Language Models Can Infer Action Semantics for Symbolic Planners from Environment Feedback

Wang Zhu **Ishika Singh** **Robin Jia** **Jesse Thomason**
Department of Computer Science
University of Southern California
{wangzhu@usc.edu, ishikasi@usc.edu, robinjia@usc.edu, jessetho@usc.edu}

Abstract

Symbolic planners can discover a sequence of actions from initial to goal states given expert-defined, domain-specific logical action semantics. Large Language Models (LLMs) can directly generate such sequences, but limitations in reasoning and state-tracking often result in plans that are insufficient or unexecutable. We propose Predicting Semantics of Actions with Language Models (PSALM), which automatically learns action semantics by leveraging the strengths of both symbolic planners and LLMs. PSALM repeatedly proposes and executes plans, using the LLM to partially generate plans and to infer domain-specific action semantics based on execution outcomes. PSALM maintains a belief over possible action semantics that is iteratively updated until a goal state is reached. Experiments on 7 environments show that when learning just from one goal, PSALM boosts plan success rate from 36.4% (on Claude-3.5) to 100%, and explores the environment more efficiently than prior work to infer ground truth domain action semantics.

1 Introduction

Symbolic planning requires extensive domain knowledge to produce a sequence of actions that achieve a specified goal. Domain knowledge comprises expert-annotated action semantics that govern the dynamics of the environment. For example, traditional symbolic methods, like Planning Domain Description Language (PDDL; Ghallab et al., 1998) solvers, take action semantics annotated in a domain file as input (Figure 2). These symbolic solvers systematically search the state space based on actions that can be executed as per these semantics and return a sequence expected to achieve specified goal conditions, if possible. However, a human expert must exhaustively define the action semantics of domain to enable symbolic planning.

We propose a novel domain induction task in which an agent must infer the action semantics of an environment without manual annotation or error correction. In this setting, domain information like object properties and action functions headers are given to an agent in natural language. The agent is then asked to infer the action semantics, that is, symbolic pre- and post-conditions, through interacting with the environment and learning from resulting feedback (Figure 1). Our proposed setting is motivated by the longer-term goal of building real-world robots that can explore a new environment (*e.g.*, an apartment) and learn to perform new tasks in the environment (*e.g.*, tidying) by building a symbolic, interpretable world representation.

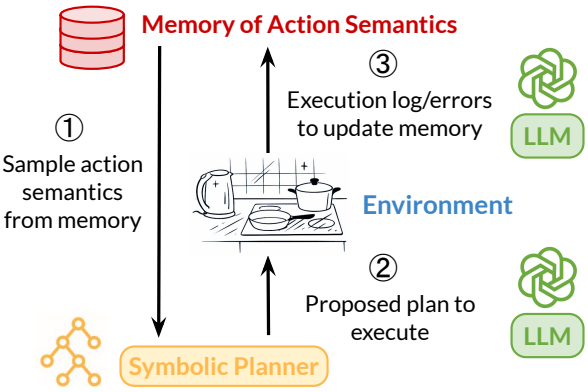


Figure 1: LLMs can propose plans and generate action semantics, but struggle with state tracking. Symbolic planners leverage specialized search algorithms, but require predefined action semantics for the environment. PSALM integrates the strengths of both.

We draw inspiration from prior work that prompts Large Language Models (LLMs) to perform robotics planning tasks. These methods rely on LLMs' ability to infer world knowledge from natural language input for generating unseen plans, rather than leveraging expert domain knowledge. For instance, ProgPrompt (Singh et al., 2023) generates program-like high-level plans from LLMs.

LLMs struggle with long-horizon planning and often generate plans that include invalid actions,

such as filling a cup before the lid is removed. Instead of using the LLM directly as a planner, we propose Predicting Semantics of Actions with Language Models (PSALM), a novel method that combines language models with symbolic solvers to *iteratively explore the environment and predict the action semantics based on environment feedback*. PSALM maintains a probabilistic memory of learned action semantics, which iteratively improves by interacting with the environment. We use LLMs to sample possible incorrect or incomplete candidate action trajectories conditioned on initial proposed action sequences from a symbolic planner, then infer action semantics based on the result of executing those trajectories. PSALM leverages LLMs' strong commonsense reasoning abilities, as well as their ability to generate syntactically valid formal semantics, while using a symbolic solver to search for ways to achieve the final goal state based on our current belief about the action semantics.

We demonstrate the effectiveness and efficiency of PSALM for domain induction in 7 symbolic reasoning environments. PSALM can achieve 100% success rate with the FAST-DOWNWARD solver, while the best LLMs, O1-preview (OpenAI, 2024) and Claude-3.5-Sonnet (Anthropic, 2024) perform less than 40% on average. Additionally, PSALM consistently induces correct domain files, and does so with substantially fewer total execution steps and environment resets than other approaches. The integration of LLMs and symbolic solvers is a promising avenue for domain induction in robotics and general reasoning agent workflows.

2 Background and Related Works

We first introduce symbolic planning, and then compare PSALM's domain induction setup with previous LLM or LLM-Modulo planning methods, as listed in Table 1.

2.1 Classical and Symbolic Planning

Classical planning algorithms have been widely applied in autonomous spacecrafts, military logistics, manufacturing, games, and robotics. The automated STRIPS planner was the first algorithm that operated the Shakey robot (Fikes and Nilsson, 1971). Classical planners require finite, deterministic, and full state information to generate guaranteed plans when a path from the initial to the goal state is possible. Some other frameworks were also shown to be useful for robot planning (Carbonell et al., 1991; Nau et al., 2003). Symbolic planning languages, such as Planning Domain Description Language (PDDL; Ghallab et al., 1998) and Answer Set Programming (ASP; Brewka et al., 2011; Lifschitz, 2002), provide a more structured and flexible way to represent problems.

We use PDDL for symbolic planning (Figure 2). We define a planning problem P as a tuple $\langle \mathcal{D}, \mathcal{O}, s^i, \mathcal{S}^g \rangle$, for $\mathcal{D}$ the domain, $\mathcal{O}$ a set of objects in the domain, s^i the initial state, and $\mathcal{S}^g$ the goal specification, *i.e.*, a set of goal states satisfying the goal conditions. A plan solution to P is a T-step sequence of actions $a_{1..T}$, which once executed at s^i would lead to a state in $\mathcal{S}^g$.

The PDDL domain file also defines the symbolic action set $\mathcal{A}$ in the environment and *predicatess* representing objects properties. Each action has a string name (*e.g.*, `Put-down`), parameters (*e.g.*, `?ob`), and *semantics*. The action semantics Φ_a of an action $a \in \mathcal{A}$ include the *preconditions* when a is valid to execute, and the *postconditions* (effects) describing what changes when an a is executed. Unlike LLM+P (Liu et al., 2023), which generates PDDL problem files from in-context examples, we infer action semantics in domain files.

2.2 LLMs for Task Planning

Several works utilize LLMs to generate plans directly given initial and goal states (Huang et al., 2022; Ichter et al., 2022), but such stochastic, generative approaches lose the success guarantees of symbolic planners. To improve the correctness of the plans, learned latent spaces (Trivedi et al., 2021) or LLMs (Singh et al., 2023; Liang et al., 2023; Hu et al., 2024) are used to generate executable programs, which introduce some symbolic structure and constraints, reaching up to 90% plan success on some simple domains, such as GRIPPERS, but these still do not guarantee plan success (Silver et al., 2023).

Recently, Kambhampati et al. (2024) proposed the LLM-Modulo network, combining LLM plan generation with symbolic planner verifiers. Using an LLM for final plan generation cannot achieve high success rate in complex domains, such as BARMAN and TERMES (Zhao et al., 2023; Hazra et al., 2024). Guan et al. (2023) have compared LLMs and symbolic planners, finding the latter is consistently better over 3 domains.

In this paper, we consider planning domain induction. We use LLMs to propose partial plans and predict domain knowledge based on execution

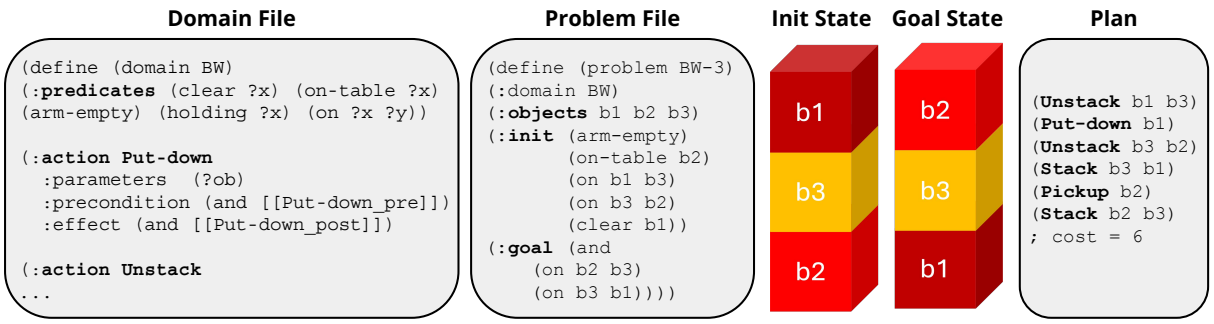


Figure 2: An example of symbolic planning information from the BLOCKSW domain, from left to right: PDDL domain file, PDDL problem file, visualization of initial and goal state for block stacking, and a potential plan.

