Towards More Likely Models for AI Planning

Turgay Caglar
Colorado State University
tcaglar@colostate.edu

Sirine Belhaj
Ecole Polytechnique de Tunisie
sirine.belhaj@ept.ucar.tn

Tathagata Chakraborti
IBM Research
tathagata.chakraborti1@ibm.com

Michael Katz
IBM Research
michael.katz1@ibm.com

Sarath Sreedharan
Colorado State University
Sarath.Sreedharan@colostate.edu

Abstract

This is the first work to look at the application of large language models (LLMs) for the purpose of model space edits in automated planning tasks. To set the stage for this sangam, we explore two different flavors of model space problems that have been studied in the AI planning literature and explore the effect of an LLM on those tasks. We empirically demonstrate how the performance of an LLM contrasts with combinatorial search (CS) – an approach that has been traditionally used to solve model space tasks in planning, both with the LLM in the role of a standalone model space reasoner as well as in the role of a statistical signal in concert with the CS approach as part of a two-stage process. Our experiments show promising results suggesting further forays of LLMs into the exciting world of model space reasoning for planning tasks in the future.

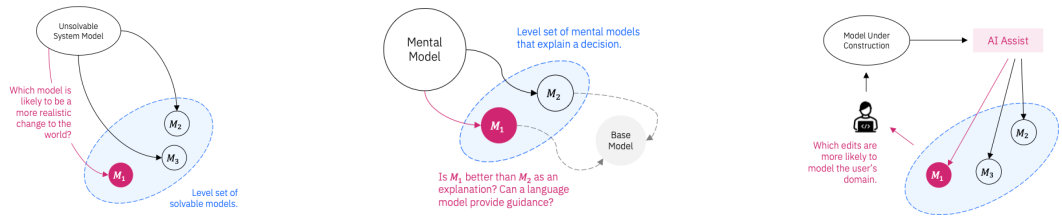
1 Introduction

AI planning or automated planning (used interchangeably) is the task of synthesizing the goal-directed behavior of autonomous agents. Traditionally, the AI planning community has looked at the classical planning problem as one of generating a plan given a model of the world [26]. Here, "model" or a "planning problem" refers to a collection of constraints describing the current state of the world (initial state), the actions available to the agent along with the conditions under which the agent can do those actions and the effect of doing those actions on the environment, and a target (goal) state for the agent to achieve. The plan is a sequence of actions that the agent can use to transform the current state to the desired goal state.

Typically, these models are represented using the planning domain definition language or PDDL [29, 44] – we will use the same in this paper. All the information to derive this solution (plan) is contained in the input model which remains static during the planning task. *But what if the model itself needs to be changed?*

This may be because it is incorrect, or incomplete, or even unsolvable. It may be because it needs to be changed to support some new behaviors. It may also be because the model is being used to describe a world that itself needs to change through the actions of an agent. In practice, the deployment of systems that can plan involves a whole gamut of challenges in authoring, maintaining, and meta-reasoning about models of planning tasks.

NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023)



(a) Unsolvability. (b) Explanations (c) Domain Authoring

Figure 1: A conceptual illustration of model space problems in AI planning. Instead of the classical planning task of computing a plan given a model, a model space task starts with a starting model $\mathcal{M}$ and a target criterion to satisfy, and the solution is a new model $\mathcal{M}_1$ where that criterion is satisfied. That criterion in Figure 1a is that the initially unsolvable model becomes solvable (or an initially invalid plan in $\mathcal{M}$ becomes valid in the new model $\mathcal{M}_1$). In Figure 1b, on the other hand, the starting model is the mental model of the user that needs to be updated and the target is a new model that can explain a given plan (or refute a given foil). In domain authoring situations, such model updates happen with the domain writer in the loop, and the starting model is the model under construction (Figure 1c). In all these cases, there are many non-unique model edits $\mathcal{M}_1 \Delta \mathcal{M}$ that can satisfy the required criterion. In this paper, we explore if LLMs can produce more likely edits in real-worldly domains.

1.1 Model Space Problems in AI Planning

We begin by enumerating different flavors of model space reasoning tasks explored in the AI planning literature. All of them involve a starting model which has something wrong with it and the solution is a new model where the problem has been resolved or the required criterion has been met.

Unsolvability

Perhaps the most difficult of model space problems, especially with humans in the loop, is that of unsolvability. This is because when a model is unsolvable, there is no artifact (such as an outputted plan) to look at for debugging purposes. While there have been a lot of efforts, including an ongoing competition [49], to *detect* unsolvability of planning tasks up-front to speed up calls to a planning module [5, 47], and attempts to compute or even learn heuristics [31, 64, 65] and produce certificates [21, 22, 20] for unsolvable tasks, to make this process as efficient as possible, these do not help to fix the issues with the model that make it unsolvable in the first place.

One of the seminal works in this category [27] framed the problem as "excuse generation" where the authors envisaged a reformulation of the input planning task where if only (i.e. an excuse) certain things about the current state were changed then it would become solvable. In addition to initial state changes, this idea was later extended [30] to cover other parts of the model and framed as a more general "planning task revision" problem.

While these works do not particularly consider a human in the loop, authors in [62, 60] have looked at the problem of explaining unsolvability of planning tasks to users explicitly as a model evolution problem, using techniques like domain abstractions (simplifications) to adjust to users with different levels of expertise. Later efforts [34] have borrowed from these concepts and tried to operationalize them for developers.

Executability

While unsolvable models produce no plans, incorrect or incomplete models produce wrong plans. Conversely, a desired plan may not be among the best (or even valid) plans in a given model. This class of model evolution problems [61, 62, 60] closely mimics the unsolvability problem but with an additional input – a plan – that must be made valid in the target model. Interestingly, since the given plan is not valid in the basis model, the basis model together with the plan (i.e. a compiled model where both are enforced) gets us back to the unsolvability situation above. We will use this approach when we deal with this class of problems later in this paper but, to be clear, we do treat it as a separate class of model space problems to study since the input involves a plan that a competent solver must be able to reason about.

Explanations

The above problems deal with one model in isolation. However, when working with humans in the loop, AI systems are often required to provide explanations of their behavior. Planning systems are no different [16, 24, 14]. The model evolution problem here involves reasoning explicitly with the model of the (system) explainer as the basis model and the mental model of the human (explainee) as the target model. This task can be formulated as one of "model reconciliation" [13] – an explanation is the model update that justifies a particular plan i.e. if both models justify a plan then there is no need for explanations. There is an overlap here with the previous tasks in terms of what kind of justifications a user is looking for: it might be a justification for a plan that the system produced and is invalid in the user model, and we end up in the unsolvability scenario again. In the worst case, the system may have to refute all possible alternatives (called "foils" [45]) and establish the optimality of a plan [13].

Interestingly, one can remove [12] the basis model in the model reconciliation formulation and produce false explanations or "lies". While this makes for a computationally harder open-ended search in the space of probable models, authors in [12] envisaged that algorithms which have looked at linguistic patterns for model evolution [54, 53] can assist in finding more probable models. This, of course, raises several ethical questions [11], especially now that LLMs can provide a stronger linguistic signal. We do not study this task here for two reasons: 1) Technically, this is not a separate class of a model reasoning problem since this ability is contained in the model reconciliation formulation; and 2) There seems to be little reason for building systems that can lie more effectively.

Domain Authoring and Design

While model evolution, in isolation, is useful for any autonomous system in a non-stationary domain, and explanations are a desired tool for any user-facing tool, a unique task in the context of planning systems we want to give a shout-out to is that of domain acquisition. Planning requires models and a significant portion of those models are acquired from domain experts. The knowledge acquisition literature in automated planning has studied this domain for decades [69] and the difficulty of acquiring domains remain a bottleneck in the adoption of planning technologies.

One subclass of domain authoring problems is *design* – here, the task is not to author a new domain but to evolve an existing one to optimize certain criteria like making the task of recognizing the goals of agents in the environment easier [35, 46, 71] or making the behavior of agents easier to interpret [36, 37]. Here as well, search techniques reveal multiple possible design options that can be enforced on a domain to achieve the desired effect. Issues of explanations, unsolvability, and executability manifest themselves in domain authoring and design tasks, with an additional component of interaction design with the domain author in the loop. Authors in [62] demonstrate this in a large-scale industrial domain on authoring models for goal-oriented conversational agents [50]. The role of an AI assist in authoring problems is especially critical in what we call "real worldly domains".

