Differentiable Learning of Lifted Action Schemas for Classical Planning

Jonas Reiter[1], **Jakob Elias Gebler**[1], **Hector Geffner**[1]

[1]RWTH Aachen University
jonas.reiter@ml.rwth-aachen.de, jakob.gebler@ml.rwth-aachen.de, hector.geffner@ml.rwth-aachen.de

Abstract

Classical planners can effectively solve very large deterministic MDPs represented in STRIPS or PDDL where states are sets of atoms over objects and relations, and lifted action schemas add or delete these atoms. This compact representation yields strong search heuristics and provides an ideal setting for structural generalization, since lifted relations and action schemas give rise to infinitely many domain instances. A central challenge is to learn these relations and action schemas from data, and recent approaches have addressed this problem using different types of observations. In this work, we develop a novel neural network architecture for learning action schemas from traces where states are fully observed but action arguments are unobserved. The problem is a simplification but an important step towards learning planning domains from sequences of images and action labels, and we aim to solve this simplification in a nearly perfect manner. The challenge lies in learning the action schemas while simultaneously identifying the action arguments from observed state changes. Our approach yields a robust differentiable component that can then be integrated into larger neuro-symbolic models. We evaluate the architecture on various planning domains, where the learned lifted action schemas must recover the ground-truth structure. Additionally, we report experiments on robustness to observation noise and on a variation related to slot-based dynamics models.

1 Introduction

Classical planners can find goal-directed plans in very large deterministic Markov decision processes (MDPs). They solve these problems from scratch, without training, by automatically deriving heuristic information to guide search from STRIPS-like representations in which states are sets of atoms over a fixed set of relations and actions add and delete atoms provided that certain preconditions hold (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2025). These reachability problems involve huge state spaces that are beyond the scope of exploration methods in reinforcement learning and Monte-Carlo tree search (MCTS).

A key challenge in classical planning is to learn state relations and action schemas from observations rather than hand-crafting them. Recent approaches address this problem in different forms. Traditional symbolic methods learn from traces consisting of full states and full actions, sometimes under partial observability or noise (Zhuo and Kambhampati 2013; Aineto, Celorrio, and Onaindia 2019; Le, Juba, and Stern 2024; Bachor and Behnke 2024; Lamanna et al. 2025). More recently, it has been shown that state relations and action schemas can be learned effectively and correctly from pure action traces alone (Gösgens, Jansen, and Geffner 2025). A limitation of these settings is that actions are assumed to reveal all of their arguments, even though some arguments are not truly observable and others are merely artifacts of the STRIPS language. For example, the action of picking up a block x in STRIPS is written as $\texttt{unstack}(x, y)$, where y is the block on which x is currently placed. The only reason y appears explicitly is that the action adds the atom $\texttt{clear}(y)$. In other planning languages, such as STRIPS+, the action can instead be written as $\texttt{unstack}(x)$, with the value of y determined uniquely by the precondition $\texttt{on}(x, y)$. The SYNTH algorithm learns action schemas from traces containing full states and actions with the minimal number of arguments needed to determine all remaining ones (Jansen, Gösgens, and Geffner 2025). Likewise, a recent SAT-based approach learns action schemas from full states and action names alone, without assuming knowledge of the action arguments or their arity (Balyo et al. 2024). For this setting, the authors show that the problem is computationally hard, namely as hard as establishing an isomorphism between two graphs.

In this work, we propose DIAS (Differentiable Induction of Action Schemas), a novel neural architecture for learning action schemas from traces in which states are fully observed, possibly with noise, but action arguments are not observed. This is the same isomorphism-hard problem as above, but addressed through gradient descent rather than repeated SAT calls. The potential advantages are scalability, robustness to noise, and the possibility of replacing full symbolic states with other types of observations, such as images. More generally, the problem we study is a simplification of the harder and more realistic task of learning a planning domain from sequences of images and action labels (Xi, Gould, and Thiébaux 2024, 2026). Rather than optimizing relative performance on that broader task, however, we aim for near-perfect performance on this simplified setting, which remains a necessary and challenging component in

its own right. We also consider two variants of the learning problem in which some or all action arguments are observed.

The proposed DIAS architecture is shown in Figure 1 and operates in two stages. First, a graph neural network (GNN) takes as input a graph defined by the atoms in state s_t at time t and their changes in the next state s_{t+1} after action a_t. It computes embeddings for each object in the state that act as keys K. For each action name a, we maintain a set of learnable query embeddings Q^a and select a list $o_1, \ldots, o_k$ of k objects into slots in $\tilde{S}^a$ via attention between K and Q^a. In addition, learnable slot probabilities w^a determine whether a slot is active. Second, for each action name a, we maintain four learnable probabilities for every lifted atom $p(x_i, x_j)$, with $i, j \in [1, k]$, in $P^a_{\mathrm{pre}+}$, $P^a_{\mathrm{pre}-}$, P^a_{add}, and P^a_{del}. These encode whether the atom is a lifted positive or negative precondition, an add effect, or a delete effect of the action. The object selection $\tilde{S}^a$ grounds the lifted action in the state, where preconditions must hold in s_t and effects must predict the next state s_{t+1}. Prediction errors are minimized with a suitable loss function.

