Evaluating Planning Model-Learning Algorithms

Anonymous submission

Abstract

Formulating domain models for model-based planning is a challenging, time consuming, and error-prone task. A number of approaches have been proposed to automatically learn domain models from a given set of observations. A key question is how to compare models learned by different approaches. Currently, there are no standard evaluation metrics or benchmarks. In this paper, we describe a set of metrics designed to assess different characteristics of a learned domain model. We then present a benchmark suite based on domain models from the International Planning Competition (IPC) and an evaluation process for using it. Four domain model learning algorithms are evaluated on this benchmark, which highlights the importance of the diverse evaluation metrics we proposed.

Introduction

Domain-independent planning is a foundational area of research in Artificial Intelligence (AI) that focuses on the automated generation of plans to achieve specific goals from a given initial state in a given environment. *Classical planning*, which is the focus of this work, is the well-studied type of domain-independent planning in which a single agent is acting in a fully observable, discrete, and deterministic environment. A crucial element of any planning system is the *domain model*, which in classical planning defines how states are represented, the set of possible actions, and the preconditions and effects of each action. Creating a domain model is a challenging, time-consuming, and error-prone task (McCluskey, Vaquero, and Vallati 2017), which is a bottleneck for the wider exploitation of planning technology in real-world applications.

To address this issue, a number of algorithms have been proposed to automatically learn domain models from a set of provided observations (Callanan et al. 2022; Aineto, Celorrio, and Onaindia 2019; Jiménez et al. 2012).[1] This task is often referred to as *domain model learning* or *model acquisition*. Despite a recent resurgence in interest in learning domain models, there is no set of agreed-upon evaluation metrics, no standard evaluation process for such algorithms, and no standard benchmark. This can be seen in Table 1, which lists the different metrics used by prior works. This paper aims to close this gap, and proposes an evaluation paradigm for domain model-learning algorithms that includes a set of metrics, a publicly-available benchmark for evaluation, and a detailed description of how to use it.

[1]See http://macq.planning.domains for more details.

A commonly used, straightforward evaluation process for domain model learning algorithms is based on comparing the *syntactic similarity* of the learned domain model to a *reference domain model*. We discuss the limitations of this evaluation method and propose, as our first contribution, an alternative evaluation process that does not require a reference domain model. This evaluation process aims to evaluate the *predictive power* and *problem-solving* ability of the learned domain model. Several specific metrics have been defined for this type of evaluation in prior works (Aineto, Celorrio, and Onaindia 2019; Juba, Le, and Stern 2021; Mordoch et al. 2024; Oswald et al. 2024), and we discuss their complementary strengths and weaknesses.

The second contribution of this work is an evaluation framework and benchmark suite based on domains from the International Planning Competition (IPC). This evaluation framework implements the evaluation process mentioned above on the introduced benchmark and outputs the proposed metric. We implemented this evaluation framework and apply it to evaluate several well-known domain model learning algorithms, namely Safe Action Model Learning (SAM) (Juba, Le, and Stern 2021), Offline Learning of Action Models (OffLAM) (Lamanna et al. 2025), Noisy Offline Learning of Action Models (NOLAM) (Lamanna and Serafini 2024), and ROSAME (Xi, Gould, and Thiébaux 2024). Our experimental results provide reference results to foster future research, as well as empirical evidence of the necessity for the different domain model evaluation metrics. The code, dataset, and evaluation process described in this work are intended to be publicly available.[2] Lastly, we briefly discuss how one may implement a more general evaluation framework that bridges over differences between how domain model learning algorithms represent the environment.

Background

A *classical planning* problem is a tuple $\mathcal{P} = \langle F, A, s_0, G \rangle$, where F is a finite set of fluents, A is a finite set of actions, s_0 is the initial state, and $G \subseteq F$ is a set of fluents. A *state* is defined by a set of propositions, representing that the fluents

[2]The code is available in the supplementary material.

Algorithm	**Syn. Sim.**	**Predict. Pow.**		**Prob. Solv.**	
		App.	**Eff.**	**Solv.**	**Fail**
ARMS (Yang, Wu, and Jiang 2007)	✓	✗	✗	✗	✗
SLAF (Amir and Chang 2008)(*)	✗	✗	✗	✗	✗
FAMA (Aineto, Celorrio, and Onaindia 2019)	✓	✓	✓	✗	✗
LOCM (Cresswell, McCluskey, and West 2013)	✗	(**)	(**)	✗	✗
OffLAM (Lamanna et al. 2025)	✓	✗	✗	✓	✓
NOLAM (Lamanna and Serafini 2024)	✓	✗	✗	✓	✓
SAM (Juba, Le, and Stern 2021)	(***)	✗	✗	✗	✗
Cond-SAM (Mordoch et al. 2024)	✓	✗	✗	✓	✗
ROSAME (Xi, Gould, and Thiébaux 2024)	✓	✗	✗	✗	✗

Table 1: Evaluation metrics used in prior works on domain model learning. These metrics are defined later in this paper. (*) Only reported runtime results. (**) Measured how many instances were required to "converged", i.e., reached where more data is not helpful, and whether the domain is isomorphic to a reference domain model. (***) Measured the number of trajectories needed until the reference domain model is learned.

in this set are true in this state and all other fluents are false. An action $a \in A$ is a tuple $a = \langle \mathit{pre}(a), \mathit{eff}(a) \rangle$, where $\mathit{pre}(a)$ is the precondition of a and $\mathit{eff}(a)$ is the effect of a. The precondition $\mathit{pre}(a)$ specifies the conditions that must hold in a state for the action a to be applicable. The effect $\mathit{eff}(a)$ specifies the changes to the state resulting from applying the action a. The precondition and effect of an action are defined each as a set of literals, which are either positive or negative propositions. An action a can be applied in states that include the positive literals in $\mathit{pre}(a)$ and do not include the negative literals in $\mathit{pre}(a)$. Applying a in a state s results in a state s' that includes all literals the positive literals in $\mathit{eff}(a)$ and all the literals in s except those appearing in a negative literal in $\mathit{eff}(a)$. Let L be the set of literals, i.e., $L = F \cup \{\neg f \mid f \in F\}$.

