Finding Semantically Guided Repairs in PDDL Domains Using LLMs

Nader Karimi Bavandpour, Pascal Bercher

College of Systems & Society, The Australian National University, Australia
{Nader.KarimiBavandpour, Pascal.Bercher}@anu.edu.au

Abstract

Repairing Planning Domain Definition Language (PDDL) models is difficult because solutions must ensure correctness while remaining interpretable to human modellers. Existing hitting set methods identify minimal repair sets from whitelist and blacklist traces, but they cannot prefer semantically meaningful fixes and the true repair may not be minimal. We propose combining large language models (LLMs) with the hitting set framework, using semantic cues in PDDL action and predicate names to guide repairs. This hybrid approach provides contrastive, counterfactual explanations of why traces fail and how domains could behave differently.

Introduction

Explainability is a central requirement for AI systems that interact with or support humans in decision making. In AI planning, this requirement is naturally addressed by the explicit representation of actions, states, and goals: planners generate solutions by reasoning over structured models of the world. Compared to black–box machine learning techniques, this explicit reasoning process makes planning inherently transparent and interpretable. However, one of the main challenges to deploying planning in practice lies in constructing the planning models themselves (Tantakoun, Zhu, and Muise 2025).

The recent success of Largarge language models (LLMs) has drawn a lot of attention to leverage it in AI planning tasks (Huang, Lipovetzky, and Cohn 2025; Katz et al. 2025; Huang, Cohn, and Lipovetzky 2024; Oswald et al. 2024; Guan et al. 2023). Recent surveys (Tantakoun, Zhu, and Muise 2025) highlight the potential of LLMs to support the construction and refinement of planning models. While verifiable planning modules remain the backbone of reliability, robustness, and explainability, LLMs can act as assistants to reduce the manual burden of defining domain models. We believe that one promising avenue for this is through *domain repair*, where the goal is to identify a set of modifications to a domain such that a given set of positive traces becomes executable and a set of negative traces becomes inapplicable.

Repairs can themselves be understood as explanations. Following Miller's account of contrastive explanations in the social sciences (Miller 2019), a repair answers the question of why a given trace fails in the current domain, and provides a counterfactual justification of how the domain could have behaved differently. Each repair is thus not only a technical fix, but also a form of interpretable feedback to the human modeller.

Domain repair in planning is an active research area, recently reviewed by (Bercher, Sreedharan, and Vallati 2025). Lin, Grastien, and Bercher (2023) introduced an efficient hitting set algorithm for domain repair with positive (whitelist) traces. It is also possible to use partially lifted test plans if partial information is available (Bavandpour et al. 2025). Lin et al. (2025) extended the hitting–set approach of Lin, Grastien, and Bercher (2023) to handle both positive and negative traces. In this setting, positive traces must remain or become valid plans, while negative (blacklist) traces must be rendered inapplicable. Although effective, optimization–based approaches, such as the cited works, are limited: they cannot select the semantically most meaningful repair when multiple minimal–cardinality options exist. Moreover, the true repair set may not even be among the minimal–cardinality solutions. The VS Code plugin (Lin, Yousefi, and Bercher 2024) which extends the work by Lin, Grastien, and Bercher (2023) is an effort to address this limitation by allowing the human modeller to reject solutions and request the next solution of the hitting–set. However, this is inconvenient and limits automation. Our core idea is to leverage the fact that PDDL domains contain semantically meaningful names for actions and predicates, which LLMs can interpret to guide the repair process. This enables us to move beyond cardinality–minimality toward repairs that are also semantically plausible to human users.

Building on insights from Caglar et al. (2024), who explored LLM applications for model repair but limited their scope to initial state fixes, we propose to combine LLM guidance with the hitting set approach of (Lin et al. 2025). This hybrid aims to exploit semantic knowledge encoded in domain symbols while retaining the optimization guarantees of hitting set solvers. Our experiments evaluate this approach on the same benchmark suite as the baseline paper, allowing for a direct comparison of performance. This paper presents our initial implementation and evaluation of this idea. Extensions and further refinements will be discussed in future work.

