ProRAC: A Neuro-symbolic Method for Reasoning about Actions with LLM-based Progression and Search

Haoyong Wu, Yongmei Liu*
Department of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
wuhy259@mail2.sysu.edu.cn; ymliu@mail.sysu.edu.cn

Abstract

Large language models (LLMs) are increasingly used in agentic systems, where reliable planning requires tracking how actions change the world over long horizons. This capability is central to reasoning about actions and change (RAC), which involves checking action executability, updating states, and preserving unaffected facts. However, recent RAC and planning benchmarks show that LLMs still struggle with these basic abilities. Effective neuro-symbolic methods for improving RAC and planning directly over natural language remain underexplored. In this paper, we propose **ProRAC** (**Pro**gression-based **R**easoning about **A**ctions and **C**hange), a neuro-symbolic framework that leverages LLMs to tackle RAC and planning problems through an agentic, modular design. Central to ProRAC is a unified framework that operates in two modes: a reasoning mode that sequentially executes actions to validate states and answer queries, and a searching mode that integrates this progression into an A* search guided by a fine-tuned heuristic model to find plans. Extensive evaluations across multiple RAC and planning benchmarks show that ProRAC achieves competitive performance across different model backbones and task types.[1]

1 Introduction

The development of LLMs is increasingly moving beyond text generation and dialogue toward more complex LLM-based agents (Luo et al., 2025). In these systems, LLMs are expected not only to generate responses, but also to serve as decision-making components that decompose goals, select actions, interact with environments, and revise decisions based on feedback. A central capability for such agents is long-horizon planning (Aghzal et al., 2025), where an agent must achieve a distant goal through a sequence of interdependent actions over many intermediate states, usually under partial observability. Representative systems such as ReAct (Yao et al., 2022), SayCan (Ichter et al., 2023), and LLM-Modulo (Kambhampati et al., 2024) show that LLMs can be coupled with observations, tools, affordance models, or external verifiers to improve planning and action selection. However, all such systems rely on a more basic capability: after each candidate action, the agent must know which facts become true, which become false, and which facts remain unchanged. Without reliable state tracking, a fluent-looking action sequence can quickly become invalid.

This capability is central to reasoning about actions and change (RAC) (McCarthy and Hayes, 1981; Reiter, 2001), a classical problem in artificial intelligence concerned with reasoning about action preconditions, action effects, the evolution of world states, and the composition of actions to reach the goal. For LLM-based agents, RAC captures both step-level state-transition requirements and global search behind long-horizon planning: an agent must check executability, update the state according to action effects, preserve unaffected facts, and use the resulting state to guide later decisions. Therefore, RAC connects to agent planning at both foundational and practical levels. At the foundational level, RAC formalizes the state-transition problems that underlie agent behavior. From a practical perspective, RAC mechanisms such as progression, executability checking, and state-evolution modeling can serve as concrete modules inside modern agent systems.

In recent years, several natural-language benchmarks have been proposed to evaluate RAC and planning abilities (He et al., 2023; Handa et al., 2025; Kokel et al., 2025; Valmeekam et al., 2023). These benchmarks evaluate fundamental RAC abilities such as executability checking, projection,

*Corresponding author.
[1]Code available at: https://github.com/sysulic/ProRAC

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 22456–22474
October 24-29, 2026 ©2026 Association for Computational Linguistics

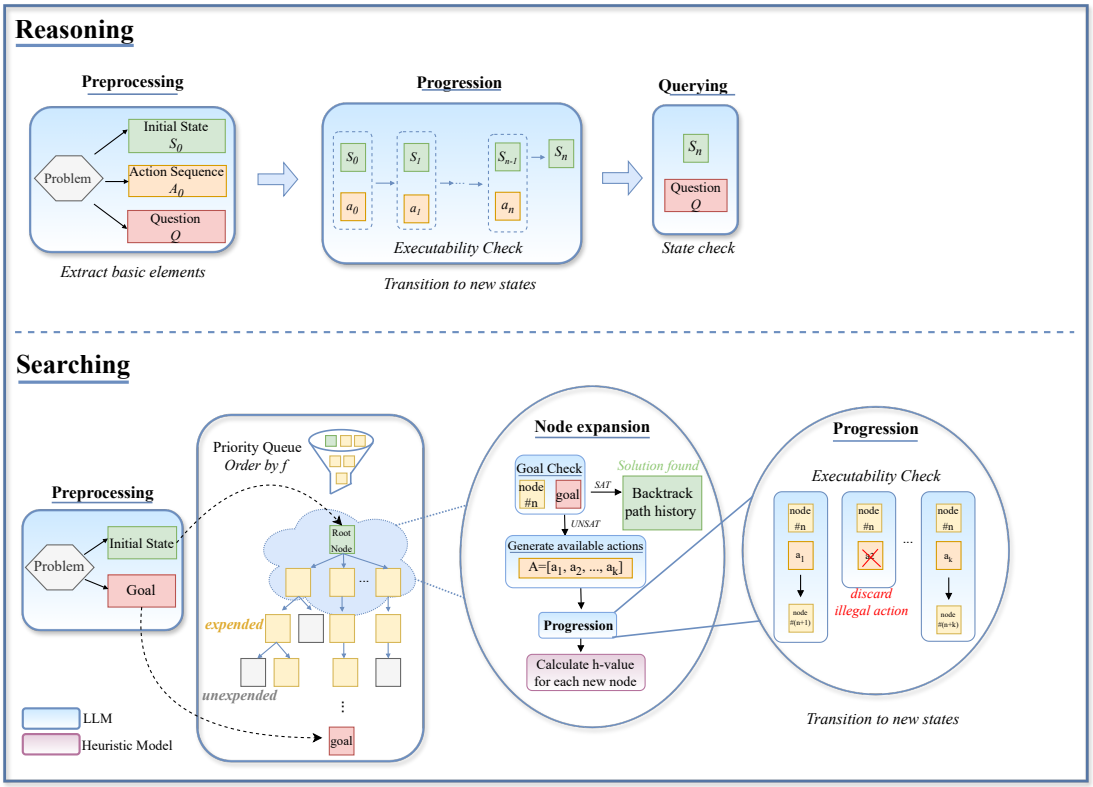


Figure 1: The framework of ProRAC. The framework has two modes: reasoning mode (for RAC tasks) and searching mode (for plan generation tasks).

state tracking, plan verification, and plan generation across different domains and question formats. Results on these benchmarks show that LLMs still perform poorly on these basic RAC capabilities, especially when local state-update errors accumulate over longer action sequences.

To improve the reasoning and planning capabilities of LLMs, several studies have proposed neuro-symbolic methods that combine neural language models with symbolic reasoning mechanisms. Neuro-symbolic AI goes beyond simply combining neural models with symbolic solvers, it covers a spectrum of integration paradigms, from neural models operating over symbolic representations to neural systems that exhibit symbolic-like reasoning behaviors (Kautz, 2022). Kautz distinguishes six architectures, including Neuro[Symbolic], Neuro | Symbolic and Symbolic[Neuro]. Neuro[Symbolic] refers to a primarily neural architecture augmented with symbolic components. Neuro | Symbolic systems use neural models to transform non-symbolic inputs into symbolic representations that are subsequently processed by a symbolic reasoning system. Symbolic[Neuro] embeds neural components within a symbolic problem-solving structure, as illustrated by AlphaGo, where MCTS invokes a neural evaluation function.

Existing neuro-symbolic methods for RAC and planning mainly rely on autoformalization (Ishay and Lee, 2025; Pan et al., 2023) or search (Hao et al., 2023; Yao et al., 2023). Autoformalization methods follow the Neuro | Symbolic paradigm by translating natural-language problems into formal representations for symbolic solvers. However, translation errors, including syntactic failures and semantic mismatches, can invalidate or misdirect subsequent symbolic solving. Search-based methods follow the Symbolic[Neuro] paradigm: an explicit search procedure organizes reasoning while invoking LLMs to perform transitions and estimate the quality of candidate states or partial trajectories. Nevertheless, such prompt-based heuristic estimates can be inaccurate, causing search to prioritize unpromising branches.

To address these limitations, we propose **ProRAC**, a unified neuro-symbolic framework for RAC and planning problems that follows the Symbolic[Neuro] paradigm: Iteration or A* search provides the explicit problem-solving structure, while LLM-based progression, executability checking, and heuristic estimation serve as neural components within it. ProRAC operates in two modes.

In the *reasoning mode*, it executes Preprocessing, Progression, and Querying to track action effects and answer RAC queries. In the *searching mode*, it integrates Progression into A* search and uses a learned heuristic model to guide natural-language planning.

We evaluate ProRAC on several benchmarks across LLMs and domains. Results show that ProRAC achieves competitive results on both RAC and planning tasks, demonstrating that explicit progression improves state reasoning and that learned heuristic guidance makes natural-language-based search more tractable.