1.2 Real Worldly Domains and Likelihood of Models

All the model space problems we talked about so far are usually solved by some compilation to a combinatorial search process [27, 13, 61] which terminates after a set of model edits satisfy the desired properties in the modified model. It is usually the case that this yields many non-unique solutions – e.g. there may be many explanations for the same plan, many ways to change an unsolvable problem into a solvable one, or many ways to fix a model in order to support an invalid plan. From the perspective of a combinatorial search process, all these are logically equivalent and hence equally likely. In fact, in preliminary studies [72], it has already been demonstrated how users perceive logically equivalent explanations generated through a model reconciliation process, differently.

Large-scale statistical models such as LLMs, on the other hand, carry a lot of domain knowledge on things we do in our everyday lives i.e. our worldly matters. For want of a better term[1], we call these real worldly domains. Broadly speaking, these include all manner of human enterprise – and

[1]While looking for a term to describe the domains describing our worldly matters, we overlooked two in particular. In scientific literature, the term "real-world domains" is often used to establish something that is real but does come with an unnecessary connotation or snark of not being something of mere academic interest aka a "toy domain". Furthermore, a so-called "real world" domain includes Mars rovers and unmanned vehicles,

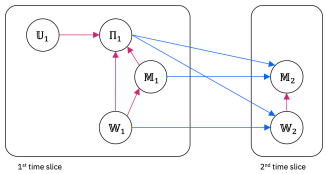


Figure 2: A DBN representing the random variables and their relations that are relevant to the problem at hand. The blue lines capture the diachronic, i.e., over time, relationships, and the maroon lines capture the synchronic ones.

consequently (planning) models describing them wherever relevant (sequential decision-making tasks) – that are described on the public internet (and not the domain describing the inner workings of a Mars rover per se). Existing works leveraging LLMs for planning have already shown promising results in the classical planning task in real worldly tasks in the home and kitchen [1, 33], and in specialized but common tasks such as service composition [38, 43]. Can LLMs do the same for model space reasoning for planning tasks? Can LLMs give statistical insight into what model edits are more likely when CS says they are equivalent? Can LLMs even bypass the CS process, as it can in certain circumstances for the classical planning task (Appendix Section B), *and do it all by itself??* These are the questions we ponder in this work.

Contributions

This is the first attempt at exploring the role of LLMs on model space search. To this end, we analyze the role of an LLM for generating more likely model edits either in relation to CS as a direct replacement for the model space reasoning task or in its role in an augmented approach with CS. The answers to these questions have major implications beyond just an academic interest in finding out the impact of LLMs on model space tasks in planning. Unlike carefully crafted planning domains used as benchmarks, such as the ones used in the International Planning Competition (IPC) [48], the deployment of planning models in real worldly domains has touchpoints with all the problems described above – explainability of outputs and failure modes, investigation of unsolvability and executability in potentially faulty models, model authoring and maintenance over time, etc. – often with the domain author in the loop [63, 62]. These models are often not written by hand but generated on the fly at runtime from input data, either through code or using knowledge compilers like [25]. An insight into the likelihood of models can empower the domain author to create and debug models with greater ease [62, 34], as well as allow automated model adaptation in fully autonomous systems in nonstationary environments [7] or in constrained creative tasks like story-telling [59, 53, 55] that have previously relied on using limited linguistic cues like antonyms and synonyms [54] for domain evolution.

2 Formal Interpretation of Model Likelihood

In this section, we aim to provide a uniform probabilistic interpretation for the types of queries we employ in this problem. Figure 2 presents a simplified dynamic Bayes network that encapsulates the scenario. This could be utilized to better comprehend and formalize the nature of the probabilities we intend to capture. Starting with the random variables, $\mathbb{M}_{1/2}$ and $\mathbb{W}_{1/2}$, these correspond to the model descriptions and the information about the true task/world at a given time step. The random variable Π_i captures the policy that determines what action will be applied at a given step, which can alter the world and the model description. $\mathbb{U}_1$ determines the use case (this roughly maps to the type of model space search problem being solved). The action combined with the use case, allows us to capture both scenarios where the focus is on updating the model description to better reflect the task (for example, domain authoring settings where the author may have misspecified something), and cases where the change also involves updating the underlying task and reflecting that change into the model

which are by no means part of our worldly matters. On the other hand, "common sense" tasks are widely used to characterize things that come naturally to humans but our worldly matters can involve much more complexity than common sense tasks – e.g. a service composition task – and we do hope to find the knowledge of those activities in the statistical signal from large-scale language models. We avoid both terms for these reasons but better suggestions are welcome.

description (for example, cases where the true task is unsolvable). Please note that for explanation tasks, we expect $\mathbb{M}_{1/2}$ to capture both the human knowledge about the task and the agent's model.

In the first time slice, we see that the actions that perform the update depend on the current model description, the task/world, and the use case. Naturally, this is a simplification of the true setting, but for the purpose of understanding the problem, this model serves as a useful abstraction. The most crucial term we are interested in measuring in this paper is the probability of an updated model description, given the prior model description and the use case:

$$P(\mathbb{M}_2 = \mathcal{M}_2 \mid \mathbb{M}_1 = \mathcal{M}_1, \mathbb{U}_1 = \mathcal{U}). \tag{1}$$

We will examine cases where the information about $\mathbb{M}_1$ and $\mathbb{U}_1$ are included as part of the prompt, and we expect the LLM to approximate the above probability expression.

Note that this presupposes multiple capabilities of the LLM. For one, it assumes that the LLM can capture prior probabilities of possible world states. Next, it assumes that it can capture the likelihood of a specific action being performed for a given use case, state, and model description. Finally, it assumes that the LLM can discern how this action affects the next state and the model description. Furthermore, even if the LLM is capable of capturing this information separately, it may not correctly estimate the above probability expression. We hope to find a model such that:

$$\mathcal{M} = \arg\max_{\mathcal{M}' \in \mathbb{M}} P(\mathbb{M}_2 = \mathcal{M}' \mid \mathbb{M}_1 = \mathcal{M}_1, \mathbb{U}_1 = \mathcal{U}), \tag{2}$$

where $\mathbb{M}$ is the set of all possible model descriptions.

3 LLMs ft. Model Space Exploration

In each of the model space search cases discussed before, we would ideally like to identify some model that satisfies Equation 2. However, to understand the current efforts in the model-space search, it might be useful to further decompose the metric into two components:

- **Objective Metric** This is the traditional metric that is being optimized by the various CS methods studied previously. In the cases we are focusing on, this is mostly a binary metric such as the solvability of a problem or the executability of the given plan. We will say a solution/model is *sound* if it satisfies the objective metric.
- **Likelihood of the Updated Model** This is the specific aspect that is currently being overlooked by existing methods. This metric corresponds to the likelihood that the updated model generated through search corresponds to a desired target model. Equation 1 provides a formalization of this probability. The likelihood of different sound models would vary based on the use case and the context.

Our goal now is to find an updated model that meets the objective metric while maximizing its likelihood. As discussed, we will use pre-trained LLMs as the source for the information about the latter measure. One can envision four different configurations to achieve this goal:

LLM-only Configuration

In this mode, we provide the entire problem to LLM. The prompt is included with enough context that the system is aware of the criteria against which the likelihood of the models need to be measured. The LLM is asked to produce an updated model that is the most likely sound model. This corresponds to asking LLM to directly approximate Equation 2.

LLM as a Post Processor

In this mode, we use CS to generate a set of potential candidate solutions that are guaranteed to be sound. The LLM is then asked to select the model that is most likely. The prompt would again be designed to include the context necessary to determine what constitutes a target model. In this case, we are effectively trying to approximate the following problem:

$$\mathcal{M} = \arg\max_{\mathcal{M}' \in \hat{\mathbb{M}}} P(\mathbb{M}_2 = \mathcal{M}' \mid \mathbb{M}_1 = \mathcal{M}_1, \mathbb{U}_1 = \mathcal{U}), \tag{3}$$

where $\hat{\mathbb{M}} \subseteq \mathbb{M}$, such that every model in $\hat{\mathbb{M}}$ meets the formal requirements to satisfy the use case $\mathcal{U}$.

Since enumerating all solutions is too expensive, we used an exhaustive search that caches solutions until a search budget of 5,000 (10,000) node expansions for unsolvability (inexecutability) and a 2-hour limit was met per problem instance. This makes the solution incomplete.