A solution to a classical planning problem is a sequence of actions that transforms the initial state s_0 into a goal state s_g where $G \subseteq s_g$.

Different algorithms have been proposed for learning classical planning domain models (Callanan et al. 2022; Aineto, Celorrio, and Onaindia 2019; Jiménez et al. 2012). The input to these algorithms is a set of *trajectories*. A *trajectory* is a sequence of observations and actions. An *observation* can be a state or some other information about the state, such as a set of predicates that hold in the state or a visual representation of a state. Domain model learning algorithms differ in the type of trajectories they learn from, the type of models they can learn, and the learning methodology they apply. In this work, we focus on arguably the most common type of planning domain model learning algorithms in which the domain model being learned is a classical planning domain, and the input trajectories are given in a symbolic representation. Examples of such learning algorithms include ARMS (Yang, Wu, and Jiang 2007), which models the learning problem as a MAX-SAT problem; Simultaneous Learning and Filtering (SLAF) (Amir and Chang 2008), which applies logical inference to filter inconsistent action models; FAMA (Aineto, Celorrio, and Onaindia 2019), which models the learning problem as a planning problem; and ROSAME (Xi, Gould, and Thiébaux 2024), which uses deep learning.

LOCM (Cresswell, McCluskey, and West 2013) and SIFT (Gösgens, Jansen, and Geffner 2024) represent a conceptually different type of domain model learning algorithm that rely on *action traces*, i.e., sequences of actions, without any knowledge of the underlying states. These algorithms learn a state representation that is based on the "state" of the objects that can be action parameters. Thus, it is not clear how can one apply them to states that were not observed in the training set without some mapping between an observed state and the internal representation of it according to LOCM/SIFT. Bridging this gap is a topic for future research.

Learning a domain model is somewhat related to *model reconciliation* (Chakraborti et al. 2017; Sreedharan et al. 2019, inter alia) and *model repair* (Bercher, Sreedharan, and Vallati 2025). Model reconciliation is the task of aligning one model with another model to "explain" the generated plans. The typically adopted metric is the number of changes to be made to align the models with the made plan observations, with the goal of minimizing modifications. Model repair is the task of modifying an initial model according to new types of constraints or observations. Domain model learning can be viewed as an extreme case of model repair where the initial model is empty. The nature of the repair task is different, however, and thus metrics and evaluation for model repair are expected to be different as well.

Problem Setting and Syntactic Similarity

We consider the following domain model learning setup. An agent is acting in an environment E that can be represented as a classical planning domain. The agent's actions are recorded in a set of trajectories. The observations and actions in the trajectories are given in a specific representation (action and fluent names and parameters), which we refer to as the *input representation*. A domain model learning algorithm is given this set of trajectories, and is expected to output a domain model for classical planning. That is, this domain model can be given as input to a classical planner, and together with an appropriate problem description, the planner can generate plans with it. The core question we consider is: **How good is the model learned by the domain model learning algorithm?**

Syntactic Similarity Metrics

Many prior works on learning domain models evaluated the learned model by syntactically comparing it to a *reference domain model* (or ground truth model), denoted by M^*. This type of evaluation relies on the assumption that a reference domain model is available and that it accurately captures the relevant aspects of the environment. This assumption is reasonable when evaluating domain model learning algorithms in some controlled environment, e.g., for research purposes. Under this assumption, several metrics have been proposed to quantify the *syntactic similarity* between the learned domain model and the reference domain model (Aineto, Celorrio, and Onaindia 2019; Mordoch et al. 2023; Xi, Gould, and Thiébaux 2024; Oswald et al. 2024). These syntactic similarity metrics typically compare the intersection or difference of the literals in the actions' preconditions and effects between the learned and reference domain models. We define these common metrics below. Let M be the evaluated domain model, a an action, and $\mathit{pre}_M(a)$ the preconditions of a according to M.

- True Positives: $\text{TP}_{\mathit{pre}}(a) = |(\mathit{pre}_M(a) \cap \mathit{pre}_{M^*}(a)|$
- False Positives: $\text{FP}_{\mathit{pre}}(a) = |((\mathit{pre}_M(a) \setminus \mathit{pre}_{M^*}(a)|$
- True Negatives: $\text{TN}_{\mathit{pre}}(a) = |L \setminus (\mathit{pre}_M(a) \cup \mathit{pre}_{M^*}(a))|$
- False Negatives: $\text{FN}_{\mathit{pre}}(a) = |(\mathit{pre}_{M^*}(a) \setminus \mathit{pre}_M(a))|$

The following standard metrics from statistical analysis can then be computed based on these values for each action:

- **Syntactic Precision**: $P_{\mathit{pre}}(a) = \frac{TP(a)}{TP(a)+FP(a)}$
- **Syntactic Recall**: $R_{\mathit{pre}}(a)\frac{TP(a)}{TP(a)+FN(a)}$

Other metrics, such as Accuracy and F1-score, can also be computed based on these values. To obtain an overall precision and recall for preconditions of the entire domain model, one can compute the average of the precision and recall values for all actions: $P_{\text{avg}} = \frac{1}{|A|}\sum_{a\in A} P(a)$ and $R_{\text{avg}} = \frac{1}{|A|}\sum_{a\in A} R(a)$, where $P(a)$ and $R(a)$ are the precision and recall of action a, respectively. The same metrics can be defined for the effects of actions, with the only difference being that the literals in the effects are used instead of those in the preconditions.

Example 1. *Consider an action* UNLOAD(ℓ, t, p)*, indicating that the truck t at location l unloads the package p. In the reference model, its preconditions are* $at(\ell, t)$ *and* $in(p, t)$*, and its effects are* $\neg in(p, t)$ *and* $at(\ell, p)$*. In the learned model it has the same effects, but with the preconditions* $at(\ell, t)$ *and* $at(\ell, p)$*. Then, its syntactic recall and precision is* 1 *for the effects and* $\frac{1}{2}$ *for the preconditions.*

A finer-grained variant of the syntactic similarity metric is to assess the *edit distance* of the learned domain model from the reference domain model (Chrpa et al. 2023). Low values indicate models that are syntactically close to each other. If two models are syntactically identical (i.e., edit distance of zero), then they are said to be *strongly equivalent*.

