ACPBench Hard: Unrestrained Reasoning about Action, Change, and Planning

Harsha Kokel, Michael Katz, Kavitha Srinivas, Shirin Sohrabi

IBM Research
{Harsha.Kokel, Michael.Katz1, Kavitha.Srinivas}@ibm.com, ssohrab@us.ibm.com

Abstract

The ACPBench dataset provides atomic reasoning tasks required for efficient planning. The dataset is aimed at distilling the complex plan generation task into separate atomic reasoning tasks in their easiest possible form, boolean or multiple-choice questions, where the model has to choose the right answer from the provided options. While the aim of ACPBench is to test the simplest form of reasoning about action and change, when tasked with planning, a model does not typically have options to choose from and thus the reasoning required for planning dictates an open-ended, generative form for these tasks. To that end, we introduce ACPBench Hard, a generative version of ACPBench, with open-ended questions which the model needs to answer. Models that perform well on these tasks could in principle be integrated into a planner or be used directly as a policy. We discuss the complexity of these tasks as well as the complexity of validating the correctness of their answers and present validation algorithms for each task. Equipped with these validators, we test the performance of a variety of models on our tasks and find that for most of these tasks the performance of even the largest models is still subpar. Our experiments show that no model outperforms another in these tasks and with a few exceptions all tested language models score below 65%, indicating that even the current frontier language models have a long way to go before they can reliably reason about planning. In fact, even the so-called reasoning models struggle with solving these reasoning tasks.

Introduction

The ability to reason and plan is the cornerstone of artificial intelligence. With the introduction of large language models, a major focus in the field is on testing their abilities in these two fields, reasoning and planning. For reasoning, the majority of work focuses on the mathematical reasoning (Cobbe et al. 2021) and logical inference (Saparov and He 2023). For planning, most work focused on the ability to produce or validate a plan (Valmeekam et al. 2023a; Stein et al. 2024). The downside of focusing on the end-to-end planning is the inability to pinpoint the reason for a black-box planner, such as an LLM-based one, to not be able to produce a solution. To tackle this gap, recent work introduced an

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

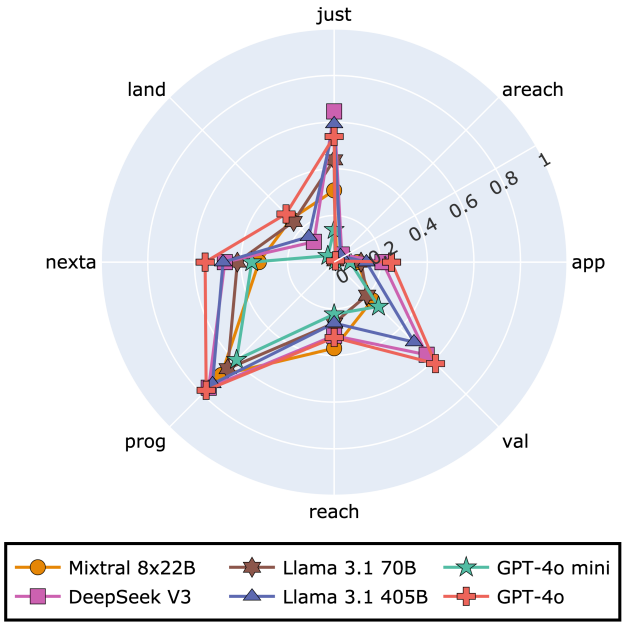


Figure 1: Task-wise accuracy of large size language models.

ACPBench dataset (Kokel et al. 2025), a benchmark for testing the reasoning abilities about action, change, and planning, separating the planning process into the atomic reasoning tasks performed by planners. The dataset featured 7 tasks of various reasoning types about action applicability, progression, reachability, sub-goals, plan justification and validation. For each of these tasks, two types of questions were constructed, boolean and multiple choice. The initial investigation aimed at devising the questions in the simplest form possible. However, the simplicity meant that the questions included additional information, which planners do not typically have, making the use of such models in planning harder. For instance, the applicability task either asks about applicability of a given action in the boolean case or chooses an applicable action among 4 variants. Even if we make the assumption that the model can answer such questions with high precision, it is not clear how this capability can be efficiently exploited for the task of generating all applicable actions in a given state.

Model	app	areach	just	land	nexta	prog	reach	val
Granite 3.1 8B base	0.00	0.01	0.20	0.09	0.18	0.42	0.32	0.12
Granite 3.1 8B	0.00	0.00	0.21	0.08	0.22	0.36	**0.33**	0.09
Llama 3.1 8B	0.00	0.00	**0.22**	0.06	**0.25**	0.40	**0.33**	0.13
DeepSeek coder 33B	**0.02**	**0.02**	0.21	0.10	0.17	0.42	0.18	**0.15**
Granite 34B code	**0.02**	0.00	0.17	**0.11**	0.18	**0.43**	0.28	0.12

Table 1: Small and medium size language models performance, best results bolded.

In this work, we build upon ACPBench to alleviate the limitations that were originally imposed. We devise open-ended, generative versions of questions for the same 7 tasks. Further, we introduce another, challenging task of finding an action that takes us closer to the goal, a task that corresponds to the ability to perform *optimal* planning (Bylander 1994). For each of these tasks, we introduce an evaluator, which scores a possible answer to the open-ended questions. We generate an evaluation set of questions based on a wide collection of 13 Planning Domain Definition Language (PDDL) (McDermott 2000) domains from ACPBench, which we call *ACPBench Hard*, in an attempt to evaluate the ability of large language models to produce reliable components for automated planners, as models that perform well on these tasks could in principle be integrated into a planner or be used directly as a policy. For all tasks, we discuss their computational complexity, as well as the complexity of validating their solutions. We devise validators for each task, based on the symbolic description of the questions. With the help of these validators, we test the performance on ACPBench Hard of a collection of modern large language models of various sizes, ranging from a few billion parameters to hundreds of billions if not trillions of parameters. We find that the performance of language models, even the largest ones, is still insufficient to be reliably used in planners, see Table 1 and Figure 1. We observe that there is no single model that outperforms all other models on all tasks of the ACPBench Hard dataset. Further, on half of the tasks, namely "atom reachability" (reach), "action reachability" (areach), "landmarks" (land), and "applicability" (app), all tested language models exhibit a very low accuracy. All tested language models score below 65% on most tasks, indicating that even the current frontier language models have a long way to go before they can reliably reason about planning. Further, even the so-called reasoning models o1 perform poorly on half of these tasks. The o1-preview model achieves 89% on the "progression" (prog) task and 80% on the "next action" (nexta) task, with the rest of the results being 66% and below, and the smaller o1-mini model outperforms it only on the "plan validation" (val) task with 78% accuracy. The much higher computational effort of the reasoning models compared to the language models do not seem to justify the somewhat moderate increase in accuracy. We note that the goal of ACPBench Hard is not simply to measure the performance of existing models - it is to provide benchmarks that help the community build *efficient* smaller models that can be used to plan; performance of state of the art computationally expensive models such as o1-preview and large LLMs are provided mostly as a reference.