LLM as a Pre-Processor

In this mode, we ask the LLM to provide a ranked order of likely model edits without considering the objective metric. The ordering can then be used by CS to compute the most likely model that would satisfy or maximize the objective metric. This approach is still guaranteed to be sound, as the CS would only return a solution if the selected model updates result in a model that meets the objective metric. In this case, we are trying to approximate the following problem:

$$\mathcal{M} = \underset{\mathcal{M}' \in \hat{\mathbb{M}},\ \mathcal{M}' \text{ is sound}}{\arg\max} V(\mathcal{M}'), \tag{4}$$

where the utility/value function $V(\mathcal{M}')$ is calculated from the LLMs approximation of the model likelihood. Specifically, we will have $V(\mathcal{M}') \propto P(\mathbb{M}_2 = \mathcal{M}' \mid \mathbb{M}_1 = \mathcal{M}_1, \mathbb{U}_1 = \mathcal{U})$ if you are trying to order based on both objective metric and the likelihood of a model description, else you will have $V(\mathcal{M}') \propto P(\mathbb{M}_2 = \mathcal{M}' \mid \mathbb{M}_1 = \mathcal{M}_1)$.

For the purposes of our implementation, we converted all the ordered edits proposed by the LLM into a set of actions that the CS can perform with different costs. In particular, we chose the cost of actions in such a way that, for an ordered sequence of l edits, the total cost of including the first i edits is always less than the cost of including the $i+1$th edit. Since the LLM cannot rank all possible edits (capped at 20 for the experiments), there is a possibility that the CS search will not be able to find a valid solution, which makes this approach incomplete in practice as well.

LLM for Search Guidance

This mode is particularly relevant if heuristic search is used. The search algorithm could leverage LLMs to obtain search guidance in the form of heuristic value. As with the previous mode, we can use LLM for getting information about both metrics and we can still guarantee correctness. The formal problem being approximated here again corresponds to the one listed in Equation 4 and the value function considered will also have similar considerations. This process requires calls to an LLM within the process of search and is known to be [23] computationally excessively prohibitive. Hence, we do not consider this configuration in our study.

In this paper, we focus primarily on evaluating two basic model space search problems, namely, addressing *unsolvability* and *plan executability*. The nature of the likelihood of the model could depend on the underlying use case in question. One can broadly identify two classes of problems, namely *model misspecification* and *updating the environment*. In the former case, the current model is misspecified and the model search is being employed to identify the true unknown underlying model. In the latter case, the current model is an exact representation of the true environment, however the model and by extension the environment doesn't meet some desired properties. The goal here becomes to then identify the set of changes that can be made to the environment such that it meets the desired property. One could equivalently think of this being a case where there are actions missing from the model that correspond to these possible changes. While both of these use cases have been considered in the literature, for simplicity the evaluation in the paper will primarily focus on the latter one. All prompts considered in the paper were written with the latter use case in mind.

4 Empirical Results

For evaluating the three approaches, we designed four novel domains so that a certain set of changes would be clearly recognized as more reasonable, i.e. more likely to be realized in the real world. We additionally assume that all changes that belong to this set (henceforth referred to as *"reasonable changes"*), will result in models with the same likelihood.

Travel Domain

Here an agent travels from a given city to another, using either a taxi or bus to travel between cities. We additionally encode which cities neighbor each other, and the initial problem only includes bus or taxi services between neighboring cities. Reasonable changes are limited to starting taxi or bus services between neighboring cities only.

Roomba

In this domain, the agent needs to clean a specified room, which requires it to travel to the target room while traversing the intermediate rooms through connecting doors. The overall floor plan is laid out using neighboring predicates and the expectation is there can only be doors between neighboring rooms. Changes are reasonable if it only involves whether a door is locked or unlocked (as opposed to connecting new rooms requiring removal of walls per se).

Barman-simple

This is a modified version of the IPC barman domain [9]. Here, the agent is expected to prepare a set of drinks, given a set of containers and ingredients. While only considering a subset of actions from the original domain, we introduce a new predicate that indicates whether a container is clean, which is a precondition for using the container for a drink. We consider solutions to be reasonable if they only involve marking containers as clean (as opposed to adding prepared drinks).

Logistics-simple

Finally, we consider a simplified version of the logistics problem where a package is transported from one collection station to a target station. Each station contains a truck that can move the package to a neighboring station. We add a new precondition that ensures that only trucks that are marked as being ready for transportation can be used to move packages. We limit reasonable changes to ones that mark trucks as being ready for transportation.

4.1 Experimental Setup

In each domain, we create a set of solvable problems of varying sizes. We then made it unsolvable by deleting a set of initial state predicates that correspond to reasonable changes. The number of such modifications ranges from 1 to 4. This means, by design, there exists a set of reasonable changes that can make the problem solvable. For the plan executability case, we chose one of the plans generated from the original solvable plan as the target plan to be made solvable. All model updates were limited to initial state changes only.

Phrasing of the prompts

Our objective is to determine whether a model space solution is reasonable in the sense of the likelihood of being realized in the real world. We captured this in the prompts by asking the LLM to generate or select the most *reasonable* set of model edits. We also tested with a more verbose prompt that explicitly mentions the ease of realizing the changes, more on this in Section 4.3.

Hypotheses

We focus on the following hypotheses, for both the unsolvability and executability settings:

Unsolvability	LLM-Only				LLM as Post Processor				LLM as Pre Processor			
Domains	GPT-3.5-turbo		GPT-4		GPT-3.5-turbo		GPT-4		GPT-3.5-turbo		GPT-4	
	Sound	Preferred	Sound	Preferred	Solutions	Preferred	Solutions	Preferred	Ratio	Preferred	Ratio	Preferred
Travel	97/245	7/97	164/245	66/164	245/245	24/245	245/245	63/245	129/245	1/129	160/245	27/160
Roomba	0/20	0/0	36/100	7/36	20/20	2/20	71/100	9/71	0/20	0/0	18/100	4/18
Logistics	61/69	0/61	65/69	1/65	69/69	10/69	69/69	0/69	56/69	0/56	65/69	4/65
Barman-S	43/61	2/43	57/61	34/57	34/61	3/34	34/61	4/34	28/61	28/28	17/61	16/17
Logistics-S	89/89	0/75	77/89	28/77	45/89	3/45	45/89	5/45	24/89	0/24	10/89	5/10
Overall	**276/484**	**9/276**	**399/564**	**136/399**	**194/484**	**39/194**	**198/564**	**78/198**	**237/484**	**29/237**	**270/564**	**56/270**

Table 1: Results from the LLM-only, LLM as post-processor, and LLM as pre-processor settings for each unsolvability domain.

Executability	LLM-Only				LLM as Post Processor				LLM as Pre Processor			
Domains	GPT-3.5-turbo		GPT-4		GPT-3.5-turbo		GPT-4		GPT-3.5-turbo		GPT-4	
	Sound	Preferred	Sound	Preferred	Solutions	Preferred	Solutions	Preferred	Ratio	Preferred	Ratio	Preferred
Travel	80/245	33/80	225/245	130/225	89/245	38/89	89/245	57/89	31/245	31/31	207/245	207/207
Roomba	0/20	0/0	57/99	31/57	12/20	12/12	16/99	12/16	0/20	0/0	67/99	11/67
Logistics	16/69	0/16	66/69	11/66	51/69	5/51	51/69	22/51	13/69	2/13	13/69	20/57
Barman-S	57/61	14/57	56/61	15/56	34/61	8/34	34/61	13/34	29/61	29/29	29/61	26/26
Logistics-S	21/89	6/21	89/89	77/89	68/89	23/68	68/89	60/68	0/89	0/0	0/89	14/18
Overall	**174/484**	**53/174**	**493/563**	**264/493**	**170/484**	**32/170**	**170/563**	**110/170**	**73/484**	**62/73**	**375/563**	**278/375**

Table 2: Results from the LLM-only, LLM as post-processor, and LLM as pre-processor settings for each executability domain.

(a) Unsolvability: Soundness vs. Edit size

(b) Unsolvability: Soundness vs. Plan size

(c) Executability: Soundness vs. Edit size

(d) Executability: Soundness vs. Plan size

Figure 3: Soundness of solutions from the LLM-only (GPT-4) approach against edit and plan sizes for unsolvability and executability settings in 564 problems across all 5 domains. Each bar represents one problem instance: a bar height of 1 indicates a sound solution, -1 otherwise. A higher concentration of negative bars will indicate deterioration in performance.

