# Can LLMs Fix Issues with Reasoning Models? Towards More Likely Models for AI Planning*

**Turgay Caglar[1], Sirine Belhaj[2], Tathagata Chakraborti[3], Michael Katz[3], Sarath Sreedharan[1]**
[1]Colorado State University
[2]Ecole Polytechnique de Tunisie
[3]IBM Research
tcaglar@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 union, 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 (Ghallab, Nau, and Traverso 2004). 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 (Haslum et al. 2019; McDermott et al. 1998) – 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.

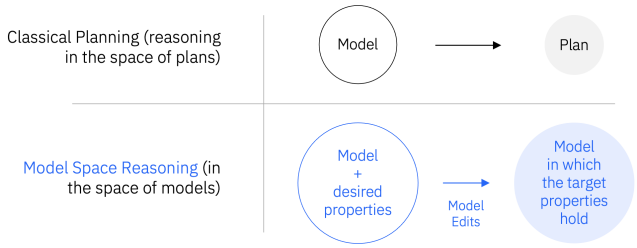



Figure 1: Classical planning versus model space problems.

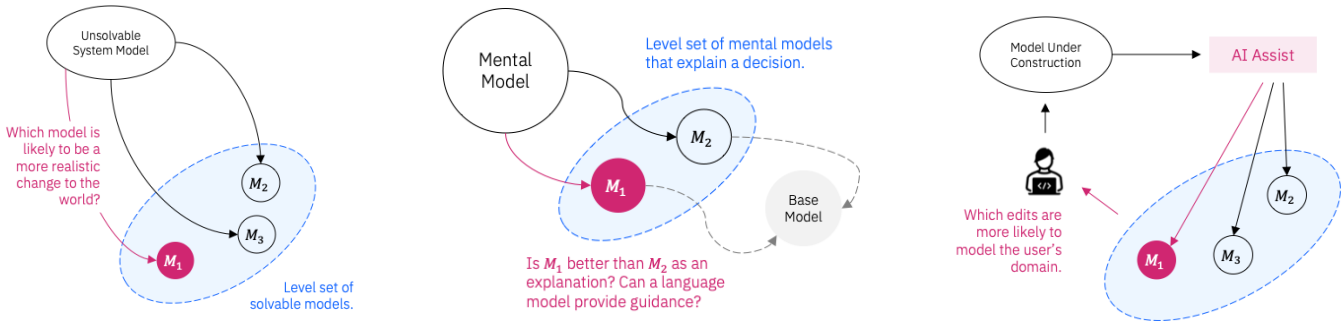
### Model Space Problems in AI Planning

We begin by enumerating the different flavors of model space reasoning 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 (Figure 1).

**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 (Muise and Lipovetzky 2023), to *detect* unsolvability of planning tasks up-front to speed up calls to a planning module (Bäckström, Jonsson, and Ståhlberg 2013; Moreira and Ralha 2017), and attempts to compute or even learn heuristics (Hoffmann, Kissmann, and Torralba 2014; Ståhlberg 2017; Ståhlberg, Francès, and Seipp 2021) and produce certificates (Eriksson, Röger, and Helmert 2017, 2018; Eriksson and Helmert 2020) 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 (Göbelbecker et al. 2010) 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

*A longer version of the paper with detailed examples of prompts is available at https://arxiv.org/abs/2311.13720.




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

Figure 2: 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 2a 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 2b, 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 2c). 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.

current state were changed then it would become solvable. In addition to initial state changes, this idea was later extended (Herzig et al. 2014) 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 (Sreedharan et al. 2020b, 2019) 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 (Käser et al. 2022) 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 (Sreedharan et al. 2020a,b, 2019) 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 (Chakraborti, Sreedharan, and Kambhampati 2020; Fox, Long, and Magazzeni 2017; Chakraborti et al. 2019). 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" (Chakraborti et al. 2017) – 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" (Miller 2019)) and establish the optimality of a plan (Chakraborti et al. 2017).

Interestingly, one can remove (Chakraborti and Kambhampat 2019a) 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 (Chakraborti and Kambhampat 2019a) envisaged that algorithms which have looked at linguistic patterns for model evolution (Porteous et al. 2015; Porteous 2016) can assist in finding more probable models. This, of course, raises several ethical questions (Chakraborti and Kambhampat 2019b), 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 (Vallati and Kitchin 2020) 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 (Keren, Gal, and Karpas 2014; Mirsky et al. 2019; Wayllace et al. 2016) or making the behavior of agents easier to interpret (Kulkarni et al. 2019, 2020). 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 (Sreedharan et al. 2020b) demonstrate this in a large-scale industrial domain on authoring models for goal-oriented conversational agents (Muise et al. 2020). The role of an AI assist in authoring problems is especially critical in what we call "real worldly domains".