	Goal	Independent of			Leveraging	Guaranteed
		Partial AS	Valid plans	Human eval	Env feedback	Plan success
Liu et al. (2023)	Problem File	✗	✓	✓	✗	✗
Hazra et al. (2024)	Plan	✓	✓	✓	✗	✗
Silver et al. (2023)	Program	✗	✓	✓	✗	✗
Arora et al. (2018)	AS	✓	✗	✓	✗	✗
Wong et al. (2023)	AS	✗	✓	✓	✗	✗
Guan et al. (2023)	AS	✓	✓	✗	✗	✗
Oswald et al. (2024)	AS	✓	✓	✓	✗	✗
PSALM	AS	✓	✓	✓	✓	✓

Table 1: Comparing the PSALM domain induction task setup to representative related works in LLM and LLM-Modulo planning; here, "AS" is short for action semantics.

feedback. Once we uncover the domain action semantics, plan success from a symbolic solver is guaranteed, provided the goal is reachable.

2.3 Domain Induction

The domain induction problem has been studied under different setups (Appendix Table 4). Arora et al. (2018) introduced several setups, including predicting pre- and post-conditions from valid plans or reusable plan fragments. Wong et al. (2023) and Chen et al. (2024) filled in missing pre- and post-conditions in the domain file given other action semantics in the same domain. Guan et al. (2023) generated domain predicates and action semantics using LLMs given detailed text action descriptions. Because the generated predicates were not aligned with the problem file, human feedback was required for error correction and evaluation.

Closest to our paper, Oswald et al. (2024) extended Guan et al. (2023)'s work by providing predicates and enabling automatic evaluation. They performed a single-round action semantics generation from LLMs and evaluated the accuracy of the generation. Using environment feedback, PSALM effectively solves their domain induction problem, achieving 100% accuracy over 7 domains. We advocate for an additional evaluation relevant for downstream, real-world deployment: the efficiency of the environment exploration.

3 Problem Formulation

Symbolic planning in PDDL requires a domain file characterizing the environment. Human experts must carefully annotate the action semantics to enable the symbolic solver to find correct solutions. We propose a novel *domain induction* task, where AI agents must find the action semantics for a new domain without human annotation or correction.

In the domain induction task, the agent knows $\mathcal{D} \setminus \bigcup_{a \in \mathcal{A}} \Phi_a$, including the predicates and the action names, but not the action semantics. In addition, the agent has access to one problem file $\langle \mathcal{O}, s^i, \mathcal{S}^g \rangle$. The goal of the agent is to learn the correct action semantics Φ_a for each action a. During the learning process, the agent interacts with the environment initialized to state s^i with an open-loop execution of planned actions, after which the environment is reset back to s^i. After learning action semantics, the agent should be able to leverage a symbolic solver for efficient and robust task

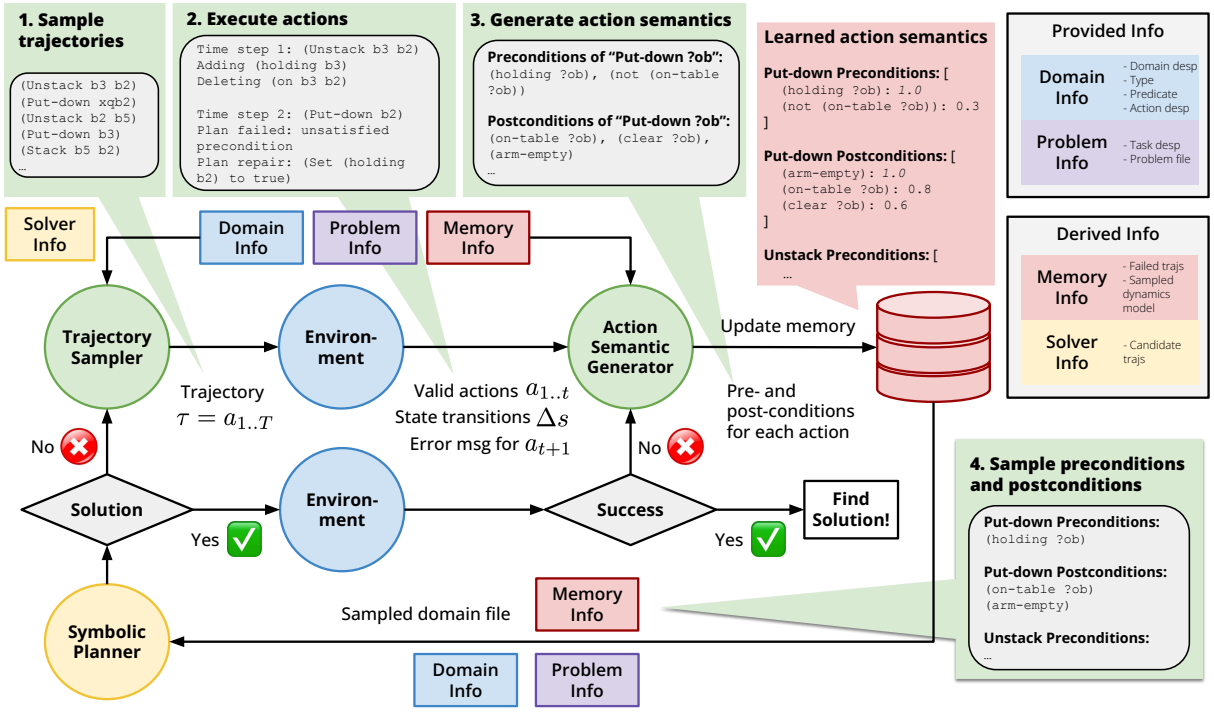


Figure 3: The pipeline of PSALM in four steps: (1) sample trajectories from a trajectory sampler; (2) execute the trajectories in the environment to get feedbacks (3) generate action semantics for each action with environment feedback, and update the memory based on the prediction; (4) sample action semantics from the memory to construct the domain file for the symbolic solver to check the success.

solving in the domain to achieve a state in $\mathcal{S}^g$.

We evaluate the domain induction learning process on three measures. (1) Accuracy (Acc): $\sum_{a\in\mathcal{A}} |\hat{\Phi}_a \cap \Phi_a| / \sum_{a\in\mathcal{A}} |\Phi_a|$, where $\hat{\Phi}_a$ is the predicted action semantics for action a. Finding a path to $\mathcal{S}^g$ is not sufficient to imply correct action semantics, but recovering ground truth action semantics is sufficient to reach $\mathcal{S}^g$ via the symbolic solver. (2) The number of resets (NR): how many times the environment returns to s_i because planned actions did not reach $\mathcal{S}^g$. (3) The number of executed steps (NES): how many total actions were executed during learning, including those causing failure.

4 Proposed Approach: PSALM

We propose the Predicting Semantics of Actions with Language Models (PSALM) framework, which leverages LLM commonsense, formal planning, and environment feedback.

4.1 Overview

Given a goal, we first use an LLM as a *trajectory sampler*, and execute the trajectories in the environment to get feedback (Figure 3). We use the LLM again, together with a rule-based parser, as the *action semantics generator* to predict action semantics, *i.e.*, preconditions and postconditions, for each action based on that feedback. We update the memory of the learned action semantics, ten sample that memory for hypothesized action semantics used as input to the symbolic solver. Finally, the symbolic solver tries to generate a plan from the goal and hypothesized action semantics.

If the symbolic solver finds a solution, we execute the plan in the environment. If the plan reaches the goal, we finish the loop; otherwise, we will pass the result of the failed plan to the LLM to predict the action semantics again. If the symbolic solver does not find a solution, we will provide some partial candidate trajectories from the symbolic solver to the trajectory sampler as a prefix sequence of actions for the plan to be generated.

4.2 Trajectory sampler

We prompt an LLM with domain and problem information (as described in §2.1), memory information representing our current beliefs about action pre- and post-conditions, and partial trajectories generated by the solver. We use templates to convert these into a natural language prompt. For example, the postconditions `(on-table ?ob) (arm-empty)` of the action `put-down` are converted to natural lan-

Model	BARMAN	BLOCKSW	FLOORTILE	GRIPPERS	STORAGE	TERMES	TYREWORLD
Direct plan generation							
O1-preview	0	30	0	90	5	0	30
GPT-4-Turbo	0	20	0	55	5	0	70
Claude 3.5 Sonnet	0	60	0	90	5	0	100
Domain induction + Symbolic planner							
GPT-4-Turbo	0	0	0	0	5	0	0
Oswald et al. (2024)	0	0	0	0	5	0	0
PSALM	**100**	**100**	**100**	**100**	**100**	**100**	**100**

Table 2: Plan success rate over 20 examples, comparing PSALM with LLM and domain induction baselines.

guage *The effects are (on-table ?ob), (arm-empty).* Because the symbolic solver we use, *i.e.*, FAST-DOWNWARD, is performing a search algorithm given a time-limit of W, if the solver finds any candidate trajectories, *i.e.*, not a complete solution but a partial solution stopped by the timer or a dead end, we include the k longest candidate trajectories as additional input to the trajectory sampler. We filter out invalid candidate trajectories that failed in previous iterations. The trajectory sampler prompt finally specifies to pick one of the candidate trajectories and generate a trajectory starting from that. We execute l trajectories sampled in this way from the. When $l > 1$, we predict action semantics for each trajectory separately (§4.3). After execution, we update the semantic hypothesis memory (§4.4).