- **H1** LLM can identify sound model updates.
- **H2** LLM can identify reasonable model updates.
- **H3** The ability to find sounds model updates improves with the capability of the LLM.
- **H4** The ability to find reasonable model updates improves with the capability of the LLM.
- **H5** The ability to produce sound, and hence reasonable solutions as a fraction of it, will be significantly outperformed by the two CS+LLM approaches.
- **H6** LLMs will provide a stronger signal, i.e. a higher fraction of sound and reasonable solutions, in public domains an LLM is likely to have seen already.
- **H7** The performance of an LLM will deteriorate with the complexity of the model space reasoning task.

Measurements

We use the OpenAI API [51] for our LLM-only approach. H1 and H2 are measured directly against the ground truth, as per the problem-generation process explained at the start of Section 4.1. For H3 and H4, we compare H1 and H2 from GPT-3.5-turbo to GPT-4. For H5, we measure H1 and H2 relative to the two CS integrations with the LLM as a pre-processor and LLM as a post-processor. For H6, we compare H1-H4 in two ways: 1) the performance in two public domains Barman and Logistics, as compared to the two novel domains Travel and Roomba; and 2) the relative performance between Logistics and Logistics-simple, the latter being a modified version of the former. Finally, for H7, we measure how H1 and H2 fares with two measures of complexity: 1) the number of model edits required to arrive at a solution; and 2) the length of the plan underlying a model space reasoning task. For unsolvability, this is known when a planning task is made unsolvable as per the problem generation process, while for executability, the plan is part of the input to the reasoning task.

4.2 Results

Tables 1 and 2 presents the outcomes for unsolvability and inexecutability setting respectively. Since both display identical trends for H1-H7, we describe them together. The only difference between the two settings is that the post-processing approach had a larger budget for expanded nodes as mentioned in Section 3, since it rarely hit the time budget. However, this did not make much difference.

In support of H1 and H2, the LLM-only approach demonstrates surprising proficiency in suggesting sound and reasonable solutions across various domains. In support of H3-H4, the LLM-only approach

Executibility	GPT-3.5-turbo		GPT-4	
Domains	Sound	Preferred	Sound	Preferred
Barman	13/33	0/13	33/33	27/33
Logistics	17/25	0/17	24/25	0/24
Overall	**30/88**	**0/30**	**57/58**	**27/57**

Table 3: The number of sound and reasonable model updates generated as a response to the more verbose query.

sees the most pronounced improvement in identifying sound model alterations, accompanied by a higher rate of reasonable solutions as well, as we upgrade to the latest LLM. The relative gain between sound and reasonable solutions is slightly counter to expectations though, since an LLM is supposed to be a stronger statistical signal on more likely updates rather than a reasoner by itself.

This surprise carries onto the comparative results with CS+LLM approaches. Contrary to H5, the LLM-only setting outperforms both CS+LLM approaches. Note that the CS+LLM approaches are guaranteed to be sound, so the deficit in the "solutions" column is between a sound solution versus no solution at all (and not sound versus unsound solutions). The only way we do not get a (sound) solution from the LLM as a Post-Processor approach is if the CS stage does not terminate within the time or memory budget (as mentioned in Section 3). Similarly, the two ways we do not get a solution for the LLM as a Pre-Processor approach is if the preferred set of reasonable edits from the LLM are not sufficient for the CS to construct a solution, or as in the previous case, the search does not terminate. While the CS+LLM approaches hit the computational curse, the LLM approach hits the curse of limited context size. Between GPT-3.5 and GPT-4, the prompt size has grown from 4,096 to 8,192 tokens, but instances surpassing the token limit could not be processed. This makes a significant dent in the numbers for the Roomba domain, especially for GPT-3.

The rate of sound solutions is much higher for public domains compared to the custom ones, which is consistent with H6. However, this trend does not carry over to whether the solutions are reasonable or not. In fact, the derived logistics domain shows much higher rate of reasonable solutions than the public logistics domain that shadows it. So results for H6 are inconclusive, and further underline the fickle nature of interfacing with LLMs. Relatedly, the trends with respect to the complexity of the tasks, also defy expectations. The rate of mistakes in constructing a sound solution is spread uniformly across the spectrum of task complexity (Figure 3).

4.3 Prompt Variations

As a way to test the effect the phrasing of our prompt had on the results, we also tried a variant of the prompt that was more explicit in what it expected to optimize for. Specifically, for generating a solvable problem variant we asked the system to: *'Select the set of changes that would be the easiest to realize in the real world'*. Table 3 shows the results of running this prompt for the LLM-only setting. The results can be compared directly to those presented in LLM-only columns of Table 1. The results are pretty similar, with the more verbose query being slightly worse off, so we do not explore this direction in much more detail.

5 Conclusion and Key Takeaways

This is the first paper to consider the use of LLMs for model space reasoning tasks for automated planning. While the problem of model space search has been studied in various contexts, the question of how to evaluate the quality of different sound model updates have mostly been left unanswered. Domain knowledge contained within LLM provides us with a powerful option to evaluate the likelihood of different model updates. In this paper, we exploited that power in 3 ways: first as a standalone end-to-end approach and the others in conjunction with a sound solver. The results reveal some intriguing trade-offs for the practitioner:

- CS approaches are limited by the complexity of search. Thus even while being theoretically sound and complete, they produce fewer solutions and hence fewer sound solutions in absolute numbers. This means that augmenting the LLM-only approach with a validator [32] will produce as a whole a more effective sound and reasonable solution generator!

- LLM approaches are limited by the size of the prompt and thus does not scale to large domains even for computationally simpler problem instances.
- The unpredictable nature of LLMs (e.g. H6 and H7) makes interfacing to LLMs unreliable.

Despite these trade-offs, the promise of an LLM across H1-H5 is undeniable. We are excited to explore further how this strong statistical signal influences domain authoring tasks, as mentioned in Section 1.2, and reduces authoring overhead for planning tasks in the future.

Acknowledgments and Disclosure of Funding

Sarath Sreedharan's research is supported in part by grant NSF 2303019.

References

[1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *Proceedings of Machine Learning Research*, 2023.

[2] M. Asai. Neural-Symbolic Descriptive Action Model from Images: The Search for STRIPS. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, 2019.

[3] M. Asai and A. Fukunaga. Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary. In *AAAI*, 2018.

[4] M. Asai and C. Muise. Learning neural-symbolic descriptive planning models via cube-space priors: The voyage home (to strips). In *IJCAI*, 2020.

[5] C. Bäckström, P. Jonsson, and S. Ståhlberg. Fast Detection of Unsolvable Planning Instances Using Local Consistency. In *SoCS*, 2013.

[6] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In *FAccT*, 2021.

[7] D. Bryce, J. Benton, and M. W. Boldt. Maintaining Evolving Domain Models. In *IJCAI*, 2016.

[8] E. Callanan, R. De Venezia, V. Armstrong, A. Paredes, T. Chakraborti, and C. Muise. MACQ: A Holistic View of Model Acquisition Techniques. *arXiv:2206.06530*, 2022.

[9] S. J. Celorrio. Domainssequential. `http://www.plg.inf.uc3m.es/ipc2011-deterministic/DomainsSequential.html#Barman`, 2011.

[10] T. Chakraborti. *Foundations of Human-Aware Planning – A Tale of Three Models*. PhD thesis, Arizona State University, 2018.

[11] T. Chakraborti and S. Kambhampat. (When) Can Bots Lie? In *AIES*, 2019.

[12] T. Chakraborti and S. Kambhampat. (How) Can AI Bots Lie? In *ICAPS Workshop on Explainable AI Planning*, 2019.

[13] T. Chakraborti, S. Sreedharan, Y. Zhang, and S. Kambhampati. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *IJCAI*, 2017.

[14] T. Chakraborti, A. Kulkarni, S. Sreedharan, D. E. Smith, and S. Kambhampati. Explicability? Legibility? Predictability? Transparency? Privacy? Security? The Emerging Landscape of Interpretable Agent Behavior. In *ICAPS*, 2019.

[15] T. Chakraborti, S. Sreedharan, and S. Kambhampati. Balancing Explicability and Explanations in Human-Aware Planning. In *IJCAI*, 2019.

[16] T. Chakraborti, S. Sreedharan, and S. Kambhampati. The Emerging Landscape of Explainable AI Planning and Decision Making. In *IJCAI*, 2020.

[17] H.-C. Chen and J.-D. Wei. Using Neural Networks for Evaluation in Heuristic Search Algorithm. In *AAAI*, 2011.

[18] L. Chrestien, T. Pevny, A. Komenda, and S. Edelkamp. Heuristic Search Planning with Deep Neural Networks Using Imitation, Attention and Curriculum Learning. *arXiv:2112.01918*, 2021.