In the rest of the paper, we will first discuss the background and related work, then in Section 4 discuss how the ACPBench Hard dataset is constructed, describing in detail all the 8 tasks. Then in Section 5, discuss how these open-ended answers are evaluated for each of the 8 tasks. Our extensive experimental evaluation is in Section 6, followed by observations we obtained from our experiments. We conclude with conclusion and next steps.

Background

We consider planning tasks $\Pi = \langle F, A, s_0, s_\star \rangle$ in the (propositional) STRIPS formalism (Bylander 1994). In such a task, F is a set of Boolean *propositions*. Each subset $s \subseteq F$ is called a *state*, and $S = 2^F$ is the *state space* of Π. The state s_0 is the *initial state* of Π. The goal $s_\star \subseteq F$ is a set of propositions, where a state s is a *goal state* if $s_\star \subseteq s$. The set A is a finite set of *actions*. Each action $a \in A$ has an associated set of *preconditions* $pre(a) \subseteq F$, *add* effects $add(a) \subseteq F$ and *delete* effects $del(a) \subseteq F$.

The semantics of STRIPS planning is as follows. An action a is *applicable* in the state s if $pre(a) \subseteq s$. Applying a in s results in the state $s[\![a]\!] := (s \setminus del(a)) \cup add(a)$. A sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ is *applicable* in s if there exists a sequence of states $s = s_1, \ldots, s_{n+1}$ such that for each $1 \leq i \leq n$ we have a_i is applicable in the state s_i and applying it results in the state s_{i+1}. If it exists, such sequence is uniquely defined, and its end state s_n is denoted by $s[\![\pi]\!]$. An applicable action sequence is a *plan* for s if $s[\![\pi]\!]$ is a goal state. A plan for s with minimal length is called *optimal*. The *perfect heuristic* for s, denoted by $h^*(s)$, or $h^*(s, \Pi)$ if the planning task is not clear from context, is the cost of an optimal plan for s. The objective of (optimal) planning is to find an (optimal) plan for the state s_0.

Related Work

The most relevant to our work is the ACPBench dataset (Kokel et al. 2025), which we build upon. ACPBench features 7 core reasoning tasks about planning, testing the ability of language models to make decisions that automated planners make. ACPBench introduces the simplest versions of these tasks in the form of boolean and multiple choice questions. While creating a simple evaluation setting, this form does not capture the decisions that the automated planners need to make, as these decisions are generative in their nature. For instance, automated planners do not check every candidate ground action whether it is applicable in the given state, they generate the set of applicable actions. In this work, therefore, we present the generative form of these tasks, requiring producing precisely the answers that automated planners produce and consume. Additionally, we present a new task, not considered by previous work, requiring to produce an action that takes us closer to the goal.

Other notable work in similar direction includes TRAC (He et al. 2023), featuring 4 tasks: projection, execution, planning, and goal recognition, as well as PlanBench (Valmeekam et al. 2023b), with 8 planning tasks including plan generation, reasoning about plan execution, and plan verification. Both benchmarks focus on a small number of

planning domains (mostly BlocksWorld and variants). Both use templates to generate natural language text. AutoPlanBench (Stein et al. 2024) alleviates the dependence on templates, leveraging LLMs to generate these natural language template and therefore were able to scale up the dataset to 12 domains. While scaling the number of domains, they limit their focus to one single task - plan generation.

Another notable dataset is ActionReasoningBench (Handa et al. 2024), featuring six tasks: fluent tracking, state tracking, action executability, effects of actions, numerical RAC, and composite questions. These tasks reason about action sequences and therefore two of the tasks overlap with three of the tasks we consider in our work. Specifically, action executability deals with questions that fall under applicability and validation tasks in our case, which we separate. The effects of actions task overlaps with our progression task.

The line of work on testing planning abilities of agents (Liu et al. 2023; Ma et al. 2024) is also somewhat relevant. In this work we focus on each individual decision instead of the overall planning abilities of agents. Having said that, our new "next action" prediction task does takes us in that direction, while focusing on optimality.

Dataset Construction

We build upon ACPBench, using the same underlined mechanisms (Kokel et al. 2025). We keep the same 13 planning domains and 7 tasks, adding one more task. In what follows, we describe the generative versions of the tasks. For each task, we describe the data stored in order to enable or speed up the evaluation of the correctness of a potential answer. To enable the evaluation, we also store the PDDL planning task $\Pi = \langle F, A, s, s_\star \rangle$ with the current state as an initial state with the question, where needed. The data will be made available upon acceptance.

1. Applicability (App) The first task deals with identifying which actions are applicable in a state. For an action to be applicable, its preconditions must hold in the state. Given a state s and the set of actions A, the subset of applicable actions would be $A(s) = \{a \in A \mid pre(a) \subseteq s\}$, easily computable by iterating over the actions. The complexity of such an iterative algorithm is $O(|F||A|)$, since $|F|$ is a theoretical upper bound on the precondition size. In practice, planning problems typically have small preconditions size. We therefore can create a generative question by simply asking the model to produce all applicable actions in a given state. For practical reasons, we impose a bound on the number of applicable actions in a state, generating questions only in cases when $|A(s)|$ is under that bound. In such cases, we can keep all applicable actions names for answer validation.

2. Progression (Prog) The next task evaluates LLMs ability to understand how the world state changes by action application. Performing an action changes the state in the following manner: The delete effects will no longer hold and the add effects will hold. Everything else remains unchanged. Given a state s and an action a, the next state is $t = (s \setminus del(a)) \cup add(a)$. The complexity of the straightforward computation is $O(|F|)$ worst case, but in practice the add and delete effects of planning problems are typically small. We construct a single generative question, asking what propositions are false in the current state but will become true after performing the action and asking which ones are true and become false. The first set is $t \setminus s$, and the second one is $s \setminus t$. We can keep the relevant sets for answer validation. Figure 2 shows a sample example of a progression question.

```
Context: There are several cities, each
  containing several locations, some of
  which are airports. There are also
  trucks, which can drive within a single
  city, and airplanes, which can fly
  between airports. The goal is to get
  some packages from various locations to
  various new locations. There are 2
  trucks and 1 airplane, as well as 4
  packages. There are 4 locations across 2
   cities. The locations are in cities as
  follows: l1-1 and l1-0 are in c1; l0-1
  and l0-0 are in c0. Currently, p2, t1,
  p1, p3, a0, and p0 are at l1-0, t0 is at
   l0-1. The available propositions are:
  (at ?obj ?loc) - ?obj is at ?loc and (in
   ?obj1 ?obj2) - ?obj1 is in ?obj2.

Inputs: Break down the outcomes of
  performing the action "load object p3
  into truck t1 at location l1-0" into two
   lists, positive effects and negative
  effects. Positive effects are the
  propositions that are false in the
  current state but will become true after
   performing the action. Negative effects
   are the propositions that are true in
  the current state and will become false
  after performing the action.
```

Figure 2: Example of a question for the progression task in ACPBench Hard. The question is composed out of the context, which contains the domain and the problem description, as well as the current state description, and the inputs, the actual question.