Prospection. Although the LLM's prompt includes the current beliefs about action preconditions, the LLM can still generate trajectories that violate these preconditions. We add *trajectory prospection*, enabling the system to do forward prediction in an open-loop fashion before actually invoking the simulator based on its symbolic understanding of the world. For a generated trajectory $\tau = a_{1:T}$, we check if is aligned with the current belief of the environment action semantics for v steps. For $v \leq T$, if any action in $a_{1:v}$ does not satisfy the preconditions in the sampled action semantics, starting from that action, we will keep randomly sampling one action until the sampled action is valid in the hypothesized action semantics, repeated to v total actions. Otherwise, if $a_{1:v}$ are all valid, we execute the original $a_{1:T}$.

Random sampler. We create an ablated PSALM that randomly samples v actions per trajectory from the longest candidate trajectory, if available, from the symbolic solver during each iteration. This random sampler ablation can also take advantage of the prospection module.

4.3 Action semantics generator

We combine LLM-based and rule-based action semantics generation given environment feedback.

LLM-based generator. We prompt the LLM to generate the action semantics of each action separately. The prompt contains information about the domain, problem, memory information, and environment feedback. For a trajectory $\tau = a_{1:T}$ failed at step $t+1$, environment feedback for the LLM action semantics generator takes the form of valid actions $a_{1:t}$, state transitions $s_{j+1} - s_j, j = 1, \ldots, t$, where s_1 is the initial state s^i, and the environment error message string for action a_{t+1}. Note that each state is a set of predicate assignments, so we can compute the difference of state transition as the difference of two sets. We assume the error message is provided by the environment. In the tested environments, when an action fails the error message specifies an unsatisfied precondition. When no action fails, the error message either states that the the goal is not reached, or the goal is reached and the PSALM inference loop exits. For each action, we prompt the LLM once to predict the preconditions and once for the postconditions.

Rule-based generator. We also write a rule-based parser for the output from the environment feedback to infer one or more missing preconditions suggested by error messages. Rule-based parser also scan state transition descriptions to derive postconditions. For each iteration, we update the action semantics hypothesis memory with both LLM-generated pre- and post-conditions and those from this rule-based generator. Unlike the LLM action semantics generator, the rule-based generator relies solely on the feedback at the current time step, making it less efficient.

Model	BARMAN	BLOCKSW	FLOORTILE	GRIPPERS	STORAGE	TERMES	TYREWORLD
GPT-4-Turbo	53	61	58	70	45	41	67
Oswald et al. (2024)	65	67	58	75	58	59	80
PSALM	**100**	**100**	**100**	**100**	**100**	**100**	**100**

Table 3: Action semantics accuracy over 20 examples, comparing PSALM with LLM to baselines.

4.4 Memory of action semantics

We keep a memory of the hypothesized action semantics. For each action's action semantics, we store two lists of predicted statements for preconditions and postconditions. Each statement ϕ is associated with a belief $p(\phi \in \Phi_a|a)$, in short $p(\phi|a)$. We use this belief as a binary probability for each statement to sample, at each iteration, the concrete action semantics input to the symbolic solver.

The first time a statement ϕ is predicted as part of the action semantics for a, the belief will be assigned to 1. Afterwards, this belief will be updated following an exponential forgetting rule. Suppose at time step t, the predicted action semantics of a is $\hat{\Phi}_{a,t}$, the belief update rule of statement ϕ is

$$p_{t+1}(\phi|a) = \begin{cases} \mathbb{1}[\phi \in \hat{\Phi}_{a,t+1}] & \text{if } p_t(\phi|a) = 0 \\ \gamma_\phi p_t(\phi|a) + (1-\gamma_\phi)\mathbb{1}[\phi \in \hat{\Phi}_{a,t+1}] & \text{else} \end{cases}$$

where γ_ϕ is the forgetting factor. If a statement ϕ has only been predicted by the LLM. Once a statement ϕ is predicted by rule-based generator, $\gamma_\phi = 1$, which does not decay. Notice that all the statements predicted in the current time step and in the memory $\phi \in \bigcup_{t' \in \{1..t+1\}} \hat{\Phi}_{a,t'}$ will be updated following the rule.

5 Experiments

We show that PSALM can recover 100% the action semantics in 7 symbolic domains, resulting in 100% planning success rate as well. The critical advantage the LLM component of PSALM is to greatly reduce the cost, *i.e.*, the number of resets and the number of executed steps, for achieving the iterative recovery of these domain semantics.

5.1 Experimental setups

We experiment on 7 symbolic domains from International Planning Competitions (Seipp et al., 2022); each defines 20 tasks that vary in number of environment objects and optimal plan length. **(1) BARMAN**: The robot is a bartender with 2 hands preparing cocktails for a customer's order, using the specified ingredients and appropriate tools; **(2) BLOCKSW**: The robot reorganizes a collection of block piles arranged on a table, into a specified configuration while adhering to the simple physics principles; **(3) FLOORTILE**: A set of robots painting color patterns on floor tiles, allowed to move around but not to step on painted tiles; **(4) GRIPPERS**: A set of robots with 2 grippers each are given a task to move objects among different rooms; **(5) STORAGE**: The robot lifts and drops crates initially stored in different areas, into a depot, using a given set of hoists; **(6) TERMES**: The robot constructs complex structures by transporting and positioning blocks, as well as using them as a means to move adjacent blocks; **(7) TYREWORLD**: The robot is assigned with changing flat tires, which involves tasks such as removing flat tires, inflating the intact tires, tightening nuts, and returning tools to the boot.

In each domain, we choose the first task that requires using all actions to reach the goal, learn action semantics from that task, then test on all 20 tasks. All experiments use the FAST-DOWNWARD (Helmert, 2006) planner, with a search time limit of $W = 30$ seconds during the PSALM loop, and unlimited search time during testing (Table 2). We use VAL[1] as simulation environment for plan validation, calculating state condition changes, and obtaining error messages. We set the maximum number of PSALM induction iterations to 1k for pure random baselines and to 100 for any method involving LLMs. We report the average over 3 runs for all the methods.

For the main results, we use GPT-4-Turbo (OpenAI, 2023) as the language model agent, with temperature 0 and one-shot prompting following Liu et al. (2023). We detail prompts in Appendix D. We use $v = 10$ prospection steps, $l = 1$ sampled trajectory, $k = 3$ candidate trajectories, $g = 5$ failed trajectories per iteration, $\gamma_\phi = 0.8$ memory forgetting factor. We perform ablation studies over these hyperparameters in the analysis. More experimental details are in Appendix B.

[1] https://github.com/KCL-Planning/VAL

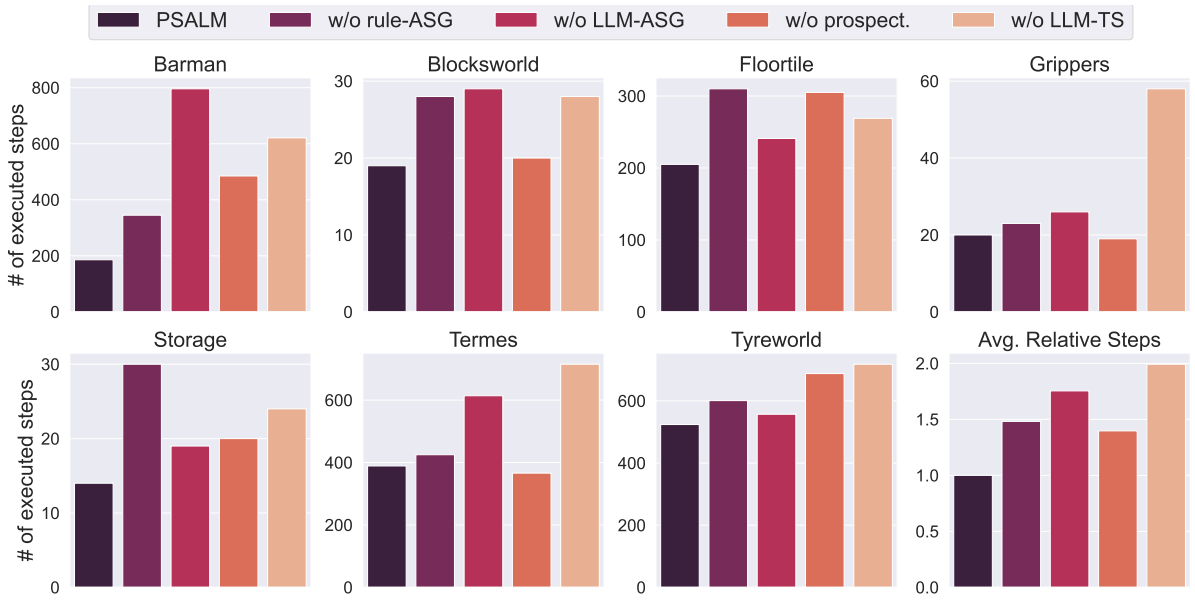


Figure 4: We compare PSALM with multiple variations over 7 domains. We report on NES and the results suggest (1) LLM as a trajectory sampler greatly reduces the execution steps; (2) LLM and rule-based action semantics generators have complementary benefits; and (3) Prospection to reject trajectories based on current action semantics hypotheses is helpful overall. TS is short for trajectory sampler and ASG is short for action semantics generator.