[19] B. Cserna, W. Ruml, and J. Frank. Planning Time to Think: Metareasoning for Online Planning with Durative Actions. In *ICAPS*, 2017.

[20] S. Eriksson and M. Helmert. Certified Unsolvability for SAT Planning with Property Directed Reachability. In *ICAPS*, 2020.

[21] S. Eriksson, G. Röger, and M. Helmert. Unsolvability Certificates for Classical Planning. In *ICAPS*, 2017.

[22] S. Eriksson, G. Röger, and M. Helmert. A Proof System for Unsolvable Planning Tasks. In *ICAPS*, 2018.

[23] P. Ferber, M. Helmert, and J. Hoffmann. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In *ECAI 2020*, 2020.

[24] M. Fox, D. Long, and D. Magazzeni. Explainable Planning. *arXiv:1709.10256*, 2017.

[25] G. Francés, M. Ramirez, and Collaborators. Tarski: An AI Planning Modeling Framework. `https://github.com/aig-upf/tarski`, 2018.

[26] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[27] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel. Coming Up With Good Excuses: What to do When no Plan Can be Found. In *ICAPS*, 2010.

[28] E. Groshev, M. Goldstein, A. Tamar, S. Srivastava, and P. Abbeel. Learning Generalized Reactive Policies Using Deep Neural Networks. In *ICAPS*, 2018.

[29] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise. An Introduction to the Planning Domain Definition Language. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2019.

[30] A. Herzig, M. V. de Menezes, L. N. De Barros, and R. Wassermann. On the Revision of Planning Tasks. In *ECAI*, 2014.

[31] J. Hoffmann, P. Kissmann, and A. Torralba. "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability. In *ECAI*, 2014.

[32] R. Howey, D. Long, and M. Fox. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *IEEE International Conference on Tools with Artificial Intelligence*, 2004.

[33] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner Monologue: Embodied Reasoning through Planning with Language Models. *CoRL*, 2023.

[34] L. G. Käser, C. Büchner, A. B. Corrêa, F. Pommerening, and G. Röger. Machetli: Simplifying Input Files for Debugging. In *ICAPS System Demonstrations Track*, 2022.

[35] S. Keren, A. Gal, and E. Karpas. Goal Recognition Design. In *ICAPS*, 2014.

[36] A. Kulkarni, S. Sreedharan, S. Keren, T. Chakraborti, D. E. Smith, and S. Kambhampati. Design for Interpretability. In *ICAPS Workshop on Explainable AI Planning*, 2019.

[37] A. Kulkarni, S. Sreedharan, S. Keren, T. Chakraborti, D. E. Smith, and S. Kambhampati. Designing Environments Conducive to Interpretable Robot Behavior. In *IROS*, 2020.

[38] LangChain. LangChain is a framework for developing applications powered by language models. `https://python.langchain.com`, 2023.

[39] T. Li, R. Chen, B. Mavrin, N. R. Sturtevant, D. Nadav, and A. Felner. Optimal Search with Neural Networks: Challenges and Approaches. In *SoCS*, 2022.

[40] C. H. Lin, A. Kolobov, E. Kamar, and E. Horvitz. Metareasoning for Planning under Uncertainty. *IJCAI*, 2015.

[41] A. Lindsay, J. Read, J. Ferreira, T. Hayton, J. Porteous, and P. Gregory. Framer: Planning Models from Natural Language Action Descriptions. In *ICAPS*, 2017.

[42] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. *arXiv:2304.11477*, 2023.

[43] J. Maeda and E. Chaki. Semantic kernel. `https://learn.microsoft.com/en-us/semantic-kernel`, 2023.

[44] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – The Planning Domain Definition Language. Technical Report CVC TR98003/DCS TR1165, New Haven, CT: Yale Center for Computational Vision and Control, 1998.

[45] T. Miller. Explanation in Artificial Intelligence: Insights from the Social Sciences. *Artificial intelligence*, 2019.

[46] R. Mirsky, K. Gal, R. Stern, and M. Kalech. Goal and Plan Recognition Design for Plan Libraries. *ACM TIST*, 2019.

[47] L. H. Moreira and C. G. Ralha. Improving Multi-agent Planning with Unsolvability and Independent Plan Detection. In *Brazilian Conference on Intelligent Systems (BRACIS)*, 2017.

[48] C. Muise. Api.planning.domains: An interface to the repository of pddl domains and problems. `http://api.planning.domains`, 2023.

[49] C. Muise and N. Lipovetzky. International plannning competition (ipc) unsolvability track. `https://unsolve-ipc.eng.unimelb.edu.au`, 2023.

[50] C. Muise, T. Chakraborti, S. Agarwal, O. Bajgar, A. Chaudhary, L. A. Lastras-Montano, J. Ondrej, M. Vodolan, and C. Wiecha. Planning for Goal-Oriented Dialogue Systems. Technical report, IBM Research AI, 2020.

[51] OpenAI. GPT-4 Technical Report. *arXiv:2303.08774*, 2023.

[52] V. Pallagani, B. Muppasani, K. Murugesan, F. Rossi, L. Horesh, B. Srivastava, F. Fabiano, and A. Loreggia. Plansformer: Generating Symbolic Plans using Transformers. *arXiv:2212.08681*, 2022.

[53] J. Porteous. Planning Technologies for Interactive Storytelling. *Handbook of Digital Games and Entertainment Technologies*, 2016.

[54] J. Porteous, A. Lindsay, J. Read, M. Truran, and M. Cavazza. Automated Extension of Narrative Planning Domains with Antonymic Operators. In *AAMAS*, 2015.

[55] J. Porteous, J. F. Ferreira, A. Lindsay, and M. Cavazza. Automated Narrative Planning Model Extension. In *AAMAS*, 2021.

[56] N. Shah and S. Srivastava. Using Deep Learning to Bootstrap Abstractions for Robot Planning. In *ICLR Workshop on Generalizable Policy Learning in Physical World*, 2022.

[57] W. Shen, F. Trevizan, and S. Thiébaux. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *ICAPS*, 2020.

[58] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling. PDDL Planning with Pretrained Large Language Models. In *NeurIPS Workshop on Foundation Models for Decision Making*, 2022.

[59] N. Simon and C. Muise. TattleTale: Storytelling with Planning and Large Language Models. In *ICAPS Workshop on Scheduling and Planning Applications*, 2022.

[60] S. Sreedharan, S. Srivastava, D. Smith, and S. Kambhampati. Why Can't You Do That HAL? Explaining Unsolvability of Planning Tasks. In *IJCAI*, 2019.

[61] S. Sreedharan, T. Chakraborti, C. Muise, and S. Kambhampati. Expectation-Aware Planning: A General Framework for Synthesizing and Executing Self-Explaining Plans for Human-AI Interaction. In *AAAI*, 2020.

[62] S. Sreedharan, T. Chakraborti, C. Muise, Y. Khazaeni, and S. Kambhampati. D3WA+ – A Case Study of XAIP in a Model Acquisition Task for Dialogue Planning. In *ICAPS*, 2020.

[63] S. Sreedharan, T. Chakraborti, Y. Rizk, and Y. Khazaeni. Explainable Composition of Aggregated Assistants. In *ICAPS Workshop on Explainable AI Planning*, 2020.

[64] S. Ståhlberg. Tailoring Pattern Databases for Unsolvable Planning Instances. In *ICAPS*, 2017.

[65] S. Ståhlberg, G. Francès, and J. Seipp. Learning Generalized Unsolvability Heuristics for Classical Planning. In *IJCAI*, 2021.

[66] Y. Sung, L. P. Kaelbling, and T. Lozano-Pérez. Learning When to Quit: Meta-reasoning for Motion Planning. In *IROS*, 2021.

[67] J. Thayer, A. Dionne, and W. Ruml. Learning Inadmissible Heuristics During Search. In *ICAPS*, 2011.

[68] X. Tian, H. H. Zhuo, and S. Kambhampati. Discovering Underlying Plans Based on Distributed Representations of Actions. *AAMAS*, 2016.

[69] M. Vallati and D. Kitchin. *Knowledge Engineering Tools and Techniques for AI Planning*. Springer, 2020.

[70] K. Valmeekam, S. Sreedharan, M. Marquez, A. Olmo, and S. Kambhampati. On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark). In *NeurIPS Workshop on Human in the Loop Learning*, 2022.

[71] C. Wayllace, P. Hou, W. Yeoh, and T. C. Son. Goal recognition design with stochastic agent action outcomes. In *IJCAI*, 2016.