3. Reachability (Reach) The reachability task evaluates if a specific fact can eventually become true by taking (possibly multiple) actions in the given state. This is a multi-step reasoning task that can help avoid exploring unfeasible options. The generative version of this question is quite simple: what proposition can never hold in any potentially reachable state. If no such propositions exist, we instruct to reply *None*.

While reachability is PSPACE-hard to answer in general (Bylander 1994) for a specific fact, we can generate some negative examples by either finding groundings of *static* predicates (unchanged by any action) that do not hold in the given state, or by (under)approximating the reachability with poly-time computable delete-relaxed reachability (Hoffmann and Nebel 2001). For practical reasons, we keep a subset of generated negative examples for speeding up answer validation. We generate question only in cases when we

either *know at least one unreachable fact* or we *know that all facts are reachable*. In case of doubt, we skip generating the question.

4. Action Reachability (AReach) The action reachability task is closely related to the atom reachability, checking whether there is a reachable state where the action is applicable. Computationally, this problem is PSPACE-hard, for the same reason as atom reachability. The generative version of the question is: what action can never become applicable, in any state reachable from the current state? For answer validation, we keep example unreachable actions, based on the evidence as in the previous case, when such actions are found. Here as well, we generate question only in cases when we either *know at least one unreachable action* or we *know that all actions are reachable*. In case of doubt, we skip generating the question.

5. Validation (Val) The validation task aims at checking whether the specified sequence of actions π is a plan. In other words, whether π is valid, applicable, and successfully achieves the intended goal from the given state s. The generative version of this question aims at identifying where the check fails. In other words, we ask to identify the first inapplicable action in a given sequence of actions. For validation purposes, we keep the index of such action.

6. Justification (Just) Action justification (Fink and Yang 1992; Salerno, Fuentetaja, and Seipp 2023) deals with the question of whether the given plan can be simplified by removing some actions. While the problem is NP-hard in general, checking whether a single action or a pair of consequent actions can be removed can be done in polynomial time. The generative version of the justification task question asks to simplify the plan by removing one or two consecutive actions and to produce the resulting simplified plan. To produce such a sequence, we start with a plan for the initial state, computed with a top-quality planner (Katz and Lee 2023a) or a diverse planner (Katz and Sohrabi 2020) when no optimal solution could be found within the bound. We check whether the plan can be simplified by removing a single or two consecutive actions. If not, we try to extend the plan by adding such an action or a pair of actions, keeping the resulting sequence a plan.

For answer validation, we keep the action or pair of actions that can be removed, together with the appearance number, for actions that appear more than once on the plan.

7. Landmarks (Land) Landmarks task tests LLMs ability to identify subgoals that are necessary to achieve the goal. In planning literature such subgoals are often called landmarks (Porteous, Sebastia, and Hoffmann 2001). Landmarks are facts that must become true sometime along every plan.

While checking whether a proposition is a landmark is PSPACE-hard (Porteous, Sebastia, and Hoffmann 2001), there are several methods that can find a subset of landmarks (Keyder, Richter, and Helmert 2010; Hoffmann, Porteous, and Sebastia 2004; Richter, Helmert, and Westphal 2008; Zhu and Givan 2003). We use the so-called RHW method (Richter, Helmert, and Westphal 2008). Further, negative evidence can be obtained from a collection of plans - a proposition that does not appear on a plan is not a landmark. We keep the sets of facts that are known to be landmarks and of facts that are known to be non-landmarks for speeding up answer validation.

8. Next Action (NextA) An additional task that does not appear in the original ACPBench is the next action task. This generative question asks what is the next action that takes us towards the goal. This task is closely related to optimal planning, since optimal plans can be produced by iteratively obtaining such actions.

While even non-optimal planning is PSPACE-hard (Bylander 1994), modern planners can often quickly find collections of optimal plans (Katz, Sohrabi, and Udrea 2020; Katz and Lee 2023b). Clearly, the first actions of these plans would be correct answers. Further, the cost of these optimal plans can be used to check other applicable actions in the state, by producing an optimal plan for the states obtained by applying these actions in the question state. We keep the sets of actions that are correct answers and of actions that are known to not take us closer to the goal for speeding up answer validation. Further, we can store the optimal cost $h^*(s)$ of achieving the goal from the current state.

Answer Evaluation

Evaluating answers to boolean or multiple-choice questions simply amounts to looking up the correct answer and comparing. Evaluating open-ended answers, on the other hand, might not be as easy. This is due to the fact that sometimes, there is not one single correct answer that can be stored with the question. Looking at the tasks at hand, while in some cases we can store the complete correct answer, in other we must resort to performing some computation in order to evaluate whether the returned answer is correct. In what follows, we describe how the answer is evaluated for each task.

1. Applicability (App) In this task, we store all applicable action names per question. The answer evaluation therefore amounts to a simple comparison between the given answer and the correct one. Since we ask for all applicable actions, we chose to assign a score 1 if the set of all actions in the answer equals to the set of all applicable actions. Otherwise, we assign 0. The complexity of validating the answer is therefore $O(1)$ if we impose a constant threshold on the number of applicable actions when the question is created, otherwise it is $O(|A|)$.

2. Progression (Prog) Here, we store both correct sets of propositions per question. An answer validation amounts to a simple comparison between the given answer and the correct one. Since we test the ability to produce all action effects, we score 1 if both all positive and all negative effects were correctly identified. Otherwise, we give the score 0. The complexity of validating the answer is therefore $O(|F|)$.

3. Reachability (Reach) If the answer P is *None*, we score it as 1 if the set of kept negative examples is empty and otherwise as 0. This is due to the fact that if the set of kept negative examples N is not empty, we *know* that there exist unreachable propositions. If, however, N is empty, we

know that there exists no unreachable proposition, otherwise we would not create a question. Otherwise, if P is in the subset of kept propositions N that are known to be unreachable, we can return the score 1. In case P is not *None* and is not in N, in order to check whether P is reachable, we need to solve a planning task $\Pi' = \langle F, A, s_0, \{P\}\rangle$, where the goal is to achieve the atom P and the rest as in the planning task in the question. We can use any off-the-shelf planner to generate a plan for this task. If a plan exists, we score the answer 0, otherwise 1. The answer validation is therefore PSPACE-complete in this case.