5.2 Comparison with baselines

As shown in Table 2, PSALM achieve 100% planning success rate, outperforming multiple LLM direct plan generation baselines. Other domain induction baselines using LLMs, such as Guan et al. (2023), are unable induce correct action semantics because they do not leverage environment feedback. Direct prompting for the plan of the task, even with powerful LMM models, O1-preview and Clalude-3.5 Sonnet, cannot solve any task in the complex domains such as BARMAN or TERMES.

Table 3 shows that without environment feedback, GPT-4 and Guan et al. (2023) can predict part of the action semantics of the environment, but cannot recover the full action semantics correctly. Given only partially correct action semantics, the solver will either fail to find a plan, or generate a plan that fails when executed. We provide the prompt templates for plan generation and domain induction as in Appendix D.

5.3 Ablations on using LLM and prospection

Figure 4 compares PSALM with multiple baselines over 7 domains, including no rule-based action semantics generator (w/o rule-ASG), no LLM action semantics generator (w/o LLM-ASG), no prospection (w/o prospect.) and use random trajectory sampler (w/o LLM-TS). Each achieves 100% Acc, so we focus on number of executed steps (NES).

LLM as a sampler greatly reduces the execution steps. We find that the LLM trajectory sampler reduces the execution steps over all domains. The random sampler, even with prospection, lacks the commonsense reasoning knowledge for choosing trajectories likely to be solutions to the problem.

LLM and rule-based action semantics predictions have complementary benefits. Comparing the purple bars (w/o rule-ASG) and the magenta bars (w/o LLM-ASG), rule-based action semantics generator is more important in FLOORTILE, STORAGE and TYREWORLD, while LLM action semantics generator is more important in the other four domains. With the commonsense prior knowledge provided by the LLM, and the exactness of the information from the rule-based parser, combining both predictions is always a better solution.

Prospection induces redundant actions sometimes, but is necessary overall. For certain domains with a small number of actions $|\mathcal{A}|$, such as GRIPPERS ($|\mathcal{A}| = 3$) and TERMES ($|\mathcal{A}| = 7$), a large number of prospection steps slows search. However, prospection is overall helpful. The w/o prospect. bar shows around 1.4 times execution steps on average *vs*. PSALM. Especially, in the BARMAN domain, prospection reduces more than a half of the execution steps.

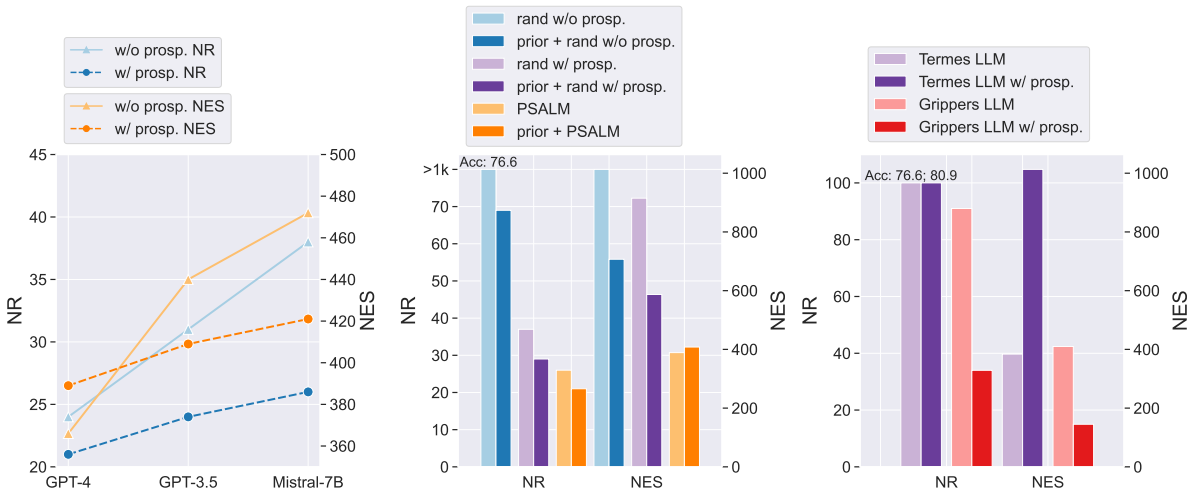


Figure 5: Additional analysis for PSALM. (Left) We vary the type of LLM and show that PSALM works with GPT-3.5 and Mistral-7B on the TERMES domain. (Middle) Using the LLM prior before trajectory sampling (darker bars) enables the random baselines to work better compared to not having the prior (lighter bars), though it can adversely affect the full PSALM method. (Right) Experiments where we remove the error message from input to the LLM action semantics generator. Without error messages, PSALM works only on easy domains. For the experiments that fail to find a solution of the problem, we show the action semantics accuracy on top of the bar.

5.4 Analysis

PSALM can work on less powerful and public LLMs. Figure 5 left shows PSALM can work with 100% plan success rate on GPT-3.5 and Mistral-7B, though with more resets (NR) and number of execution steps (NES). Prospection reduces the gap between the different LLMs.

LLM prior enables random baselines. We experiment with using the LLM to predict the action semantics $\hat{\Phi}_{0,a}$ for each action a, before having any sampled trajectory, and then start the iteration with the memory of the dynamics model initialized from $\hat{\Phi}_{0,a}$. This prediction is solely based on the name of the action and its parameter. Figure 5 middle shows on the TERMES domain the LLM prior can enable the random baseline without prospection to work within 1k resets, and greatly reduce NR and NES for the random baseline with prospection. However, on more complex domains such as BARMAN, the LLM prior does not enable the random baselines, *i.e.*, failing to find a solution within 1k resets, with and without prospection. The finding implies we can perform domain induction on some easier domains with limited access, or even one call, of language models. On the other hand, this naive LLM prior is harmful to PSALM, because it inserts noisy predictions of action semantics which requires additional resets to erase.

Error messages are crucial for domain induction. Error messages are obtainable from simulation environments, or in real world deployment if human operators are available to describe failure. We study whether PSALM can succeed without receiving error messages about failed actions. Note that without the error message, the action semantics generator can only be LLM-based and cannot utilize rule-based inference. Figure 5 right shows that on slightly complex domains like TERMES, PSALM cannot predict the correct dynamics model, *i.e.*, the accuracy is not 100. PSALM works on very simple domains like GRIPPERS, but it requires 7x more NES to learn. Moreover, prospection is essential when we have no error message.

6 Conclusion

In conclusion, we propose a novel domain induction problem in PDDL, where agents automatically infer the action semantics for a new domain without human annotation. We introduce a simple but strong framework PSALM, which combines the commonsense reasoning ability of large language models with the precision of symbolic solvers for domain induction. To update the action semantics in a memory, PSALM uses LLMs as agents for trajectory sampling and action semantics prediction. We demonstrate the effectiveness and efficiency of leveraging LLMs with environment feedback for domain induction over 7 domains.

Limitations

Though PSALM shows to be relatively robust across multiple LLMs, we do not claim that PSALM can work with *any LLM*. Additionally, the error message, which might be hard to get from some real-world environment, is crucial in the PSALM framework given current LLM reasoning abilities. Future works may explore methods without the error message as the input. To apply the domain induction setup to the real world, the environment predicates and object types have to be annotated or derived from the environment. The domain induction problem without these annotations is more challenging, but expert definitions of these domain components represent substantially less effort than fully specifying action pre- and post-conditions. We will leave this extension for future studies.

References

Saeid Amiri, Sujay Bajracharya, Cihangir Goktolgal, Jesse Thomason, and Shiqi Zhang. 2019. Augmenting knowledge through statistical, goal-oriented human-robot dialog. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Anthropic. 2024. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet.

Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc M ´ Etivier, and Sylvie Pesty. 2018. A Review of Learning Planning Action Models. *Knowledge Engineering Review*, 33.

Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. 2011. Answer set programming at a glance. *Commun. ACM*.

Jaime Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton, and Manuela Veloso. 1991. Prodigy: An integrated architecture for planning and learning. *SIGART Bull.*

Guanqi Chen, Lei Yang, Ruixing Jia, Zhe Hu, Yizhou Chen, Wei Zhang, Wenping Wang, and Jia Pan. 2024. Language-augmented symbolic planner for open-world task planning. In *Robotics: Science and Systems Conference (RSS)*.

Richard E. Fikes and Nils J. Nilsson. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*.

Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. PDDL - the planning domain definition language. *Technical Report, Tech. Rep.*

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Muzhi Han, Yifeng Zhu, Song-Chun Zhu, Ying Nian Wu, and Yuke Zhu. 2024. Interpret: Interactive predicate learning from language feedback for generalizable task planning. In *Robotics: Science and Systems (RSS)*.

Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. 2024. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20123–20133.

Malte Helmert. 2006. The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246.