[72] Z. Zahedi, A. Olmo, T. Chakraborti, S. Sreedharan, and S. Kambhampati. Towards Understanding User Preferences for Explanation Types in Explanation as Model Reconciliation. In *HRI Late Breaking Report*, 2019.

[73] H. H. Zhuo, Y. Zha, S. Kambhampati, and X. Tian. Discovering Underlying Plans Based on Shallow Models. *ACM TIST*, 2020.

A Limitations

The proposed method has several limitations that need to be acknowledged. Firstly, its effectiveness is inherently limited by the capabilities of the LLMs it uses. As of the writing of the paper, LLMs have a number of known limitations that could prevent them from identifying the most likely models. Some of the issues include hallucination, lack of knowledge about specialized domains, the fact that it is an unsound reasoner, and so on (cf. [70, 6]). Secondly, it is currently hard to make the prediction generation more specific to a task or a user. This arises from various challenges, including practical limitations on fine-tuning the model to these specific settings or the inability to include all the relevant information in the prompt due to limitations in prompt size and context windows.

In Section 1.1, we noted the various flavors of model space problems in AI planning. We also noted how some of them overlap – e.g. unsolvability, executability, and explanations in domain authoring tasks – and how some of them are contained in others as a strict subset – e.g. explanations and lies. In evaluating the proposed method, we only focused on two prominent use cases, namely unsolvability, and executability. Additionally, we only considered a specific type of model update namely adding new predicates into the initial state. While theoretically, this model update can subsume any other model change, the ability of LLMs to identify likely model updates could differ based on the type of model updates considered.

Furthermore, the current study is limited to a set of domains where the reasonable or most likely changes were determined by the authors. We limited testing to a few LLMs and only considered two of the four possible configurations. It is also worth noting that effective solutions for model space search may involve additional challenges that are not being evaluated here. For example, domain authoring tasks also involve a human in the loop, which introduces additional dimensions of study beyond just figuring out which model edits are more likely – such as in figuring out how to communicate those edits effectively to the domain author. Such considerations are out of the scope of this paper. Similarly, the bastardized explainability problem that is able to generate lies, or conversely, a likely models approach that can actually catch those lies also have additional dimensions of interest, such as in mental modeling and computational ethics, which is also out of the scope of this work. We hope that this initial foray into this topic opens up future works in these directions.

In the future, we hope to address many of the limitations of the current evaluation. This includes expanding the number of use cases studied, considering various model updates, and comparing between all the possible configurations. We will also look at the possibility of testing these methods in tasks, where we can correctly quantify the likelihood of these models. For unsolvability, this might involve focusing on scenarios where the cost of various actions possible in that setting is can be at least quantified accurately. For use cases such as domain authoring, this might correspond to cases where the ground truth is known and as such one can correctly determine what the missing or incorrect model components could be. We also hope to run user studies to evaluate the model updates generated by the method.

B Model Space Problems versus Other Meta-Reasoning Tasks in AI Planning

We include here some additional pointers to relevant works that either explore the evolving role of language models in planning or address other meta-reasoning tasks for planning.

Meta-Reasoning for Planning Tasks

Reasoning about a planning model rather than using that model as immutable input to plan with, can be viewed as a form of meta-reasoning. Indeed, there is a long history of work on meta-reasoning for planning tasks. However, these primarily involve a trade-off of the time taken to arrive at a solution versus the quality of the solution. Typically, in this setting, a planner can choose to stop looking for better solutions, and potentially settle for a suboptimal solution, if it believes that there is (computationally) no point in carrying on. Such approaches have been used for policy optimization in Markov Decision Processes [40], motion planning [66], planning in temporal domains [19], heuristic search [67], and so on. However, this thread of work does not aim to change the model itself to better suit a given criterion, and that is our aim.

Model Space to State Space Compilations in Human-Aware Planning

One meta-reasoning task that looks to change the model is "human-aware planning" – this is explicitly formulated as a planning task of finding a plan [10], and potentially some directive with it, given a basis model and the mental model of the human(s) in the loop. In this paradigm, the directive accompanying the plan may be an update to the mental model (i.e. an explanation of the plan). In contrast to the traditional meta-reasoning approaches that trade-off computation time with solution quality, the reasoning task in human-aware planning trades off the solution quality in the basis model with how it will be perceived in the mental model [15].

At this point, we want to make it clear that even though, conceptually, the model space reasoning problems described in this paper are looking for solutions (new models) in the space of models, and classical planning tasks are looking for solutions (plans) in the space of plans, these are not technically equivalent to plan-space and state-space search approaches used in planning [26]. Indeed, if the reasoning task is compiled to be represented by a state-space representation, then both plans and models can be searched for in the space of states. The approach in [61] does exactly that for the explicability-explanations trade-off originally envisaged explicitly in model-space search in [15]. We do the same in our compilations for unsolvability and executability for LLM as a pre-processor, while for LLM as a Post Processor, we use the original model space search from [13].

Automated Planning ft. Neural Networks & LLMs

Finally, there is a long history of work incorporating statistical models and machine learning, particularly deep neural networks, in planning tasks. Historically, these works have only considered the classical planning task of computing a plan given a model. To that end, researchers have looked at any and all aspects of the classical planning task of computing a plan given a model.

Learning heuristics

Heuristics play a key role in speeding up CS by providing goal-directed guidance – this is typically achieved by solving a simplified version of the planning task at every search node and using information from that solution as guidance. The better the approximation or simplification, the better the heuristic guidance. As an alternative to the (human-made) approximation approach for devising heuristics, given experience from previous searches, one can train a model to learn a heuristic [39]. One of the early applications attempts to learn heuristics for automated planning using deep neural networks was in [17]. Many researchers [57, 23, 18] have since tried to replicate this idea with varying levels of success – the computation overhead of a learned heuristic during the search process remains an inhibiting factor.

Scaling up

Automated planning, even in its simplest form, is computationally expensive [26]. Recent work [28] have looked at training models on simpler problems and using them to scale up to problems of higher complexity where the learned approaches might not have all the nice guarantees of a traditional solver but, on the other hand, can at least solve the problem with some level of quality instead of timing out. Relatedly, learning approaches can also be used to scale up planning using model abstractions [56].

Transition functions

There have been several attempts to learn the transition function of planning tasks in the form of PDDL directly from images [3, 2, 4] or text [41]. There is an entire world of model learning for planning [8] using both statistical as well as non-statistical learning techniques which we do not get into here. Model learning as a task, although involves producing a model at the end, is distinctly different from model space reasoning tasks in that the target there is to produce a model that is maximum likelihood given an input dataset (versus evolving a given model to meet a set of desired properties).

End-to-end

Finally, with the increasing effectiveness of large-scale language models, researchers are actively exploring whether a planning task can be done end-to-end using LLMs [70, 58, 52]. Some recent

approaches have even tried to produce PDDL representations using large language models [42] – not quite end-to-end but the final step from PDDL to plan is lossless. Perhaps one of the earliest approaches to using language models in classical planning tasks is [68], where authors used word embeddings to achieve a lower fidelity planning task they term "plan completion". Follow-up works [73] to this have also attempted to use other deep networks to this end. The task of composing sequences, especially in the context of service composition using natural language as input, has also received much attention [38, 43]. These are similar in fidelity to planning in real worldly domains such as the one discussed previously in [1, 33]. While still largely underwhelming in terms of the accuracy of the output plans, these works do demonstrate rapid improvement. This is an intriguing development in planning research, especially as a way to bypass the computationally expensive combinatorial search process when possible.