To summarize, our main contributions are:

- We propose ProRAC, a framework that addresses both RAC reasoning and planning problems within a progression-based architecture.
- We propose a training paradigm that uses symbolic planners to generate supervision for a lightweight neural heuristic, enabling reliable guidance for natural-language planning.
- We show that ProRAC achieves competitive results compared with existing methods across four benchmarks and multiple task settings.

2 Related Work

Agent-Style LLM Planning A growing body of work studies how LLMs can be used for building agents that reason, act, and revise their decisions through interaction or feedback. ReAct (Yao et al., 2022) interleaves reasoning traces with actions and observations, allowing LLMs to ground intermediate decisions in environment feedback. SayCan (Ichter et al., 2023) combines LLM-generated instructions with learned affordance functions so that robot actions are both semantically plausible and physically executable. LLM-Modulo (Kambhampati et al., 2024) follows a generate-then-verify paradigm in which external critics validate and refine LLM-generated plans. Recently, long-horizon and embodied-agent systems extend this agentic design to web navigation, open-world exploration, and multi-agent robotic cooperation, often combining planning with execution feedback, correction, verification, memory, or communication modules (Erdogan et al., 2025; Nayak et al., 2024; Zhang et al., 2024).

Search-Based LLM Planning Search-based planners further explore multiple candidate trajectories before committing to a solution. Tree of Thoughts (ToT) (Yao et al., 2023) performs tree search over intermediate textual thoughts and uses prompted self-evaluation to prioritize branches. RAP (Hao et al., 2023) formulates reasoning as planning with an LLM-based world model and applies Monte Carlo Tree Search, while LATS (Zhou et al., 2024) combines tree search with environment feedback and self-reflection.

Symbolic Planning and Autoformalization Symbolic methods can reliably solve RAC and planning problems when correct formal specifications are available, but they typically require manually defined predicates, action schemas, and domain rules (Reiter, 2001; Kowalski and Sergot, 1986; Gelfond and Lifschitz, 1993; Giunchiglia and Lifschitz, 1998). Autoformalization methods reduce this burden by using LLMs to translate natural-language problems into formal representations before invoking symbolic solvers (Pan et al., 2023; Olausson et al., 2023; Callewaert et al., 2025; Ishay and Lee, 2025). In particular, LLM+AL translates natural-language planning problems into action-language representations and then employs a symbolic solver to derive solutions. However, LLMs still face significant challenges in producing reliable formal translations, particularly on complex reasoning problems (Zhong et al., 2021).

Calculating Heuristics with LLMs Heuristic guidance is essential for search-based planning. Classical domain-independent heuristics such as h_{add} and h_{max} are derived from delete relaxation (Bonet and Geffner, 1999), but they require formal state representations and cannot be directly applied to natural-language problems. Recent work explores LLM-generated or LLM-refined heuristics (Liu et al., 2024; Ling et al., 2025), but these approaches can require substantial API calls and manually curated heuristic targets.

3 Overview of ProRAC

Figure 1 illustrates the overview of our framework. To address RAC and plan generation problems, the framework operates in two modes: **Reasoning** (the action sequence is given) and **Searching** (heuristic search for complex planning). Crucially, the *Progression* module—originally designed for reasoning, serves as the fundamental state transition operator within the search process.

The entire reasoning/searching process involves three key components: *Preprocessing*, *Progression*, and *Search*/*Querying*. To minimize our frame-

Domain	Model	IO	0-CoT	2-CoT	SC	ProRAC
			TRAC			
Blocksworld	GPT-4o	93.33	93.64	95.55	95.55	**97.77**
	GPT-4o-mini	85.92	82.96	84.44	88.14	**85.92**
	DeepSeek-v3.2	95.55	98.52	98.52	91.85	**100**
			ActionReasoningBench			
Depots	GPT-4o	82.16	67.51	81.43	80.93	**90.77**
	GPT-4o-mini	75.54	73.25	73.72	74.83	**81.78**
	DeepSeek-v3.2	76.50	79.24	82.23	88.81	**93.06**
Driverlog	GPT-4o	77.02	82.62	87.52	**90.97**	86.00
	GPT-4o-mini	75.91	84.33	81.35	**87.30**	81.28
	DeepSeek-v3.2	83.76	83.13	84.07	81.85	**90.35**
Mystery	GPT-4o	71.02	72.38	70.05	67.04	**80.21**
	GPT-4o-mini	61.06	60.87	61.78	59.25	**69.85**
	DeepSeek-v3.2	66.18	67.08	67.47	68.80	**84.89**
Grippers	GPT-4o	91.21	90.70	89.38	91.89	**95.46**
	GPT-4o-mini	83.44	81.39	82.00	85.14	**87.46**
	DeepSeek-v3.2	87.46	90.83	91.65	89.17	**97.73**
Satellite	GPT-4o	75.07	79.04	80.30	82.04	**90.58**
	GPT-4o-mini	63.18	66.21	70.89	71.25	**83.12**
	DeepSeek-v3.2	78.47	73.68	69.35	79.06	**89.41**
Spanner	GPT-4o	79.17	75.37	87.28	82.12	**87.93**
	GPT-4o-mini	81.04	80.64	**81.35**	79.95	79.42
	DeepSeek-v3.2	80.78	82.86	80.78	85.89	**89.07**
Visitall	GPT-4o	82.15	81.95	83.69	86.18	**90.30**
	GPT-4o-mini	73.60	76.10	68.40	69.17	**80.51**
	DeepSeek-v3.2	76.69	77.14	74.83	76.36	**87.92**

Table 1: Performance(%) on TRAC and ActionReasoningBench. IO=direct prompting, 0CoT=Zero-Shot CoT, 2CoT=Two-Shot CoT, SC=Self-Consistency. **Bold** = best per domain-model row. Performances per task can be found at Appendix A.1.

work's reliance on extensive prompt engineering, we adopt a domain-independent strategy, employing a minimal set of two fixed demonstrations for both executability and state checks, regardless of the domain. Moreover, the instructions for each module remain consistent regardless of the specific task.

3.1 Preprocessing

The preprocessing step initializes the problem solving. For the Reasoning mode, the LLM extracts the initial state S_0, the given action sequence A, and the query q. For the Searching mode, where the action sequence is unknown, the LLM extracts the initial state S_0 and the goal state G. This step is technically necessary because some benchmarks (such as ActionReasoningBench) embed the action sequences within the questions.

3.2 Progression

The core component shared by both modes is **Progression**. In this step, the current states S_i and each action a_i in action sequence A are sequentially provided to the LLM to generate the corresponding new states S_{i+1}, that is:

$$S_{i+1} = T_\theta(S_i, a_i) \quad (i = 0, \ldots, k-1) \quad (1)$$

where $T_\theta(\cdot)$ represents the transition function implemented by the LLM, that is, the single-step progression from the current state to the next one.

This process is performed iteratively, with the LLM explicitly instructed to return the states of all objects, reflecting the progression principle in symbolic methods, where the initial state is continuously updated based on the effects of each action.

Besides, it is possible that certain actions in the given sequence cannot be executed because their preconditions are not satisfied in the current state. We add an additional executability check before taking each action. The executability check asks the LLM whether all preconditions of the action are satisfied in the current state. This module is also used in the Searching mode.

In the Reasoning mode, this function is applied iteratively to a fixed sequence A. In the Searching mode, it is invoked dynamically during node expansion to generate successor nodes.

Domain	Model	IO	0-CoT	2-CoT	SC	ProRAC
	GPT-4o	100	96.66	96.66	76.66	**100**
Ferry	GPT-4o-mini	**93.33**	86.66	93.33	73.33	90
	DeepSeek-v3.2	100	100	93.33	100	**100**
	GPT-4o	**100**	96.66	100	66.66	96.66
Swap	GPT-4o-mini	**80**	80	80	50	76.66
	DeepSeek-v3.2	**100**	100	100	100	96.66
	GPT-4o	86.66	90	86.66	70	**96.66**
Logistics	GPT-4o-mini	86.66	80	83.33	60	**86.66**
	DeepSeek-v3.2	**100**	93.33	100	100	93.33
	GPT-4o	90	86.66	76.66	66.66	**90**
Rovers	GPT-4o-mini	**90**	86.66	83.33	63.33	76.66
	DeepSeek-v3.2	83.33	83.33	93.33	80.00	**93.33**
	GPT-4o	73.33	83.33	83.33	56.66	**93.33**
Floortile	GPT-4o-mini	86.66	76.66	73.33	63.33	**93.33**
	DeepSeek-v3.2	**100**	100	100	100	96.66
	GPT-4o	90	80	100	66.66	**96.66**
Goldminer	GPT-4o-mini	93.33	96.66	76.66	70	**73.33**
	DeepSeek-v3.2	96.66	96.66	**100**	96.66	96.66

Table 2: Performance(%) on ACPBench, bool questions. IO=direct prompting, 0CoT=Zero-Shot CoT, 2CoT=Two-Shot CoT, SC=Self-Consistency. **Bold** = best per domain-model row. Performances per task/domain can be found at appendix A.1.

