Evaluating Planning Model Learning Algorithms Across Different Representations

Anonymous Authors

Abstract

Automated planning is a prominent approach to sequential decision-making. A crucial aspect of domain-independent planning is the domain model, which provides to a planning engine the necessary application knowledge needed to synthesize solution plans. A domain model typically includes a state representation and an action model, which defines the set of possibly actions and the preconditions and effects of each action. Formulating domain-models is a challenging, time consuming, and error-prone task. For this reason, a number of approaches have been proposed to automatically learn complete (or partial) domain models from a set of provided observations. This raises the question of how to compare models learned by different approaches; there is a lack of evaluation metrics, and the complexity is exacerbated since learning algorithms may output models with different representations of states and actions. To foster the use of model learning approaches, in this paper we describe a set of metrics designed to assess different characteristics of models to be compared. Further, to bridge the potential representation gap between different learned models, we propose an encoder-decoder mechanism that allows to map two models regardless of their encoding, and we demonstrate how this encoder-decoder mechanism can be leveraged on to systematically compare models using the proposed metrics. Finally, suggest a benchmark suite based on existing domain models from the International Planning Competition (IPC) and an evaluation process for using it.

1 Introduction

Domain-independent planning is a foundational area of research in Artificial Intelligence (AI) that focuses on the automatic 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 colloquial name for a well-studied type of domain-independent planning in which a single agent is acting in a fully observable, discrete, and deterministic environment. Most research on classical planning has focused on developing efficient algorithms for solving planning problems, and assumed the existence of a *domain model* specified in a formal language such as the Planning Domain Definition Language (PDDL) (Haslum et al., 2019). The domain model in classical planning defines how states are represented, the set of possible actions, and the preconditions and effects of each action. However, creating a domain model is a challenging, time-consuming, and error-prone task (McCluskey, Vaquero, and Vallati, 2017). This is a bottleneck for the wider dissemination 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*, *model acquisition*, and *action model learning*. Despite a recent resurgence in interest in learning domain models, there is no standard evaluation process for such algorithms, no set of agreed-upon evaluation metrics, and no standard benchmark. This paper aims to close this gap, and proposes an evaluation paradigm for domain model learning algorithms, a set of metrics, and a publicly available benchmark for evaluation.

We begin our exposition by describing a straightforward evaluation process for action model learning algorithms, which 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 an alternative evaluation process that aims to evaluate the *predictive power* and *problem-solving* ability of the learned domain model. Several specific evaluation metrics are defined for this type of evaluation based on prior works (Aineto, Celorrio, and Onaindia, 2019; Juba, Le, and Stern, 2021; Mordoch et al., 2024; Oswald et al., 2024), and we discuss their strengths and weaknesses.

The proposed metrics are designed for evaluating models that use the same representation of states and actions. However, in practice, action model learning algorithms may output models that use different representations of states and actions. For example, some algorithms output models in a grounded representation (Stern and Juba, 2017), while others output models in a lifted representation (Aineto, Celorrio, and Onaindia, 2019; Juba, Le, and Stern, 2021; Xi, Gould, and Thiébaux, 2024). Some represent states based only the parameters of executed actions (Cresswell and Gregory, 2011), or different languages to support human engineers (McCluskey et al., 2010). Finally, some require a richer symbolic representation of states (Juba, Le, and Stern, 2021), while others use a visual representation of

[1] A comprehensive list of such algorithms is listed in https://macq.planning.domains.

states (Asai and Fukunaga, 2018; Xi, Gould, and Thiébaux, 2024). [Roni: All: add references and refine the above and add more examples of representation gap.] We propose an evaluation paradigm that addresses this representation gap challenge. In this paradigm, an evaluated domain model learning algorithm is obliged to define encoder and decoder functions to bridge the representation gap. We adapt the domain model evaluation metrics defined above to use these bridging functions, and show how to define such functions for several domain model learning algorithms. Finally, we describe a benchmark suite and suggested evaluation process for using it based on existing domain models from the International Planning Competition (IPC). We implemented this evaluation process and executed it on several domain model learning algorithms, including [Roni: TODO: what do we have]. The code, dataset, and evaluation process described in this work is intended to be publicly available. [2]

2 Background

We focus on *classical planning* problems, which is a well-studied type of planning problem in which a single agent is acting in a fully observable, discrete, and deterministic environment. A classical planning problem is defined as 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, $s_0 \in S$ is the initial state, and $G \subset F$ is a set of fluents. [Roni: For all: I modified the above definition to:(1) not have S as the set of states but have F as the set of fluents, and (2) not have G as a set of states but as a subset of fluents (that must be achieved). If anyone objects please do not change but rather comment here on the change you would like and why.] A *state* is defined by a set of propositions, representing that the conjunction of fluents in this set are true in this state. An action $a \in A$ is defined as a tuple $a = \langle pre(a), \mathit{eff}(a) \rangle$, where $pre(a)$ is the precondition of a and $\mathit{eff}(a)$ is the effect of a. The precondition $pre(a)$ specifies the conditions that must hold in a state for the action a to be applicable, while 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. We denote by L as the set of literals, i.e., $L = \{\ell | \exists f \in F : \ell = f \vee \ell = \neg f\}$. [Roni: Not sure about the] 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 $G \subseteq s_g$. [Roni: Minor modification to goal state to reflect the change above]

Planning domains are usually represented in a lifted representation, where the actions are defined with respect to a set of object types. The lifted representation of a planning domain is defined as a tuple $\mathcal{D} = \langle O, P, A \rangle$, where O is a set of object types, P is a set of predicates, and A is a set of actions. Actions and predicates are parameterized by objects, and preconditions and effects are defined accordingly. Popular classical planning systems, such FastDownward (Helmert, 2006), support such a lifted representation.