4. Action Reachability (AReach) For an action a, we construct a planning task $\Pi' = \langle F, A, s_0, pre(a)\rangle$ with the goal being the preconditions of the action a, and the rest as in the planning tasks in the question. We run a planner on this task, checking if a plan exists. If yes, we score the answer 0, otherwise 1. If the answer is *None*, we score it as 0 if there exist unreachable actions and otherwise as 1. As in the previous case, if no unreachable actions are kept, we *know* that all actions are reachable, otherwise we would not create a question. The answer validation is therefore PSPACE-complete in this case as well.

5. Validation (Val) In this case, we only need to compare the index in the answer to the correct index of the first inapplicable action. We give a 0/1 score based on that comparison. The complexity of validating the answer is therefore $O(1)$.

6. Justification (Just) To check whether the returned sequence is correct, we slightly relaxed the constraint in the question. We check whether it is a proper subsequence of the provided plan and whether it is a plan. If both are true, we give a score of 1, otherwise, we score the answer 0. The complexity of validating the answer is therefore $O(|\pi||F|)$ for the plan π.

7. Landmarks (Land) Invalid propositions are scored 0. A proposition $p \in s_0 \cup s_\star$ is a trivial landmark and is also scored 0. Given a valid proposition $p \in F \setminus (s_0 \cup s_\star)$, if it is known not to be a landmark (e.g., there exists a plan that does not traverse any state in which p holds), we assign it a score of 0. Otherwise, in order to check if p is a non-trivial landmark, we construct a planning task $\Pi' = \langle F', A', s'_0, s'_\star\rangle$ as follows. The set of propositions is extended with a proposition p_{ach} that is intended to indicate that p was never achieved along a sequence of actions. $F' = F \cup \{p_{nach}\}$ Thus, for each action $a \in A$ such that $p \in add(a)$, we construct a new action with extended delete effect to include p_{nach}. Formally, for an action a, a' is defined as follows. $pre(a') = pre(a)$, $add(a') = add(a)$, and $del(a') = del(a) \cup \{p_{nach}\}$ if $p \in add(a)$, otherwise $del(a') = del(a)$. The extended action set is therefore $A' = \{a') \mid a \in A$. Finally, both the initial state and the goal are extended with p_{nach}: $s'_0 = s_0 \cup \{p_{nach}\}$, $s'_\star = s_\star \cup \{p_{nach}\}$, indicating that we are interested in plans that do not achieve p along their path, as any action that achieves p will delete p_{nach} from the state, making the goal not reachable. We use an off-the-shelf planner to check if there is a plan for Π'. If there is one, it corresponds to a plan for Π that does not make p true and therefore p is not a landmark, and we assign the score of 0. If Π' is found unsolvable, then p is a landmark and we assign the score of 1. The answer validation is therefore PSPACE-complete in this case as well.

```
NAME: /[a-zA-Z][a-zA-Z0-9-_]*/
LPAR : "("
RPAR : ")"
LSPAR: "["
RSPAR: "]"
COMMA: ","
WS: /[ \n]/
action_none : "None"
action_name : LPAR NAME (WS NAME)* RPAR
action_list : (action_name WS?)*
prog_list :  action_name* (COMMA
 action_name)*
progression_list : LSPAR prog_list RSPAR
 LSPAR prog_list RSPAR
act : action_name | action_none
index: /[0-9]+[0-9]*/
```

Figure 3: Grammar used for parsing the model response.

8. Next Action (NextA) For an action a, if it is in the set of kept correct answers, we score it 1 and if it is in the set of known incorrect answers, we score it 0. Otherwise, if it is applicable, we apply it to the current state s and obtain the state t. We find the optimal plan costs $h^*(s)$ and $h^*(t)$ for these two states with the help of an optimal planner and score the answer 1 if $h^*(s) - h^*(t) = 1$. Otherwise, we score it 0. The answer validation is therefore PSPACE-complete in this case as well.

Experiments

We evaluated the following 14 language/reasoning models, with the aim to cover small, medium, and large models:

- small size: *Granite 3.1 8B base* and *Granite 3.1 8B* (instruct) (Granite Team 2024), as well as *Llama 3.1 8B* (instruct) (Dubey et al. 2024),
- medium size: *DeepSeek coder 33B* (instruct) (Guo et al. 2024) and *Granite 34B code* (instruct) (Mishra et al. 2024),
- large: *Mixtral 8x22B* (instruct) (MistralAI 2024), *Llama 3.1 70B* (instruct) and *Llama 3.1 405B* (instruct) (Dubey et al. 2024), *DeepSeek V3* (DeepSeek-AI et al. 2025b) and *DeepSeek R1* (DeepSeek-AI et al. 2025a) *GPT-4o mini* and *GPT-4o* (OpenAI et al. 2024a), *o1 mini* and *o1-preview* (OpenAI et al. 2024b).

All models were either accessed using API or hosted locally using hugging face transformer library on machines with 2 $A100$ 80 GB GPU. It is important to note that there is a significant difference in the energy consumption and evaluation cost between various models. While smaller models are relatively cheap, the larger language models such as Llama 3.1 405B and GPT-4o are prohibitively expensive to be used as planner components. The reasoning models such as o1 and even the cheaper DeepSeeek R1 are even more

Model	app	areach	just	land	nexta	prog	reach	val
Mixtral 8x22B	0.10	0.02	0.31	0.26	0.32	0.68	0.37	0.23
Llama 3.1 70B	0.12	0.02	0.44	0.20	0.42	0.65	0.28	0.20
GPT-4o mini	0.07	0.01	0.14	0.04	0.35	0.59	0.22	0.27
Llama 3.1 405B	0.14	0.04	0.59	0.15	0.48	0.74	0.26	0.48
GPT-4o	0.25	0.01	0.54	0.29	0.55	0.78	0.32	0.62
DeepSeek V3	0.21	0.05	**0.65**	0.12	0.47	0.76	0.32	0.56
DeepSeek R1	0.05	0.01	0.52	0.20	0.36	0.77	0.24	0.53
o1 mini	0.38	0.06	0.44	0.38	0.64	0.70	0.60	**0.78**
o1-preview	**0.44**	**0.12**	0.46	**0.56**	**0.80**	**0.89**	**0.66**	0.26

Table 2: Large size language and reasoning models accuracy. Bold entries indicate the best overall model, while the underlined entries indicate the best language model.

expensive than the language models. Therefore, our experiments with these reasoning models are intended mostly for providing a frame of reference.