3.3 Querying

After step two, we obtain the final state S_n. In the third step, we ask the LLM whether S_n satisfies the query q obtained from step one. The query may contain multiple propositions, and the LLM returns true only if S_n satisfies all propositions in q. If there is any proposition that cannot be satisfied, the LLM returns false as the answer.

3.4 Searching

To tackle complex planning tasks, we introduce a *Search mode* based on the A* algorithm, where the heuristic is provided by a fine-tuned language model rather than hand-designed or prompt-based. This enables the agent to perform deliberative planning by integrating the LLM as a transition model and a fine-tuned model as a heuristic evaluator.

Heuristic Model Fine-tuning We fine-tuned a Qwen3-1.7B model (Yang et al., 2025) as the heuristic model with LoRA fine-tuning (Hu et al., 2022). The heuristic function $h(n)$ is trained on a symbolic corpus generated via an automated pipeline, covering the Blocksworld, Depots, and Logistics domains. We first create diverse problem instances using PDDL instance generators and solve them with Fast Downward (Helmert, 2006) to obtain optimal plans. These plans are then decomposed into state-action trajectories, where each intermediate state is annotated with its goal distance (heuristic value) calculated by PyperPlan (Alkhazraji et al., 2020). Then we translate the data from PDDL language to natural language. By training on these samples, the model learns to map the semantic context of a state and its corresponding goal to a heuristic value. More details about the data generation process can be found in the Appendix A.5.

Search Process The agent explores the state space via a priority queue ordered by

$$f = g + h \tag{2}$$

where g denotes the length of the action sequence (path) from the initial state S_0 to the current state.

Each node expansion follows a reflect-then-act cycle: (1) **Node Selection & Goal Check**: The queue pops the node with the minimum f value and invokes the LLM to verify if its state satisfies G. If so, the search terminates. (2) **Expansion**: For unsatisfied states, the LLM proposes candidate actions, which are validated via the executability check module and processed by the progression module to generate verified successor states. (3) **Evaluation**: The heuristic model assigns h-values to new nodes to update the queue.

4 Experiments

4.1 Benchmarks

We evaluated ProRAC on several RAC and planning benchmarks. We selected diverse domains from different benchmarks, ensuring that these domains do not overlap with each other. These domains cover a wide range of scenarios, including logistics and transportation, block stacking, and grid-based navigation.

Domain	Method	DeepSeek-v3.2				GPT-5-mini			
		SR	OR	OSR	AGN	SR	OR	OSR	AGN
Blocksworld	**ProRAC**	85	82.35	69.99	141	90	80	**72**	183
	ToT	0	0	0	244	0	0	0	276
	BFS	5	100	5	131.85	7.5	0	0	129.54
	LLM+AL	0	0	0	—	0	0	0	—
	IO	70	60	42	—	90	70	63	—
	ReAct	90	83.33	**75**	—	80	56.25	45	—
	LLM-Modulo	90	82.35	74.11	—	75	60	45	—
Depots	**ProRAC**	75	65	48.75	164.93	55	75	41.25	217.06
	ToT	0	0	0	175	0	0	0	223.7
	BFS	0	0	0	184.95	0	0	0	202.1
	LLM+AL	0	0	0	—	0	0	0	—
	IO	60	2	1.2	—	5	0	0	—
	ReAct	75	86.67	**65**	—	35	57.14	19.99	—
	LLM-Modulo	85	50	42.5	—	70	64.28	**51.42**	—
Logistics	**ProRAC**	90	88.90	**80.01**	147	80	75	**60**	156.9
	ToT	0	0	0	176	0	0	0	127.3
	BFS	5	0	0	364.4	0	0	0	379
	LLM+AL	0	0	0	—	0	0	0	—
	IO	80	65	52	—	65	60	39	—
	ReAct	90	83	74.70	—	25	80	20	—
	LLM-Modulo	85	64.70	54.90	—	75	66.67	50	—

Table 3: Performance on PlanBench across three domains. SR=Success Rate(%), OR=Optimality Rate(%), OSR=Optimal Success Rate(%), AGN=Average Generated Nodes. **Bold** = best per domain-model group. ToT=Tree-of-Thought, IO=direct prompting. A dash (-) indicates that the metric is not applicable.

TRAC focuses on the Blocksworld domain with three core tasks: executability (verifying action preconditions), projection (tracking state changes after action sequences), and plan verification (assessing whether a sequence achieves the goal).

ActionReasoningBench spans seven domains including Depots, Driverlog, Mystery, Grippers, Satellite, Spanner, and Visitall. While some tasks align with TRAC, it poses significantly higher complexity due to substantially larger numbers of propositions per query.

ACPBench evaluates six distinct domains: Ferry, Swap, Logistics, Rovers, Floortile, and Goldminer. Its applicability and progression tasks align with TRAC's executability and projection, while its validation task introduces more rigorous tripartite classification, requiring agents to distinguish whether an action sequence is valid (no undefined actions), applicable (executable in the current state), or constitutes a complete plan (reaches the goal).

PlanBench evaluates plan generation in the Blocksworld, Depots and Logistics domain. Given an initial state and goal, the language model must generate an action sequence that transforms the initial state into the goal state.

4.2 Methods

RAC Tasks We used GPT-4o (OpenAI, 2024), GPT-4o-mini, and DeepSeek-v3.2 (DeepSeek-AI et al., 2025) as the backbone models for evaluation. For comparison, we adopted direct prompting (IO), Zero-Shot-CoT (Kojima et al., 2022), Two-Shot-CoT (Wei et al., 2022), and Self-Consistency (Wang et al., 2023) prompting as baselines. Two-Shot-CoT prompts followed a process similar to ProRAC: identifying corresponding objects, checking action executability, executing actions to obtain the final state, and comparing the query with the final state. ToT is excluded in RAC tasks because action sequences in reasoning tasks are pre-defined, making multi-path reasoning redundant.

Planning Task We employ DeepSeek-v3.2 and GPT-5-mini as the base models. We compare our A* search strategy against the following baselines: direct prompting, LLM+AL, ToT, BFS, ReAct and LLM-Modulo. RAP and LATS are excluded because their MCTS approaches are better suited to simulation games better and incurs significant computational cost. For A*, ToT, and BFS, we set the node budget to 500, the maximum search depth to 20, and the branching factor to 3. Further implementation details are provided in Appendix A.4.

4.3 Main Results of RAC tasks

Table 1 and 2 present the results of ProRAC and baseline methods on RAC tasks.

ProRAC achieves competitive results compared with prompting baselines across benchmarks. On TRAC, direct prompting achieves reasonable performance, but ProRAC yields a further marked

improvement. On ActionReasoningBench where action sequences can be as long as 19 steps, neither Zero-Shot nor CoT methods achieve satisfactory results, whereas ProRAC attains significantly better performance across domains and tasks. The Mystery domain remains the most challenging, likely because it involves discrete fuel levels with strict threshold semantics, where LLMs are known to struggle with ordinal arithmetic.

ACPBench reveals a systematic difficulty with sequence validation. LLMs perform relatively well on applicability and progression tasks, but neither ProRAC nor baselines excel on the validation task, with performance declining notably in domains like Logistics and Floortile. We observed that determining action sequence validity poses particular challenges for LLMs, which occasionally misclassify valid sequences as invalid. Detailed per-task breakdowns in Appendix A.1 reveal that baseline LLM methods perform poorly on certain tasks, with accuracy below 50% in some cases.

Action-wise decomposition mitigates LLMs' limited working memory. The results across all three benchmarks validate the robustness and effectiveness of our approach. A key reason ProRAC is effective is that it decomposes multi-step reasoning into single-step progressions, avoiding the need to track long reasoning chains within a single context.

4.4 Main Results of Planning Task

Table 3 reports performance on PlanBench using four primary metrics: **Success Rate (SR)**, the percentage of solved instances; **Optimality Rate (OR)**, the ratio of optimal plans among successful solutions; **Optimal Success Rate (OSR)**, calculated as $SR \times OR$; and **Average Generated Nodes (AGN)**, the mean number of nodes generated during search (lower is better). Additional metrics including Average Expanded Nodes (AEN) and Average Effective Branching Factor (AEBF) are provided in Appendix A.1.