Zichao Hu, Francesca Lucchetti, Claire Schlesinger, Yash Saxena, Anders Freeman, Sadanand Modak, Arjun Guha, and Joydeep Biswas. 2024. Deploying and evaluating llms to program service mobile robots. *IEEE Robotics and Automation Letters*, 9(3):2853–2860.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, Proceedings of Machine Learning Research.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. 2022. Do as i can, not as i say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. 2024. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Forty-first International Conference on Machine Learning*.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy

Zeng. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*.

Vladimir Lifschitz. 2002. Answer set programming and plan generation. *Artificial Intelligence*.

Xinrui Lin, Yangfan Wu, Huanyu Yang, Yu Zhang, Yanyong Zhang, and Jianmin Ji. 2024. CLMASP: Coupling large language models with answer set programming for robotic task planning. *ArXiv*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. LLM+P: Empowering large language models with optimal planning proficiency. *ArXiv preprint*.

Angelos Mavrogiannis, Christoforos Mavrogiannis, and Yiannis Aloimonos. 2023. Cook2ltl: Translating cooking recipes to ltl formulae using large language models. *arXiv preprint arXiv:2310.00163*.

Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. 2003. Shop2: An htn planning system. *J. Artif. Intell. Res. (JAIR)*.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. 2023. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5153–5176, Singapore. Association for Computational Linguistics.

OpenAI. 2023. Gpt-4 technical report. *ArXiv preprint*, abs/2303.08774.

OpenAI. 2024. Introducing openai o1-preview. https://openai.com/index/introducing-openai-o1-preview.

James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. 2024. Large language models as planning domain generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 423–431.

Vittorio Perera, Robin Soetens, Thomas Kollar, Mehdi Samadi, Yichao Sun, Daniele Nardi, René van de Molengraft, and Manuela Veloso. 2015. Learning task knowledge from dialog and web access. *Robotics*.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4226–4252, Mexico City, Mexico. Association for Computational Linguistics.

Gabriel Sarch, Yue Wu, Michael Tarr, and Katerina Fragkiadaki. 2023. Open-ended instructable embodied agents with memory-augmented large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*.

Jendrik Seipp, Álvaro Torralba, and Jörg Hoffmann. 2022. PDDL generators.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. 2023. Generalized planning in pddl domains with pretrained large language models. In *AAAI Conference on Artificial Intelligence*.

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

Pavel Smirnov, Frank Joublin, Antonello Ceravola, and Michael Gienger. 2024. Generating consistent pddl domains with large language models. *ArXiv preprint*.

Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J. Lim. 2021. Learning to synthesize programs as interpretable and generalizable policies. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models - a critical investigation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Li Siang Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S. Siegel, Jiahai Feng, Noa Korneev, Joshua B. Tenenbaum, and Jacob Andreas. 2023. Learning adaptive planning representations with natural language guidance. In *The International Conference on Learning Representations (ICLR)*.

Li Zhang, Peter Jansen, Tianyi Zhang, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024. PDDLEGO: Iterative planning in textual environments. In *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics (*SEM 2024)*, pages 212–221, Mexico City, Mexico. Association for Computational Linguistics.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. In *Advances in Neural Information Processing Systems*, volume 36, pages 31967–31987. Curran Associates, Inc.

Wang Zhu, Jesse Thomason, and Robin Jia. 2023. Chain-of-questions training with latent answers for robust multistep question answering. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore. Association for Computational Linguistics.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael L. Littman, and Stephen H. Bach. 2024. Planetarium: A rigorous benchmark for translating text to structured planning languages. *arXiv*.

A More Related Works

As shown in Table 1, the previous works have not leveraged environment feedback and guaranteed plan success as PSALM. To discuss the related works comprehensively in using LLM in planning and PDDL generation over many setups, we extend the related work as in Table 4.

A.1 LLM as an Idea Generator in Planning

LLMs are widely used as idea generators, not only in planning, but also in other reasoning areas, such as first-order logic (Olausson et al., 2023), multistep reasoning (Zhu et al., 2023), decomposition (Prasad et al., 2024) and formal language (Mavrogiannis et al., 2023).

The major difference between LLM idea generators in planning is whether LLMs generate ideas on the plan itself or on the action semantics. Previous works explore different ways of using LLMs to generate the plan itself, such as Monte Carlo tree search over possible plans (Hazra et al., 2024), or applying other verifier or postprocessor after the plan generation (Lin et al., 2024). These approaches are better than the direct prompting using LLM, while they still cannot guarantee plan success.

Alternatively, using LLMs to generate action semantics followed by a symbolic planner often results in an all-or-nothing issue during plan generation. Either the action semantics are fully captured, leading to a successful plan, or the planner fails to produce a correct plan for the domain at all. Most previous works focus on error analysis. For instance, Smirnov et al. (2024) attempts to generate syntactically correct but not semantically guaranteed PDDL domain using LLMs, by exhaustively listing syntax errors in the prompt to correct the generated domains. Oswald et al. (2024) utilizes LLM to generate PDDL actions given detailed text action descriptions and analysis of the errors. Other works like InterPreT (Han et al., 2024) leverages language feedback from human during embodied interaction for domain predicates and action semantics prediction. They explore complex visual environments and show how to accomplish certain human specified goal with robots, such as "stack red block on coaster". Unlike all previous works, PSALM shows how we can achieve 100% success rate on a certain type of task in the domain, by using LLMs to learn the action semantics from one example.

A.2 LLM for PDDL Generation

There are two types of PDDL generation for LLM, generating the PDDL problem file or the domain file.

Generating the problem file requires to get access to the domain file, recent works such as LLM+P (Liu et al., 2023), PDDLEGO (Zhang et al., 2024) demonstrate the effectiveness of LLM to generate problem file for simple domains. The Planetarium (Zuo et al., 2024) benchmark provides large-scale study of the LLM capability in generating PDDL problem files.

Unlike the success on generating problem file, LLMs, even the most powerful ones, performs badly on generating domain file (Smirnov et al., 2024; Oswald et al., 2024). We formally define the domain induction problem, its evaluation measures, and provide a first-step solution PSALM to predicting the action semantics in the domain file in the text-based environments.

A.3 LLM-based Memory Augmentation

Knowledge acquisition for task planning through dialog and web access has been studied in the past. Previous works (Perera et al., 2015; Amiri et al., 2019) construct a knowledge base in an open domain to refer to for grounding user utterance. Recent works have shown LLM based memory augmentation to be effective. Sarch et al. (2023) builds a memory of language instruction and corresponding plan to retrieve from for prompting the LLM with a new instruction, where the retrieved interaction might inform planning. PSALM build probabilistic memory storing the belief for domain action semantics prediction.

B Experiment Details

B.1 Task Selection and Domain Continual Learning

For learning action semantics, we choose the first task that requires using all actions to reach the goal. Specifically, for FLOORTILE, BARMAN, TERMES and TYREWORLD domains, we choose task #1. For GRIPPERS and BLOCKSW domains, we choose task #2. For STORAGE domain, we choose task #3, out of the 20 tasks.

If PSALM learns from a task that only requires partial actions to reach the goal. The planner may be able to generate successful plans for that tasks without knowing the entire domain. To overcome this issue, we also experiment on a setup where

	Goal	Independent of			Role of LLM	Final planner
		Partial AS	Valid plans	Human eval		
Liu et al. (2023)	PF	✗	✓	✓	PF generator	-
Zhang et al. (2024)	PF	✓	✓	✓	PF generator	-
Hazra et al. (2024)	Plan	✓	✓	✓	Plan sampler	Symbolic + LLM
Valmeekam et al. (2023)	Plan	✓	✓	✗	Plan sampler	LLM
Lin et al. (2024)	Plan	✓	✓	✗	Plan sampler	Symbolic + LLM
Silver et al. (2023)	Program	✗	✓	✓	Code generator	LLM
Arora et al. (2018)	AS	✓	✗	✓	-	-
Wong et al. (2023)	AS	✗	✓	✓	AS generator	Symbolic + LLM
Guan et al. (2023)	AS	✓	✓	✗	AS generator	Symbolic / LLM
Oswald et al. (2024)	AS	✓	✓	✓	AS generator	Symbolic
Han et al. (2024)	AS	✓	✓	✗	PF generator + AS generator	Symbolic
PSALM	AS	✓	✓	✓	Plan sampler + AS generator	Symbolic

Table 4: Comparing the PSALM domain induction task setup to representative related works in LLM and LLM-Modulo planning; here, "PF" is short for problem file and "AS" is short for action semantics. We consider the final planner using tree or heuristic search as symbolic.

PSALM learns from task #1 for all domains and perform a continual learning setup. We report the learning process as below. Instead of directly learning the entire STORAGE domain from task #3 with 14 executed steps, we start the learning from task #1 with 13 executed steps, and recover 78.9% of the action semantics conditions. Then, based on what we learned from task #1, it takes 10 steps to recover 81.6% of the action semantics conditions learning from task #2, and 11 more steps to recover the entire domain from task #3. The continual learning process can be applied to real world exploration of certain domain as well. Thus, the "success without learning the entire domain" effect is not a limitation but a benefit of PSALM.