For each task, we evaluated a 2-shot prompting with static examples from outside the evaluation set. The two examples come from the grid and logistics domains, one each per task. This allows to exemplify the expected response format. Additionally, we instructed the language models to produce their response in a particular format. Still, the tested language models do not necessarily adhere to the instructions or the example format. Therefore, to be able to extract the answer from the response, we developed a lenient *grammar* based parser, which would discard tokens if they did not fit expected token values. Figure 3 shows the grammar used for parsing the responses for our 8 tasks. For example, in the *progression task*, the response is a `progression_list`, which is two lists, one for the positive effects and one for the negative effects. The response for the *justification* task is a `action_list`, which is a list of actions (or the updated plan) and the response for the *validation* task is an `index` which is the index of the first inapplicable action. Using the grammar with a parser that discards tokens that were not consistent with the grammar helps significantly in post-processing these open-ended responses.

Focusing first on the small and medium size language models, Table 1 shows the accuracy of these models on the 8 tasks in our dataset. A conclusion we can draw from the figure is that there are three categories of tasks according to model performance. First, *progression* seems to be the easiest tasks among the 8 tasks. Even so, the highest accuracy reached is 43% by Granite 34B code. The second category is the hard tasks of *applicability* and *action reachability*. In fact, none of these models could score above 2% for these two tasks. The third category includes the other 5 tasks with an average performance between 8.8% and 28.8%. Hence, ACPBench Hard seems to be a difficult dataset for the small and medium size language models.

Moving on to large size models, Table 2 shows the accuracy of the tested large language and reasoning models. Looking first at the language models, the top part of the figure, we observe that there is no single model that outperforms all other models. GPT-4o is the top performer, with best results in 5 out of 8 tasks, but it is outperformed by 5% by Mixtral 8x22B on the *atom reachability* task, and by

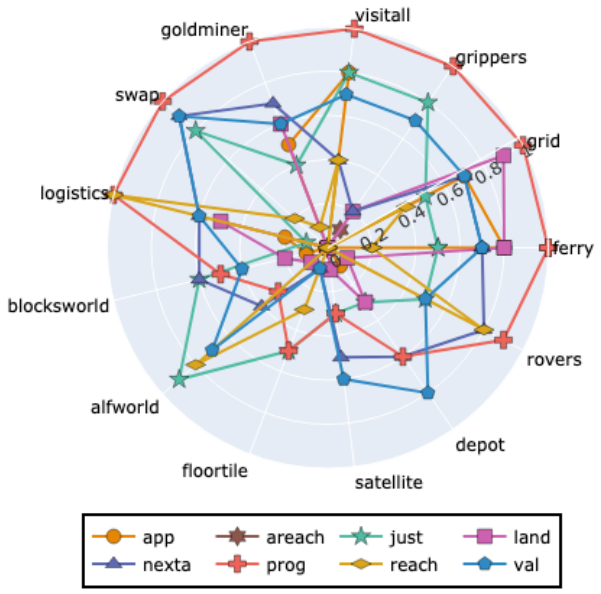


Figure 4: Domain-wise accuracy of GPT-4o.

DeepSeek V3 on the *action reachability* task by 4% and on the *justification* task by 11%. While large reasoning models show a better performance across most tasks, on the *justification* task the DeepSeek V3 model remains the top performer. The second observation we can make is that similar to the small and medium size language models, there are tasks such as *progression* that is relatively easy for all models, and there are tasks such as *action reachability* that is difficult for all models, and even o1-preview achieves only 12%. For other tasks, some models do better than others. Our third observation is that with the exception of what seems to be the easiest for these models *progression* task, there are only three scores above 65%, and all of them are of reasoning models, indicating that the ACPBench Hard dataset is difficult even for large models.

One of these tasks, specifically the new *next action* task shows a higher than expected accuracy for such a computationally hard task. In an attempt to better understand the phenomena, Table 3 shows the per-domain split. The bolded and underlined values depict the top accuracy values per domain across the tested models. Note that the accuracy is not uniform across domains. Some more complex domains like *depot* showing better performance across models than than easy domains such as *grippers*, that is hard for all models. There are some domains, like *logistics* and *ferry* that are hard for some models and easy for other. There are domains like *floortile* and *alfworld* that are very hard for almost all models.

Going deeper, we look into performance across different domains. Figure 4 presents the performance of the top-performing language model GPT-4o on various tasks, across the existing domains. For other models performance we refer to the supplementary material. Observe that GPT-4o exhibits high performance on the *progression* task across most

Model	depot	goldminer	satellite	swap	alfworld	ferry	logistics	blocks	grid	floortile	grippers	rovers	visitall
Mixtral 8x22B	0.4	0.2	0.4	0.8	0.1	0.3	0.4	0.3	0.3	0.1	0.2	0.4	0.3
Llama 3.1 70B	0.6	0.4	0.5	0.7	0.1	0.2	0.4	0.5	0.5	0.4	**0.3**	0.3	0.5
GPT-4o mini	0.4	0.4	0.4	0.7	0.0	0.5	0.1	0.4	0.6	0.1	0.2	0.3	0.5
DeepSeek V3	0.5	0.4	0.2	**1.0**	0.2	0.7	**0.8**	0.6	0.6	0.0	0.1	0.5	0.5
Llama 3.1 405B	0.6	0.4	0.4	**1.0**	0.4	0.4	0.6	0.2	0.6	0.0	**0.3**	0.6	0.7
GPT-4o	0.6	0.7	0.5	0.9	0.4	0.7	0.6	0.6	0.7	0.1	0.2	0.8	0.4
DeepSeek R1	0.4	0.5	0.2	0.9	0.0	0.4	0.6	0.5	0.4	0.0	0.1	0.2	0.5
o1 mini	**0.9**	0.8	0.5	0.9	0.5	**1.0**	0.4	0.6	**0.9**	0.3	0.2	0.4	**0.9**
o1-preview	0.8	**0.9**	**0.8**	0.9	**0.9**	**1.0**	**0.8**	**0.8**	**0.9**	**0.8**	0.1	**0.9**	0.8

Table 3: Accuracy of large size models on the *next action* task, split by domain. Bold entries indicate the best overall model, while the underlined entries indicate the best language model.