The syntactic similarity metrics have several limitations. First, they require the existence of a domain reference model. Such a model is rarely available when applying automated planning technology in real-world applications, where a PDDL specification of the environment is not given. This was also the setup in the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) (Chrpa et al. 2017). Second, comparing to a reference model implies there is a single best model for the environment. What constitutes a *best model* for an environment is not well-defined, and assessing the quality of a model is very challenging (McCluskey, Vaquero, and Vallati 2017). In fact, one may say that any domain model that is consistent with the observations is equally likely to be the "real one", as the learning algorithm is not given any incentive to prefer one over the other. Lastly, there may be multiple domain models for a given environment that are identical for all practical purposes, yet different syntactically. Thus, it is not clear whether the syntactic similarity of the learned model to a reference model is a good indicator of how *useful* a learned domain model is. Next, we discuss what constitutes a useful domain model and how to evaluate it without the need for a reference domain model.

Metrics for Evaluating Domain Models

We consider two main dimensions when evaluating the "usefulness" of a learned domain model:

- **Predictive power.** Aim to assess a domain model's ability to predict the applicability of actions in the environment and the outcomes of applying them.
- **Problem-solving ability.** Aim to assess a domain model's ability of generating applicable plans in the environment, within given resource bounds.

These dimensions may not be aligned with the syntactic similarity metrics, even if there exists a reference model that enables computing them. For example, if a domain maintains logical invariants on its states, such as mutex (mutual exclusion) conditions, that the reference model does not explicitly define. The learned model may include these invariants as additional preconditions that only rule out the action in environment states that are unreachable in actual trajectories. Similarly, a learned domain model may be very *similar syntactically* to a reference domain model but not effective in solving problems from the corresponding real-world environment. For example, the learned domain may miss a single crucial effect or include an extra precondition that cannot be met in a key state, which prevents it from solving many problems correctly. In contrast, a domain model may be very different from a reference domain model, e.g., adding many extraneous preconditions to some actions, yet very *effective* in solving problems in the application domain since these extraneous preconditions are often true in the real world.

The two introduced dimensions are also not necessarily aligned with each other. For example, consider a learned domain model that includes many copies of the same action from the reference domain model, each with different parameters and preconditions, in order to capture different aspects of that action's behavior. While this may be useful for predicting the applicability of actions in the environment, it may hinder the ability of a planner to find plans for problems in the application domain with this model, due to its complexity (e.g., large branching factor).

Next, we describe concrete metrics to quantify the usefulness of a learned domain model with respect to its predictive power and problem-solving ability. Computing these metrics do not require a reference domain model. Thus, they can be used to compare two models directly. However, computing these metrics requires that it is possible to attempt to perform actions in the environment E and observe if the action is applicable as well as the resulting state (if it is).

Predictive Power Metrics

The predictive power metrics, also referred to as *semantic* domain model metrics (Aineto, Celorrio, and Onaindia 2019; Mordoch et al. 2024; Le, Juba, and Stern 2024), measure how well a learned model is able to predict the applicability of actions and their effects in the environment.

We define two types of predictive power metrics: *action applicability* metrics and *predicted effects* metrics. The former measures the ability of the learned model to predict whether an action is applicable in a given state, while the latter measures the ability of the learned model to predict the effects of an applicable action in a given state. Computing the predictive power metrics requires a dataset of states that we denote by S_{test}. This dataset is intended to represent the distribution of states of interest. Thus, using S_{test} may actually be beneficial over using all the possible states, as planning domains can have many "unnatural" states (Grundke, Röger, and Helmert 2024) which may bias the evaluation. E.g., in Blockworld we can theoretically define that two blocks can be on top of each other, i.e., $\text{on}(a, b) \wedge \text{on}(b, a)$, but such a state does not make sense. Thus, if the learned model is inaccurate in such states, it may not be important. Obtaining a representative S_{test} is a challenge. One way to do so is by observing an agent acting in the environment, e.g., operated by an expert.

Predicted applicability For a domain model M and action a, we denote by $app_M(a, S_{test})$ and $app(a, S_{test})$ the set of states in S_{test} in which a is applicable according to M and E, respectively. Using this notation, we define the following predicted applicability metrics as follows for some action a:

- $\text{TP}_{app}(a) = |app_M(a, S_{test}) \cap app(a, S_{test})|$
- $\text{FP}_{app}(a) = |app_M(a, S_{test}) \setminus app(a, S_{test})|$
- $\text{TN}_{app}(a) = |S_{test} \setminus (app_M(a, S_{test}) \cup app(a, S_{test}))|$
- $\text{FN}_{app}(a) = |app(a, S_{test}) \setminus app_M(a, S_{test})|$

In words, $\text{TP}_{app}(a)$, $\text{FP}_{app}(a)$, $\text{TN}_{app}(a)$, and $\text{FN}_{app}(a)$ are the number of states in S_{test} where a is applicable in the environment and according to the learned model, a is not applicable in the environment but is applicable in the learned model, a is not applicable according to both environment and learned model, and a is applicable in the environment but inapplicable according to the learned model, respectively. From these metrics, one can compute the precision and recall for action applicability of every action a in the learned model as $P_{app}(a) = \frac{TP_{app}(a)}{TP_{app}(a)+FP_{app}(a)}$ and $R_{app}(a) = \frac{TP_{app}(a)}{TP_{app}(a)+FN_{app}(a)}$, respectively. When $TP_{app}(a) = FP_{app}(a) = 0$ it means the domain model never allowed a to be applied. We define $P_{app}(a) = 1$ and $R_{app}(a) = 0$ in such cases.