Uninformed search and prompt-based heuristics both fail to scale. BFS barely solves any instance, confirming that exhaustive search without guidance is computationally intractable. ToT, which relies on prompt-based heuristics, performs even worse at 0% SR on both models despite exhausting its 500-node budget, demonstrating that prompt-based scoring with LLM is insufficiently accurate to direct search toward valid solutions.

Accurate heuristics are the decisive factor for tractable planning. Neither the search framework alone nor a prompt-based heuristic alone is sufficient, it is the combination of A* with a trained heuristic that makes search tractable. ProRAC solves the majority of instances while generating significantly fewer nodes, validating that a compact fine-tuned model provides more reliable guidance than a large LLM prompted at inference time. This finding is further corroborated by the controlled ablation in Section 5.1, where replacing the trained heuristic with a prompted alternative causes SR to drop to zero.

Auto-formalization remains a fundamental bottleneck. Both language models fail to solve any instance without human intervention. The primary failure mode is the generated program's inability to pass the satisfiability (SAT) check, and LLMs frequently fail to rectify existing errors across multi-round iterations—highlighting that the brittleness of auto-formalization is a systemic issue rather than a model-specific one.

Feedback-based methods improve solvability but cannot guarantee solution quality. LLM-Modulo and ReAct both achieve non-trivial success rates, showing that feedback signals provide meaningful guidance for planning. Their advantage is especially clear when the backbone LLM can already generate a plausible complete plan, because feedback can be applied directly to the whole trajectory. Nevertheless, such feedback is not always reliable. Some generated plans are even *shorter* than the optimal solution, yet the LLM-based checker still incorrectly accepts them as valid. Moreover, when a correct but suboptimal plan is produced, multi-round feedback often fails to refine it toward the optimum. This suggests that feedback signals are useful for improving solvability, but their utility is fundamentally bounded.

Plan-level feedback and heuristic-guided search capture complementary strengths. Table 3 shows that ProRAC does not consistently outperform ReAct and LLM-Modulo. In some cases, such as Blocksworld and Depots, feedback-based methods achieve higher success or optimal-success rates than ProRAC. This is because ReAct and LLM-Modulo revise complete candidate plans, allowing them to preserve global trajectory coherence when the backbone LLM already produces a near-correct solution. In contrast, ProRAC performs systematic search over partial states, making planning more explicit and controllable but also dependent on action proposal and heuristic ranking under the node budget. These results suggest that plan-level feedback

Error Type	Definition and Example	Percentage
Incorrect arguments or ordered values	The predicate is correct, but the model uses an incorrect entity or ordered values. For example, it updates fuel from `f6` to `f7` instead of the correct predecessor `f5`.	41.86%
Missed fluent preservation	The model fails to preserve an unaffected fluent in the successor state. For example, `switch-on` incorrectly removes an instrument's `calibration-target` relation.	25.58%
Missed add effects	The model omits a fluent that the action should add. For example, `take-image` fails to add the corresponding `have-image` fluent.	20.93%
Missed delete effects	The model retains a fluent that the action should remove, leaving mutually incompatible facts in the state.	6.98%
Hallucinated unrelated fluents	The model adds a fluent unrelated to the current action and unsupported by the previous state.	4.65%

Table 4: Distribution and examples of progression errors.

and heuristic-guided search are complementary.

4.5 Progression Error Analysis

To characterize the failure modes of ProRAC, we manually analyzed 318 generated successor states from 40 instances sampled from the Mystery and Satellite domains. We identified 43 step-level progression errors, corresponding to an error rate of 13.52%.

As shown in Table 4, we categorize these errors into five types: incorrect arguments or ordered value, Missed fluent preservation, missed add effects, missed delete effects, and hallucinated unrelated fluents. The first three categories account for 88.37% of the observed errors. These errors primarily involve the precision of local state updates: the model may use an incorrect entity or ordinal value, omit an action effect, or fail to preserve an unaffected fluent. In contrast, unrelated hallucinations are rare, indicating that action-wise progression largely keeps generation grounded in the relevant state representation.

5 Ablation Study

5.1 Effect of Heuristic Source

Domain	Method	SR	AEN	AGN
Depots	ProRAC w/ PH	0	187	500
	ProRAC	**75**	**72.67**	**164.93**
Logistics	ProRAC w/ PH	0	196.2	439
	ProRAC	**90**	**85.3**	**147**

Table 5: Ablation on heuristic source. PH = Prompted Heuristic (LLM-scored at inference time), TH = Trained Heuristic. All results use DeepSeek-v3.2 as the backbone. AEN = Average Expanded Nodes; AGN = Average Generated Nodes.

To isolate the contribution of heuristic quality within our search framework, we compare A* search under two heuristic sources: prompted heuristics (PH), where the backbone LLM scores candidate states at inference time, and trained heuristics (TH), where the fine-tuned model provides estimates. As shown in Table 5, ProRAC w/ PH exhausts its 500-node generation budget without finding any solution on the Depots domain, while ProRAC with trained heuristics achieves 75%. The differences in AEN values confirm that the search was actively misled. In domains with complex multi-way object dependencies, prompt-based heuristics produce systematic directional errors that cause A* to repeatedly expand incorrect branches until the budget is exhausted. The trained heuristic, supervised on symbolic trajectories with exact h_{add} targets, maintains reliable correlation with true plan cost and makes A* tractable. These results confirm that heuristic quality, not the search framework itself, is the decisive factor.

5.2 Heuristic Estimation Accuracy

Domain	Mean h^*	MAE
Blocksworld	10.919	3.247
Depots	8.630	2.541
Logistics	9.289	0.774

Table 6: Heuristic estimation error of the learned heuristic model. Mean h^* is the average ground-truth cost-to-goal per domain, MAE = Mean Absolute Error.

To quantify the accuracy of the learned heuristic model, we construct a new set of evaluation instances using the same pipeline as training data generation, and measure the model's estimation error against ground-truth heuristic values h^*. We

report Mean Absolute Error (MAE) and Mean h^* per domain.

The estimation errors align with the search behavior in Table 3. Logistics has the lowest MAE, matching ProRAC's strong performance in this domain. Blocksworld and Depots have similar MAE, but their planning outcomes differ. This shows that heuristic accuracy is necessary but not sufficient for guaranteed planning efficiency. Final planning performance also depends on domain structure and on the backbone LLM's ability to propose executable actions and produce accurate progressions.

6 Discussion

Our evaluation focuses on existing RAC benchmarks based on structured "toy" domains. We view this work as a necessary step toward addressing more complex scenarios: if LLMs cannot reliably solve basic RAC problems in well-defined settings, their success in open-ended, uncertain real-world contexts is even less likely. Thus, these benchmarks should not be viewed as endpoints, but as controlled testbeds for isolating the state-transition abilities required by more complex agents. The modules in ProRAC are intended to be transferable: once validated in well-defined RAC settings, they can be integrated into broader agent pipelines.

The connection between RAC and agent planning is not only conceptual, but also methodological. Modern agents often fail because an error in a single action step propagates through the remaining trajectory. Progression can serve as a state-update module that maintains an explicit world state after each action, preserves unaffected facts, and exposes the updated state to subsequent reasoning steps. Executability checking can act as an action-level validation module, filtering actions whose preconditions are not satisfied before they are executed. By integrating these RAC components into an agent loop, the system can obtain step-level control over action effects and reduce the risk that local errors propagate through a long trajectory.

Search and learned heuristics further extend this integration. Instead of relying only on sampling, self-reflection, or external feedback after a complete plan has been generated, an agent can use search to compare partial trajectories before committing to a full plan. These components can be combined with existing agent-planning frameworks: ReAct-style agents can use them to maintain and branch over explicit intermediate states, and verifier-based agents can use them to provide fine-grained feedback on partial plans.

7 Conclusion

In this work, we presented ProRAC, a progression-based neuro-symbolic framework for RAC and planning. Across TRAC, ActionReasoningBench, ACPBench, and PlanBench, ProRAC improves both state reasoning and plan generation by combining explicit progression with heuristic-guided search. The ablation study further shows that search performance depends critically on heuristic quality, with prompt-based scoring failing to guide A* effectively despite substantial node expansion. These results suggest that ProRAC provides a useful foundation for building more reliable LLM-based planning systems. An important future direction is to extend ProRAC to long-horizon planning systems, where reliable progression and heuristic-guided search are crucial for maintaining coherent state transitions over long action sequences.

Limitations

First, the heuristic model is fine-tuned on automatically generated symbolic data specific to the target domains. While this ensures high precision within those domains, its zero-shot generalization to entirely unseen planning environments remains a challenge.

Second, constrained by computational resources and expenses, the models used in our evaluation are primarily proprietary models. Future iterations of ProRAC could investigate the efficacy of open-source LLMs, and explore techniques such as fine-tuning and reinforcement learning to enhance the RAC capabilities of these models.

