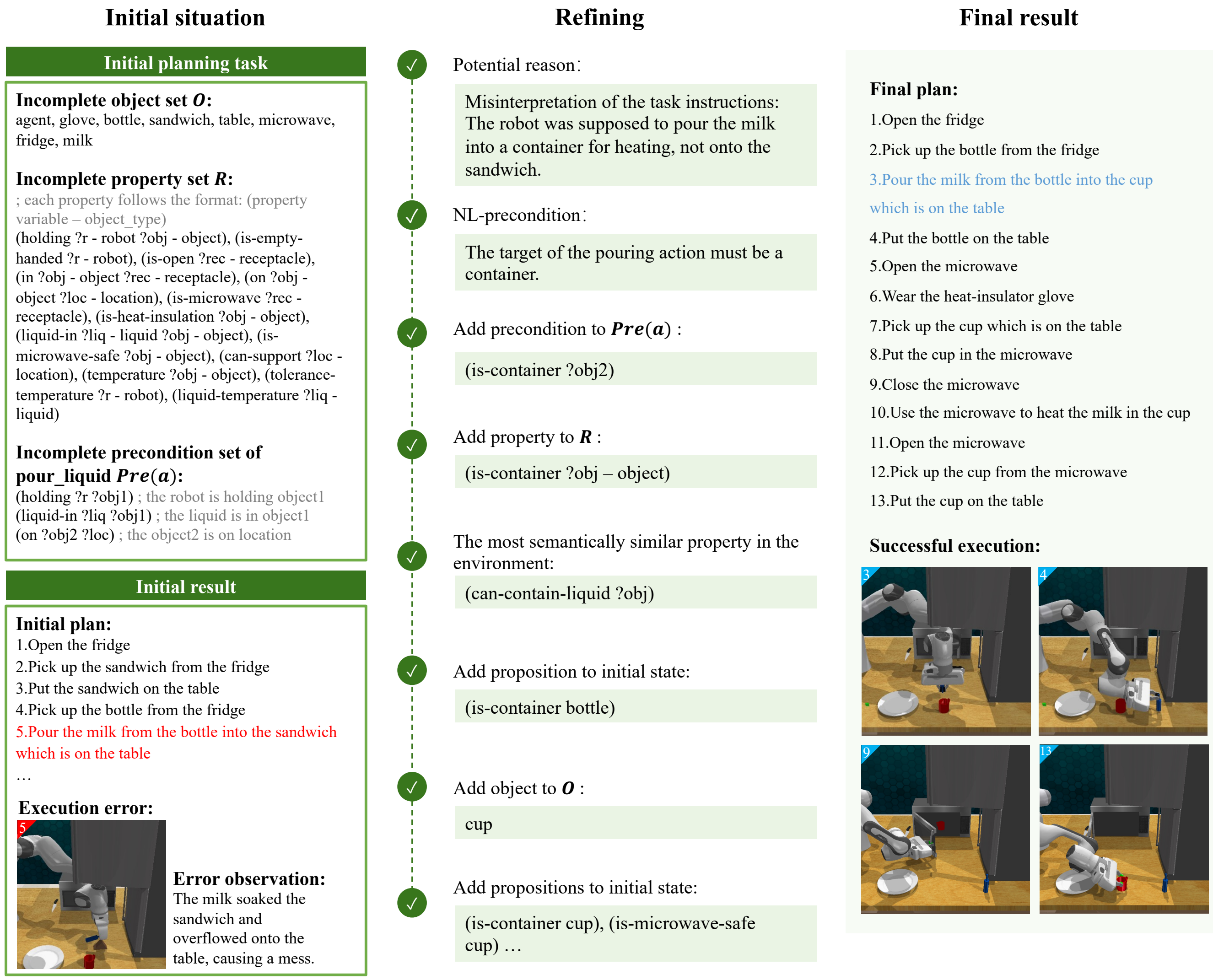


Fig. 3: Qualitative result of LASP for the task of *heating milk*. LASP is able to refine incomplete knowledge within the planning task and find a plan to accomplish the task. The snapshots of the execution actions are taken from our simulation platform built on PyRep [10]. Comments are indicated in gray text and start with a semicolon.

translate natural language descriptions into PDDL expressions. If the initial PDDL expressions are poorly readable by the LLM, our approach may suffer a performance drop. The failure cases of *heating milk* are due to this reason. In these failure examples, the action precondition generator has synthesized the natural language described precondition, e.g., “the target for the milk pouring action is a container” or “the target vessel must be suitable for containing liquids”, indicating *the target of the pouring action should be a container*. However, the property completeness evaluator falsely inferred these preconditions described in natural language can be expressed using existing properties, leading to the failures. This problem may be solved by translating the PDDL expressions in the prompts into natural language.