### 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 (Göbelbecker et al. 2010; Chakraborti et al. 2017; Sreedharan et al. 2020a) 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 (Zahedi et al. 2019), 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 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 (Ahn et al. 2023; Huang et al. 2023), and in specialized but common tasks such as service composition (LangChain 2023; Maeda and Chaki 2023). 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 an extensive and systematic exploration of the role of LLMs in model space search. To this end, we analyze the effectiveness 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) (Muise 2023), 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 (Sreedharan et al. 2020c,b). 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 (Francés, Ramirez, and Collaborators 2018). An insight into the likelihood of models can empower the domain author to create and debug models with greater ease (Sreedharan et al. 2020b; Käser et al. 2022), as well as allow automated model adaptation in fully autonomous systems in nonstationary environments (Bryce, Benton, and Boldt 2016) or in constrained creative tasks like story-telling (Simon and Muise 2022; Porteous 2016; Porteous et al. 2021) that have previously relied on using limited linguistic cues like antonyms and synonyms (Porteous et al. 2015) 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 3 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 $\Pi_i$ 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

[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, 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.

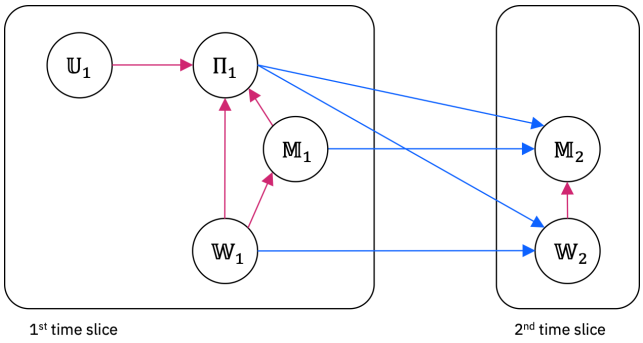



Figure 3: A DBN representing the random variables and their relations that are relevant to the problem at hand. The blue lines capture the diachronic, i.e., over time, relationships, and the maroon lines capture the synchronic ones.

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 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:

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

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 (see Figure 4) 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. We use the OpenAI API (OpenAI 2023) for this approach.

**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:

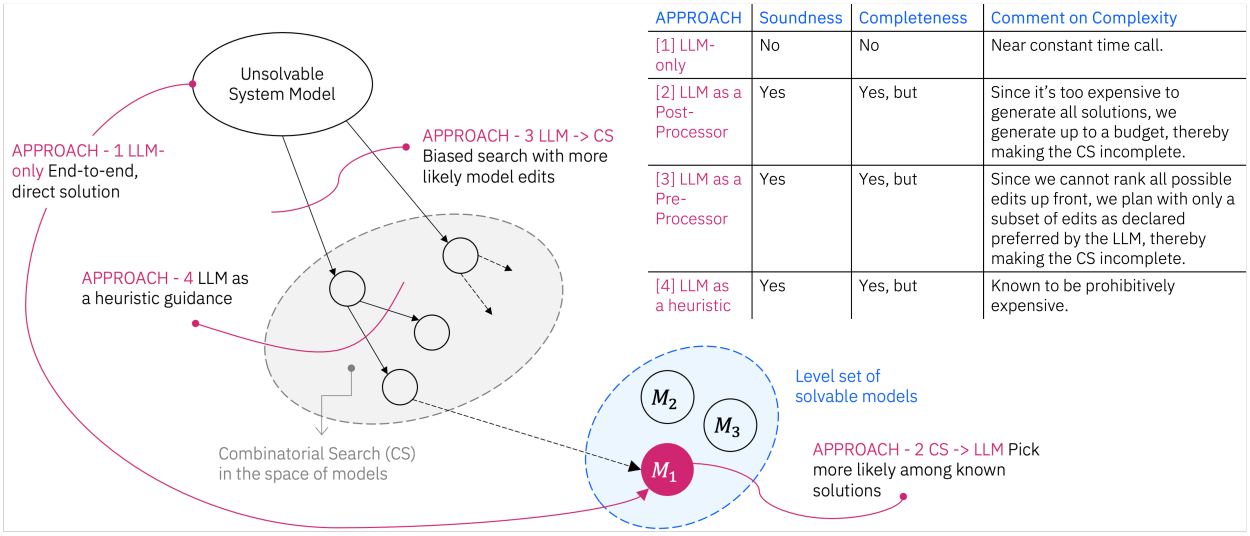
$$\mathcal{M} = \underset{\mathcal{M}' \in \hat{\mathbb{M}}}{\arg\max}\, P(\mathbb{M}_2 = \mathcal{M}' \mid \mathbb{M}_1 = \mathcal{M}_1, \mathbb{U}_1 = \mathcal{U}), \tag{3}$$

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:

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




| APPROACH | Soundness | Completeness | Comment on Complexity |
|---|---|---|---|
| [1] LLM-only | No | No | Near constant time call. |
| [2] LLM as a Post-Processor | Yes | Yes, but | Since it's too expensive to generate all solutions, we generate up to a budget, thereby making the CS incomplete. |
| [3] LLM as a Pre-Processor | Yes | Yes, but | Since we cannot rank all possible edits up front, we plan with only a subset of edits as declared preferred by the LLM, thereby making the CS incomplete. |
| [4] LLM as a heuristic | Yes | Yes, but | Known to be prohibitively expensive. |

Figure 4: Different points of contact with LLMs and the CS process. While Approach-4 is known to be too expensive, we explore Approaches 1-3 in this paper in terms of the soundness and likelihood of solutions.