Predicted effects For domain M, action a, and state s, we denote by $a_M(s)$ and $a(s)$ the state resulting from applying a in s according to M and E, respectively. Based on this, we define the following predicted effect metrics for every state $s \in S_{test}$ and action a, as follows:

- $\text{TP}_{eff}(s, a) = |(a_M(s) \setminus s) \cap (a(s) \setminus s)|$
- $\text{FP}_{eff}(s, a) = |(a_M(s) \setminus s) \setminus a(s)|$
- $\text{TN}_{eff}(s, a) = |s \cap a_M(s) \cap a(s)|$
- $\text{FN}_{eff}(s, a) = |(a_M(s) \cap s) \setminus a(s)|$

For the purpose of the above computation, a state includes all the literals true in it, i.e., both positive propositions and negative ones. Importantly, we consider above only state-action pairs s and a where a is applicable in s in the environment and in the learned model. Otherwise, the outcome of applying a in s is not defined. To obtain the predicted effects metrics per action, we simply average over all the states in S_{test}. Similar to the predicted applicability metric, when $TP_{eff}(a) = FP_{eff}(a) = 0$, we define $P_{eff}(a) = 1$ and $R_{eff}(a) = 0$.

Aggregated Metrics The above precision and recall metrics are computed per action. It is often beneficial to aggregate over all actions and obtain precision and recall metrics for predicted applicability or predicted effects over the entire learned domain. One way to do so is to *average* the precision and recall obtained for each action. Alternatively, one may *sum* the TP, FP, TN, and FN separately, and then compute the precision and recall. That is, compute precision and recall based on $TP_{app} = \sum_a TP_{app}(a)$, $FP_{app} = \sum_a FP_{app}(a)$, $TN_{app} = \sum_a TN_{app}(a)$, and $FN_{app} = \sum_a FN_{app}(a)$. We refer to the result of the first aggregation method as the *average precision and recall* of the chosen metric (predicted applicability or predicted effects). We refer to the result of the second aggregation method as the *cumulative precision and recall* of the chosen metric. Both are reasonable options. In our evaluation framework described below, we used the average aggregation method.

Problem-Solving Metrics

Neither the syntactic similarity metrics nor the predictive power metrics are sufficient for evaluating the *operationality* (McCluskey, Vaquero, and Vallati 2017) of the learned domain model, i.e., its ability to solve problems. Even small syntactical changes in domain models can result in significant performance gaps (Vallati and Chrpa 2019; Vallati et al. 2021). We propose two metrics that are designed to measure a model's ability to solve problems: *solving ratio* and *false plan ratio*. Both metrics are defined with respect to a set of problems Π_{test} and a planner. The solving ratio metric is the fraction of problems in Π_{test} that can be solved with the learned model by the given planner within a fixed limit on the available computational resources — CPU runtime and memory. We say that a problem is *solved* if a plan is found within the allowed computational resources, and that executing this plan in the environment is possible and achieves the intended goals. The false-plan-ratio metric is defined as the fraction of problems in Π_{test} that are solved by the learned model but cannot be executed in the environment or do not achieve the intended goals. This reflects the reliability of the learned model for producing plans. Additional problem-solving metrics can also be considered that quantify the runtime and solution quality of returned solutions.

The straightforward nature of the above metrics is not without limitations. The ability to solve a problem with a given domain depends on external factors such as the set of test problems, the planner used, the runtime and memory budget allowed for it to run, and the computer and OS that executed the planner. Ideally, the above metrics would be run on a diverse set of test problems, planners, resource limits, and computing machines. In practice, this might be difficult to implement, but one is advised to at least run all evaluated domains on the same setup and provide an appropriate disclaimer to the concluded result.

Benchmarks and an Evaluation Process

Domain	$\|A\|$	$\|P\|$	Types	Const.	A arity		P arity		Del.	Inj.
					min	max	min	max	pre.	ass.
barman	12	15	9	no	2	6	1	2	no	yes
blocksworld	4	5	1	no	1	2	0	2	no	yes
childsnack	6	13	6	yes	2	4	1	2	no	no
depots	5	6	9	no	3	4	1	2	no	yes
elevators	6	8	5	no	3	5	2	2	no	yes
ferry	3	5	2	no	2	2	0	2	no	yes
floortile	7	10	3	no	3	4	1	2	no	yes
goldminer	7	12	1	no	1	2	0	2	yes	no
grippers	3	4	4	no	3	4	2	3	no	yes
matchingbw	10	10	2	no	2	3	1	2	yes	yes
miconic	4	6	2	no	2	2	1	2	no	yes
nomystery	3	6	5	no	3	6	2	3	no	yes
npuzzle	1	3	2	no	3	3	1	2	no	yes
parking	4	5	2	no	3	3	1	2	no	yes
rovers	9	25	7	no	2	6	1	3	no	no
satellite	5	8	4	no	2	4	1	2	yes	yes
sokoban	2	4	3	no	3	5	1	3	no	yes
spanner	3	6	5	no	3	4	1	2	no	yes
tpp	4	7	7	no	3	7	2	3	no	no
transport	3	5	6	no	3	5	2	2	no	yes

Table 2: Details about the benchmark domains that are relevant to domain model learning algorithms. Columns from left to right: domain name; number of lifted actions, lifted predicates , and types; if there are constants; the min. and max. arity of the actions; the min. and max. arity of the predicates; if a literal can be a delete effects without being a preconditions; and if the injective binding assumption holds (Juba, Le, and Stern 2021).

As a standard benchmark, we propose to use the set of 20 classical planning domains adopted in all previous IPC learning tracks. These domains serve as the environment and as a reference domain model if one wishes to compute the syntactic similarity metrics. These domains are well known in the planning community and have available problem generators. The number of lifted actions, predicates, and object types in these domains varies in $[1, 12]$, $[3, 25]$, and $[1, 9]$, respectively. See Table 2 for further details about the domains.

Step 1: Creating the training data. The first step in our evaluation process is to obtain training data T_{train}, i.e., trajectories recording interactions with the environment. In our evaluation process, we generate these training trajectories by simulating the behavior of an agent in the environment, mixing goal-oriented and exploration-oriented behavior, as follows. First, we generate feasible problems using existing generators (Seipp, Torralba, and Hoffmann 2022). Then, we solve every generated problem using a planner with the reference domain model (from the IPC). The agent then begins to follow the generated plan, but interleaves a random action with some probability p_{rnd}. After executing a random action, a new plan is computed by the planner from the resulting state (using the reference domain model). If the random action execution leads to an unsolvable state, we backtrack and perform a different random action.