Different algorithms have been proposed for learning domains for classical planning. 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 can learn from, the type of action models they can learn, and the representation of the learned action model.

[Christian: I'm actually not sure any of this is needed. We're probably pushing over 50 related works in the area (cf. MACQ or similar surveys), and there's no way we'll cover the full space of approaches. A few exemplary ones, and then a pointer to a survey should be fine. Unless we need the background of how some of these work, it's about half-a-page of space we could save. Now the *metrics* used in those papers, is something related.] [Roni: I agree this needs to be shortened. TODO for me. Would be very good to have a list of the metrics used in each paper. Great idea. TODO by someone? a table or so with metric and list of papers using it?]

The ARMS algorithm (Yang, Wu, and Jiang, 2007) requires as observations the initial and final state of each of the provided trajectories, with optional use of intermediate states if available. For each observed action, ARMS lifts the action and constructs a set of constraints on the fluents involved in its preconditions and effects. These constraints are then resolved using a weighted MAX-SAT solver to generate the action model with the highest weight. The Simultaneous Learning and Filtering (SLAF) (Amir and Chang, 2008) learns lifted action models from trajectories even if some of the states in them are not observed. SLAF uses logical inference to filter inconsistent action models and requires observing all actions in the trajectory. FAMA (Aineto, Celorrio, and Onaindia, 2019) can also handle missing observations and outputs a lifted planning domain model. It frames the task of learning an action model as a planning problem, ensuring that the returned action model is consistent with the provided observations. NOLAM (Lamanna and Serafini, 2024) can learn lifted action models even from noisy trajectories. LOCM (Cresswell and Gregory, 2011) and LOCM2 (Cresswell, McCluskey, and West, 2013), and its extension to learn action costs (Gregory and Lindsay, 2016) analyze only the sequences of actions in the given trajectories, ignoring any information about the states between them. Based on less structured input knowledge, Framer (Lindsay et al., 2017) is an approach for learning planning domain models from natural language descriptions of activity sequences.

The Safe Action Model Learning (SAM) learning algorithms (Stern and Juba, 2017; Mordoch, Juba, and Stern, 2023; Juba, Le, and Stern, 2021; Juba and Stern, 2022; Le, Juba, and Stern, 2024; Mordoch et al., 2024) are a family of action model learning algorithms that return action models that guarantee that plans generated with them are applicable in the actual action model. An action model having this property is referred to as *safe action model*. The SAM family of algorithms addresses learning safe action models under different settings: the action models are lifted (Juba, Le, and Stern, 2021), with stochastic (Juba and Stern, 2022) or conditional effects (Mordoch et al., 2024), and even action

[2] A link to this is omitted to preserve anonymity.

models containing numeric preconditions and effects (Mordoch, Juba, and Stern, 2023). Le, Juba, and Stern (2024) extended their approach to support learning safe action models in a partially observable environment. ESAM (Juba, Le, and Stern, 2021) is an extension of the SAM algorithm that learns action models from domains in which there is ambiguity regarding the mapping of objects to parameters. To resolve this ambiguity, ESAM outputs a model with additional *proxy actions* that impose additional preconditions and parameter changes on actions in which such ambiguity cannot be resolved.

LatPlan (Asai and Fukunaga, 2018) and ROSAME-I (Xi, Gould, and Thiébaux, 2024) are conceptually different since they learn propositional action models from trajectories where the states in the given trajectories are given as images, as opposed to a conjunction of fluents. LatPlan is a fully unsupervised system that uses a variational autoencoder as a differentiable approximation to convert image states into discrete propositional symbols, enabling classical planning in a learned latent space. ROSAME-I, in contrast, assumes a predefined set of propositions and action signatures. It simultaneously trains a classifier to identify propositions from images and learns a lifted, first-order action model over the given symbolic vocabulary.

[Roni: To all: do not worry about shortening the background section. I will do this later.]

3 Problem Setting and Syntactic Similarity

In this work we consider the following domain model learning setup. An agent is acting in an environment that we assume can be represented as a classical planning domain denoted by M^*. The agent's actions are recorded in a set of trajectories, which are sequences of observations and actions. The observations and actions in the trajectories are given in a specific representation, 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 in 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. [Gregor: Maybe add: These plans then should be applicable in the environment M^* and should lead to the goal set in the problem description.][Roni: I intentionally did not add this, as it is not always the case and we talk later about cases where we need a "decoder" to translate the plans generated by the planner and the thing that can be executed by the agent in the environment.]

The core question is then: How good is the model learned by the domain model learning algorithm? Many prior works evaluated their domain model learning algorithms by comparing the learned domain model to a *reference domain model* (or *ground truth model*). The reference domain model is assumed to be correct, i.e., equivalent to M^*, and that the domain model returned by the evaluated domain model learning algorithm usese the same input representaiton (action names, object types, etc.).

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 predicates in the actions' preconditions and effects between the learned and reference domain models. We define these common metrics below. Let M and M^* be the evaluated and reference domain models, respectively, and let a be an action. We denote by $pre_M(a)$ the preconditions of action a according to domain M.

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

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

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

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.

[Roni: TODO: Add an example] [Gregor: Proposed this "pathological" example] [Roni: Not sure about this. It is just an example, not a "corner case".] As an example, consider a delivery scenario. The true model M^* has predicates at, in, and $contains$ to describe that a truck or package is at a location, a package is in a truck, and a truck contains a package. The action unload has three parameters ℓ, t, and p (location, truck, and package). In M^* its preconditions are $at(\ell, t)$ and $in(p, t)$, while its effects are $\neg in(p, t)$, $\neg contains(t, p)$, and $at(p, t)$. A learned model could conceivably have the same effects, but the preconditions $at(\ell, t)$ and $contains(t, p)$. It would have syntactic recall and precision for the effects of 1, but for the preconditions it would be $\frac{1}{2}$.