Planning Formalism

Since our focus is on repairing lifted PDDL domains, we introduce the lifted planning formalism. A lifted planning problem is defined as a tuple $\Pi = (\mathcal{P}, \mathcal{A}, \alpha, \mathcal{O}, s^I, G)$, where the domain is $\mathcal{D} = (\mathcal{P}, \mathcal{A}, \alpha)$ and the task is $\mathcal{T} = (\mathcal{O}, s^I, G)$.

Objects, Types, and Variables. Let $\mathcal{O}$ be the set of objects in the planning task. We consider a set of variables $\mathcal{V}$, each acting as a placeholder for an object. The type of a variable $v \in \mathcal{V}$ is written as $v|t$, and the set of objects associated with t is shown by $\mathcal{O}[\![t]\!] \subseteq \mathcal{O}$. We say that $t \in \mathcal{T}$ is a *subtype* of $t' \in \mathcal{T}$ iff $\mathcal{O}[\![t]\!] \subseteq \mathcal{O}[\![t']\!]$.

Predicates and Atoms A predicate symbol $P \in \mathcal{P}$ and a tuple of $k \in \mathbb{N}_0$ typed variables forms an atom $\mathbf{p} = P(v_1|t_1, \ldots, v_k|t_k)$. We denote by $\mathcal{L}$ the set of all atoms in Π.

Variable Substitution. A variable substitution function $\varrho : \mathcal{V} \to \mathcal{O}$ maps each typed variable $v|t$ to an object $\varrho(v|t) \in \mathcal{O}[\![t]\!]$ of the same type t.

Facts. Given an atom $\mathbf{p} \in \mathcal{L}$ and a substitution function ϱ, a *fact* is obtained by grounding $\mathbf{p}$, that is, by replacing each parameter $(v_1, \ldots, v_k)$ with the corresponding objects given by ϱ: $f = \varrho(\mathbf{p}) = P(\varrho(v_1), \ldots, \varrho(v_k))$. The set of all grounded atoms is denoted by $\mathcal{F}$, and any set of facts constitutes a *state*. s^I and G are called the initial state and the goal description, respectively, each of which is a set of facts.

Action Schemas. Let $\mathcal{A}$ denote the set of action schemas. An action schema $\mathbf{a} = A(v_1|t_1, \ldots, v_k|t_k)$ is defined by a unique name A and a tuple of n variables. Each schema is associated with a mapping

$$\alpha(\mathbf{a}) = (\mathit{prec}^+(\mathbf{a}), \mathit{prec}^-(\mathbf{a}), \mathit{eff}^+(\mathbf{a}), \mathit{eff}^-(\mathbf{a}))$$

whose codomain is $(2^{\mathcal{L}})^4$, representing a tuple of four sets of compatible atoms, as defined below.

Definition 1 (Compatible Atoms)**.** For an action schema $\mathbf{a}$, the set of *compatible atoms* $\mathcal{L}^{\mathbf{a}}$ contains all atoms whose set of parameters is a subset of the parameters of $\mathbf{a}$. As an example, $P(v_1|t_1)$ is a compatible atom for the action schema $A(v_1|t_1, v_2|t_2)$, but $Q(v_3|t_3)$ is not.

Actions. Given an action schema $\mathbf{a}$ and a substitution function ϱ, the corresponding *action* is obtained by replacing each parameter of $\mathbf{a}$ according to ϱ, and is denoted $a = \mathbf{a}[\varrho]$. Actions describe transitions in the state space. An action a is *applicable* in a state s iff $\mathit{prec}^+(a) \subseteq s$ and $\mathit{prec}^-(a) \cap s = \emptyset$. Applying an applicable action a in s produces the successor state

$$s' = (s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a),$$

which we denote by $s \to_a s'$.