To generate diverse goal-oriented behavior, we use a greedy planner configuration for some problems and an optimal configuration for others. Let p_{opt} be the ratio of problems in which the optimal configuration is used. Specifically, we adopted the FastDownward planner (Helmert 2006) with lazy greedy best-first search, the FF heuristic (Hoffmann and Nebel 2001), and the context-enhanced additive heuristic (Helmert and Geffner 2008) for the greedy configuration, and A^* with LM cut (Karpas and Domshlak 2009) for the optimal configuration.

To emphasize the importance of using both optimal and suboptimal planners to generate trajectories, consider the following example. In the BARMAN domain, an agent can either CLEAN a previously used shot and then FILL it (which requires 2 actions), or just REFILL a used shot (which requires only 1 action); while both alternatives are possible in an heuristic plan, the REFILL action must be executed in an optimal one. Thus, generating both optimal and suboptimal plans increases the likelihood of "sampling" more diverse set of actions. Finally, to produce heterogeneous trajectories, we also generated trajectories from problems with different numbers of objects of each type.[3] For our benchmark, we generated 10 trajectories using the above process with $p_{rnd} = 0.2$ and $p_{opt} = 0.3$. The obtained trajectories (T_{train}) includes every lifted action at least once, and every trajectory includes between 5 to 45 states and 3 to 107 objects.

Step 2: Running the evaluated learning algorithm. Next, we run the evaluated domain model learning algorithms with the generated training set T_{train}, obtaining the learned domain model M. For completeness, we also compute at this step the syntactic similarity metrics w.r.t the reference domain from the IPC, despite their inherent limitations, as they are common in the literature.

Step 3: Computing the predictive power metrics. We create a set of trajectories T_{test} similar to how T_{train} were created. First we generate problems in the environment and solve a ratio $p_{opt} = 0.3$ of them optimally with the reference domain model and planning configurations used to generate T_{train}. Then, we created trajectories by simulating an agent that alternates between performing actions in the plan and random actions with $p_{rnd} = 0.2$. For our benchmark, we generated 100 such trajectories. All the states in these trajectories are used as the set of test states (S_{test}) for computing the predictive power metrics.

Step 4: Computing the problem-solving metrics. We generated 10 problems for every domain and included in Π_{test} the problems that FastDownward could solve using the greedy configuration with the reference domain model, within a time limit of 60 seconds and 16 GB of RAM. The resulting set of problems is then used to compute the problem-solving metrics using the same planner with the learned models and validating the produced plans through the default plans validator provided in Unified Planning (Micheli et al. 2025) .

Experimental Results

We run our evaluation process to four state-of-the-art domain model learning algorithms, namely SAM (Juba, Le,

[3]An exception to this is the NPUZZLE domain, where there is a single object type, and increasing it leads to problems that are too difficult to be solved.

	Syntactic $P\uparrow$				Syntactic $R\uparrow$			
Domain	OffLAM	SAM	ROSAME	NOLAM	OffLAM	SAM	ROSAME	NOLAM
barman	**0.95**	0.54	0.84	0.53	**1.00**	**1.00**	0.89	**1.00**
blocksworld	**1.00**	0.66	**1.00**	0.58	**1.00**	**1.00**	**1.00**	**1.00**
childsnack	**0.97**	0.63	**0.97**	0.66	0.86	**0.90**	0.86	0.86
depots	**0.98**	0.71	**0.98**	0.67	**1.00**	**1.00**	**1.00**	**1.00**
elevators	**0.81**	0.50	0.70	0.36	**1.00**	**1.00**	**1.00**	**1.00**
ferry	**0.93**	0.75	**0.93**	0.62	**1.00**	**1.00**	**1.00**	**1.00**
floortile	**0.83**	0.41	0.73	0.31	**1.00**	**1.00**	0.90	**1.00**
goldminer	**0.80**	0.39	0.56	0.36	**0.98**	**0.98**	0.81	**0.98**
grippers	**1.00**	0.77	0.97	0.77	**1.00**	**1.00**	0.97	**1.00**
matchingbw	**0.95**	0.60	0.79	0.54	**1.00**	**1.00**	0.78	**1.00**
miconic	**1.00**	0.70	**1.00**	0.56	**1.00**	**1.00**	**1.00**	**1.00**
nomystery	**0.94**	0.66	**0.94**	0.50	**1.00**	**1.00**	**1.00**	**1.00**
npuzzle	0.88	0.70	**0.88**	0.58	**1.00**	**1.00**	**1.00**	**1.00**
parking	**0.89**	0.60	0.87	0.53	**1.00**	**1.00**	0.81	**1.00**
rovers	**0.83**	0.61	0.61	0.51	**0.88**	**0.88**	0.78	**0.88**
satellite	**1.00**	0.75	**1.00**	0.75	**1.00**	**1.00**	0.96	**1.00**
sokoban	0.88	0.57	**0.89**	0.48	**1.00**	**1.00**	**1.00**	**1.00**
spanner	**0.93**	0.71	**0.93**	0.60	**1.00**	**1.00**	**1.00**	**1.00**
tpp	**0.95**	0.29	0.39	0.34	**1.00**	0.78	0.49	0.89
transport	**0.93**	0.71	**0.93**	0.55	**1.00**	**1.00**	**1.00**	**1.00**

Table 3: Syntactic similarity precision (P) and recall (R). .