In a slightly different fashion, Chrpa et al. (2023) proposed an approach to assess the edit distance of the learned domain model with regard to the reference one. Low distance values indicate models that are syntactically close to each other, and if two models are syntactically identical (i.e., edit distance of zero), then they are said to be *strongly equivalent* (Chrpa et al., 2023).

The above approach to evaluating domain models has several limitations. First, in some evaluation setups a reference model may not be available at all. This is the setting, for instance, of the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) (Chrpa et al., 2017), and, of course, when applying automated planning to real-world problems. Second, comparing to a reference model implies there is a single best model for the envi-

ronment. 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). Lastly, there may be multiple domain models for a given environment that are fully identical for all practical purposes yet different syntactically. Thus, it is not clear that the syntactic similarity of the learned model to a reference model is a good indicator of how *useful* a learned domain model is. This limitation has been observed in prior works (Aineto, Celorrio, and Onaindia, 2019; Juba, Le, and Stern, 2021; Mordoch et al., 2024). Next, we discuss what constitutes a useful domain model and how to evaluate it without the need for a reference domain model.

[Roni: Tried to compile above the comments from Mauro and Gregor (which are commented out below). Please check if it is ok.]

4 Metrics for Evaluating Domain Models

[Roni: TODO: Add assupmtions on what we know (objects, types, etc.)]

[Gregor: In the previous section, we strongly argue that there might not be a "best" reference model of the environment available, which we can base our comparison on. But in this section, we still use the reference model M^* to compare with – we just do it in a more indirect fashion. Should we start this section with a discussion of what we can still assume in terms of access? And where the parallels to classical machine learning are?
From my point-of-view: (1) Having a true model of the actual environment to compare against is problematic in a realistic setting (why would we try to learn it otherwise?). (2) We can assume to have some samples, i.e., plans available from that environment, maybe also counter example plans. (3) There are two types of evaluations that we can conduct: the ICKEPS style, where we have an actual real-world problem that needs modelling, resp. whose model we need to learn (here we don't have a model!) and on the other hand our test-bed evaluation where the sole purpose is to determine whether the learning algorithm is good.
In the second case, we create an artificial environment and specify it using a PDDL domain. And then run our experiments in that domain. For these evals, it is totally ok to assume that we have a model to compare against. But: we should still be representation agnostic, which syntactic measures are not.
Let's take the analogy to ML here: they typically want to analyze how well we can learn from a specific real world setting (and thus have no model). While new ML-techniques can (and are?) also tested on artificial test-bed setting and datasets, where the ground truth, i.e., generation of all the data is known and can be used for evaluation.] [Roni: I like this distinction. Our focus should initially be on case (2), and then later talk about case (1).]

[Gregor: One general problem: the syntactic metrics all work solely on the level of the domain representation. But the metrics we propose here all require some type of actual states – i.e. they are dependent on at least the set of objects that exists. Or even more they are dependent on the full problem instances. I assume we then have a selection of "test" problem instances as in standard machine learning?][Roni: Yes, different metrics require differet "test sets". I'll write about this below, let me know if this works] We consider two main dimensions when evaluating the "usefulness" of a learned domain model: its *predictive power* and its *ability to enable solving problems*.

- **Predictive power.** Aim to learn a domain model that allows predicting the applicability and outcome of actions in the environment.
- **Problem-solving ability.** Aim to learn a domain model that enables the generation of applicable plans in the environment within reasonable resources bounds.

Observe that these dimensions are not necessarily aligned with the syntactic similarity metrics defined above, even if a reference model exists that enables computing it. A domain model may predict accurately when actions are applicable and the effects of applying them, while still being very different from a given reference model. This can happen, for example, when the learned model encodes mutex (mutual exclusion) conditions that the reference model does not explicitly define. [Gregor: maybe: "does only define implicitly"?][Roni: Editted. Is it clearer?] [Gregor: I've also put this issue already in the example above. Is this more illustrative?] 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 precondition that prevent it from solving many problems correctly. In contrast, a domain model may be very different from a reference domain model, e.g., adding many redundant preconditions to some actions, yet very *effective* in solving problems in the application domain since these redundant preconditions are often true in the real world.

The two dimensions described here — predictive power and problem-solving ability — are also not necessarily aligned. Indeed, a domain model may be very *predictive* of behavior of the agent in the reference domain model, yet too complex to be used for solving problems by any existing planning engine. Assume, for example, that a learned domain model that has many copies of the same action in 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 reference domain model, it may hinder the ability of a planning engine to find plans for problems in the application domain with this model, due to its complexity (e.g., large branching factor). Therefore, different metrics are required to evaluate these two dimensions. We describe such metrics below.

4.1 Predictive Power Metrics

While the syntactic similarity metrics are easy to compute, they do not accurately reflect the *predictive power* of the learned model. Predictive power metrics, referred to sometimes as *semantic* domain model metrics (Aineto, Celorrio, and Onaindia, 2019; Mordoch et al., 2024; Le, Juba, and Stern, 2024), are based on the idea that a learned model

should be able to predict the applicability of actions and their effects in the environment.

[Christian: Up until this point, I thought it was headed towards the predictive power of explaining the trajectory data. Given that it's about action applicability, this needs to be clarified much earlier on (e.g., intro). Also means that we don't have a notion of how well the learned model predicts the trajectories?]