Third, ProRAC incurs additional inference cost because progression and search require multiple model calls. Token usage grows with action-sequence length and the number of generated nodes. This overhead increases latency, limits cost-efficiency, and constrains deployment under strict computational budgets.

Acknowledgments

We acknowledge support from the Natural Science Foundation of China under Grant No. 62476295.

References

Mohamed Aghzal, Erion Plaku, Gregory J. Stein, and Ziyu Yao. 2025. A survey on large language models for automated planning. *ArXiv*, abs/2502.12435.

Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. 2020. Pyperplan. https://doi.org/10.5281/zenodo.3700819.

Blai Bonet and Hector Geffner. 1999. Planning as heuristic search: New results. In *European Conference on Planning*.

Benjamin Callewaert, Simon Vandevelde, and Joost Vennekens. 2025. VERUS-LM: a versatile framework for combining llms with symbolic reasoning. *ArXiv*, abs/2501.14540.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. DeepSeek-V3.2: Pushing the frontier of open large language models. *ArXiv*, abs/2512.02556.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. *ArXiv*, abs/2503.09572.

Michael Gelfond and Vladimir Lifschitz. 1993. Representing action and change by logic programs. *J. Log. Program.*, 17(2/3&4):301–321.

Enrico Giunchiglia and Vladimir Lifschitz. 1998. An action language based on causal explanation: Preliminary report. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*, pages 623–630. AAAI Press / The MIT Press.

Divij Handa, Pavel Dolin, Shrinidhi Kumbhar, Tran Cao Son, and Chitta Baral. 2025. ActionReasoningBench: Reasoning about Actions with and without Ramification Constraints. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 8154–8173. Association for Computational Linguistics.

Weinan He, Canming Huang, Zhanhao Xiao, and Yongmei Liu. 2023. Exploring the capacity of pretrained language models for reasoning about actions and change. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 4629–4643. Association for Computational Linguistics.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

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, and 26 others. 2023. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR.

Adam Ishay and Joohyung Lee. 2025. LLM+AL: bridging large language models and action languages for complex reasoning about actions. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 24212–24220. AAAI Press.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil B. Murthy. 2024. LLMs can't plan, but can help planning in llm-modulo frameworks. *ArXiv*, abs/2402.01817.

Henry A. Kautz. 2022. The Third AI Summer: Aaai robert s. engelmore memorial lecture. *AI Magazine*, 43(1):105–125.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Harsha Kokel, Michael Katz, Kavitha Srinivas, and Shirin Sohrabi. 2025. ACPBench: Reasoning About Action, Change, and Planning. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 26559–26568. AAAI Press.

Robert A. Kowalski and Marek J. Sergot. 1986. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95.

Hongyi Ling, Shubham Parashar, Sambhav Khurana, Blake Olson, Anwesha Basu, Gaurangi Sinha, Zhengzhong Tu, James Caverlee, and Shuiwang Ji. 2025. Complex LLM planning via automated heuristics discovery. *ArXiv*, abs/2502.19295.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2024. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32201–32223. PMLR.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, and 7 others. 2025. Large language model agent: A survey on methodology, applications and challenges. *ArXiv*, abs/2503.21460.

J. McCarthy and P.J. Hayes. 1981. Some philosophical problems from the standpoint of artificial intelligence. In Bonnie Lynn Webber and Nils J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 431–450. Morgan Kaufmann.

Siddharth Nayak, Adelmo Morrison Orozco, Marina Ten Have, Vittal Thirumalai, Jackson Zhang, Darren Chen, Aditya Kapoor, Eric Robinson, Karthik Gopalakrishnan, James Harrison, Brian Ichter, Anuj Mahajan, and Hamsa Balakrishnan. 2024. LLaMAR: Long-horizon planning for multi-agent robots in partially observable environments. *ArXiv*, abs/2407.10031.

Theo Olausson, Alex Gu, Benjamin Lipkin, Cedegao E. Zhang, Armando Solar-Lezama, Joshua B. 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, EMNLP 2023, Singapore, December 6-10, 2023*, pages 5153–5176. Association for Computational Linguistics.

OpenAI. 2024. GPT-4o System Card. *CoRR*, abs/2410.21276.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 3806–3824. Association for Computational Linguistics.

Raymond Reiter. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press.

Stuart J. Russell and Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach*, 4 edition. Pearson, Hoboken, NJ.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *ArXiv*, abs/2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. 2024. Building cooperative embodied agents modularly with large language models.

In *International Conference on Learning Representations*.

Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. 2021. AR-LSAT: investigating analytical reasoning of text. *CoRR*, abs/2104.06598.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62138–62160. PMLR.

A Appendix

Model	PR	EXE	PV
4o	94.73	96.58	88.69
4o-0CoT	96.19	97.22	87.51
4o-2CoT	97.77	95.56	93.33
4o-SC	97.78	95.56	93.33
4o-ProRAC	**100**	**97.77**	**95.56**
4o-mini	**95.56**	73.33	88.89
4o-mini-0CoT	95.56	68.88	84.44
4o-mini-2CoT	93.33	73.33	86.67
4o-mini-SC	88.89	**84.44**	**91.11**
4o-mini-ProRAC	93.33	80	84.44
v3.2	97.77	100	88.88
v3.2-0CoT	100	100	95.56
v3.2-2CoT	100	100	95.56
v3.2-SC	97.77	88.89	88.89
v3.2-ProRAC	**100**	**100**	**100**

Table 7: Performance (%) of ProRAC and baseline methods on TRAC benchmark. PR=Projection, EXE=Executability, PV=Plan Verification. **Bold** indicates the best performance.

A.1 Performance Per Task

Table 7, 8 and 9 and demonstrate the results of ProRAC and baseline methods TRAC and ActionReasoningBench, divided by tasks and domains.

Table 10 and 11 report results on the multiple-choice questions (MCQs) in ACPBench, comparing ProRAC with baseline methods.

Table 12 reports additional metrics on PlanBench, including AEN and AEBF.

A.2 Token Usage

In Table 13 to Table 18, we report the average token usage of the ProRAC method on RAC and planning tasks, where token usage refers to the total number of input and output tokens.

A.3 Model Availability and Inference Details

Table 15 summarizes the model snapshots, API providers, and their availability. All model calls used a temperature of 0. We did not explicitly specify `top_p`, a random seed, or a separate decoding strategy; all unspecified parameters followed the default settings of the corresponding API provider. In particular, `top_p` was 1.0. For GPT-5-mini, `reasoning_effort` was set to `medium`.

A.4 Implementation Details

ReAct We implement a ReAct-style agent adapted to the closed-world RAC setting. At each step, the agent receives the current world state as its observation, generates a candidate action, executes it via the progression module, and receives the updated state as the next observation. This process iterates for a maximum of 10 rounds. Since RAC tasks have no external environment to query, the observation at each step is the LLM-generated state description produced by the same progression module used in ProRAC. The agent is prompted to select the most plausible next action given the current state and the goal, following the standard Thought-Action-Observation format. If a valid goal state is reached within 10 rounds, the instance is counted as solved; otherwise it is marked as failure.

LLM-Modulo We implement a LLM-Modulo-style framework consisting of two components: a generator LLM and a prompt-based critic. The generator is first prompted to produce a complete plan given the initial state and goal. The critic then evaluates the plan along two dimensions: (1) executability, checking whether each action's preconditions are satisfied in sequence; (2) optimality, checking whether a shorter valid plan is likely to exist given the problem structure. Both checks are implemented via prompting the same LLM rather than an external symbolic verifier. The critic's verdict and natural-language feedback are appended to the conversation history and passed back to the generator for revision. This loop runs for a maximum of 10 iterations. The final plan from the last iteration is used for evaluation.