	Predicted applicability $P\uparrow$				Predicted applicability $R\uparrow$			
Domain	OffLAM	SAM	ROSAME	NOLAM	OffLAM	SAM	ROSAME	NOLAM
barman	**1.00**	**1.00**	0.90	**1.00**	**0.95**	0.92	0.85	0.92
blocksworld	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
childsnack	0.92	**1.00**	0.90	0.92	**1.00**	0.86	**1.00**	0.86
depots	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.96	**1.00**	0.96
elevators	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.87	0.60	0.87
ferry	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
floortile	**1.00**	**1.00**	0.96	**1.00**	**1.00**	0.93	0.89	0.93
goldminer	**1.00**	**1.00**	0.84	**1.00**	**1.00**	0.98	0.82	0.98
grippers	**1.00**	**1.00**	0.91	**1.00**	**1.00**	0.92	**1.00**	0.92
matchingbw	**1.00**	**1.00**	0.87	**1.00**	**0.93**	**0.93**	0.69	**0.93**
miconic	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
nomystery	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.67	**1.00**	0.67
npuzzle	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
parking	**1.00**	**1.00**	0.85	**1.00**	**1.00**	0.89	0.90	0.89
rovers	**1.00**	**1.00**	0.92	**1.00**	**1.00**	0.75	0.67	0.77
satellite	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.77	**1.00**	0.77
sokoban	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
spanner	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
tpp	**1.00**	**1.00**	0.61	**1.00**	**1.00**	0.27	0.65	0.29
transport	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 4: Predicted applicability precision (P) and recall (R).

and Stern 2021), OffLAM (Lamanna et al. 2025), NOLAM (Lamanna and Serafini 2024), and ROSAME (Xi, Gould, and Thiébaux 2024). Highlights of the results of our evaluation are described below. For a more detailed analysis see the supplementary material. Due to the stochasticity of ROSAME, we run it with 5 different random seeds and report the average results. All the experiments were run on a CPU Apple M1 Pro with 16 GB of RAM.

Syntactic similarity. The syntactic precision and recall of the effects was one for most algorithms and domains. The syntactic similarity results for preconditions, presented in Table 3 were more interesting. In all our tables, we highlighted in bold the best results in each domain and metric. As can be seen, in general OffLAM and ROSAME yielded the best results in terms of syntactic precision while OffLAM, NOLAM, and SAM provided the best syntactic recall. The low syntactic precision (< 0.8) of SAM and NOLAM is understandable as they allow negative preconditions, which are not needed in any domain, while OffLAM and ROSAME assume such preconditions do not exist. ROSAME's low syntactic recall in some domains is likely due to its design to address noisy observations, which we do not currently consider in our training trajectories. ROSAME's standard deviation for syntactic recall is 0 in half the domains and ranges from 0.01 to 0.16 in other domains; similarly for other metrics, showing ROSAME performance does not significantly vary across runs. Also worth noting are the CHILDSNACK results, where SAM outperformed all others in terms of syntactic recall. This is because it is currently the only algorithm that supports constants, which are used in this domain. SAM advantage in this domain is evident in most other metrics, for the same reason.

Predicted applicability. Table 4 shows the predicted applicability results. As can be seen, despite poor syntactic precondition precision results both SAM and NOLAM provide almost perfect predicted applicability. This provides empirical evidence of the syntactic metrics' drawback: the applicability of an action is not necessarily affected by the syntactic precision and recall of its preconditions. For example, in domain TPP, the applicability precision achieved by NOLAM equals 1, despite the preconditions syntactic recall equal to

	Predicted effects $P\uparrow$				Predicted effects $R\uparrow$			
Domain	OffLAM	SAM	ROSAME	NOLAM	OffLAM	SAM	ROSAME	NOLAM
barman	**1.00**	**1.00**	0.94	**1.00**	**1.00**	**1.00**	0.95	**1.00**
blocksworld	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
childsnack	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
depots	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
elevators	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
ferry	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
floortile	**1.00**	**1.00**	0.98	**1.00**	**1.00**	**1.00**	0.97	**1.00**
goldminer	**1.00**	**1.00**	0.84	**1.00**	**1.00**	**1.00**	0.90	**1.00**
grippers	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
matchingbw	**1.00**	**1.00**	0.98	**1.00**	**1.00**	**1.00**	0.84	**1.00**
miconic	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
nomystery	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
npuzzle	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
parking	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.86	**1.00**
rovers	**1.00**	**1.00**	0.93	**1.00**	**1.00**	**1.00**	0.92	**1.00**
satellite	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
sokoban	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
spanner	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
tpp	**1.00**	**1.00**	0.64	**1.00**	**1.00**	**1.00**	0.57	**1.00**
transport	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 5: Predicted effects precision (P) and recall (R).

80%; similarly for SAM in CHILDSNACK, and ROSAME in BARMAN, DEPOTS and MATCHINGBW.

Predicted effects. Table 5 shows the predicted effects results. All algorithms except ROSAME achieved perfect precision and recall across all domains. By design, SAM and OffLAM provides effects syntactic precision equal to 1, which also holds for NOLAM as the trajectories are not noisy, since an action effect is learned only when a literal becomes true/false after observing such action execution in some trajectory transition. This prevents SAM and OffLAM from learning unnecessary effects, thus making the false positives for the predicted effects equal to 0 and the precision equal to 1. Interestingly, in domain FLOORTILE, TPP, and MATCHINGBW, the predicted effects precision achieved by ROSAME equals 1 despite the syntactic precision being lower than 1 for both positive and negative effects. This is due to some actions with unnecessary effects (which lowers the syntactic precision) and inconsistent preconditions; indeed, actions with inconsistent preconditions are never executable according to both the environment and learned model, which prevents such actions to be considered when measuring the predicted effects metric. Similarly, when an action has inconsistent preconditions and missing effects,

	Solv. % ↑				False % ↓			
Domain	OffLAM	SAM	ROSAME	NOLAM	OffLAM	SAM	ROSAME	NOLAM
barman	**1.00**	0.80	0.04	0.80	**0.00**	**0.00**	0.38	**0.00**
blocksworld	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**
childsnack	0.00	**1.00**	0.00	0.00	1.00	**0.00**	1.00	1.00
depots	**1.00**	**1.00**	0.80	**1.00**	**0.00**	**0.00**	0.20	**0.00**
elevators	**1.00**	0.10	0.10	0.10	**0.00**	**0.00**	**0.00**	**0.00**
ferry	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**
floortile	**1.00**	0.90	0.02	0.90	**0.00**	**0.00**	0.40	**0.00**
goldminer	0.00	**1.00**	0.00	**1.00**	1.00	**0.00**	0.60	**0.00**
grippers	**1.00**	**1.00**	0.60	**1.00**	**0.00**	**0.00**	0.40	**0.00**
matchingbw	**1.00**	**1.00**	0.00	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**
miconic	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**
nomystery	**1.00**	0.00	**1.00**	0.00	**0.00**	**0.00**	**0.00**	**0.00**
npuzzle	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**
parking	**1.00**	0.90	0.18	0.90	**0.00**	**0.00**	0.20	**0.00**
rovers	**1.00**	0.20	0.00	0.30	**0.00**	**0.00**	0.04	**0.00**
satellite	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**
sokoban	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**
spanner	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**
tpp	**1.00**	0.00	0.00	0.00	**0.00**	**0.00**	0.12	**0.00**
transport	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**0.00**	**0.00**	**0.00**

Table 6: Problem solving (Solv.%) and false plan ratio (False %).

the predicted effects recall keeps unchanged whereas the effects syntactic recall is lower than 1. This is the case for SAM in domain TPP, and ROSAME in domains BARMAN, FLOORTILE, PARKING and ROVERS.