Currently, our approach assumes that an environment can provide the robotic agent with the true property values at request. In real-world applications, vision-language models (or even humans) may wrongly estimate the property value of an object in the environment. In the future, we intend to estimate the property values from noisy data using vision-language models and leverage relationships between different properties as well as robotic interactions with the environment to gradually reduce the uncertainty of the estimated values. This would make LASP applicable to real-world scenarios.

Besides, our approach can update the planner’s knowledge base only after an action error occurs. In contrast, humans can update their knowledge by reading books (e.g., a manual) or watching demonstrations without explicitly triggering an execution error. Humans can also predict the future states ahead to verify their actions. In the future, we aim to incorporate these capabilities into our method.

Additionally, the assumption that the action effects are complete in our approach may not hold in the practical applications. In the future, we are going to refine the incomplete action effects with the help of additional information like demonstrations.

VI. Conclusion

In this paper, we introduce a novel framework that integrates pre-trained large language models to enable conventional symbolic planners to operate in an open-world environment where only incomplete knowledge of action preconditions, objects, and properties is initially available. LASP keeps the advantages of interpretability, verifiability, and composability that symbolic planning possesses. In addition, the observation-based reasoning mechanism enables LASP to correct errors encountered during plan execution. Experiments demonstrate that LASP is proficient in solving planning problems in the open-world setting, performing well even in situations where there are multiple gaps in the knowledge.

Acknowledgments

This research project is partially supported by the Innovation and Technology Commission of the HKSAR Government under the InnoHK initiative, Hong Kong General Research Fund (11202119 and 11208718), Innovation and Technology Commission (GHP/126/21GD), and Guangdong, Hong Kong and Macao Joint Innovation Project (2023A0505010016). Jia Pan is the corresponding author. The authors would like to thank Jingqian Liu, Xuebo Ji, and Wentao Chen for their help in drawing and discussions. Also, we would like to thank blind reviewers for their constructive suggestions.

References

[1] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl— the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

[2] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[3] Yan Ding, Xiaohan Zhang, Saeid Amiri, Nieqing Cao, Hao Yang, Andy Kaminski, Chad Esselink, and Shiqi Zhang. Integrating action knowledge and llms for task planning and situation handling in open worlds. *arXiv preprint arXiv:2305.17590*, 2023.

[4] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[5] Alba Gragera, Raquel Fuentetaja, Ángel García-Olaya, and Fernando Fernández. A planning approach to repair domains with incomplete action effects. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 153–161, 2023.

[6] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[7] Jörg Hoffmann. The metric-ff planning system: Translating“ignoring delete lists”to numeric state variables. *Journal of artificial intelligence research*, 20:291–341, 2003.

[8] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[9] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[10] Stephen James, Marc Freese, and Andrew J Davison. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.

[11] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*, pages 12888–12900. PMLR, 2022.

[12] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[13] Songtuan Lin, Pascal Bercher, et al. Change the world-how hard can that be? on the computational complexity of fixing planning models. In *IJCAI*, pages 4152–4159, 2021.

[14] Songtuan Lin, Alban Grastien, and Pascal Bercher. Towards automated modeling assistance: An efficient approach for repairing flawed planning domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12022–12031, 2023.

[15] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[16] Zeyi Liu, Arpit Bahety, and Shuran Song. Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*, 2023.

[17] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487, 2022.

[18] OpenAI. Gpt-4v(ision) system card. https://cdn.openai.com/papers/GPTV_System_Card.pdf, 2023.

[19] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida,

Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[20] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[21] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. In *ICLR 2022-Tenth International Conference on Learning Representations*, 2022.

[22] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[23] Sarath Sreedharan, Siddharth Srivastava, David Smith, and Subbarao Kambhampati. Why can't you do that hal? explaining unsolvability of planning tasks. In *International Joint Conference on Artificial Intelligence*, 2019.

[24] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[25] Huaxiaoyue Wang, Gonzalo Gonzalez-Pumariega, Yash Sharma, and Sanjiban Choudhury. Demo2code: From summarizing demonstrations to synthesizing code via extended chain-of-thought. *arXiv preprint arXiv:2305.16744*, 2023.

[26] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.