Model	Depots				Driverlog				Mystery				Grippers			
	AE	EFF	FT	ST	AE	EFF	FT	ST	AE	EFF	FT	ST	AE	EFF	FT	ST
4o	**97.74**	79.1	78.13	73.68	77.14	80.95	70	80	75	45.5	89.48	74.1	95.24	90.38	95.92	83.33
4o-0CoT	97.37	48.84	84.38	39.47	82.86	90.95	76.67	80	82.1	54.55	88.9	64	95.24	90.38	93.88	83.33
4o-2CoT	97.37	74.41	75	78.94	**94.28**	90.47	73.33	92	80	52.2	79	69	95.24	89	90	83.3
4o-SC	97.37	69.77	75	81.58	91.42	90.47	**90**	**92**	75	43.47	84.21	65.51	95.24	88.46	93.87	**90**
4o-ProRAC	92.10	**93.02**	**93.75**	**84.21**	80	**100**	80	84	**82.5**	**69.56**	**89.48**	**79.31**	**97.61**	**100**	**97.59**	86.67
4o-mini	**94.7**	58.14	67.74	81.6	77.14	90.5	80	56	65	52.2	68.42	58.62	**92.9**	84.62	89.6	66.67
4o-mini-0CoT	89.5	53.5	71	79	71.43	95.24	86.67	**84**	65	39.13	**84.21**	55.17	83.33	86.54	85.71	70
4o-mini-2CoT	92.1	76.64	65.62	60.52	85.71	85.72	**90**	64	70	39.13	79	**59**	91	79	88	70
4o-mini-SC	89.47	51.16	71.87	**86.84**	**80**	95.23	90	84	72.5	39.13	73.68	51.72	92.85	84.61	89.79	73.33
4o-mini-ProRAC	81.57	**87.5**	**84.38**	73.68	77.14	**100**	80	68	**75**	**65.22**	73.68	**65.51**	88.1	**92.30**	**89.79**	**79.67**
v3.2	84.73	72.09	78.13	71.05	91.42	85.71	84.61	73.33	77.5	39.13	89.47	58.62	95.24	94.23	93.87	80
v3.2-0CoT	92.10	72.09	84.37	68.42	85.29	85.71	82.61	78.94	77.5	47.82	**94.73**	48.27	90.47	96.15	100	80
v3.2-2CoT	97.36	76.64	81.25	73.68	82.85	90	88.46	75	80	52.17	89.47	48.27	83.33	96.15	93.87	83.33
v3.2-SC	**94.73**	84.21	86.84	89.47	**94.29**	90.47	86.67	56	77.5	47.83	94.73	55.17	95.23	96.15	97.95	83.33
v3.2-ProRAC	92.10	**90.69**	**100**	**89.47**	91.42	**100**	**90**	**80**	**90**	**82.60**	84.21	**82.75**	**97.61**	**100**	**100**	**93.33**

Table 8: Performances(%) of ActionReasoningBench on Depots, Driverlog, Mystery and Grippers, divided by tasks and domains. AE=Action Executability, EFF=Effects, FT=Fluent tracking, ST=State Tracking. **Bold** indicates the best performance.

A.5 Details About the Heuristic Model Training

Based on our data generation pipeline, we generate 76026 training examples from three domains, with heuristic values calculated using h_{add}.

We use the Hugging Face Transformer library (Wolf et al., 2020) to build the code. We employ Qwen3-1.7B as the backbone transformer model. The model is fine-tuned on the training data. To accommodate long-horizon sequences, the maximum sequence length is set to 2048 tokens. Other parameters are displayed in table 19. All experiments, including LoRA fine-tuning and model inference, were conducted on a single workstation equipped with an NVIDIA A100 GPU. The host system features 10 vCPUs powered by an Intel Xeon processor and 72GB of system RAM.

A.6 Details about EBF

Effective branching factor (EBF) is a metric used to evaluate the quality of the heuristic function (Russell and Norvig, 2020). If A* search generates n nodes for a problem with solution depth d, then b^* is the branching factor that a uniform tree of depth d must have to contain $n+1$ nodes, satisfying the following relationship:

$$n + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d \quad (3)$$

By solving this equation we can obtain the EBF. We calculate the Average EBF (AEBF) for instances that can be solved using searching methods. In practice, we employ the bisection method to numerically solve for b^*.

An AEBF value closer to 1 indicates a more informed and powerful heuristic, representing near-optimal search efficiency with minimal redundant exploration.

A.7 Domain Independent Instructions

Initial State Extractor Prompt

You are a PDDL expert, and you are familiar with the classical IPC domains. Based on the domain description, the current state and the example, extract all objects' states. The domain of the example may be different from the current state, but the idea of finishing the task is the same. The format of the example must be always the same. Your answer should use the same format as the example to think and generate your answer. Your answer should contain all objects' states, and each object's state should contain all predicates belonging to this object mentioned in the PDDL domain description. Pay attention that different kinds of objects have different predicates, don't mix them up. Besides, Some objects may have some predicates that are not mentioned in the domain description, that means these

Model	Satellite				Spanner				Visitall			
	AE	EFF	FT	ST	AE	EFF	FT	ST	AE	EFF	FT	ST
4o	75.68	79.12	78.72	66.77	75	84.48	85	72.22	69.23	86.05	**93.33**	80
4o-0CoT	81.08	83.33	85.11	66.67	69.44	74.07	83	75	64.1	83.72	93.33	**86.67**
4o-2CoT	75.67	83.3	87.23	75	**94.44**	88.88	**93.61**	72.22	64.1	90.69	93.33	86.67
4o-SC	78.37	83.33	**91.48**	75	80.55	81.48	91.48	75	74.36	83.72	93.33	93.33
4o-ProRAC	**91.89**	**100**	89.36	**81.08**	91.66	**88.88**	85.10	**86.11**	**94.87**	**93.02**	86.67	86.67
4o-mini	56.8	62.5	72.34	61.11	77.8	74.1	89.4	**82.86**	66.67	74.42	86.67	66.67
4o-mini-0CoT	51.35	66.67	78.23	68.6	77.14	84.62	89.4	71.43	66.67	74.42	90	73.33
4o-mini-2CoT	75.67	62.5	**78.72**	66.67	85.71	**85.71**	**90**	64	53.84	69.76	**90**	60
4o-mini-SC	70.27	70.83	74.46	69.44	88.88	81.48	80	69.44	51.28	72.09	80	73.33
4o-mini-ProRAC	**86.48**	**91.66**	76.59	**77.78**	**88.88**	81.48	72.34	75	**82.05**	**80**	86.67	**73.33**
v3.2	86.49	75	82.97	69.44	86.11	85.18	85.18	66.66	82.05	81.39	83.33	60
v3.2-0CoT	72.97	79.16	78.72	63.88	86.11	85.18	85.18	**75**	87.17	81.39	80	60
v3.2-2CoT	72.97	62.5	80.85	61.11	88.88	85.18	85.18	63.88	84.61	81.39	80	53.33
v3.2-SC	81.9	79.17	82.97	72.22	94.44	88.46	88.46	72.22	89.74	79.07	83.33	53.33
v3.2-ProRAC	**86.48**	**100**	**92.9**	**78.26**	**100**	**92.59**	**91.48**	72.22	**92.30**	**86.04**	**86.67**	**86.67**

Table 9: Performances(%) of ActionReasoningBench on Satellites, Spanner and Visitall, divided by tasks and domains. AE=Action Executability, EFF=Effects, FT=Fluent tracking, ST=State Tracking. **Bold** indicates the best performance.

predicates are false (not true) in the current state. You should also include these predicates correctly with the correct negation. Don't miss any predicates.
So, first you need what kind of objects are mentioned in the PDDL domain description, and what predicates belongs to each object. Second, you need to confirm how many objects are mentioned in the current state. Third, extract all objects' states with all predicates belonging to each object mentioned in the PDDL domain description. Remember you need to return all all objects' states with all predicates belong to them. Finally, organize all objects' states into a new paragraph as the end of your answer. Each sentences should be a simple sentence that only contain one object, divided by a period. Your last paragraph must use this format: 'box' with no other content, and put all objects' states inside the box. Remember, don't use any markdown formatting and latex format, the box is a strict format requirement, you can't change it, and your final answer must put inside the box.

Question Extractor Prompt

The raw text above contains two parts: an action sequence and a question. Based on the the example, your task is to extract and organize the question from the raw text. The domain of the example may be different from the current state, but the format of the example must be always the same. Your answer should use the same format as the example to think and generate your answer. Normally the text has an action sequence in the front, followed by the question, but not all the time. Sometimes the question will contain an if clause, where another action is mentioned. You should ignore the if-clause and only extract the question.
So, first you need to confirm which parts of the raw text is the action sequence, and which parts is the question. Second, discard the action sequence and extract the question and organize the question into a new paragraph as the end of your answer. Your last paragraph must use this format: 'box' with no other content, and put the question you extracted inside the box. Remember, don't use any markdown formatting and latex format, the box is a strict format requirement, you can't change it, and your final answer