Throughout this paper, we use boldface (e.g., $\mathbf{p}$, $\mathbf{a}$) for atoms and action schemas, and regular typeface (e.g., f, a) for facts and actions.

Solutions. Let $\gamma = \langle a_1, \ldots, a_k \rangle$ be an action sequence. We write $s \to_\gamma^* s'$ to denote that s' results from applying γ to s via a state trajectory $\langle s_0, \ldots, s_k \rangle$ where $s_0 = s$, $s_k = s'$, and each action is applicable in its preceding state. A solution to a planning problem is an action sequence $\gamma = \langle a_1, \ldots, a_k \rangle$ such that $s_I \to_\gamma^* s'$ for some s' with $G \subseteq s'$, and each a_i is a grounding of some action schema $\mathbf{a} \in \mathcal{A}$.

The Repair Problem

We begin by introducing the notation and syntax used to define possible repair operations for a given planning domain. Next, we describe how a set of such repairs can be applied to produce a modified domain. Based on these concepts, we then formalize the domain repair problem in terms of the defined repair operations and a set of positive and negative plans.

In a planning domain $\mathcal{D} = (\mathcal{P}, \mathcal{A}, \alpha)$, an *atomic repair* is a modification denoted by $r[\![\mathbf{a}, \mathbf{p}, c, \mathit{op}]\!]$. Here, $\mathbf{a} \in \mathcal{A}$ is an action schema, $\mathbf{p} \in \mathcal{P}$ is an atom compatible with $\mathbf{a}$, $c \in \{\mathrm{prec}^+, \mathrm{prec}^-, \mathrm{eff}^+, \mathrm{eff}^-\}$ indicates whether the change concerns a positive/negative precondition or effect, and $\mathit{op} \in \{+, -\}$ specifies whether the component is added or removed. We write $\mathcal{D} \Rightarrow_r \mathcal{D}'$ to indicate that applying r to $\mathcal{D}$ yields $\mathcal{D}' = (\mathcal{P}, \mathcal{A}, \alpha')$, where α' is resulted by applying r to α.

A *repair set* δ for a domain is a finite collection of one or more atomic repairs. We say that δ is *valid* if and only if it contains no two repairs $r, r' \in \delta$ such that one reverses the effect of the other. Specifically, two repairs $r = r[\![\mathbf{a}, \mathbf{p}, c, \mathit{op}]\!]$ and $r' = r'[\![\mathbf{a}', \mathbf{p}', c', \mathit{op}']\!]$ are considered to undo each other if $\mathbf{a} = \mathbf{a}'$, $\mathbf{p} = \mathbf{p}'$, $c = c'$, and $\mathit{op} \neq \mathit{op}'$.

Let $\mathcal{D}$ be a domain and δ a valid repair set for $\mathcal{D}$. Applying the repairs in δ in any order yields the same modified domain $\mathcal{D}'$. We use $\mathcal{D} \Rightarrow_\delta^* \mathcal{D}'$ to indicate that $\mathcal{D}'$ is obtained from $\mathcal{D}$ by applying the valid repair set δ.

Definition 1 (Domain Repair Problem)**.** *The domain repair problem is defined as a pair $\mathcal{R} = (\mathcal{D}, \mathbb{T})$, where $\mathcal{D}$ denotes a planning domain and $\mathbb{T} = \{\mathbf{T}_1, \ldots, \mathbf{T}_n\}$ for some $n \in \mathbb{N}$. Each element $\mathbf{T}_i$ is a triple $(\Pi_i, \mathbb{P}_i, \mathbb{E}_i)$. Here, $\Pi_i = (\mathcal{D}, \mathcal{T}_i)$ denotes the planning problem; $\mathbb{P}_i$ is a finite set of positive plans π_k^+ for Π_i; and $\mathbb{E}_i$ is a finite set of pairs (π_k^-, b_k) associated with Π_i. Each π_k^- denotes a negative plan (a sequence of actions considered undesirable) for Π_i, and b_k is an integer equal or less than the length of π_k^- which specifies the index of the first action in π_k^- which must be inapplicable.*