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 action in a given state. Unlike the syntactic similarity metrics, which are computed based on the action model itself, the predictive power metrics require a dataset of states that we denote by S. This dataset is intended to represent the distribution of states of interest in the domain. One way to create this dataset is by running a planner on a set of test problems with the real action model.[Gregor: Or to actually observe states that appear in the actual environment during execution.]

[Christian: We still need the original domain in order to compute the predictive power. Some of the criticisms lofted at the existing metrics may need to be scaled back because of this – there's no way to know if the predicted actions that are applicable are the right ones, without having a reference model.]

[Roni: TODO: Clarify this does not consider complexity of the domain (which will be addressed in the next metric)]

[Gregor: Issue: what happens if we either cannot get the states themselves, i.e., no symbolic description or if the state description of the actual model differs from the one of the learned model? (say: predicates are named differently?)
Can we be more radical here? Maybe it is sufficient to have plans and non-plans for the real model? Then we could check whether these plans are executable in the learned model and whether the non-plans are not. One issue: this mixes applicability and effects.]

[Christian: Ya, I think many of the definitions start to fall apart if we don't have several assumptions down – same objects, same types, same fluent/action symbols with their signatures, etc.]

Predicted applicability For a domain model M and action a, we denote by $app_M(a, S)$ the set of states in S in which a is applicable according to M. [Gregor: This is easy to compute for STRIPS/SAS+, but hard if you have things like disjunctive preconditions. It should be #P-hard] [Roni: Hmm. I don't see what this is not linear in the size of S and the preconditions of a in M: we iterate over every state in S and then check whether a is applicable in it according to M. I guess if we have some universals in the preconditions this might be harder?] Based on this notation, we define the following predicted action applicability metric as follows for some action a:

- $\text{TP}_{app}(a) = |app_M(a, S) \cap app_{M^*}(a, S)|$
- $\text{FP}_{app}(a) = |app_M(a, S) \setminus app_{M^*}(a, S)|$
- $\text{TN}_{app}(a) = |S \setminus (app_M(a, S) \cup app_{M^*}(a, S))|$
- $\text{FN}_{app}(a) = |app_{M^*}(a, S) \setminus app_M(a, S)|$

[Leonardo: Should we keep TN? (since is not mentioned in precision/recall), similarly for predicted effects] [Gregor: Hm. That is strange. Should we also define precision and recall for the negative class? Does that make sense? This would be how good the approach is to determine non-applicable actions. I.e.:

$$P^{-}_{app}(a) = \frac{TN_{app}(a)}{TN_{app}(a) + FN_{app}(a)} \quad (1)$$

$$R^{-}_{app}(a) = \frac{TN_{app}(a)}{TN_{app}(a) + FP_{app}(a)} \quad (2)$$

] [Leonardo: I am not sure since it's binary classification. I think precision/recall for the negative class is dual w.r.t. the positive class ones] In words, $\text{TP}_{app}(a)$ is the number of states in S where a is applicable according to both the learned model and the real model, $\text{FP}_{app}(a)$ is the number of states in S where a is applicable according to the learned model but not the real model, $\text{TN}_{app}(a)$ is the number of states in S where a is not applicable according to both models, and $\text{FN}_{app}(a)$ is the number of states in S where a is applicable according to the real model but not the learned model. From these metrics, one can compute the precision and recall of the learned model for action applicability as follows,

$$P_{app}(a) = \frac{TP_{app}(a)}{TP_{app}(a) + FP_{app}(a)} \quad (3)$$

$$R_{app}(a) = \frac{TP_{app}(a)}{TP_{app}(a) + FN_{app}(a)} \quad (4)$$

and the overall action applicability, precision, and recall by averaging over all actions.

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

- $\text{TP}_{eff}(s, a) = |(a_M(s) \setminus s) \cap (a_{M^*}(s) \setminus s)|$
- $\text{FP}_{eff}(s, a) = |(a_M(s) \setminus s) \setminus a_{M^*}(s)|$
- $\text{TN}_{eff}(s, a) = |s \cap a_M(s) \cap a_{M^*}(s)|$
- $\text{FN}_{eff}(s, a) = |(a_M(s) \cap s) \setminus a_{M^*}(s)|$ [Gregor: there is a start too many (this is always $\emptyset$) I assume it should be $(a_M(s) \cap s) \setminus (a_{M^*}(s) \cap s)$ – the facts that are not changed by a_M, but not counting the ones changes by M_{M^*}][Roni: Good catch! I fixed it in a slightly different way, let me know if you agree or not.][Christian: The TN and FN aren't looking correct...but after spending some minutes with it, I think it is ;). Should state that a "negative" here indicates a literal that remains unchanged.][Gregor: Yep, but it does – $a_M(s)$ is the next state. If you take the intersection with the previous one, you get exactly all literals that have not changed.] [Leonardo: should we denote $a_M(s)$ as 's' obtained after executing a in s'? I'm not

sure currently is formally correct since we used a to denote actions in the previous paragraph.]

[Roni: Aggregate only over action applicable according to both Rationale: if a model specifies an action does not applicable thne its effects are not defined. TODO: Add a discussion about this]

For the purpose of the above computation, a state includes all the literals true in it, i.e., both positive propositions and negative ones. One can aggregate the above metrics over all states and actions, [Gregor: This might be computationally a problem – so we sample?!][Roni: Hmm. I am not sure. Is it worse than low-order polynomial in S and number of actions?] [Gregor: No, but $|S|$ can be exponential in the number of facts/predicates that we have.] [Roni: You're right. I had at some stage some text on separating S from the set of all states, having the set of states be implicitly defined by the set of fluents. I modified the PDDL definition above, and will add that we need a "test set" of states somewhere here.] [Leonardo: I agree, currently in the implemented evaluation the 'test set' is the set of states in a *test* set of trajs obtained interleaving optimal planning and random actions. A point is that some actions are executable in a few states, which are unlikely to be sampled randomly; moreover random sampling can generate invalid states] compute the overall values, and compute aggregated precision and recall values accordingly. The difference between the syntactic similarity of effects and the predicted effects metrics is subtle. It manifests when processing observations in which some literal ℓ is observed in the state before an action a, it is not an [Gregor: posisitve i.e. adding?] effect according to the learned model, but it is an effect according to the real model. This will count as a false positive in the syntactic similarity metrics yet not count as a false positive in the predicted effects metrics.