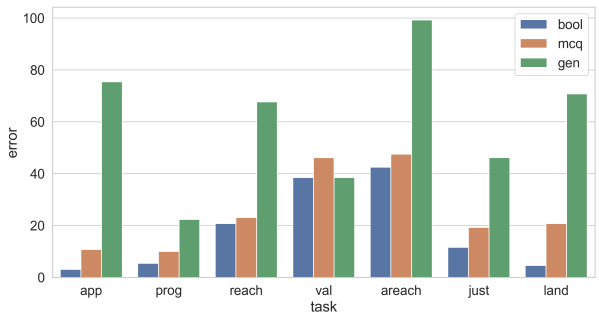


Figure 5: Comparison of prediction error for GPT-4o on different question formats.

domains, showing somewhat lower accuracy in *satellite*, *alfworld*, and *floortile*. *Action reachability* and *landmarks* tasks, on the other hand, pose significant challenges to the model. Notably, the model consistently generated accurate answers for the *visitall* and *grid* domains across most tasks. In the remaining domains, however, the performance varied across tasks, indicating that none of these domains are particularly easy for the language model.

Finally, we evaluate the hardness of our proposed open-ended variants by comparing the performance of GPT-4o on these questions (gen) to the two formats from ACPBench: boolean (bool) and multiple choice (mcq). Figure 5 depicts the comparison in terms of model errors, the complement of the accuracy. As can be seen, model error in generative format of the task is significantly higher than bool and mcq except for the *validation* task. Note that we are showing results from one of the largest models for this analysis. The gap in performance is even more prolonged in other models.

Observations

In this section, we provide some observations obtained from looking at individual answers provided by the models.

The 2-shot setting seemed to be sufficient for the language models to mostly follow the instructions on the answer syntax. The number of parsing errors was typically negligible, with a single exception of Llama 3.1 8B on the applicable actions task, where in 4 cases the model made up objects that did not follow the naming convention in the question and in 5 cases the model ran out of context generating the same actions over and over again.

While atom and action reachability are very much related tasks, the latter seems to be much harder for the tested models. It is not surprising, as action reachability requires an additional reasoning step about the atoms in the action preconditions and their reachability. Further, while atom reachability focuses on single atoms, action reachability requires reasoning about the entire precondition, that often consists of multiple atoms. The reasoning should now account for their interplay, as they need to hold in the same reachable state. In fact, action reachability seems to be consistently the most difficult task across the tested models. Here, the models needed to provide an action that can never become applicable, in any reachable state or None if there are no such actions. Interestingly, only the OpenAI models were able to correctly identify the latter case, and these were the majority of their correct answers. For instance, 9 out of 16 correct answers of o1-preview and all the 8 correct answers of o1 mini were *None*. Most of the other correct answers recognized that a block cannot be (un)stacked on itself, which accounts for 5 out of 6 correct answers for DeepSeek V3, 4 out of 5 for Llama 3.1 405B, and 4 out of 16 for o1-preview.

Surprisingly, the second hardest task is action applicability. While one of the core tasks in planning, it seems to be quite challenging even for the reasoning models, the largest of which scored 44%, while DeepSeek R1 scoring only 5%. The smaller language models fail miserably on this task, with only DeepSeek coder 33B and Granite 34B code showing performance slightly above 0. Even the largest language models barely reach 25%. One reason for that is the strict requirement for producing precisely the set of all applicable actions. While missing actions can hinder completeness and cause missing existing solutions, extra made up actions can lead to producing incorrect solutions, which is arguably worse. Interestingly, if we only required not to make up actions, but allowed producing subsets of real answers, scoring according to Jaccard similarity, the score of the best performing model o1 preview would go up to 57%. The largest

absolute increase would be for Mixtral, going from 10% to 38%. Using such action generation in a planner would correspond to loosing completeness, while keeping the soundness of the planner.

Looking at the landmarks task, among the smaller models, the best performing is Granite 34B. It gives quite uniform answers. Most of its correct answers are in the ferry domain, where it correctly identifies the ferry location as a landmark. In logistics, it recognizes a package need to be in the truck in the goal city before it can be delivered. The best performing large language model GPT-4o achieves 29% accuracy, producing a diverse set of answers. In ferry, it sometimes correctly identifies ferry being empty as a landmark, sometimes the need for a car to be onboard, and sometimes a ferry location was identified as a landmark. In logistics, trucks or packages at particular locations were identified as landmarks. In grid, it was more uniform, reporting in most cases holding a key as a landmark. In goldminer, all correctly identified landmarks were some location being clear. The best performing reasoning model o1-preview achieves 56% accuracy. It correctly reports twice in satellite and 6 times in swap the absence of non-trivial landmarks. The model that is best at recognizing that is o1 mini, with 3 cases in visitall, 4 cases in satellite, 5 in swap, and once in alfworld. The only other model that can identify such cases is Mixtral 8x22B, with 3 cases in visitall.

The justification task is among the few tasks where the best among the language models is not GPT-4o, but DeepSeek V3. In fact, it performs much better than even the reasoning models. The second best performer is Llama 3.1 405B. Interestingly, in multiple cases, the Llama model produces a valid plan, which is shorter than the given plan, but not its subsequence, and therefore not a correct answer. In some cases, the Llama model produced a valid plan that changes the order of the given plan actions. The aim of the justification task, however, is not to produce a valid plan, but to recognize unnecessary actions on a sequence.

The validation task requires an index to be returned, which makes it harder to investigate the source of the mistakes made. Most language models do not perform well on this task, with only the largest models go above 30%. Interestingly, the reasoning model o1-preview scores much lower than o1 mini (26% vs 78%). To investigate the source of its mistakes, we check the absolute difference between the given answer and the correct one. In 86% of the cases that o1-preview incorrectly answered the question, it errored by 1. It is important to note that it was mistaken in both directions, giving both lower and higher than the correct indices. An additional observation is that several models favor the answers 1, 10, and 100. For the majority of smaller models, most their answers are one of these three numbers.

The progression task is mostly simple for the reasoning model o1-preview, which reaches 89%, but even this model makes mistakes. For example, it does not recognize reasonable positive effects, such as stacking a block on top of another block makes the top block clear. Not surprisingly, it misses less reasonable negative effects, such as taking a picture with a rover camera makes the camera not calibrated. Surprisingly, it invents unreasonable effects such as communicating the rock data from rover to the lander would empty the rover's store and delete the effect of rocks being analyzed at the waypoint.

Conclusions and Future Work

In this work, we have extended the existing benchmark dataset ACPBench with an additional computationally challenging task of finding an action that takes us closer to the goal, as well as with open-ended questions for all the tasks, old and new. These open-ended questions reflect precisely the questions answered by symbolic planners during the planning process and therefore we believe that this new dataset, ACPBench Hard is a good benchmark for testing reasoning abilities that are required for planning. We performed an empirical investigation of a collection of large language models of various sizes and found that this new benchmark set is significantly more challenging for these models. In fact, on some of the tasks, the performance of all these models drops to 0% and on most tasks it is below 60%. For smaller and medium size models the average performance is around 16%, while for largest language models the average performance is around 40%. Even the largest reasoning model o1-preview only reaches 52% on average. We conclude that even the largest and best performing models have a very long way to go before they can be reliably used for planning.