Problem-solving. OffLAM achieved the highest solving ratio across all domains except for CHILDSNACK and GOLDMINER. Notably, in TPP, it is the only algorithm to produce a model capable of solving the entire test set of problems. The difficulty in solving problems for SAM and NOLAM is mainly due to the learned domains having unnecessary negative preconditions. As expected, the plans produced with the domains learned by SAM are never false plans; this is guaranteed by the SAM safety property (Juba, Le, and Stern 2021). In the CHILDSNACK domain, both OffLAM and NOLAM produced inapplicable plans (i.e., the false plans ratio equals 1) due to a missing precondition involving the unsupported domain constant. In the GOLDMINER domain, the plans produced with the models learned by OffLAM are inapplicable due to a missing negative effect that was never observed in the input trajectories, which affects the preconditions of subsequent plan actions. Interestingly, SAM and NOLAM, learned such unobserved negative effect as a negative precondition, which allows computing valid plans. It is worth noting that, the SAM solving ratio in domains GOLDMINER and CHILDSNACK equals 1 despite the syntactic precision of 0.39 and 0.63, respectively, and the syntactic recall being lower than 1; similarly for NOLAM in domain GOLDMINER and OffLAM in domain ROVERS. Moreover, the solving ratio of OffLAM is 0 in domain CHILDSNACK despite the syntactic precision and recall higher than 0.8, i.e. 0.97 and 0.86, respectively; similarly for ROSAME in 5 out of 20 domains. These results provide clear evidence of the syntactic similarity drawbacks previously discussed.

Results summary. Our results indicate that the proposed metrics allow to capture the strengths of learning algorithms. For example, SAM provides the best applicability precision and false plans ratio, while OffLAM achieves the best applicability recall and solving ratio, and SAM, OffLAM, and NOLAM provide the same performance in terms of predicted effects precision and recall. While in general ROSAME provided weaker results, we emphasize that ROSAME is designed to handle noisy observations, while currently in our evaluation framework observations are noise-free. Moreover, we did not perform any hyper-parameter tuning, which is often crucial in deep learning based approach. Hyper-parameter tuning requires a validation set, and it is unclear how to correctly split the training set accordingly in our setting.

Discussion

We explored three families of metrics: syntactic similarity, predictive power, and problem-solving ability. There is an interesting trade-off between these families of metrics. The syntactic similarity is the easiest to compute, as it only requires comparing the learned domain with a reference domain. However, they require a reference domain model and, as discussed above, are arguably the least useful. The predictive power metrics characterize well a desirable property of a domain model, and are still relatively easy to compute. The downside of these metrics is that in contrast to classical machine learning settings, the "data distribution" in planning is highly *non-stationary* (not "i.i.d."): Although the states in the benchmarks were selected to be representative of those visited on trajectories collected using the reference model, this may change. Indeed, the states encountered on a trajectory generated by a planner using a given learned domain-model representation depend on the model. Planners can exploit deficiencies in the learned representation to systematically visit states that are erroneously modeled, if those errors make goals easier to achieve. Thus, the predictive power metrics can be a poor proxy for the utility of the learned model for planning. The problem-based metrics fill in this gap, but, of course, they are the hardest to compute as they require running a planner to compute them.

Conclusion and Future Work

To support the evaluation of model learning approaches, we proposed a comprehensive evaluation process and a suite of metrics. Our framework includes three complementary families of metrics: syntactic similarity, predictive power, and problem-solving ability. We analysed the strengths and limitations of each metric family and how they jointly capture different aspects of model quality, and we described our implementation of an evaluation process. Empirical evaluation of four state-of-the-art domain model learning algorithms on our proposed benchmark based on domains from the IPC learning track reveals that each algorithm exhibits distinct strengths and weaknesses. The proposed benchmark and evaluation framework are publicly available, providing a foundation for systematic and reproducible research in this area. Future work will focus on extending the benchmarks and adapting the evaluation metrics to settings involving online or incremental learning (Lamanna et al. 2021; Sreedharan and Katz 2023; Benyamin et al. 2025; Ng and Petrick 2019; Chitnis et al. 2021; Jin et al. 2022; Verma, Karia, and Srivastava 2023; Karia et al. 2023).

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Amir, E.; and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33: 349–402.

Benyamin, Y.; Mordoch, A.; Shperberg, S. S.; and Stern, R. 2025. Integrating Reinforcement Learning, Action Model Learning, and Numeric Planning for Tackling Complex Tasks. arXiv:2502.13006.

Bercher, P.; Sreedharan, S.; and Vallati, M. 2025. A Survey on Model Repair in AI Planning. In *International Joint Conference on Artificial Intelligence*.

Callanan, E.; Venezia, R. D.; Armstrong, V.; Paredes, A.; Chakraborti, T.; and Muise, C. 2022. MACQ: A Holistic View of Model Acquisition Techniques. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proc. of IJCAI*, 156–163. IJCAI.

Chitnis, R.; Silver, T.; Tenenbaum, J. B.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Glib: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling. In *AAAI Conference on Artificial Intelligence*, 11782–11791.

Chrpa, L.; Dodaro, C.; Maratea, M.; Mochi, M.; and Vallati, M. 2023. Comparing Planning Domain Models Using Answer Set Programming. In *European Conference on Logics in Artificial Intelligence*, 227–242.