[Gregor: One more argument for these predictive power metrics overall: They allow us to differentiate between "cautious" or "safe" learning (e.g. Juba, Le, and Stern (2021)) and optimistic approaches (e.g. Bachor and Behnke (2024); that is don't add a precondition until proven it must exist). Safe model learning will always have $FP_{app}(a) = 0$ at the cost of high $FN_{app}(a)$, while optimistic learning will have $FN_{app}(a) = 0$, but high $FP_{app}(a)$.
Which of the two is better depends on your concrete application — in critical domains (air traffic, medicine) you want safe model, but in cases where you want to only get some plan that might work and you can always backtrack if it actually does not (transport? travelling?) you are ok with this.]

[Roni: Add a concrete example of how to compute these metrics] [Yarin: I can add an example after we verify the different formulas, example for each or only for Predicted effects ?]

[Gregor: I wanted to add one more point that I got from Pascal Bercher:
If we view domain model learning as an algorithmic problem, we are supposed to only judge the output of the learner based on the inferences that can be drawn explicit from the given trajectories. If we do anything else we implicitly evaluate the bias of the learners towards the type domain models we "usually" have in the input domains.]

4.2 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. It is well-known that 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 address this issue: *solving ratio* and *false plans ratio*. Both metrics are defined with respect to a set of problems P in the environment M^*. The solving ratio metric is defined as the ratio of problems in P that can be solved with the learned model by a given planning 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 limited computational resources, and that plan is valid according to the environment. The false plans ratio metric is defined as the ratio of problems in P that are solved by the learned model but are not valid according to the environment. This reflects the reliability of using plans with the learned model.

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. [Gregor: Theorem provers and proof assistants, e.g., Lean, sometimes use "heartbearts" (https://florisvandoorn.com/carleson/docs/Lean/Util/Heartbeats.html) instead of time. They measure elementary orations and bound their number. This ensures reproducibility across different machines. For us this could e.g. be number of expanded states. Or the number of times an effect is applied – this would also include computation effort for heuristics.] 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.

[Roni: TODO: Add metric on the runtime of solving the problem.] [Roni: TODO: Discussion: what about a metric on how many real plans can be validated by the learned model.][Mauro: Yes! validation capability of the learned model with regards to plans generated with the reference model!] [Roni: TODO: Maybe: add some unsolvability detection metrics?] [Christian: In our work on aligning (which is very closely related, it seems!), we computed both (1) if the plans found using P and M validate on M* and (2) if the plans found using P and M* validate on M. You need assumptions on the action names+parameters, but then can test (via VAL) both ways.] [Gregor: Agree! I have been using this in teaching as well: generate plans for both P and M^*.]

[Gregor: Proposal:
Let M and M^* be the evaluated and reference domain models, respectively, and let a be an action. Let $\mathcal{P}(M, k)$ be the set of all plans of M that have at most k many actions (or cost k) and that do not visit the same state s more than once

during their execution. We then define three metrics:

- $TP_{plan}(k) = |\mathcal{P}(M,k) \cap \mathcal{P}(M^*,k)|$
- $FP_{plan}(k) = |\mathcal{P}(M,k) \setminus \mathcal{P}(M^*,k)|$
- $FN_{plan}(k) = |\mathcal{P}(M^*,k) \setminus \mathcal{P}(M,k)|$

Lastly we could define $FN_{plan}(k) = |A|^k - TP_{plan}(k) - FP_{plan}(k) - FN_{plan}(k)$. We only consider plans up to length k, as computing the set of all plans that a model admits might be computationally infeasible. To determine $\mathcal{P}(M,k)$, we could use a top-k planner Katz et al. (2018); Speck, Mattmüller, and Nebel (2020); von Tschammer, Mattmüller, and Speck (2022). Alternatively one can generate plans for one model and validate it in the other using, e.g., VAL.
The main advantage of this evaluation metric is that it is (mostly) agnostic w.r.t. the representation of the problem chosen by the learner. E.g. if the learned model uses different predicates or an entirely different formalism (numeric, images, ...) to represent states. The only technical requirement is that we are able to "translate" the initial state and goal states of the problems in M^* to the problems in M.
This however does not solve the issue of potentially different representations of the actions, which we will discuss in the next section.] [Leonardo: I have currently implemented the alternative: generate plans with a 'reference' model and validate them in the learned model. I liked Gregor proposal to define also this metric in terms of TP/FP/FN, I would slightly rephrase it as:

- $TP_{plan} = |\Pi(s,G) \cap \Pi^*(s,G)|$
- $FP_{plan} = |\Pi(s,G) \setminus \Pi^*(s,G)|$
- $FN_{plan} = |\Pi^*(s,G) \setminus \Pi(s,G)|$

where Π and Π^* are the set of solution plans in M and M^* for going from state s to a goal state in G]

Note that all the metrics described in this section do not require a reference model, but instead are with respect to the actual environment. Thus, they can be used to compare two models directly, providing statistics on which one is better according to different aspects of the environment. [Christian: I think the above is misleading. We need to have states (fluents matching the learned model), action applicability (for predictive power), etc, etc. We do need a reference model! What's changed is that we aren't using syntactic comparisons anymore, but the M* model is certainly still there.] [Gregor: I think you could relax this a bit more. Iirc the point here is that you don't need to have a symbolic model of M^* to evaluate. It would be enough to "try" the actions in the actual physical world. For action applicability this would be enough. Well in both cases, you still need to be able to correlate an initial state in the real world with a symbolically represented initial state in the learned model. For example, if you observe only action sequences (think LOCM), you have no means to correlate the initial state that your learner thought the trajectories have with the actual one. You are missing a kind-of anchoring or "grounding" in the actual model M^*. This is what Roni's group proposes in the Encoder-Decoder Mechanism section.]