C Sample Prompts

C.1 LLM only Setting for Unsolvability

```
Prompt Template:

"given the following problem and domain files:" + domain_content + "," +
problem_content + "Come up with most reasonable set of additions that you
can make to the initial state that will make it solvable. I want you to only
list the predicates to be added to the initial states without any
explanation or additional sentences in the beginning."

Example values:

domain_content:
   (define (domain domaingotocity)
   (:requirements :typing)
   (:types city - object)
   (:predicates
       (at ?x - city)
       (has_taxi ?x ?y - city)
       (has_bus ?x ?y - city)
       (neighboring ?x ?y - city)
   )
   (:action use_taxi
       :parameters (?from ?to - city)
       :precondition (and
         (at ?from)
         (has_taxi ?from ?to)
       )
       :effect (and
         (not (at ?from))
         (at ?to))
     )
   (:action use_bus
       :parameters (?from ?to - city)
       :precondition (and
         (at ?from)
         (has_bus ?from ?to)
       )
       :effect (and
         (not (at ?from))
         (at ?to))
     )
   )
   problem_content:
   (define (problem problemgotocity)
```

```
    (:domain domaingotocity)
    (:objects city_a - city
    city_b - city city_c - city city_d - city
    city_e - city city_f - city city_j - city
    city_l - city city_o - city city_r - city
    city_s - city city_t - city city_v - city
    city_x - city)
    (:init
    (at city_a)
    (has_bus city_a city_b)
    (has_bus city_a city_d)
    (has_bus city_d city_j)
    (has_bus city_l city_v)
    (has_bus city_t city_e)
    (has_taxi city_e city_o)
    (has_taxi city_e city_x)
    (has_taxi city_f city_s)
    (has_taxi city_r city_l)
    (has_taxi city_s city_c)
    (neighboring city_a city_b)
    (neighboring city_a city_d)
    (neighboring city_b city_c)
    (neighboring city_d city_j)
    (neighboring city_e city_o)
    (neighboring city_e city_x)
    (neighboring city_f city_s)
    (neighboring city_l city_v)
    (neighboring city_r city_l)
    (neighboring city_s city_c)
    (neighboring city_t city_e)
    (neighboring city_t city_o))
    (:goal (at city_c))
    )
output:
(:init (at city_a)
(has_bus city_a city_b)
(has_bus city_a city_d)
(has_bus city_d city_j)
(has_bus city_l city_v)
(has_bus city_t city_e)
(has_taxi city_e city_o)
(has_taxi city_e city_x)
(has_taxi city_f city_s)
(has_taxi city_r city_l)
(has_taxi city_s city_c)
(neighboring city_a city_b)
(neighboring city_a city_d)
(neighboring city_b city_c)
(neighboring city_d city_j)
(neighboring city_e city_o)
(neighboring city_e city_x)
(neighboring city_f city_s)
(neighboring city_l city_v)
(neighboring city_r city_l)
(neighboring city_s city_c)
(neighboring city_t city_e)
(neighboring city_t city_o)
(at city_o)
(at city_x))
```

C.2 LLM as Post Processor for Unsolvability

```
Prompt Template:

"Given the following problem, domain files, and options list:
- Problem: {uns_problem_string}
- Domain: {domain_string}
- Options: {option_list}
Pick the most reasonable option from the list that you can apply to the
initial state to make the problem solvable. Only provide the number of
the option selected and no other information (exclude even the term
option)."

Example values:

   uns_problem_string:
   (define (problem problemgotocity)
       (:domain domaingotocity)
       (:objects city_a - city
       city_b - city city_c - city
       city_d - city city_e - city
       city_f - city city_j - city
       city_l - city city_o - city
       city_r - city city_s - city
       city_t - city city_v - city
       city_x - city)
       (:init (at city_a)
       (has_bus city_a city_b)
       (has_bus city_a city_d)
       (has_bus city_d city_j)
       (has_bus city_l city_v)
       (has_bus city_t city_e)
       (has_taxi city_e city_o)
       (has_taxi city_e city_x)
       (has_taxi city_f city_s)
       (has_taxi city_r city_l)
       (has_taxi city_s city_c)
       (neighboring city_a city_b)
       (neighboring city_a city_d)
       (neighboring city_b city_c)
       (neighboring city_d city_j)
       (neighboring city_e city_o)
       (neighboring city_e city_x)
       (neighboring city_f city_s)
       (neighboring city_l city_v)
       (neighboring city_r city_l)
       (neighboring city_s city_c)
       (neighboring city_t city_e)
       (neighboring city_t city_o))
       (:goal (at city_c))
   )
   domain_string:
   (define (domain domaingotocity)
       (:requirements :typing)
       (:types city)
       (:predicates (at ?x - city)
       (has_bus ?x - city ?y - city)
       (has_taxi ?x - city ?y - city)
       (neighboring ?x - city
```

```
    ?y - city))
    (:action use_bus
        :parameters
        (?from - city ?to - city )
        :precondition
        (and (at ?from)
        (has_bus ?from ?to))
        :effect
        (and (not (at ?from))
        (at ?to))
    )
     (:action use_taxi
        :parameters
        (?from - city ?to - city )
        :precondition (and (at ?from)
        (has_taxi ?from ?to))
        :effect (and (not (at ?from))
        (at ?to))
    )
)
```

```
options:

["Option 1: {'has_taxi city_a city_c'}",
"Option 2: {'has_taxi city_b city_c'}",
"Option 3: {'has_bus city_d city_c'}",
"Option 4: {'has_taxi city_a city_s'}",
"Option 5: {'has_bus city_a city_s'}",
"Option 6: {'has_bus city_b city_c'}",
"Option 7: {'has_taxi city_b city_s'}",
"Option 8: {'has_taxi city_d city_s'}",
"Option 9: {'has_bus city_b city_f'}",
"Option 10: {'at city_s'}",
"Option 11: {'has_bus city_a city_c'}",
"Option 12: {'has_bus city_a city_f'}",
"Option 13: {'has_taxi city_j city_s'}",
"Option 14: {'has_bus city_j city_f'}",
"Option 15: {'has_bus city_d city_f'}",
"Option 16: {'has_taxi city_d city_c'}",
"Option 17: {'has_taxi city_j city_f'}",
"Option 18: {'at city_f'}",
"Option 19: {'has_bus city_d city_s'}",
"Option 20: {'has_bus city_j city_s'}"]

output:
4
```

C.3 LLM as Pre Processor Setting for Unsolvability

```
Prompt Template:

prompt = f"Given the following problem and domain file:
Problem:{uns_problem_string} Domain:\n {domain_string}
Come up with a list of twenty predicates that are currently
missing from the initial state. Order the predicates in such
a way that the predicates in the top correspond to changes that
are most reasonable to make (the predicate will added to the
existing initial state). Only list the initial state predicate,
one predicate in a line, and provide no other information.
Do not include any number in the list and do not include any
text before the list."


Example values:

    uns_problem_string:
    (define (problem problem_barman)
    (:domain barman)
    (:objects cocktail_a - cocktail
    ingredient_a - ingredient
    ingredient_b - ingredient
    shaker_a - shaker shot_a - shot)
    (:init (cocktail-part1 c
    ocktail_a ingredient_a)
    (cocktail-part2 cocktail_a ingredient_b)
    (contains shaker_a ingredient_a)
    (contains shaker_a ingredient_b)
    (empty shot_a) (unshaked shaker_a))
    (:goal (contains shot_a cocktail_a))
    )



domain_string:
 (define (domain barman)
(:requirements :strips :typing)
(:types
beverage container - object
ingredient cocktail - beverage
shot shaker - container
)
(:predicates
(empty ?c - container)
(contains ?c - container ?b - beverage)
(clean ?c - container)
(unshaked ?s - shaker)
(shaked ?s - shaker)
(cocktail-part1 ?a - cocktail ?b - ingredient)
(cocktail-part2 ?a - cocktail ?b - ingredient)
)

(:action shake
:parameters (?b - cocktail ?d1 ?d2 - ingredient ?s - shaker)
:precondition (and
(contains ?s ?d1)
```

```
(contains ?s ?d2)
(unshaked ?s))
:effect (and
(not (unshaked ?s))
(not (contains ?s ?d1))
(not (contains ?s ?d2))
(shaked ?s)
(cocktail-part1 ?b ?d1)
(cocktail-part2 ?b ?d1)
(contains ?s ?b))
)

(:action pour-shaker-to-shot
:parameters (?b - cocktail ?d - shot ?s - shaker ?d1 ?d2 - ingredient)
:precondition (and
(shaked ?s)
(empty ?d)
(clean ?d)
(contains ?s ?b)
(cocktail-part1 ?b ?d1)
(cocktail-part2 ?b ?d2)
)
:effect (and
(not (clean ?d))
(not (empty ?d))
(contains ?d ?b)
))
)
```