B.2 Task Example Initial and Goal States

We provide the visualization of the initial and goal state for the selected training example per domain. The natural language description of the actions are from Liu et al. (2023).

BARMAN. Figure 6 shows the initial and goal state for task #1. The domain allows 12 actions: grasp a container, leave a container on the table, fill a shot glass with an ingredient, refill a shot glass with an ingredient, empty a shot glass, clean a shot glass, pour an ingredient from a shot glass to a clean shaker, pour an ingredient from a shot glass to a used shaker, empty a shaker, clean a shaker, shake a cocktail in a shaker, and pour from a shaker to a shot glass.

BLOCKSW. Figure 7 shows the initial and goal state for task #2. The domain allows 4 actions: pick up a block from the table, put down a block on the table, stack a block on top of another block, unstack a block from on top of another block.

FLOORTILE. Figure 8 shows the initial and goal state for task #1. The domain allows 7 actions: change the spray gun color paint the tile that is up from the robot, paint the tile that is down from the robot, move up, move down, move right, move left.

GRIPPERS. Figure 9 shows the initial and goal state for task #2. The domain allows 3 actions: move from one room to another, pick up an object using the gripper, drop an object that it is carrying.

STORAGE. Figure 10 shows the initial state for task #3. The goal state is to move the crate to any location in the hoist store areas. The domain allows 5 actions: lift a crate using a hoist from a store area to an area, drop a crate from the hoist onto a surface in a store area, move a hoist from one store area to another connected store area, move a hoist from a store area to a transit area, move a hoist from a transit area to a store area.

TERMES. Figure 10 shows the initial and goal state for task #1. The goal state shows the height of the blocks in each position. The domain allows 7 actions: move from a position to another, move up from a position to another, move down from a position to another, place a block at a neighboring position from the robot's current position, remove a block at a neighboring position from the robot's

current position, create a block at the depot, destroy a block at the depot.

TYREWORLD. Figure 12 shows the initial and goal state description for task #1. The domain allows 13 actions: open a container, close a container, fetch an object inside a container, put an object into a container, loosen a nut on a hub, tighten a nut on a hub, jack-up a hub, jack-down a hub, unfasten a nut on a hub, fasten a nut on a hub, remove a wheel from a hub, put a wheel onto a hub, inflate a wheel using a pump.

B.3 Other Hyperparameters and Details

The total cost of the API calls of PSALM over 7 domains is around $80 for 1 run. We do not put the pure random baselines in Figure 4, because none of the pure random baselines complete any task within 1k iterations.

Besides, we only use the cheapest configuration of FAST-DOWNWARD, *i.e.*, we are not using very smart symbolic solver. FAST-DOWNWARD has various advanced search configurations including using A* search (`seq-opt-lmcut`), using heuristics plus hill climbing algorithms (`seq-sat-fd-autotune-1`). We use the `lama-first` configuration, which is close to lazy greedy search designed to find solutions quickly without much regard for plan cost.

The IPC domains and tasks (Seipp et al., 2022) are licensed under the MIT License. The FAST-DOWNWARD planner is distributed under the GNU GPLv2 license, while the VAL validator is licensed under the BSD 3-Clause, with copyright held by the University of Strathclyde, King's College London, and Schlumberger Ltd. Our use of the dataset and software complies with the terms of their respective licenses.

C Ablation studies

We perform ablation studies on prospection steps, number of sampled trajectories, number of candidate trajectories, and number of failed trajectories per run, to show their influence on the PSALM framework. The results validate our choice of the hyperparameters in the main results.

Few prospection steps is enough for PSALM. We vary prospection steps v from $0, 1, 5, 10$ in Figure 13a. Notice the random baseline does not find a solution when $v = 0, 1, 5$, and reaches an Acc of 78.5. We conclude the number of prospection steps matters more when the trajectory sampler is random, while a small number of prospection is enough for language model trajectory sampler.

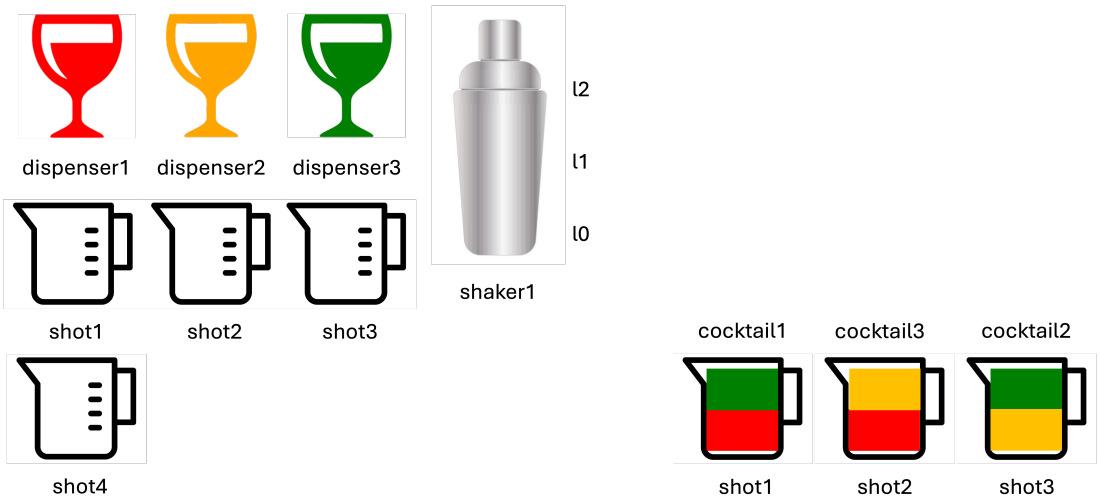
One sampled trajectory per run is enough. We vary the number of sampled trajectories l from $1, 3, 5$ in Figure 13b. For both LLM and random samplers, We see the total number of steps grows linearly with number of sampled trajectory, which means LLMs are hard to learn more information from just sampling more each run, and one sampled trajectory per run is enough.

More candidate trajectories help on prospection. We vary the number of candidate trajectories k from $0, 1, 3$ in the LLM prompt. The results in Figure 13c suggest including candidate trajectory in the prompt is beneficial, while one candidate trajectory is enough for pure LLM prediction, more candidate trajectories help LLM with prospection. On the other hand, as the candidate trajectories can be very long for certain domain like BARMAN, and thus result in a very long prompt, we do not experiment with more than 3 candidate trajectories in the prompt.

Choose the number of failed trajectories with care. We vary the number of failed trajectories g from $0, 1, 5$ in the LLM prompt. To select g failed trajectories, we first filter trajectories with no less than 3 steps, and then sample g trajectories from the filtered ones for the LLM prompt to avoid generating those trajectories as prefix. The results in Figure 13d suggest that multiple failed trajectories help LLM action semantics prediction, but only one failed trajectory could be harmful. We hypothesize the one failed trajectory in the input might distract the model from following the candidate trajectories.

D LLM Prompts

We list the prompt template for LLM trajectory sampler (Figure 14) and LLM action semantics predictor (Figure 15), and provide one complete example for each. Figure 16, 17 is the example for trajectory sampler, and Figure 18, 19 is the example for action semantics predictor.

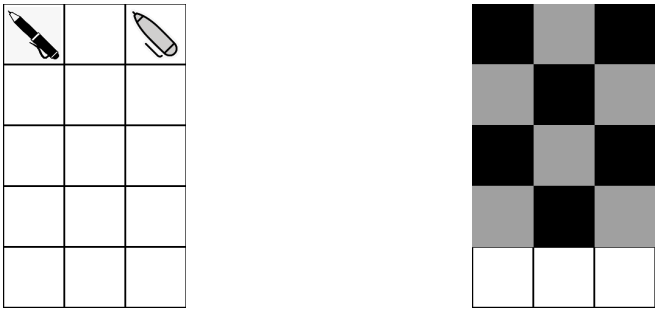


(a) (b)

Figure 6: BARMAN task #1 initial state (a) and goal state (b).

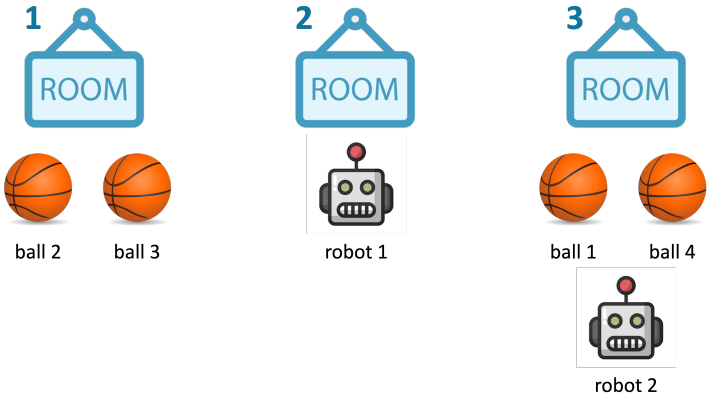
(a) (b)

Figure 7: BLOCKSW task #2 initial state (a) and goal state (b).



(a) (b)

Figure 8: FLOORTILE task #1 initial state (a) and goal state (b).



(a)

(b)

Figure 9: GRIPPERS task #2 initial state (a) and goal state (b).