Definition 2 (Solution to the Repair Problem)**.** *A solution to $\mathcal{R}$ is a valid repair set δ that transforms the original domain $\mathcal{D}$ into a modified domain $\mathcal{D}'$ through the sequence of repair operations $\mathcal{D} \Rightarrow_\delta^* \mathcal{D}'$. This repair must satisfy the following: for every index i with $1 \leq i \leq n$, all positive plans $\pi^+ \in \mathbb{P}_i$ are executable in the updated planning task $\Pi_i' = (\mathcal{D}', \mathcal{T}_i)$, and each pair $(\pi_k^-, b_k) \in \mathbb{E}_i$ meets the condition that π_k^- is not a valid plan for Π_i', with the action at position b_k being the first that cannot be applied.*

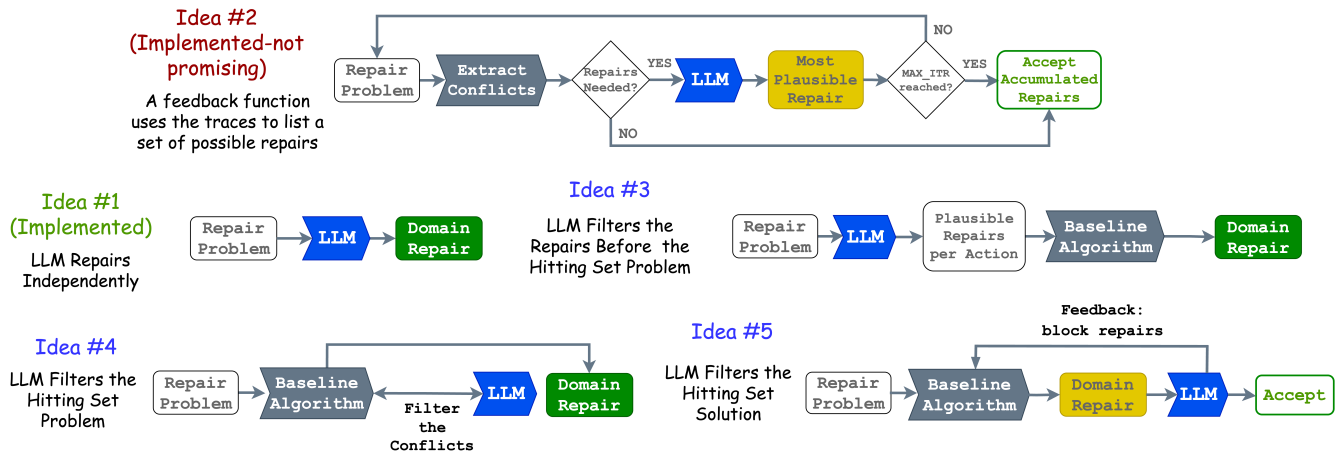

Figure 1: Different LLM strategies in domain repair to infuse semantic knowledge in search.

Solving the Repair Problem

Our baseline approach (Lin et al. 2025) proposes a sound algorithm based on conditional hitting sets to solve the domain repair problem. They report precision and recall against known ground truth repairs. To obtain ground truth, they perturb IPC domains by randomly adding or removing preconditions and effects, which allows direct computation of these metrics.

The Baseline Approach

Here, we briefly restate their approach at a level sufficient to motivate our LLM–based extensions. The algorithm executes all provided traces. For each positive trace, one of the unsatisfied preconditions (if any) encountered along the run in each trace is recorded as a *flaw*. For a negative trace (π_k^-, i_k), a flaw is recorded if the action at index i_k in π_k^- is applicable. If some earlier action at index $j < i_k$ is inapplicable, the trace is handled analogously to a positive one: repairs are generated so that the prefix becomes executable and the first inapplicable action occurs exactly at position i_k.