C.4 LLM only Setting for Executability

```
 Prompt Template:

"given the following problem and domain and plan files:" + domain_content + ","
+ problem_content + ","+ plan_content + "," + "Come up with most reasonable set
of additions that you can make to the initial state that will make the plan
executable. I want you to only list the predicates to be added to the initial
states without any explanation or additional sentences in the beginning."

 Example values:

domain_content:
(define (domain cleaning)
(:requirements :typing)
(:types room door -object)
(:predicates
    (connects ?x - room ?y - room ?z - door)
    (is_open ?x - door)
    (at ?x - room)
    (is_dirty ?x - room)
    (is_unlocked ?x - door)
    (neighboring ?x ?y - room)
    (is_clean ?x - room)
)
(:action open_door
    :parameters (
    ?x - door
    )
    :precondition (and
      (is_unlocked ?x)
    )
    :effect (and
      (is_open ?x)
    )
)
(:action go
    :parameters (
    ?from - room
    ?to - room
    ?x - door
    )
    :precondition (and
      (at ?from)
      (connects ?from ?to ?x)
      (is_open ?x)
      (neighboring ?from ?to)
    )
    :effect (and
      (at ?to)
      (not(at ?from))
    )
)
(:action clean
    :parameters (
    ?x - room
    )
    :precondition (and
```

```
      (at ?x)
      (is_dirty ?x)
    )
    :effect (and
      (is_clean ?x)
    )
)
)


problem_content:
(define (problem problem_cleaning_robot)
    (:domain cleaning)
    (:objects door_a - door door_b - door door_c - door door_d - door
    room_a - room room_b - room room_c - room room_d - room room_e - room
    room_f - room room_g - room room_h - room)
    (:init (at room_a) (connects room_a room_a door_b)
    (connects room_a room_b door_a) (connects room_a room_c door_d)
    (connects room_a room_d door_c) (is_dirty room_b)
    (neighboring room_a room_b) (neighboring room_a room_h)
    (neighboring room_c room_d) (neighboring room_d room_g)
    (neighboring room_f room_a) (neighboring room_f room_d)
    (neighboring room_h room_e))
    (:goal (is_clean room_b))
)
plan_content:
(open_door door_a )
(go room_a room_b door_a )
(clean room_b )
```

C.5 LLM as Post Processor for Executability

```
 Prompt Template:

 "Given the following problem, domain files, and options list:
    - Problem: {uns_problem_string}
    - Domain: {domain_string}
    - Options: {option_list}


Pick the most reasonable option from the list that you can apply to the initial
state to make the following plan executable.
- Plan: {original_plan}
Only provide the number of the option selected and no other information
(exclude even the term option)."

 Example values:

    uns_problem_string:
    (define (problem problemgotocity)
        (:domain domaingotocity)
        (:objects city_a - city city_b - city city_c - city city_d - city
        city_e - city city_f - city city_g - city city_h - city city_i - city
        city_j - city)

        (:init (at city_a) (has_bus city_a city_b) (has_bus city_a city_f)
        (has_bus city_a city_i) (has_bus city_a city_j)
        (has_bus city_h city_g) (has_bus city_j city_f) (has_taxi city_c city_d)
        (has_taxi city_c city_e) (has_taxi city_d city_e)
        (has_taxi city_g city_f) (has_taxi city_h city_a)
        (has_taxi city_j city_a) (neighboring city_a city_b)
        (neighboring city_a city_f) (neighboring city_a city_i)
        (neighboring city_a city_j) (neighboring city_b city_c)
        (neighboring city_c city_d) (neighboring city_c city_e)
        (neighboring city_d city_e) (neighboring city_f city_c)
        (neighboring city_g city_f) (neighboring city_h city_a)
        (neighboring city_h city_g) (neighboring city_j city_a)
        (neighboring city_j city_f))
        (:goal (at city_e))
    )

    domain_string:
        (define (domain domaingotocity)
    (:requirements :typing)
    (:types city - object)
    (:predicates
        (at ?x - city)
        (has_taxi ?x ?y - city)
        (has_bus ?x ?y - city)
        (neighboring ?x ?y - city)
    )

      (:action use_taxi
        :parameters (?from ?to - city)
        :precondition (and
          (at ?from)
          (has_taxi ?from ?to)
        )
```

```
    :effect (and
      (not (at ?from))
      (at ?to)
    )
  )

  (:action use_bus
    :parameters (?from ?to - city)
    :precondition (and
      (at ?from)
      (has_bus ?from ?to)
    )
    :effect (and
      (not (at ?from))
      (at ?to)
    )
  )
)

option_list:
['Option 1: (has_bus city_b city_c)',
'Option 2: (has_bus city_b city_c) (has_bus city_c city_d)',
'Option 3: (has_bus city_b city_c) (has_bus city_b city_i)',
'Option 4: (has_bus city_b city_c) (has_bus city_g city_d)',
'Option 5: (has_bus city_g city_i) (has_bus city_b city_c)',
'Option 6: (has_bus city_b city_c) (has_bus city_h city_c)',
'Option 7: (has_taxi city_e city_d) (has_bus city_b city_c)',
'Option 8: (has_bus city_b city_c) (has_taxi city_f city_j)',
'Option 9: (has_taxi city_h city_i) (has_bus city_b city_c)',
'Option 10: (has_bus city_b city_c) (neighboring city_b city_i)',
'Option 11: (has_bus city_b city_c) (neighboring city_c city_b)',
'Option 12: (has_bus city_b city_c) (has_bus city_c city_f)',
'Option 13: (has_bus city_d city_d) (has_bus city_b city_c)',
'Option 14: (has_bus city_b city_c) (has_bus city_e city_g)',
'Option 15: (has_bus city_b city_c) (has_bus city_f city_a)',
'Option 16: (has_bus city_b city_c) (has_bus city_f city_h)',
'Option 17: (has_bus city_b city_c) (has_bus city_g city_e)',
'Option 18: (has_bus city_b city_c) (has_bus city_j city_c)',
'Option 19: (has_bus city_j city_i) (has_bus city_b city_c)',
'Option 20: (has_bus city_b city_c) (has_taxi city_a city_b)']

original_plan:
(use_bus city_a city_b)
(use_bus city_b city_c)
(use_taxi city_c city_e)
; cost = 3 (unit cost)
```

C.6 LLM as Pre Processor Setting for Executability

```
Prompt Template:

prompt = f"Given the following problem, domain, and plan file:
Problem: {uns_problem_string}
Domain: {domain_string}
Plan: {solv_init_plan_string}
Come up with a list of twenty predicates that are currently missing from the initial state
to make the plan executable. Order the predicates in such a way that the predicates in
the top correspond to changes that are most reasonable to make (the predicate will added
to the existing initial state). Only list the initial state predicate, one predicate in a
line, and provide no other information. Do not include any number in the list and do
not include any text before the list."

Example values:

   uns_problem_string:
   (define (problem problem_barman)
   (:domain barman)
   (:objects cocktail_a - cocktail
   ingredient_a - ingredient
   ingredient_b - ingredient
   shaker_a - shaker shot_a - shot)
   (:init (cocktail-part1 c
   ocktail_a ingredient_a)
   (cocktail-part2 cocktail_a ingredient_b)
   (contains shaker_a ingredient_a)
   (contains shaker_a ingredient_b)
   (empty shot_a) (unshaked shaker_a))
   (:goal (contains shot_a cocktail_a))
   )



   domain_string:
   (define (domain barman)
(:requirements :strips :typing)
(:types
beverage container - object
ingredient cocktail - beverage
shot shaker - container
)
(:predicates
(empty ?c - container)
(contains ?c - container ?b - beverage)
(clean ?c - container)
(unshaked ?s - shaker)
(shaked ?s - shaker)
(cocktail-part1 ?a - cocktail ?b - ingredient)
(cocktail-part2 ?a - cocktail ?b - ingredient)
)

(:action shake
:parameters (?b - cocktail ?d1 ?d2 - ingredient ?s - shaker)
:precondition (and
(contains ?s ?d1)
(contains ?s ?d2)
```

```
(unshaked ?s))
:effect (and
(not (unshaked ?s))
(not (contains ?s ?d1))
(not (contains ?s ?d2))
(shaked ?s)
(cocktail-part1 ?b ?d1)
(cocktail-part2 ?b ?d1)
(contains ?s ?b))
)

(:action pour-shaker-to-shot
:parameters (?b - cocktail ?d - shot ?s - shaker ?d1 ?d2 - ingredient)
:precondition (and
(shaked ?s)
(empty ?d)
(clean ?d)
(contains ?s ?b)
(cocktail-part1 ?b ?d1)
(cocktail-part2 ?b ?d2)
)
:effect (and
(not (clean ?d))
(not (empty ?d))
(contains ?d ?b)
))
        )

    solv_init_plan_string:
    (shake cocktail_a ingredient_a ingredient_b shaker_a)
    (pour-shaker-to-shot cocktail_a shot_a shaker_a ingredient_a ingredient_a)
    ; cost = 2 (unit cost)
```