We see these results as an opportunity for improvement. By identifying which tasks need more focus, we pave the way to better performance of future models on these tasks, possibly with advanced prompting techniques, beyond multiple shots, as multiple shots were often found detrimental in our experiments. Another way to improve models' performance on these tasks would be fine-tuning on training data, similarly to what was done in ACPBench. There, a fine-tuned on boolean and multiple-choice questions training data smaller model has performed on par with the largest and best-performing frontier models. In our preliminary experiments, however, that model did not perform well on our dataset, tending to return answers that it was trained on - yes/no, A, B, C, or D. Creating training data for generative questions, possibly with a chain of thought, would therefore be a promising area of future research. Another avenue for future research would be extending the benchmark to additional new tasks such as object counting in a current state.

References

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. 69(1–2): 165–204.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; Hesse, C.; and Schulman, J. 2021. Training verifiers to solve math word problems. arXiv:2110.14168 [cs.LG].

DeepSeek-AI; Guo, D.; Yang, D.; Zhang, H.; Song, J.; et al. 2025a. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948.

DeepSeek-AI; Liu, A.; Feng, B.; Xue, B.; Wang, B.; Wu, B.; et al. 2025b. DeepSeek-V3 Technical Report. arXiv:2412.19437.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; Goyal, A.; et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

Fink, E.; and Yang, Q. 1992. Formalizing Plan Justifications. In *Proceedings of the Nineth Biennial Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI 1992)*.

Granite Team, I. 2024. Granite 3.0 Language Models.

Guo, D.; Zhu, Q.; Yang, D.; Xie, Z.; Dong, K.; Zhang, W.; et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196*.

Handa, D.; Dolin, P.; Kumbhar, S.; Baral, C.; and Son, T. C. 2024. ActionReasoningBench: Reasoning about Actions with and without Ramification Constraints. *CoRR*, abs/2406.04046.

He, W.; Huang, C.; Xiao, Z.; and Liu, Y. 2023. Exploring the Capacity of Pretrained Language Models for Reasoning about Actions and Change. In *ACL*. Association for Computational Linguistics.

Hoffmann, J.; and Nebel, B. 2001. RIFO Revisited: Detecting Relaxed Irrelevance. In Cesta, A.; and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 127–135. AAAI Press.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. 22: 215–278.

Katz, M.; and Lee, J. 2023a. K* and Partial Order Reduction for Top-quality Planning. In Barták, R.; Ruml, W.; and Salzman, O., eds., *Proceedings of the 16th Annual Symposium on Combinatorial Search (SoCS 2023)*. AAAI Press.

Katz, M.; and Lee, J. 2023b. K* Search Over Orbit Space for Top-k Planning. In Elkind, E., ed., *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI 2023)*. IJCAI.

Katz, M.; and Sohrabi, S. 2020. Reshaping Diverse Planning. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9892–9899. AAAI Press.

Katz, M.; Sohrabi, S.; and Udrea, O. 2020. Top-Quality Planning: Finding Practically Useful Sets of Best Plans. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9900–9907. AAAI Press.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and Complete Landmarks for And/Or Graphs. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 335–340. IOS Press.

Kokel, H.; Katz, M.; Srinivas, K.; and Sohrabi, S. 2025. ACPBench: Reasoning about Action, Change, and Planning. In Shah, J.; and Kolter, Z., eds., *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2025)*. AAAI Press.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; Zhang, S.; Deng, X.; Zeng, A.; Du, Z.; Zhang, C.; Shen, S.; Zhang, T.; Su, Y.; Sun, H.; Huang, M.; Dong, Y.; and Tang, J. 2023. AgentBench: Evaluating LLMs as Agents. *CoRR*, abs/2308.03688.

Ma, C.; Zhang, J.; Zhu, Z.; Yang, C.; Yang, Y.; Jin, Y.; Lan, Z.; Kong, L.; and He, J. 2024. AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents. *CoRR*, abs/2401.13178.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. 21(2): 35–55.

Mishra, M.; Stallone, M.; Zhang, G.; Shen, Y.; Prasad, A.; et al. 2024. Granite Code Models: A Family of Open Foundation Models for Code Intelligence. *CoRR*, abs/2405.04324.

MistralAI. 2024. Mixtral 8x22B. https://mistral.ai/news/mixtral-8x22b/.

OpenAI; :; Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; et al. 2024a. OpenAI GPT-4o System Card. arXiv:2410.21276.

OpenAI; :; Jaech, A.; Kalai, A.; Lerer, A.; Richardson, A.; et al. 2024b. OpenAI o1 System Card. arXiv:2412.16720.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the Extraction, Ordering, and Usage of Landmarks in Planning. In Cesta, A.; and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 174–182. AAAI Press.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982. AAAI Press.

Salerno, M.; Fuentetaja, R.; and Seipp, J. 2023. Eliminating Redundant Actions from Plans using Classical Planning. In Marquis, P.; Son, T. C.; and Kern-Isberner, G., eds., *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*, 774–778. IJCAI Organization.

Saparov, A.; and He, H. 2023. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*. OpenReview.net.

Stein, K.; Fišer, D.; Hoffmann, J.; and Koller, A. 2024. AutoPlanBench: Automatically generating benchmarks for LLM planners from PDDL. arXiv:2311.09830 [cs.AI].

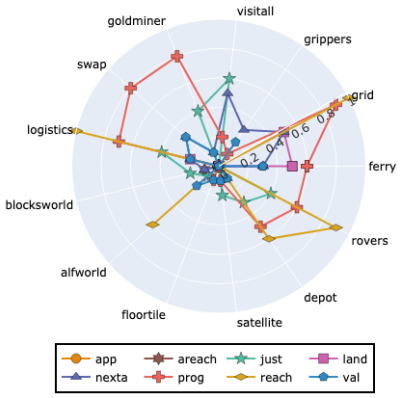
Valmeekam, K.; Marquez, M.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2023a. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 38975–38987.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023b. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*.

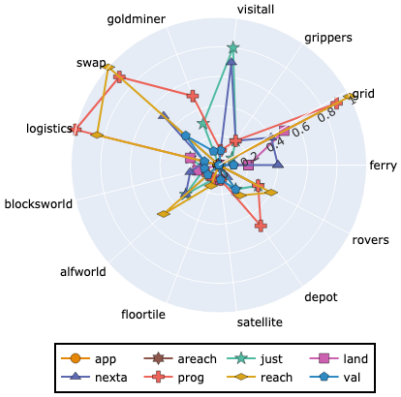
Zhu, L.; and Givan, R. 2003. Landmark Extraction via Planning Graph Propagation. In *ICAPS 2003 Doctoral Consortium*, 156–160.