For each flaw, the algorithm enumerates a finite set of candidate repairs. The set of candidates for a single flaw is called a *conflict*; at least one element of the conflict must be selected to eliminate that flaw. Consider a missing positive precondition p_m for action a_3 in the sequence $\gamma = \langle a_1, a_2, a_3 \rangle$, and suppose a_1 has p_m in its negative effects. The repair candidates include: (i) remove the missing precondition from a_3, that is $r_1[\![\mathbf{a_3}, \mathbf{p_m}, \text{prec}^+, -]\!]$ for some ϱ so that $a_3 = \mathbf{a}_3[\varrho]$ and $p_m = \mathbf{p}_m[\varrho]$; (ii) add p_m to the positive effects of a_2, that is $r_2[\![\mathbf{a_2}, \mathbf{p_m}, \text{eff}^+, +]\!]$ whenever a matching ϱ such that $a_2 = \mathbf{a}_2[\varrho]$ and $p_m = \mathbf{p}_m[\varrho]$; exists; (iii) remove p_m from the negative effects of a_1, that is $r_3[\![\mathbf{a_1}, \mathbf{p_m}, \text{eff}^-, -]\!]$ for some ϱ so that $a_1 = \mathbf{a}_1[\varrho]$ and $p_m = \mathbf{p}_m[\varrho]$;. These give a conflict $\theta = \{r_1, r_2, r_3\}$.

For negative traces, conflicts are constructed so that the target action becomes inapplicable at its specified position. For example, one can add a new required precondition q that is false in the corresponding state, or remove earlier effects that would otherwise establish an existing precondition.

Different conflicts may interact. Some repairs may contradict one another, or jointly re–introduce earlier flaws. The baseline therefore augments the plain collection of conflicts to a more complex object Θ that encodes applicability conditions and mutual exclusions, producing a *conditional* hitting set instance. Note that the details of forming Θ is out of scope here. A solver then returns a minimal–cardinality hitting set, called a *diagnosis*. The algorithm iterates: it (1) extracts flaws from the current domain using the given positive and negative traces, (2) constructs Θ, (3) solves for a diagnosis, and (4) applies the corresponding repairs to obtain a modified domain. The process repeats until all positive traces succeed and all negative traces fail at their specified positions.

The baseline does not exploit semantic cues encoded by the modeller in predicate, action, or domain names within the PDDL file. Its hitting–set solver optimizes only the size of the repair set; when multiple diagnoses share the same cardinality, it returns an arbitrary one. Moreover, the ground truth repair need not be cardinality–minimal, so it can be missed under this objective.

LLM–Guided Search

We propose five strategies for using LLMs to steer the search toward human–plausible repairs, as Figure 1 shows. Because a planning domain can admits multiple semantically consistent fixes, precision and recall against the ground truths should be viewed as proxies for alignment with modeller intent rather than definitive correctness. Note that ideas 1 and 2 are implemented, but only Idea 1 is evaluated here. Ideas 3–5 are deferred to follow–up work.

Idea 1. This simple approach treats the LLM as a knowledge–engineering assistant. We specify the allowed repair operators, show the full domain to preserve global context, and ask the model to summarize the semantics it infers from action and predicate names. It then proposes a small set of repairs with brief rationales. This variant is implemented, and its results are reported in the next section.