Model	Ferry			Swap			Logistics			Rovers			Floortile			Goldminer		
	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL
4o	100	100	100	**100**	100	100	**100**	90	70	**100**	90	80	90	90	40	90	90	90
4o-0CoT	100	100	90	100	90	100	100	90	80	90	90	80	100	80	70	60	90	90
4o-2CoT	100	100	90	100	100	100	100	100	60	70	**100**	60	90	90	70	100	100	**100**
4o-SC	70	80	80	80	50	70	50	90	70	60	80	60	50	80	40	60	80	60
4o-ProRAC	**100**	**100**	**100**	90	**100**	**100**	**100**	**100**	**90**	**100**	90	**80**	**100**	**90**	**90**	**100**	**100**	90
4o-mini	90	100	**90**	60	100	**80**	**100**	80	80	**100**	90	80	100	100	60	**100**	90	**90**
4o-mini-0CoT	100	100	60	70	100	70	100	**90**	50	80	**100**	80	90	100	40	100	**100**	90
4o-mini-2CoT	100	100	80	**90**	100	50	100	80	70	80	90	80	100	100	20	100	90	40
4o-mini-SC	60	80	80	40	60	50	40	90	50	60	90	40	50	80	60	70	90	50
4o-mini-ProRAC	**100**	**100**	70	70	**100**	60	90	**90**	**80**	80	70	**80**	**100**	**100**	**80**	**80**	70	70
v3.2	100	100	100	100	100	**100**	100	**100**	**100**	80	90	80	100	100	100	100	**100**	90
v3.2-0CoT	100	100	100	100	100	100	100	90	90	80	90	80	100	100	100	90	100	100
v3.2-2CoT	90	90	100	100	100	100	100	100	**100**	100	90	90	100	100	100	100	100	100
v3.2-SC	100	100	100	100	100	100	100	100	100	80	80	80	100	100	100	100	100	90
v3.2-ProRAC	**100**	**100**	**100**	**100**	**100**	90	**100**	90	90	**100**	**90**	**90**	90	**100**	**100**	**100**	90	**100**

Table 10: Performance(%) of ProRAC and baseline methods on the ACPBench, BOOL questions. APP=Applicability, PROG=Progression, VAL=Validation. **Bold** indicates the best performance.

Model	Ferry			Swap			Logistics			Rovers			Floortile			Goldminer		
	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL
4o	90	100	40	**100**	90	**100**	80	100	40	40	80	30	90	90	40	90	100	**90**
4o-0CoT	90	100	40	80	100	60	70	100	50	50	80	40	90	80	60	60	90	50
4o-2CoT	90	100	70	90	100	90	80	100	50	50	80	**50**	90	90	60	90	80	30
4o-SC	100	100	40	100	90	50	80	100	50	90	80	50	90	90	70	90	100	60
4o-ProRAC	**100**	**100**	**90**	90	**100**	70	**100**	**100**	70	**100**	**80**	40	**100**	**90**	**70**	**100**	**100**	60
4o-mini	90	100	30	70	100	70	**60**	90	20	**70**	70	20	80	80	40	**100**	90	**80**
4o-mini-0CoT	90	100	30	70	100	70	60	90	20	70	60	20	80	90	30	70	90	40
4o-mini-2CoT	90	100	40	70	100	70	50	100	40	50	80	50	80	90	30	90	90	40
4o-mini-SC	90	90	40	80	100	60	90	80	40	70	80	30	90	80	40	100	80	30
4o-mini-ProRAC	**100**	**100**	**70**	**80**	90	**70**	**90**	**100**	**60**	**90**	**80**	**50**	**100**	**90**	**60**	**100**	**100**	60
v3.2	100	100	100	100	80	**100**	100	100	90	100	**100**	**100**	100	100	90	**100**	100	90
v3.2-0CoT	100	100	100	80	90	80	90	90	**100**	80	90	90	100	90	90	100	100	**100**
v3.2-2CoT	100	100	100	100	70	100	100	100	90	100	100	90	100	90	100	90	100	100
v3.2-SC	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
v3.2-ProRAC	**100**	**100**	**100**	**100**	**100**	90	**90**	**100**	90	**100**	90	90	**100**	**100**	**100**	90	**100**	90

Table 11: Performance(%) of ProRAC and baseline methods on the ACPBench, MCQ questions. APP=Applicability, PROG=Progression, VAL=Validation. **Bold** indicates the best performance.

must put inside the box.

Executability Check Prompt

You are a PDDL expert, and you are familiar with the classical IPC domains. Based on the domain description, the example, the current state and the action, determine whether this action is executable or not. The domain of the example may be different from the current state, but the idea of finishing the task is the same. The format of the example must be always the same. Your answer should use the same format as the example to think and generate your answer. An action is executable if and only if all preconditions of this action is satisfied in the current state. If any preconditions of the action is not satisfied, the action is not executable.
So, first you need to confirm objects that are affected by action. Second, confirm the preconditions of this action. Third, check all preconditions one by one.
After your analysis, please your final answer in a new paragraph as the end of your answer. If executable, return True, else return False. Your last paragraph should use

Domain	Method	DeepSeek-v3.2						GPT-5-mini					
		SR	OR	OSR	AEN	AGN	AEBF	SR	OR	OSR	AEN	AGN	AEBF
Blocksworld	**ProRAC**	85	82.35	69.99	71	141	1.65	90	80	72	130	183	1.71
	ToT	0	0	0	74	244	—	0	0	0	85	276	—
	BFS	5	100	5	131.85	131.85	3.27	7.5	0	0	129.54	129.54	3.00
	LLM+AL	0	0	0	—	—	—	0	0	0	—	—	—
	IO	70	60	42	—	—	—	90	70	63	—	—	—
	ReAct	90	83.33	75	—	—	—	80	56.25	45	—	—	—
	LLM-Critic	90	80	72	—	—	—	75	60	45	—	—	—
Depots	**ProRAC**	75	65	48.75	72.67	164.93	1.72	55	75	41.25	105.55	217.06	1.80
	ToT	0	0	0	58	175	—	0	0	0	69	223.7	—
	BFS	0	0	0	184.95	184.95	—	0	0	0	202.1	202.1	—
	LLM+AL	0	0	0	—	—	—	0	0	0	—	—	—
	IO	60	2	1.2	—	—	—	5	0	0	—	—	—
	ReAct	75	86.67	65	—	—	—	35	57.14	19.99	—	—	—
	LLM-Modulo	85	50	42.5	—	—	—	70	64.28	51.42	—	—	—
Logistics	**ProRAC**	90	88.9	80.01	85.3	147	1.8	80	75	60	96.2	156.9	1.95
	ToT	0	0	0	58	176	—	0	0	0	67	127.3	—
	BFS	5	0	0	364.4	364.4	12	0	0	0	379	379	—
	LLM+AL	0	0	0	—	—	—	0	0	0	—	—	—
	IO	80	65	52	—	—	—	65	60	39	—	—	—
	ReAct	90	83	74.7	—	—	—	25	80	20	—	—	—
	LLM-Modulo	85	64.7	54.9	—	—	—	75	66.67	50	—	—	—

Table 12: Full results on PlanBench across three domains. SR=Success Rate(%), OR=Optimality Rate(%), OSR=Optimal Success Rate(%), AEN=Avg. Expanded Nodes, AGN=Avg. Generated Nodes, AEBF=Avg. Effective Branching Factor. **Bold** = best per domain-model group. A dash (-) indicates that the metric is not applicable.

Domain	Model	PR	EXE	PV
Blocksworld	GPT-4o	17690.0	9845.2	15173.2
	GPT-4o-mini	18360.5	8290.3	12266.1
	DeepSeek-v3.2	17689.0	9724.1	15611.4

Table 13: Average token usage on TRAC. PR=Projection, EXE=Executability, PV=Plan Verification.

this format: 'boxTrue' with no other content, and put your final answer inside the box. Remember, don't use any markdown formatting and latex format, the box is a strict format requirement, you can't change it, and your final answer must put inside the box.

Progression Prompts

You are a PDDL expert, and you are familiar with the classical IPC domains. Based on the domain description, the current state, the action and the example, take the action and return the new state. The domain of the example may be different from the current state, but the idea of finishing the task is the same. The format of the example must be always the same. Your answer should contain all objects' states, and each object's state should contain all predicates that belongs to this object mentioned in the PDDL domain description, including the properties that is not affected by the action. Remember, your answer should use natural language to describe the state of each object. Besides, some predicates of an object may not be mentioned in the current state, that means these predicates are false (not true) in the current state. You should also include these predicates correctly with the correct negation. Don't miss any predicates.
So, first you need to confirm the effects of this action. Second, you need to confirm which objects are effected by the action. Third, take the action and return all objects' state with all predicates mentioned in the PDDL domain description. Remember you need to return all predicates of all objects, including these that are not affected by the action.
After your analysis, please organize all objects' state as the new state in a new paragraph as the end of your answer. Each sentences should be a simple sentence that only contain one object, divided by a period. Your last paragraph must use this format: 'box' with no other content, and put all objects' states inside the box. Remember, don't use any markdown formatting and