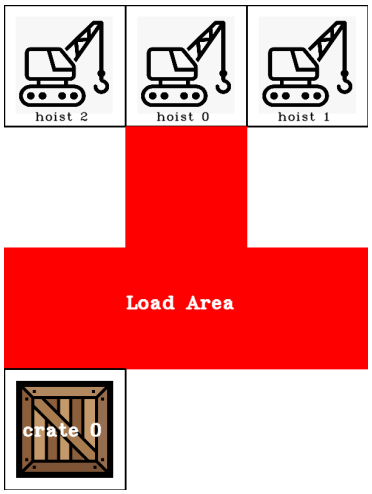


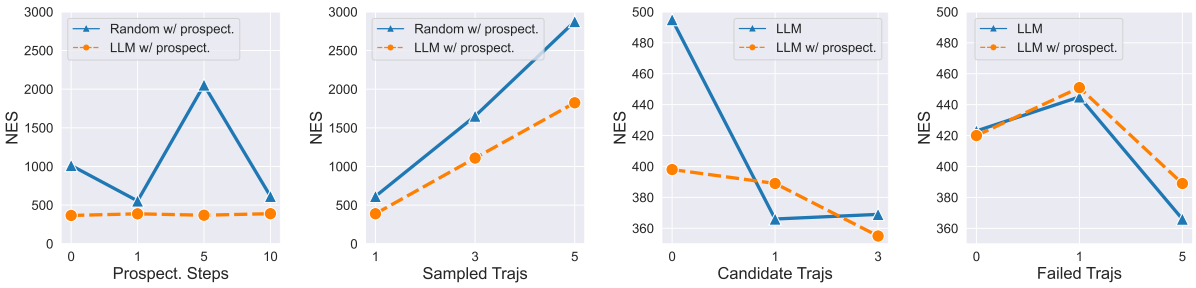
Figure 10: STORAGE task #3 initial state.

(a) (b)

Figure 11: TERMES task #1 initial state (a) and goal state (b).

```
You have a jack, a pump, a wrench, a boot, 1 hubs, 1 nuts, 1 flat tyres, and 1 intact tyres.
The jack, pump, wrench, and intact tyres are in the boot.
The boot is unlocked but is closed.
The intact tyres are not inflated.
The flat tyres are on the hubs.
The hubs are on the ground.
The nuts are tight on the hubs.
The hubs are fastened.
Your goal is to replace flat tyres with intact tyres on the hubs. Intact tyres should be inflated.
The nuts should be tight on the hubs. The flat tyres, wrench, jack, and pump should be in the boot. The boot should be closed.
```

Figure 12: TYREWORLD task #1 initial state and goal state description.



(a) (b) (c) (d)

Figure 13: Ablation studies for PSALM. (a). Varying the number of prospection steps v. Few prospection steps is enough for PSALM. (b). Varying the number of sampled trajectories l. One sampled trajectory per run is enough. (c). Varying the number of candidate trajectories k passed from the symbolic solver to the LLM trajectory sampler. More candidate trajectories help when prospection is used. (d). Varying the number of failed trajectories g shown to the LLM trajectory sampler. A certain amount of failed trajectories is required for the LLM.

[[domain_nl]]

The environment is defined with the Planning Domain Definition Language (PDDL) as
[[domain_type_pred]]
Here are the actions you can perform in the environment:
[[action_description]]

An example planning problem is:
[[context_nl]]

which means the initial state is:
[[context_init]]

A plan for the example problem is:
[[context_sol]]

Now I have a new planning problem and its description. Can you provide a executable plan, in the way of a sequence of behaviors, to solve the problem?

The new planning problem is:
[[task_nl]]

which means the initial state is:
[[task_init]]

[[cand_action_list]][[failed_traj]]Please provide the list of actions in the format as described in the example.
Feel free to use try other actions not in the example when their preconditions are satisfied.
Please output a clean, complete list of actions, including the option and the starting actions you choose, to solve the problem.
Don't use the word repeat in the response.

A plan for the new planning problem is:

Figure 14: LLM trajectory sampler prompt template

[[domain_nl]]

The current guess of the preconditions and effects for actions are:
[[action_description]]

Please infer missing or correct wrong [[postfix]] for action [[action]] in CNF, given the PDDL types and predicates:
[[domain_type_pred]]

To help you predict the [[postfix]], we provide a running example in this domain. For the problem:
[[task_nl]]

The initial state of the task in PDDL is:
[[task_init]]

A plan (list of actions) runs in the simulator like:
[[trajectory]]

Please use the information from the simulator output carefully, and output the CNF in a clean format of "(and (...) (...))", where each parentheses contains some predicates.
Note that you should not specify types, e.g., "- object", in preconditions and effects.

The [[postfix]] for [[action]] are:

Figure 15: LLM action semantics predictor prompt template

You are a robot with a gripper that can move objects between different rooms.

The environment is defined with the Planning Domain Definition Language (PDDL) as
PDDL Types:
room object robot gripper
PDDL Predicates:
(at-robby ?r - robot ?x - room)
(at ?o - object ?x - room)
(free ?r - robot ?g - gripper)
(carry ?r - robot ?o - object ?g - gripper)

Here are the actions you can perform in the environment:
Move from one room to another: move (?r - robot ?from ?to - room). The effects are (at-robby ?r ?to), (not (at-robby ?r ?from)).
Pick up an object using the gripper: pick (?r - robot ?obj - object ?room - room ?g - gripper). The preconditions are (at-robby ?r ?room), (at ?obj ?room), (free ?r ?g). The effects are (not (free ?r ?g)), (not (at ?obj ?room)), (carry ?r ?obj ?g).
Drop an object that it is carrying: drop (?r - robot ?obj - object ?room - room ?g - gripper). The preconditions are (carry ?r ?obj ?g). The effects are (at ?obj ?room), (free ?r ?g), (not (carry ?r ?obj ?g)).

An example planning problem is:
You control 2 robots, each robot has a left gripper and a right gripper.
There are 4 rooms and 6 balls.
robot2 is in room3. robot1 is in room2.
ball1 is in room3. ball2 is in room1. ball3 is in room3. ball4 is in room2. ball5 is in room4. ball6 is in room4.
The robots' grippers are free.
Your goal is to transport the balls to their destinations.
ball1 should be in room4.
ball2 should be in room1.
ball3 should be in room1.
ball4 should be in room2.
ball5 should be in room1.
ball6 should be in room1.

which means the initial state is:
((at-robby robot1 room2)
(free robot1 rgripper1)
(free robot1 lgripper1)
(at-robby robot2 room3)
(free robot2 rgripper2)
(free robot2 lgripper2)
(at ball1 room3)
(at ball2 room1)
(at ball3 room3)
(at ball4 room2)
(at ball5 room4)
(at ball6 room4))

A plan for the example problem is:
(pick robot2 ball1 room3 lgripper2)
(pick robot2 ball3 room3 rgripper2)
(move robot2 room3 room1)
(drop robot2 ball3 room1 rgripper2)
(move robot2 room1 room4)
(pick robot2 ball5 room4 rgripper2)
(drop robot2 ball1 room4 lgripper2)
(pick robot2 ball6 room4 lgripper2)
(move robot2 room4 room1)
(drop robot2 ball6 room1 lgripper2)
(drop robot2 ball5 room1 rgripper2)

Now I have a new planning problem and its description. Can you provide a executable plan, in the way of a sequence of behaviors, to solve the problem?

The new planning problem is:
You control 2 robots, each robot has a left gripper and a right gripper.
There are 3 rooms and 4 balls.
robot2 is in room3. robot1 is in room2.
ball1 is in room3. ball2 is in room1. ball4 is in room3. ball3 is in room1.
The robots' grippers are free.
Your goal is to transport the balls to their destinations.
ball1 should be in room2.
ball2 should be in room2.
ball3 should be in room3.
ball4 should be in room3.

which means the initial state is:
((at-robby robot1 room2)
(free robot1 rgripper1)
(free robot1 lgripper1)
(at-robby robot2 room3)
(free robot2 rgripper2)
(free robot2 lgripper2)
(at ball1 room3)
(at ball2 room1)
(at ball3 room1)
(at ball4 room3))

Figure 16: LLM trajectory sampler prompt example

We provide some options of actions to start from.
Please choose ONLY one option, and continue the list of actions to solve the task.
Option 0:
(move robot2 room3 room1)
(pick robot2 ball2 room1 lgripper2)
(pick robot2 ball3 room1 rgripper2)
(move robot2 room1 room3)
Option 1:
(move robot2 room3 room1)
(pick robot2 ball2 room1 lgripper2)
(pick robot2 ball3 room1 rgripper2)

Here we listed the previous explored but failed trajectories. Please AVOID generating the same trajectories.
Failed traj 0:
(pick robot2 ball1 room3 rgripper2)
(move robot2 room3 room2)
(drop robot2 ball1 room2 rgripper2)
(move robot2 room2 room3

(pick robot2 ball4 room3 lgripper2)
(move robot2 room3 room2)
(drop robot2 ball4 room2 lgripper2)
(move robot1 room2 room1)
(pick robot1 ball2 room1 rgripper1)
(move robot1 room1 room2)
(drop robot1 ball2 room2 rgripper1)
(move robot1 room2 room1)
(pick robot1 ball3 room1 lgripper1)
(move robot1 room1 room3)
(drop robot1 ball3 room3 lgripper1)
Plan Repair Advice: The goal is not satisfied
(Set (at ball4 room3) to true)
Failed traj 1:
(pick robot2 ball1 room3 rgripper2)
(pick robot2 ball4 room3 lgripper2)
(move robot2 room3 room2)
(drop robot2 ball1 room2 rgripper2)
(drop robot2 ball4 room2 lgripper2)
(move robot1 room2 room1)
(pick robot1 ball2 room1 rgripper1)
(pick robot1 ball3 room1 lgripper1)
(move robot1 room1 room3)
(drop robot1 ball3 room3 lgripper1)
(move robot1 room3 room2)
(drop robot1 ball2 room2 rgripper1)
Plan Repair Advice: The goal is not satisfied
(Set (at ball4 room3) to true)

Please provide the list of actions in the format as described in the example.
Feel free to use try other actions not in the example when their preconditions are satisfied.
Please output a clean, complete list of actions, including the option and the starting actions you choose, to solve the problem.
Don't use the word repeat in the response.