Chrpa, L.; McCluskey, T. L.; Vallati, M.; and Vaquero, T. 2017. The Fifth International Competition on Knowledge Engineering for Planning and Scheduling: Summary and Trends. *AI Mag.*, 38(1): 104–106.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

Gösgens, J.; Jansen, N.; and Geffner, H. 2024. Learning Lifted STRIPS Models from Action Traces Alone: A Simple, General, and Scalable Solution. *arXiv preprint arXiv:2411.14995*.

Grundke, C.; Röger, G.; and Helmert, M. 2024. Formal Representations of Classical Planning Domains. In *International Conference on Automated Planning and Scheduling (ICAPS'24')*, 239–248.

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

Helmert, M.; and Geffner, H. 2008. Unifying the Causal Graph and Additive Heuristics. In *ICAPS*, 140–147.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Jiménez, S.; De la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review*, 27(4): 433–467.

Jin, M.; Ma, Z.; Jin, K.; Zhuo, H. H.; Chen, C.; and Yu, C. 2022. Creativity of AI: Automatic symbolic option discovery for facilitating deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 7042–7050.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 379–389.

Karia, R.; Verma, P.; Vipat, G.; and Srivastava, S. 2023. Epistemic Exploration for Generalizable Planning and Learning in Non-Stationary Stochastic Settings. In *NeurIPS 2023 Workshop on Generalization in Planning*.

Karpas, E.; and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1728–1733.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; and Traverso, P. 2021. Online Learning of Action Models for PDDL Planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4112–4118.

Lamanna, L.; and Serafini, L. 2024. Action Model Learning from Noisy Traces: a Probabilistic Approach. In *ICAPS*, 342–350.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A. E.; and Traverso, P. 2025. Lifted action models learning from partial traces. *Artificial Intelligence*, 339.

Le, H. S.; Juba, B.; and Stern, R. 2024. Learning Safe Action Models with Partial Observability. In *AAAI Conference on Artificial Intelligence*, 20159–20167.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *Proceedings of the Knowledge Capture Conference, K-CAP*, 14:1–14:8.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29: 102012.

Mordoch, A.; Scala, E.; Stern, R.; and Juba, B. 2024. Safe learning of pddl domains with conditional effects. In *International Conference on Automated Planning and Scheduling*, volume 34, 387–395.

Mordoch, A.; Stern, R.; Scala, E.; and Juba, B. 2023. Safe Learning of PDDL Domains with Conditional Effects. In *Reliable Data-Driven Planning and Scheduling (RDDP) workshop in ICAPS*.

Ng, J. H. A.; and Petrick, R. P. 2019. Incremental Learning of Planning Actions in Model-Based Reinforcement Learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 3195–3201.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large Language Models as Planning Domain Generators. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 423–431. AAAI Press.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL Generators. https://doi.org/10.5281/zenodo.6382173.

Sreedharan, S.; Hernandez, A. O.; Mishra, A. P.; and Kambhampati, S. 2019. Model-Free Model Reconciliation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 587–594.

Sreedharan, S.; and Katz, M. 2023. Optimistic exploration in reinforcement learning using symbolic model estimates. *Advances in Neural Information Processing Systems*, 36: 34519–34535.

Vallati, M.; and Chrpa, L. 2019. On the Robustness of Domain-Independent Planning Engines: The Impact of Poorly-Engineered Knowledge. In *International Conference on Knowledge Capture (K-CAP)*, 197–204.

Vallati, M.; Chrpa, L.; McCluskey, T. L.; and Hutter, F. 2021. On the importance of domain model configuration for automated planning engines. *Journal of Automated Reasoning*, 65(6): 727–773.

Verma, P.; Karia, R.; and Srivastava, S. 2023. Autonomous capability assessment of sequential decision-making systems in stochastic settings. *Advances in Neural Information Processing Systems*, 36: 54727–54739.

Xi, K.; Gould, S.; and Thiébaux, S. 2024. Neuro-Symbolic Learning of Lifted Action Models from Visual Traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 653–662.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.

Reproducibility Checklist

1. General Paper Structure

1.1. Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes/partial/no/NA) yes

1.2. Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes/no) yes

1.3. Provides well-marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper (yes/no) yes

2. Theoretical Contributions

2.1. Does this paper make theoretical contributions? (yes/no) no

If yes, please address the following points:

2.2. All assumptions and restrictions are stated clearly and formally (yes/partial/no)

2.3. All novel claims are stated formally (e.g., in theorem statements) (yes/partial/no)

2.4. Proofs of all novel claims are included (yes/partial/no)

2.5. Proof sketches or intuitions are given for complex and/or novel results (yes/partial/no)

2.6. Appropriate citations to theoretical tools used are given (yes/partial/no)

2.7. All theoretical claims are demonstrated empirically to hold (yes/partial/no/NA)

2.8. All experimental code used to eliminate or disprove claims is included (yes/no/NA)

3. Dataset Usage

3.1. Does this paper rely on one or more datasets? (yes/no) yes

If yes, please address the following points:

3.2. A motivation is given for why the experiments are conducted on the selected datasets (yes/partial/no/NA) yes

3.3. All novel datasets introduced in this paper are included in a data appendix (yes/partial/no/NA) yes

3.4. All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no/NA) yes

3.5. All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations (yes/no/NA) yes

3.6. All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available (yes/partial/no/NA) yes

3.7. All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing (yes/partial/no/NA) yes

4. Computational Experiments

4.1. Does this paper include computational experiments? (yes/no) yes

If yes, please address the following points:

4.2. This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting (yes/partial/no/NA) NA

4.3. Any code required for pre-processing data is included in the appendix (yes/partial/no) yes

4.4. All source code required for conducting and analyzing the experiments is included in a code appendix (yes/partial/no) yes

4.5. All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/-partial/no) yes

4.6. All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes/-partial/no) yes

4.7. If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results (yes/partial/no/NA) yes

4.8. This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks (yes/partial/no) yes

4.9. This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics (yes/partial/no) yes

4.10. This paper states the number of algorithm runs used to compute each reported result (yes/no) yes

4.11. Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information (yes/no) yes

4.12. The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank) (yes/partial/no) no

4.13. This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments (yes/partial/no/NA) NA