Domain	Model	AE	EFF	FT	ST
Depots	GPT-4o	71397.3	132618.7	119082.2	128189.1
	GPT-4o-mini	28267.2	139813.1	125846.4	228869.6
	DeepSeek-v3.2	67893.3	13907.3	13998.4	23056.3
Driverlog	GPT-4o	56235.1	122722.3	131508.3	154645.6
	GPT-4o-mini	37458.0	135640.0	137008.7	247215.9
	DeepSeek-v3.2	72956.2	134748.2	135740.9	152536.0
Mystery	GPT-4o	27536.4	129441.6	108892.2	119711.3
	GPT-4o-mini	34707.7	140919.9	115632.3	140708.7
	DeepSeek-v3.2	52018.2	154325.9	124528.7	135614.9
Grippers	GPT-4o	53484.1	102722.8	90214.0	92092.8
	GPT-4o-mini	36699.8	93827.7	84937.7	85413.2
	DeepSeek-v3.2	48684.1	114143.0	101528.1	99274.0
Satellite	GPT-4o	57087.9	141581.9	126484.7	150650.7
	GPT-4o-mini	38100.0	137764.3	119468.4	156937.9
	DeepSeek-v3.2	34231.0	156267.5	129245.5	12399.5
Spanner	GPT-4o	36436.6	146172.9	117108.5	140922.8
	GPT-4o-mini	35699.0	136561.9	112374.7	206300.7
	DeepSeek-v3.2	66230.1	116477.0	125967.6	140303.8
Visitall	GPT-4o	115357.9	168374.5	114369.4	194912.2
	GPT-4o-mini	45840.7	134343.4	129421.9	124165.9
	DeepSeek-v3.2	85906.3	27808.4	25333.7	156193.0

Table 14: Average token usage on ActionReasoningBench. AE=Action Executability, EFF=Effects, FT=Fact Trajectory, ST=State Trajectory.

Model Snapshot	**Provider**	**Availability**
`deepseek-v3.2`	DeepSeek official API	Available until 2026-07-24
`gpt-4o-2024-11-20`	Third-party API provider	Available until 2026-10-23
`gpt-5-mini-2025-08-07`	Third-party API provider	Available until 2026-12-11

Table 15: Model snapshots, API providers, and availability used in our experiments.

latex format, the box is a strict format requirement, you can't change it, and your final answer must put inside the box.

Generate Available Actions

You are a PDDL expert and you are familiar with classical IPC planning domain. Your task is to generate all executable actions available in the current state. An executable action is defined as an action whose execution prerequisites are all satisfied in the current state. Your answer must satisfy all the following requirements:
(1)Each action must be written in PDDL symbolic format, and actions are separated by a period, such as 'pick-up(A). stack(B, C)'. Each action is parallel and does not interfere with one another. (2)The action you give must strictly follow the domain description, especially the preconditions of each action. DON'T generate any action that violates the preconditions at the current state. (3) Do not generate actions that have already been executed in the previous step, and do not generate the inverse actions of the actions executed in the previous step. For example, if the previous action is "(unstack red blue)", then its inverse action would be "(stack red blue)". Such inverse actions should not be generated in order to avoid producing duplicate states.
After your analysis, you should organize all available actions into a new paragraph as the end of your answer without any other content. Your last paragraph must be in the following format: 'box' and put all actions in the box. For example, 'box(pick-up red). (stack yellow blue)'. Remember, don't use any markdown and latex formatting. The only allowed format in your last

Domain	Model	APP	PROG	VAL
Ferry	GPT-4o	25968.4	25475.6	190886.7
	GPT-4o-mini	26257.4	25619.8	125555.2
	DeepSeek-v3.2	25450.9	25240.7	183732.5
Swap	GPT-4o	24519.3	23197.3	78205.3
	GPT-4o-mini	25784.6	24427.7	53880.7
	DeepSeek-v3.2	20989.0	23177.1	73839.2
Logistics	GPT-4o	27626.3	29994.3	327929.6
	GPT-4o-mini	29653.3	30542.5	228794.4
	DeepSeek-v3.2	30264.5	31122.5	161181.1
Rovers	GPT-4o	40338.4	41593.7	232325.3
	GPT-4o-mini	38677.8	39536.8	97158.5
	DeepSeek-v3.2	42053.9	43148.6	564587.2
Floortile	GPT-4o	28832.4	28754.8	96151.6
	GPT-4o-mini	25345.7	27309.2	85199.7
	DeepSeek-v3.2	30580.6	31617.1	105344.4
Goldminer	GPT-4o	30697.1	33558.9	237816.9
	GPT-4o-mini	29735.6	32207.2	59248.6
	DeepSeek-v3.2	34240.6	36094.8	80121.1

Table 16: Average token usage on ACPBench (Boolean questions). APP=Applicability, PROG=Progression, VAL=Validation.

paragraph of your answer is box.... You can't change the format, this is the strict requirement. DON'T USE ANY MARKDOWN and latex format.

State Check Prompt

You are a PDDL expert, and you are familiar with the classical IPC domains. Based on the domain description, the current state, the question and the example, You are required to determine whether the question is true or not. The domain of the example may be different from the current state, but the idea of finishing the task is the same. The format of the example must be always the same. Your answer should use the same format as the example to think and generate your answer.
There is two possible situations in the question. The first one is that the question is a goal description. The second one is that the question is a fact query. If the question is a goal description (case1), you need to check whether the current state is the goal state. If the question is a fact query (case2), you need to check whether the fact is true in the current state.
A state is the goal state if and only if all propositions in the goal state is satisfied. A fact is true if and only if all propositions in the fact is satisfied. If there is any proposition in the goal state or the fact is not satisfied, the state is not the goal state and the fact is not true.
So, first need to confirm which case the question belongs to. If it is a goal description (case1), you need to check whether the current state is the goal state. If it is a fact query (case2), you need to check whether the fact is true in the current state. Second, you need to check every proposition in the question one by one by comparing the propositions with the current state.
After your analysis, please your final answer in a new paragraph as the end of your answer. If all propositions are satisfied, return True, else return False. Your last paragraph should use this format: '`\box{True/False}`' with no other content and put your final answer inside the '`\box{}`'. Remember, don't use any markdown formatting and latex format, the `\box{}` is a strict format requirement, you can't change it, and your final answer must put inside the `\box{}`.

Domain	Model	APP	PROG	VAL
Ferry	GPT-4o	30594.1	35170.2	520208.5
	GPT-4o-mini	30225.7	34995.0	114640.5
	DeepSeek-v3.2	30745.0	33756.4	246106.7
Swap	GPT-4o	24905.4	29988.2	79292.2
	GPT-4o-mini	23686.3	32140.6	81349.5
	DeepSeek-v3.2	26439.2	31171.9	87923.7
Logistics	GPT-4o	37760.9	45324.1	282483.6
	GPT-4o-mini	36998.6	44458.1	294400.1
	DeepSeek-v3.2	40673.4	45615.9	203715.5
Rovers	GPT-4o	54515.8	49840.3	251711.5
	GPT-4o-mini	51074.7	52512.4	84770.8
	DeepSeek-v3.2	52909.3	50900.3	486298.0
Floortile	GPT-4o	28158.7	35398.3	85490.9
	GPT-4o-mini	26446.8	34704.1	123459.7
	DeepSeek-v3.2	33955.7	36661.9	125156.9

Table 17: Average token usage on ACPBench (multiple-choice questions). APP=Applicability, PROG=Progression, VAL=Validation. Goldminer results are not available for this variant.

Model	Domain	Total Tokens
GPT-5-mini	Blocksworld	2,279,871.5
	Depots	2,595,776.1
	Logistics	2,248,593.6
DeepSeek-v3.2	Blocksworld	3,512,014.7
	Depots	3,768,290.5
	Logistics	4,120,926.3

Table 18: Average token usage on PlanBench per domain.

Category	**Hyperparameter**	**Value**
LoRA Specs	LoRA rank	64
	LoRA alpha	16
	Dropout Prob.	0.05
Optimization	Optimizer	AdamW
	Peak Learning Rate	2e-4
	Warmup Ratio	0.03
	Learning Rate Scheduler	Cosine
Training Setup	Total Epochs	3
	Per-device Batch Size	4
	Gradient Accumulation Steps	4
	Mixed Precision	FP16
	Random Seed	42

Table 19: Hyperparameters of LoRA fine-tuning.

State Score Prompt (ToT and ablation study Only)

You are a PDDL expert and you are familiar with classical IPC planning domain. Based on the domain description, the current state and the goal state, score this state, you are required to score this state from 0-10. Your answer must satisfy all the following requirements: (1)Your score should be a float number from 0-10 with one digit. (2)Your scoring criteria is based on how close the state is to the goal state. The closer the current state is to the goal state, the lower the score; the further away the current state is from the goal state, the higher the score. If the current state is the goal state, you should return 0. After your analysis, you should organize your final answer in the new state into a new paragraph as the end of your answer without any other content. Your last paragraph must be in the following format: '\box{}' and put the score in the box. For example, '\box{0}'. Remember, don't use any markdown and latex formatting. The only allowed format in your last paragraph of your answer is \box{}. You can't change the format, this is the strict requirement.)