5 Bridging Representation Gaps

Some domain model learning algorithms output a classical planning domain model that uses a different representation than the input representation. Such a model can still be used by a planner, yet its input and output may be incompatible with that of the reference domain model. In some cases, these representation differences are minimal, e.g., using different ordering of the parameters or object types. In other cases, the differences are more significant. For example, the ESAM algorithm (Juba, Le, and Stern, 2021) outputs a domain model with additional *proxy actions* that are used to resolve ambiguities in the mapping of objects to parameters. [Roni: Pascal mentioned a paper that learned macro actions. TODO: add ref for it here (with relevant short text)] The proxy actions are not part of the original domain model, and they are not used in the reference domain model. [Roni: More examples?]

[Mauro: OpMaker2 generates classical planning models in OCL language McCluskey et al. (2010)]

[Yarin: added LOCM:] Similarly, the LOCM algorithm (Cresswell, McCluskey, and West, 2013) induces finite-state machines solely from action sequences-with no access to intermediate state information or predicate names-and outputs action schemas using *proxy predicates*. These proxy predicates are not part of the original domain but are introduced as a mechanism to infer the semantics of the original predicates.

The syntactic similarity metrics are not relevant in such cases, where the representation is intentionally different from the reference model. Next, we describe how to apply the other metrics – predictive power and problem-solving – in such cases. Note that we limit the discussion to the case where the learned domain is still a planning domain, as opposed to images or other more involved representations.

5.1 The Encoder-Decoder Mechanism

Let R_{M^*} denote the input representation and let R_M denote the representation of the learned domain model. The given trajectories are given in R_{M^*}, while the output of a planner that uses the learned domain model is in R_M. To allow the comparison of the learned domain model with the reference domain model, we need to bridge the gap between these two representations. To this end, we require every domain model learning algorithm to output a representation *encoder* and a representation *decoder* in addition to the learned model. We describe these two components below. The *encoder* transforms states and actions in R_{M^*} to corresponding states and actions in the learned domain representation R_M. The *decoder* performs the reverse transformation, mapping states and actions in R_M to corresponding states and actions in R_{M^*}.

Formally, let A and A_M be the set of actions in the real and learned domain models, respectively, and let S and S_M be the set of states in the real and learned domain models, respectively. The power set is denoted by $P(X)$, which is the set of all subsets of X. The encoder is required to implement the following set of functions:

- $encodeAction : S \times A \to P(A_M)$, returns the set of ac-

tions in M that represent the application of a[Yarin: $a \in A$] in a given state s[Yarin: $s \in S$].

- $encodeState : S \to S_M$, returns the state in M that represents the state s in M^*. [Yarin: returns the state $s \in S$ as its representation in S_M]

The decoder is required to implement the following functions:

- $decodeAction : A_M \to A$, returns the action in M^* corresponding to the action in M. [Yarin: returns the action $a \in A_M$ as its representation in A]
- $decodeState : S_M \to S$, returns the state in the input representation (S) that represents a given state in S_M. [Yarin: returns the state $s \in S_M$ as its representation in S]

5.2 Predictive Power Metrics with an Encoder-Decoder

For the predicted applicability metric, we modify the way $app_M(a, S)$ is computed. The main difference is that a single action in the reference domain model may be represented by multiple actions in the learned model. Thus, we define $app_M(a, S)$ to be the number of states in S in which *there exists* an action a' in the learned model such that a' is applicable in the state in M that encodes s.

[Roni: TODO: Add a formal definition of the app_M method using the encoder and decoder methods]

Similarly, for the predicted effects metric, we need to decode the state resulting from $a_M(s)$ instead of using it as-is.

[Roni: TODO: Add a formal definition of the app_M method using the encoder and decoder methods]

5.3 Problem-Solving Metrics with an Encoder-Decoder

Adapting the problem-solving metrics to use the encoder-decoder mechanism is straightforward. The encoder is used to encode the problem from the input representation to the learned model representation. Then, the planner is run on the encoded problem with the learned model. Finally, the decoder is used to decode the solution plan from the learned model representation to the input representation. This allows checking if a plan has been found and if it is valid according to the reference domain model.

5.4 Case Studies

Next, we describe how the encoder-decoder mechanism can be implemented for several action model learning algorithms. [Yarin: Some action model learning algorithms operate directly in the original input representation R_{M^*}, without transforming the state or action spaces. For these algorithms, the learned model uses the same symbols and structure as the input data. As such, there is no need for additional encoding or decoding: the identity function suffices for both.]

SAM (Stern and Juba, 2017)[Yarin: or is ut juba2021safe ?]: [Roni: Either me or someone from my group will write this one] Encoder: identity Decoder: identity

FAMA (Aineto, Celorrio, and Onaindia, 2019) - Leonardo L.?: Encoder: identity Decoder: identity - maybe change to parameter names?