Idea 2. Extract conflicts from action traces as in the base-

									Idea #1			Baseline		
Domain	Words	Lines	Tasks	POS-Sum	NEG-Sum	POS-Len	NEG-Len	Flaws	Repairs	Prec	Rec	Repairs	Prec	Rec
FLOORTILE	348	109	20	1	5	27.00	1.00	4	3.20	**1.00**	**0.75**	2.00	0.70	0.35
FREECELL	597	215	80	62	60	43.15	38.00	4	4.00	0.56	0.50	3.00	**0.73**	**0.55**
GED	784	310	20	20	9	14.30	8.67	10	4.00	0.13	0.06	4.00	**0.90**	**0.36**
HIKING	442	140	20	7	2	20.43	13.50	4	2.60	**0.73**	**0.45**	2.00	0.40	0.20
LOGISTICS00	282	99	28	28	1	47.54	1.00	4	3.60	**0.90**	**0.75**	2.00	0.50	0.25
LOGISTICS98	276	97	35	27	6	67.96	1.33	4	3.40	**0.85**	**0.70**	3.00	0.60	0.45
MPRIME	307	97	35	30	29	11.10	4.66	2	3.60	0.15	0.30	2.00	**1.00**	**1.00**
SCANALYZER	323	94	20	14	1	42.57	4.00	2	1.20	**0.90**	0.50	2.00	0.70	**0.70**
SLITHERLINK	410	133	20	1	2	19.00	14.00	2	2.40	**0.53**	**0.60**	2.00	0.50	0.50
SOKOBAN	239	77	20	2	3	40.50	17.67	2	2.80	0.43	0.60	2.00	**1.00**	**1.00**
TETRIS	477	144	17	5	5	24.60	20.80	4	3.20	0.29	0.20	2.00	**0.80**	**0.40**
THOUGHTFUL	1283	469	20	15	1	135.27	2.00	10	4.80	0.32	0.14	3.00	**0.53**	**0.16**
TIDYBOT	2157	656	20	4	5	29.25	19.20	12	3.60	0.60	**0.17**	2.00	**1.00**	**0.17**
WOODWORKING08	972	300	30	30	5	20.77	6.00	6	3.60	**0.73**	**0.40**	7.00	0.17	0.20
WOODWORKING11	976	302	20	7	1	62.29	9.00	6	2.80	**0.53**	0.23	3.00	**0.53**	**0.27**

Table 1: Experimental results comparing idea #1 with the baseline algorithm. Summary statistics over 5 runs per domain. **Words** and **Lines** count PDDL size; **Tasks** is the number of planning problems; **POS-Sum/NEG-Sum** are positive/negative plan counts; **POS-Len/NEG-Len** are their average lengths; **Flaws** and **Repairs** are detected issues and fixes; **Prec/Rec** denote precision and recall.

line, take their union, and iteratively ask the LLM to choose the most plausible repair, apply it to the domain, and repeat. Note that by taking union of the conflicts we are creating a long flat set of repairs, each of which fix at least one of the flaws found in the action traces. While we are not ready to report on its performance, preliminary tests suggest that the flat candidate pool can become large, which degrades LLM selection quality and makes this approach less promising.

Idea 3. Combine LLM proposals with minimality via the hitting–set solver. After exposing the full domain for context, we query the LLM for a list of plausible repairs at the action level. The hitting–set solver then selects a cardinality–minimal subset, ensuring that the final repairs are optimized toward consistency across conflicts. This separation allows the LLM to focus on local, per–action suggestions, avoiding confusion from reasoning over global repair sets, while the optimization stage consolidates them. This could improve the results compared to Idea 1, with the added benefit of shorter and more efficient prompts.

Idea 4. A symmetric decomposition to Idea 3 that operates at the level of conflicts rather than actions. For each flaw, the LLM filters or ranks its candidate repairs, and we keep conflict sets separate instead of forming a single union. This yields more focused prompts and allows us to test whether action–centric (Idea 3) or flaw–centric filtering is more natural for LLMs.

Idea 5. Use the LLM as a post–hoc judge. The baseline algorithm proposes a minimal repair; the LLM then accepts or vetoes it on semantic grounds and requests alternatives if needed. This mirrors the human–in–the–loop design of the VS Code plugin (Lin, Yousefi, and Bercher 2024) for the predecessor algorithm (Lin, Grastien, and Bercher 2023), with the LLM replacing the human reviewer.