A Domain-wise Accuracy of Tested Models

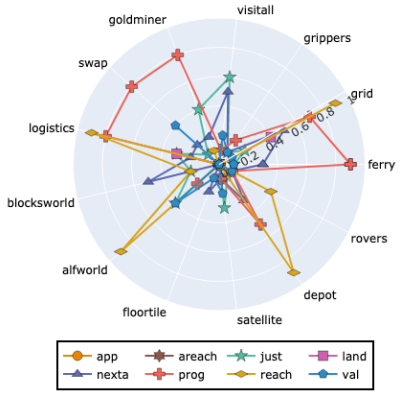
Figures 6 and 7 show the domain-wise accuracy of tested small and medium language models, while Figure 8 presents the domain-wise accuracy of tested large language and reasoning models.



(a) Granite 3.1 8B base.

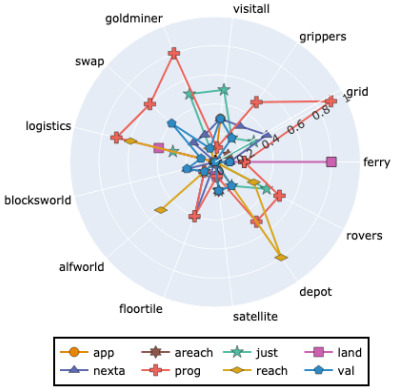


(b) Granite 3.1 8B.

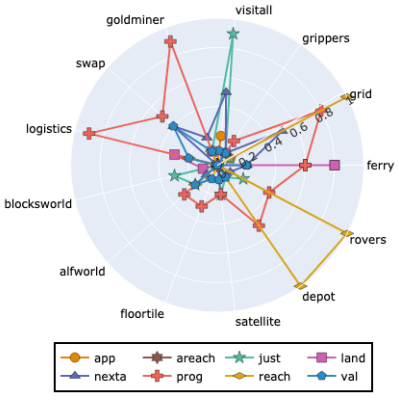


(c) Llama 3.1 8B.

Figure 6: Domain-wise accuracy of small size language models.

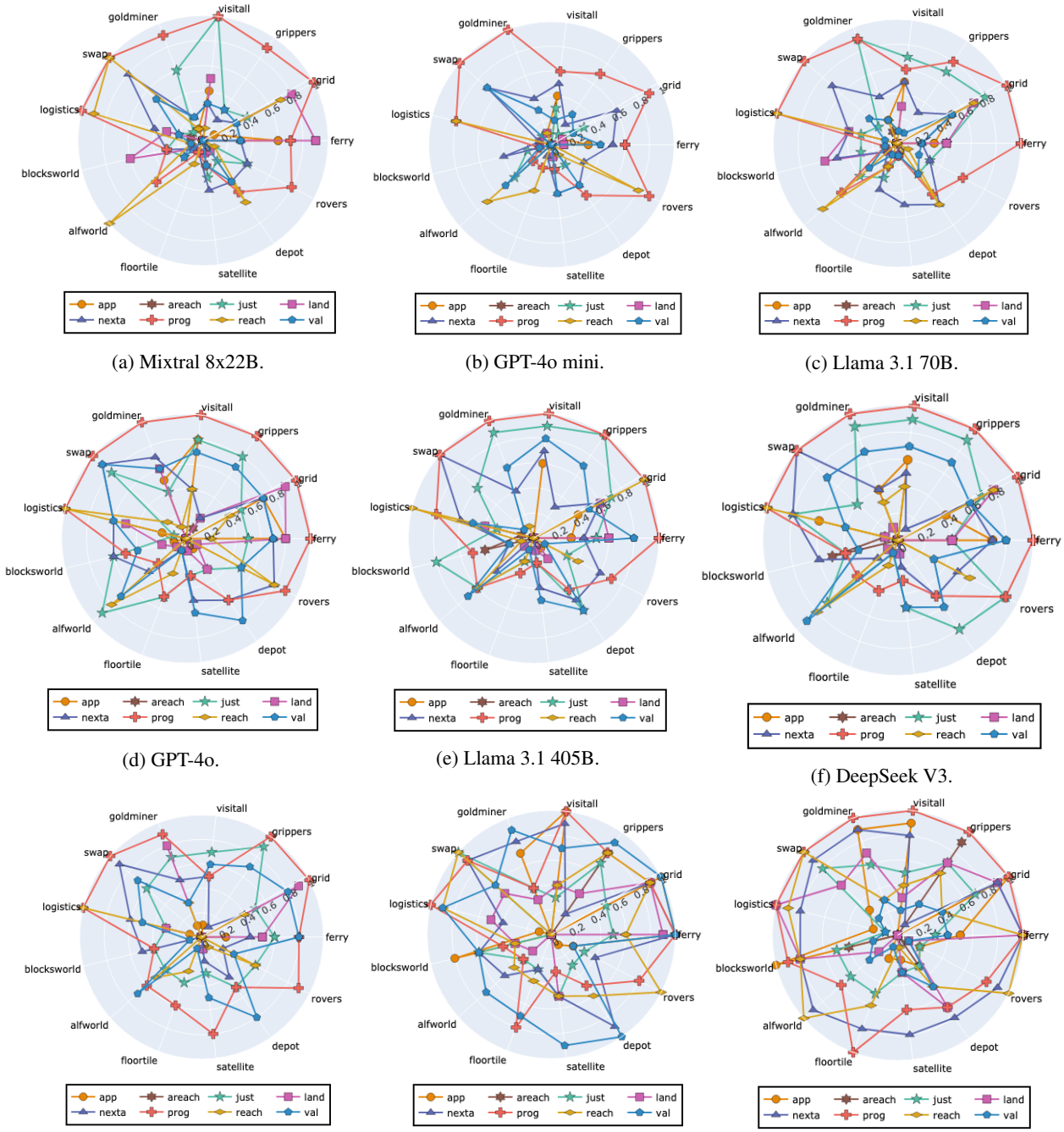


(a) DeepSeek coder 33B.



(b) Granite code 34B.

Figure 7: Domain-wise accuracy of small and medium size language models.



(a) Mixtral 8x22B. (b) GPT-4o mini. (c) Llama 3.1 70B.

(d) GPT-4o. (e) Llama 3.1 405B. (f) DeepSeek V3.

(g) DeepSeek R1. (h) o1 mini. (i) o1 preview.

Figure 8: Domain-wise accuracy of large language and reasoning models.