ESAM (Juba, Le, and Stern, 2021): [Roni: Either me or someone from my group will write this one] Encoder: identity for both state and actions Decoder: translate proxy actions to original actions

ROSAME-I (Xi, Gould, and Thiébaux, 2024): [Roni: Maybe Yarin or Argaman can help with this one] Encoder: encode to a numeric vector of ones and zeros Decoder: map action parameters

OBSERVER[Yarin: add cite]: [Roni: Anyone can help here?] Encoder: from binding to grounded Decoder: from grounded predicates and actions to the binding they represent

LOCM (Cresswell, McCluskey, and West, 2013):[Yarin: I will complete this] Encoder: identity for action name and proxy for state predicates Decoder: translate proxy predicates to original predicates

[Yarin: Add NOLAM too?] [Roni: Most important: anyone can help here?]

5.5 An Evaluation Paradigm

[Roni: Leonardo will merge this into the next section] Using the above metrics and the encoder-decoder mechanism, we can define an evaluation paradigm for action model learning algorithms. The evaluation paradigm consists of the following steps: First, we generate a set of trajectories in the input representation for learning. This set of trajectories can be generated by running a planner on a set of problems in the reference domain model or via a random walk. Then, this set of trajectories is split in 80/20 ratio, using 80% of the trajectories for training and 20% for testing. The evaluated learning algorithm is given the training set of trajectories and is required to output a learned domain model, an encoder, and a decoder. [Roni: TODO for myself: Complete this subsection] This 80/20 split is repeated k times following a standard k-fold validation process.

[Christian: An eval-heavy ICAPS'er may take issue with some of the suggestions (random walks are hard to do right, the benchmark problem sets aren't IID, etc, etc. I'd say we're probably mostly fine (someone can always find some issue if they try hard enough), but we *should* comment on the disconnect of trace -vs- state. Many approaches require traces, but the evals require state sets. Taking the latter to just be states along traces is a deliberate choice that warrants discussion.]

6 Benchmarks and Evaluation

[TODO: Leonardo is in charge of this section.]

[TODO: Describe the benchmark suite and how to use it.]

[TODO: If we have time: show results for at least some of the algorithms on the benchmark suite.]

[Leonardo:

6.1 Benchmark

We validate the proposed metrics by experimenting with some state-of-the-art approaches on a newly generated

benchmark for offline learning classical planning domains. we consider the set of 20 classical planning domains adopted in all previous IPC learning tracks (Fern, Khardon, and Tadepalli, 2011; Vallati et al., 2015; Taitler et al., 2024), with a number of operators in $[1, 12]$, predicates in $[3, 25]$, and object types in $[1, 9]$[3]. For each domain, we produced a set of 10 trajectories from a set of 10 small-medium size problems as described in the following. Firstly, we randomly generated a problem using existing generators (Seipp, Torralba, and Hoffmann, 2022). Then, we run FastDownward (Helmert, 2006) to generate a solution plan, and generated the trajectory by executing the plan starting from the initial state of the problem. It is worth noting that problem generators available in the literature can be biased in terms of initial states and goals. For example, in domain ferry the ferry is always empty in the initial state; in domain floortile the goal is always to paint all even tiles white and all odd ones black, leaving an empty extra row to place the robot at the end. To mitigate such biases, we randomly sample a *subtrajectory* from the original generated one, resulting in a subtrajectory where the initial and final states are also randomized. However, some operators are unlikely to appear in a randomly sampled subtrajectory; for example, in domain goldminer, the operator pick-gold is always executed at the end of a solution plan. Hence, we jointly sample the initial and final states of each subtrajectory such that the initial or final state is the same as the original trajectory with probability 0.33. Moreover, some operators are not likely to be executed in an heuristic plan; for example, in domain barman, it is possible to either clean and fill a shot, which requires 2 actions, or just refill a (possibly dirty) shot. To increase the chance of executing more efficient actions in a plan, we *optimally* solved 30% of the problems. Finally, to obtain heterogeneous trajectories, we generated problems with all different settings in terms of the number of objects of each type; however, npuzzle and blocksworld does not respect this requirement, since there is only one object type and linearly increase it leads to problem that are too difficult to solve. We also randomized the trajectory length in $[5, 30]$, and the number of objects in $[3, 107]$[1]. – IN PROGRESS –

6.2 Evaluation paradigm?

6.3 Experimental analysis

]

7 Discussion

[Roni: For all: please read and let me know what you think by adding comments in the text (e.g., add text like this "[[YourName: bla bla bla"]]) or editting.] The proposed evaluation paradigm is designed to be flexible and extensible. In particular, it can be extended to support other types of representations, such as images or other types of data. Another possible extension is to allow the action model learning algorithm to output a domain model that uses a more general type of planning. For example, the action model learning algorithm may output a domain model that uses a probabilistic (Xi, Gould, and Thiébaux, 2024) or partially observable representation (Le, Juba, and Stern, 2024). Adapting the predictive power metrics to such cases may not be trivial, but the problem-solving metrics can be adapted in a straightforward manner.

Going beyond discrete representations is also possible. The predicted applicability metrics can be computed as-is, considering both numeric and discrete preconditions. More involved is the predicted effects metric, which may require defining some loss function for the numeric effects, as learning exactly the same numeric effects as in the reference domain model seems unlikely. Note that in all of these variants and generalizations, the encoder-decoder mechanism allows seamless use of the problem-solving metrics.