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 (Ferber, Helmert, and Hoffmann 2020) 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 paths. Along the paths, obstacles such as walls, chairs, or tables may be present. If a path is blocked, the agent can not move to an adjacent cell. Changes are reasonable if they involve removing chairs or tables that obstruct the path and adding 'path clear' to the corresponding cells.

| Unsolvability Domains | LLM-Only | | | | LLM as Post Processor | | | | LLM as Pre Processor | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 Domains | LLM-Only | | | | LLM as Post Processor | | | | LLM as Pre Processor | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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.

**Barman-simple** This is a modified version of the IPC barman domain (Celorrio 2011). 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.

## 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 Appendix Section C.

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

**H1** LLM can identify sound model updates.

**H2** LLM can identify reasonable model updates.

**H3** The ability to find sound model updates improves with the capability of the LLM.

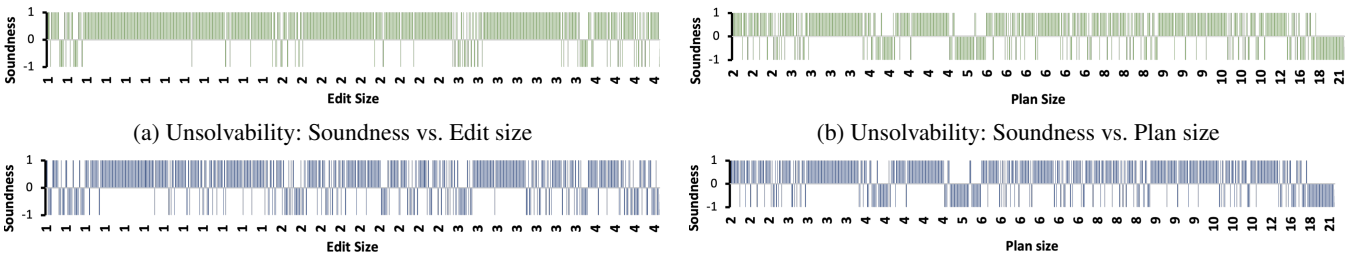
**H4** The ability to find reasonable model updates improves with the capability of the LLM.

**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** H1 and H2 are measured directly against the ground truth, as per the problem-generation process explained at the start of Section 4. 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.




(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 5: 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.

## 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 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 5).

## 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 contrast to early attempts (Gragera and Pozanco 2023) to use LLMs for model corrections, which were constrained to limited settings and models that are no longer the state of the art, we find LLMs to be surprisingly competent at this task. 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 (Howey, Long, and Fox 2004) 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, and reduces authoring overhead for planning tasks in the future.

## Acknowledgements

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

## References

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

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

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

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

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

Chakraborti, T.; and Kambhampat, S. 2019b. (When) Can Bots Lie? In *AIES*.

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

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

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

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

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

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

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

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. *arXiv:1709.10256*.

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

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

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

Gragera, A.; and Pozanco, A. 2023. Exploring the Limitations of using Large Language Models to Fix Planning Tasks. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

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

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

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

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

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

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

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

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

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

LangChain. 2023. LangChain is a framework for developing applications powered by language models. https://python.langchain.com.

Maeda, J.; and Chaki, E. 2023. Semantic Kernel. https://learn.microsoft.com/en-us/semantic-kernel.

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

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

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

Moreira, L. H.; and Ralha, C. G. 2017. Improving Multiagent Planning with Unsolvability and Independent Plan Detection. In *Brazilian Conference on Intelligent Systems (BRACIS)*.

Muise, C. 2023. API.Planning.Domains: An interface to the repository of PDDL domains and problems. http://api.planning.domains.

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

Muise, C.; and Lipovetzky, N. 2023. International Plannning Competition (IPC) Unsolvability Track. https://unsolve-ipc.eng.unimelb.edu.au.

OpenAI. 2023. GPT-4 Technical Report. *arXiv:2303.08774*.

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

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

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

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

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

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

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

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

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

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

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

Wayllace, C.; Hou, P.; Yeoh, W.; and Son, T. C. 2016. Goal Recognition Design with Stochastic Agent Action Outcomes. In *IJCAI*.

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