A plan for the new planning problem is:

Figure 17: LLM trajectory sampler prompt example (cont')

You control a robot that can take the following actions to build complex structures.

The current guess of the preconditions and effects for actions are:
Move from a position to another.: move (?from - position ?to - position ?h - numb). The preconditions are (NEIGHBOR ?from ?to), (at ?from), (height ?from ?h). The effects are (not (at ?from)), (at ?to).
Move up from a position to another.: move-up (?from - position ?hfrom - numb ?to - position ?hto - numb). The preconditions are (at ?from), (height ?from ?hfrom), (SUCC ?hfrom ?hto), (NEIGHBOR ?from ?to), (height ?to ?hto), (SUCC ?hto ?hfrom). The effects are (not (at ?from)), (not (height ?from ?hfrom)), (height ?to ?hto), (at ?to), (height ?from ?hfrom), (not (height ?to ?hfrom)), (not (height ?to ?hto)), (height ?to ?hfrom).
Move down from a position to another.: move-down (?from - position ?hfrom - numb ?to - position ?hto - numb). The preconditions are (height ?to ?hto), (at ?from), (height ?from ?hfrom), (SUCC ?hfrom ?hto), (NEIGHBOR ?from ?to), (SUCC ?hto ?hfrom). The effects are (not (at ?from)), (not (height ?from ?hfrom)), (height ?from ?hto), (at ?to), (height ?from ?hfrom), (not (height ?from ?hto)), (not (height ?to ?hfrom)), (height ?to ?hto), (not (height ?to ?hto)), (height ?to ?hfrom).
Place a block at a neighboring position from the robot's current position.: place-block (?rpos - position ?bpos - position ?hbefore - numb ?hafter - numb). The preconditions are (at ?rpos), (SUCC ?hafter ?hbefore), (NEIGHBOR ?rpos ?bpos), (has-block), (height ?bpos ?hbefore). The effects are (not (has-block)), (not (height ?rpos ?hbefore)), (height ?bpos ?hafter), (not (height ?bpos ?hbefore)), (height ?rpos ?hafter).
Remove a block at a neighboring position from the robot's current position.: remove-block (?rpos - position ?bpos - position ?hbefore - numb ?hafter - numb). The preconditions are (at ?rpos), (SUCC ?hafter ?hbefore), (NEIGHBOR ?rpos ?bpos), (height ?bpos ?hbefore), (has-block), (SUCC ?hbefore ?hafter). The effects are (not (has-block)), (height ?bpos ?hafter), (not (height ?bpos ?hbefore)), (has-block).
Create a block at the depot.: create-block (?p - position). The preconditions are (IS-DEPOT ?p), (at ?p). The effects are (has-block).
Destroy a block at the depot.: destroy-block (?p - position). The preconditions are (has-block), (at ?p), (IS-DEPOT ?p). The effects are (not (has-block)).

Please infer missing or correct wrong preconditions for action move in CNF, given the PDDL types and predicates:
```
(:types
    numb - object
    position - object
)
(:predicates
    (height ?p - position ?h - numb)
    (at ?p - position)
    (has-block)
    (SUCC ?n1 - numb ?n2 - numb)
    (NEIGHBOR ?p1 - position ?p2 - position)
    (IS-DEPOT ?p - position)
)
```

To help you predict the preconditions, we provide a running example in this domain. For the problem:
The robot is on a grid with 4 rows and 3 columns.
pos-0-0 pos-0-1 pos-0-2
pos-1-0 pos-1-1 pos-1-2
pos-2-0 pos-2-1 pos-2-2
pos-3-0 pos-3-1 pos-3-2
The robot is at pos-2-0.
The depot for new blocks is at pos-2-0.
The maximum height of blocks is 3.
Your goal is to build blocks so that the height at pos-1-2 is 3.
You cannot have an unplaced block at the end.

The initial state of the task in PDDL is:
```
(:init
    (height pos-0-0 n0)
    (height pos-0-1 n0)
    (height pos-0-2 n0)
    (height pos-1-0 n0)
    (height pos-1-1 n0)
    (height pos-1-2 n0)
    (height pos-2-0 n0)
    (height pos-2-1 n0)
    (height pos-2-2 n0)
    (height pos-3-0 n0)
    (height pos-3-1 n0)
    (height pos-3-2 n0)
    (at pos-2-0)
    (SUCC n1 n0)
    (SUCC n2 n1)
    (SUCC n3 n2)
    (NEIGHBOR pos-0-0 pos-1-0)
    (NEIGHBOR pos-0-0 pos-0-1)
    (NEIGHBOR pos-0-1 pos-1-1)
    (NEIGHBOR pos-0-1 pos-0-0)
    (NEIGHBOR pos-0-1 pos-0-2)
    (NEIGHBOR pos-0-2 pos-1-2)
    (NEIGHBOR pos-0-2 pos-0-1)
    (NEIGHBOR pos-1-0 pos-0-0)
    (NEIGHBOR pos-1-0 pos-2-0)
    (NEIGHBOR pos-1-0 pos-1-1)
    (NEIGHBOR pos-1-1 pos-0-1)
    (NEIGHBOR pos-1-1 pos-2-1)
    (NEIGHBOR pos-1-1 pos-1-0)
    (NEIGHBOR pos-1-1 pos-1-2)
    (NEIGHBOR pos-1-2 pos-0-2)
    (NEIGHBOR pos-1-2 pos-2-2)
    (NEIGHBOR pos-1-2 pos-1-1)
    (NEIGHBOR pos-2-0 pos-1-0)
    (NEIGHBOR pos-2-0 pos-3-0)
    (NEIGHBOR pos-2-0 pos-2-1)
```

Figure 18: LLM action semantics predictor prompt example

```
    (NEIGHBOR pos-2-1 pos-1-1)
    (NEIGHBOR pos-2-1 pos-3-1)
    (NEIGHBOR pos-2-1 pos-2-0)
    (NEIGHBOR pos-2-1 pos-2-2)
    (NEIGHBOR pos-2-2 pos-1-2)
    (NEIGHBOR pos-2-2 pos-3-2)
    (NEIGHBOR pos-2-2 pos-2-1)
    (NEIGHBOR pos-3-0 pos-2-0)
    (NEIGHBOR pos-3-0 pos-3-1)
    (NEIGHBOR pos-3-1 pos-2-1)
    (NEIGHBOR pos-3-1 pos-3-0)
    (NEIGHBOR pos-3-1 pos-3-2)
    (NEIGHBOR pos-3-2 pos-2-2)
    (NEIGHBOR pos-3-2 pos-3-1)
    (IS-DEPOT pos-2-0)
)

A plan (list of actions) runs in the simulator like:
(create-block pos-2-0)
Checking next happening (time 1)
Adding (has-block)

(move pos-2-0 pos-3-0 n0)
Checking next happening (time 2)
Deleting (at pos-2-0)
Adding (at pos-3-0)

(move pos-3-0 pos-3-1 n0)
Checking next happening (time 3)
Deleting (at pos-3-0)
Adding (at pos-3-1)

(move pos-3-1 pos-3-2 n0)
Checking next happening (time 4)
Deleting (at pos-3-1)
Adding (at pos-3-2)

(move pos-3-2 pos-2-2 n0)
Checking next happening (time 5)
Deleting (at pos-3-2)
Adding (at pos-2-2)

(move pos-2-2 pos-1-2 n0)
Checking next happening (time 6)
Deleting (at pos-2-2)
Adding (at pos-1-2)

(place-block pos-1-2 pos-1-2 n0 n1)
Checking next happening (time 7)
Plan failed because of unsatisfied precondition in:
(place-block pos-1-2 pos-1-2 n0 n1)
Plan failed to execute
Plan Repair Advice:
(place-block pos-1-2 pos-1-2 n0 n1) has an unsatisfied precondition at time 7
(Set (neighbor pos-1-2 pos-1-2) to true)

Please use the information from the simulator output carefully, and output the CNF in a clean format of "(and (...) (...))", where each parentheses contains some predicates.
Note that you should not specify types, e.g., "- object", in preconditions and effects.

The preconditions for move is:
```

Figure 19: LLM action semantics predictor prompt example (cont')