[Yarin: We can add a section on trajectories with not successful actions too @Argaman]

[TODO: Maybe talk about metrics for online learning] The metrics and evaluation paradigm outlined in this paper are for *offline* learning of action models, where the learning algorithm is given a set of trajectories and is required to output a domain model. *Online learning* of action models have also been studied in the literature (Lamanna et al., 2021; Sreedharan and Katz, 2023; Benyamin et al., 2025; Ng and Petrick, 2019; Chitnis et al., 2021; Verma, Karia, and Srivastava, 2023; Karia et al., 2023; Jin et al., 2022)[TODO: More citations?][Yarin: added, to many?], where the learning algorithm is required to learn a domain model by actively interacting with the environment. Similar to other online tasks, one may distinguish between the *cumulative regret* of the learning process, which can be the number of actions performed for learning or the number of problems failed to solve while collecting observations. Specific metrics and evaluation for online learning of action models, however, is beyond the scope of this work.

Finally, models can also be compared not only in terms of their characteristics, but also in terms of the processes used to learn or generate them (Vallati and McCluskey, 2021), a perspective that is commonly used for assessing conceptual models.

7.1 Relation to Model Repair

There is a strong relation between model learning and model repair. Following the definition provided in Bercher, Sreedharan, and Vallati (2025), model learning approaches start from a minimalistic (potentially empty) model and extend it on the basis of some provided input. The goal is to bring such an initial model to a model that can be used in practice. Instead, model repair starts from compilable/operational models, and refine them according to some constraints in order to maintain the operationality of the models with regards to changing requirements or demands. Constraints can come under different forms, such as the request to include/exclude some given plans in the solution space of a model, to ensure that a given plan is optimal for the model, or to constraint the solution space.

This relationship also extends to approaches such as model reconciliation (see, e.g., (Chakraborti et al., 2017; Sreedharan et al., 2019)), where one model needs to be

[3]Further details about the benchmarks are provided in the supplementary material.

aligned with another model to maximize the explainability of generated plans. In reconciliation approaches, the usual metric considered is the number of changes to be made to align the models, with the goal of minimizing modifications.

8 Conclusion Future Work

[Roni: I'll fix this text later] We presented a new paradigm for evaluating action model learning algorithms across different representations. In this paradigm, each action model learning algorithm is required to output an action model and an encoder-decoder pair. The encoder-decoder pair is used to bridge representation gaps and enable measuring the problem-solving capabilities of the learned action models. We proposed an evaluation scheme that leverages the encoder-decoder pair to systematically compare learned action models and described several evaluation metrics. A benchmark suite was also provided to facilitate the evaluation of action model learning algorithms, based on existing domain models from the International Planning Competition (IPC). We demonstrated our evaluation paradigm by applying it to several action model learning algorithms, including SAM, ESAM, FAMA, and ROSAME.

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.

Asai, M., and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the aaai conference on artificial intelligence*, volume 32, 6094–6101.

Bachor, P., and Behnke, G. 2024. Learning Planning Domains from Non-Redundant Fully-Observed Traces: Theoretical Foundations and Complexity Analysis. In *38th AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20028–20035.

Benyamin, Y.; Mordoch, A.; Shperberg, S. S.; and Stern, R. 2025. Integrating reinforcement learning, action model learning, and numeric planning for tackling complex tasks.

Bercher, P.; Sreedharan, S.; and Vallati, M. 2025. A survey on model repair in ai planning. In *Proceedings of the 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11782–11791.

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.

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.

Cresswell, S., and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

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.

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning* 84(1-2):81–107.

Gregory, P., and Lindsay, A. 2016. Domain model acquisition in domains with action costs. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 149–157.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; Muise, C.; Brachman, R.; Rossi, F.; and Stone, P. 2019. *An introduction to the planning domain definition language*.

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

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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 7042–7050.

Juba, B., and Stern, R. 2022. Learning probably approximately complete and safe action models for stochastic worlds. In *AAAI*, 9795–9804.

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*.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A novel iterative approach to top-k planning. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 132–140. AAAI Press.

Lamanna, L., and Serafini, L. 2024. Action model learning from noisy traces: a probabilistic approach. In *ICAPS*, 342–350. AAAI Press.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; and Traverso, P. 2021. Online learning of action models for pddl planning. In *IJCAI*, 4112–4118.

Le, H. S.; Juba, B.; and Stern, R. 2024. Learning safe action models with partial observability. In *AAAI Conference on Artificial Intelligence*, volume 38, 20159–20167.

Lindsay, A.; Read, J.; Ferreira, J. F.; Hayton, T.; Porteous, J.; and Gregory, P. 2017. Framer: Planning models from natural language action descriptions. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

McCluskey, T. L.; Cresswell, S. N.; Richardson, N. E.; and West, M. M. 2010. Action knowledge acquisition with opmaker2. In *Agents and Artificial Intelligence*, 137–150.

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.

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*.

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

Mordoch, A.; Juba, B.; and Stern, R. 2023. Learning safe numeric action models. In *AAAI Conference on Artificial Intelligence*, 12079–12086.

Ng, J. H. A., and Petrick, R. P. 2019. Incremental learning of planning actions in model-based reinforcement learning. In *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.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic top-k planning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9967–9974. AAAI Press.

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

Sreedharan, S.; Hernandez, A. O.; Mishra, A. P.; and Kambhampati, S. 2019. Model-free model reconciliation. In *Proc. of IJCAI*, 587–594. IJCAI.

Stern, R., and Juba, B. 2017. Efficient, safe, and probably approximately complete learning of action models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; et al. 2024. The 2023 international planning competition.

Vallati, M., and Chrpa, L. 2019. On the robustness of domain-independent planning engines: The impact of poorly-engineered knowledge. In *Proceedings of the 10th International Conference on Knowledge Capture, K-CAP*, 197–204.

Vallati, M., and McCluskey, T. L. 2021. A quality framework for automated planning knowledge models.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3):90–98.

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.

von Tschammer, J.; Mattmüller, R.; and Speck, D. 2022. Loopless top-k planning. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 380–384. AAAI Press.

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.