Experiments

We use the same problem set as the baseline paper (Lin et al. 2025) to ensure a direct comparison, and we report the same metrics of precision and recall. Runtime results are omitted for now, as they are negligible (comparable to the latency of a single OpenAI API call) but will be included in future work. The baseline numbers in our tables are taken directly from the published paper; we did not re–run their implementation locally. Our experiments employ the model `gpt-4o-2024-08-06`, where the suffix denotes the training cutoff date.

Despite its simplicity, Idea 1 improves both precision and recall in roughly half of the benchmark domains. We also report word and line counts of the PDDL files as a proxy for domain complexity, although no clear correlation with LLM performance emerges. Crucially, Idea 1 does not exploit information from action traces, so we expect that incorporating our other proposed methods will further improve results across all domains. An additional insight concerns the number of repairs returned by the LLM: it is lower than the baseline in some domains and higher in others. When fewer repairs are produced, combining with the baseline is necessary, as any sound repair set necessarily has a cardinality greater than or equal to the minimum–cardinality set. When more repairs are produced, recall does not consistently improve over the baseline, which indicates unexploited information in the action traces. Trace availability and length vary by domain: some domains include 62 traces with an average length of 43 steps, whereas a different domain has a single trace of length 135. Given this sizes, we do not expect the LLM alone to exploit trace information reliably, and we believe integrating the LLM with hitting–set methods that consume traces is therefore essential for consistent gains.

Conclusion & Future Work

We introduced the use of large language models (LLMs) to exploit semantic cues in PDDL domains, guiding repairs toward solutions that are preferable for human modellers. Our approach leverages action and predicate names to improve semantic alignment. Among five proposed strategies, a simple single–shot prompt outperformed the baseline on half the benchmark domains. A limitation is that benchmarks may overlap with LLM training data; future work will test on unpublished domains for a more reliable evaluation.

Acknowledgements

Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

Bavandpour, N. K.; Lauer, P.; Lin, S.; and Bercher, P. 2025. Repairing Planning Domains Based on Lifted Test Plans. In *Proc. of the 28th ECAI*.

Bercher, P.; Sreedharan, S.; and Vallati, M. 2025. A Survey on Model Repair in AI Planning. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence (IJCAI 2025)*.

Caglar, T.; Belhaj, S.; Chakraborti, T.; Katz, M.; and Sreedharan, S. 2024. Can LLMs Fix Issues with Reasoning Models? Towards More Likely Models for AI Planning. In *Proc. of the 38th AAAI*, 20061–20069.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. In *Proc. of the 37th NeurIPS*.

Huang, S.; Cohn, T.; and Lipovetzky, N. 2024. Chasing Progress, Not Perfection: Revisiting Strategies for End-to-End LLM Plan Generation. *CoRR*.

Huang, S.; Lipovetzky, N.; and Cohn, T. 2025. Planning in the Dark: LLM-Symbolic Planning Pipeline Without Experts. In *Proc. of the 39th AAAI*, 26542–26550.

Katz, M.; Kokel, H.; Muise, C.; Sohrabi, S.; and Sreedharan, S. 2025. Make Planning Research Rigorous Again! *CoRR*, abs/2505.21674.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In *Proc. of the 37th AAAI*, 12022–12031.

Lin, S.; Grastien, A.; Shome, R.; and Bercher, P. 2025. Told You That Will Not Work: Optimal Corrections to Planning Domains Using Counter-Example Plans. In *Proc. of the 39th AAAI*, 26596–26604.

Lin, S.; Yousefi, M.; and Bercher, P. 2024. A Visual Studio Code Extension for Automatically Repairing Planning Domains. In *Demo at the 34th ICAPS*.

Miller, T. 2019. Explanation in artificial intelligence: Insights from the social sciences. *AIJ*, 267: 1–38.

Oswald, J. T.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large Language Models as Planning Domain Generators. In *Proc. of the 34th ICAPS*, 423–431.

Tantakoun, M.; Zhu, X.; and Muise, C. 2025. LLMs as Planning Modelers: A Survey for Leveraging Large Language Models to Construct Automated Planning Models. *CoRR*.