The selection of the objects that bind action arguments, together with the localized effects of actions on those objects, connects DIAS to a number of slot-based dynamic models in deep learning, such as recurrent independent mechanisms (RIMs) and related approaches (Goyal et al. 2021; Madan et al. 2021; Lamb et al. 2021; Goyal and Bengio 2022). In these models, slots play the role of state variables that interact sparsely. Two important differences from STRIPS dynamics are that STRIPS effects are state independent and therefore less expressive, yet the learned STRIPS action schemas generalize in ways that slot-based or variable-based dynamics cannot. The reason is that in domains such as *Blocksworld*, changing the number of objects changes not only the number of state variables, but also their set of possible values. This does not happen in STRIPS, where the state variables, namely atoms, are Boolean. We also consider a RIM-like architecture in our setting in which the STRIPS effects are replaced by a multi-layer perceptron (MLP) that takes the values of atoms over the selected objects at time t and predicts their effects. The resulting dynamic model is more expressive than STRIPS because it handles state-dependent effects, and it generalizes to larger numbers of objects in the same way, but learning is less effective and does not yield symbolic action schemas that support effective planning.

The remainder of the paper is organized as follows. We cover the necessary background and notation in Section 2, formulate the problem setting in Section 3, introduce DIAS in detail in Section 4, report experiments in Section 5, and conclude in Section 6. A more detailed discussion of related work is provided in Appendix A.1.

2 Preliminaries

We review STRIPS and the STRIPS+ variant for modeling classical planning domains.

2.1 STRIPS

In classical planning, a STRIPS problem P is defined as a pair $P = \langle D, I \rangle$ consisting of a domain D and instance information I (Geffner and Bonet 2013). Here, $D = \langle \mathcal{P}, \mathcal{A} \rangle$ is a lifted domain description containing a set of predicates $\mathcal{P} = \{p_i\}_{i=1}^{R}$ and a set of action schemas $\mathcal{A} = \{a_i\}_{i=1}^{A}$. Each predicate $p \in \mathcal{P}$ takes a set of arguments $p(y_1, \ldots, y_{|p|})$, or, more briefly, $p(\vec{y})$, where $|p|$ denotes the arity of predicate p. We denote by $\mathcal{P}_k = \{p \in \mathcal{P} \mid |p| = k\}$ the subset of predicates with arity k. Each action schema $a \in \mathcal{A}$ similarly takes a set of arguments $a(x_1, \ldots, x_{|a|})$, or, more briefly, $a(\vec{x})$, where $|a|$ denotes the arity of action schema a. An action schema is defined by its set of preconditions, add effects, and delete effects as $a(\vec{x}) = \langle \mathrm{Pre}(a), \mathrm{Add}(a), \mathrm{Del}(a) \rangle$. These sets consist of lifted atoms $p(x_1, \ldots, x_{|p|})$ over the action schema arguments.

An instance $I = \langle \mathcal{O}, s_0, g \rangle$ consists of a set of constants or objects $\mathcal{O} = \{o_i\}_{i=1}^{O}$, an initial state s_0, and a goal description g. A predicate $p(o_1, \ldots, o_{|p|})$, with objects $\vec{o}$ assigned to its arguments $\vec{y}$, is called a ground atom. Then s_0 is the set of all ground atoms true in the initial state, and g is a set of ground atoms that have to be true in every goal state.

Applying a ground action $a(o_1, \ldots, o_{|a|})$, with objects $\vec{o}$ assigned to action arguments $\vec{x}$, induces a transition from a current state s to a next state s' in the state space $\mathcal{S}$. The grounded preconditions $\mathrm{Pre}(a(\vec{o}))$ must hold in s for $a(\vec{o})$ to be applicable. Action execution then yields $s' = ((s \setminus \mathrm{Del}(a(\vec{o}))) \cup \mathrm{Add}(a(\vec{o})))$.

2.2 STRIPS+

In STRIPS, the action arguments $\langle x_1, \ldots, x_{|a|} \rangle$ must contain every variable that appears in a lifted atom $p(x_1, \ldots, x_{|p|})$ in the preconditions $\mathrm{Pre}(a)$ or effects $\mathrm{Add}(a)$ and $\mathrm{Del}(a)$. Jansen, Gösgens, and Geffner (2025) introduce STRIPS+ to relax this requirement. In STRIPS+, lifted preconditions and effects may use an additional set of variables $\vec{z}$ that is disjoint from the explicit variables $\vec{x}$ and does not appear in the action arguments. The only requirement is that the value of every variable in $\vec{z}$ is uniquely determined by the explicit action arguments $\vec{x}$ together with the action preconditions. STRIPS+ can therefore express STRIPS domains while using fewer action arguments.

Intuitively, this allows for more natural action descriptions. For example, when moving a block in *Blocksworld*, the STRIPS action `move(?block ?from ?to)` can be simplified to `move(?block ?to)` in STRIPS+, since the source location is uniquely determined by the precondition `on(?block ?from)`. In our approach, we consider input traces consisting of full states together with STRIPS+ actions, full STRIPS actions, or action names only.

3 Problem formulation

We are given a set of N state-transition triplets $\{\langle s, a, s' \rangle_i\}_{i=1}^{N}$. All transitions are applicable and are sampled from a single planning problem P under an arbitrary stochastic sampling policy. The states s and $s' \in \mathcal{S}$ are grounded STRIPS state descriptions. We consider

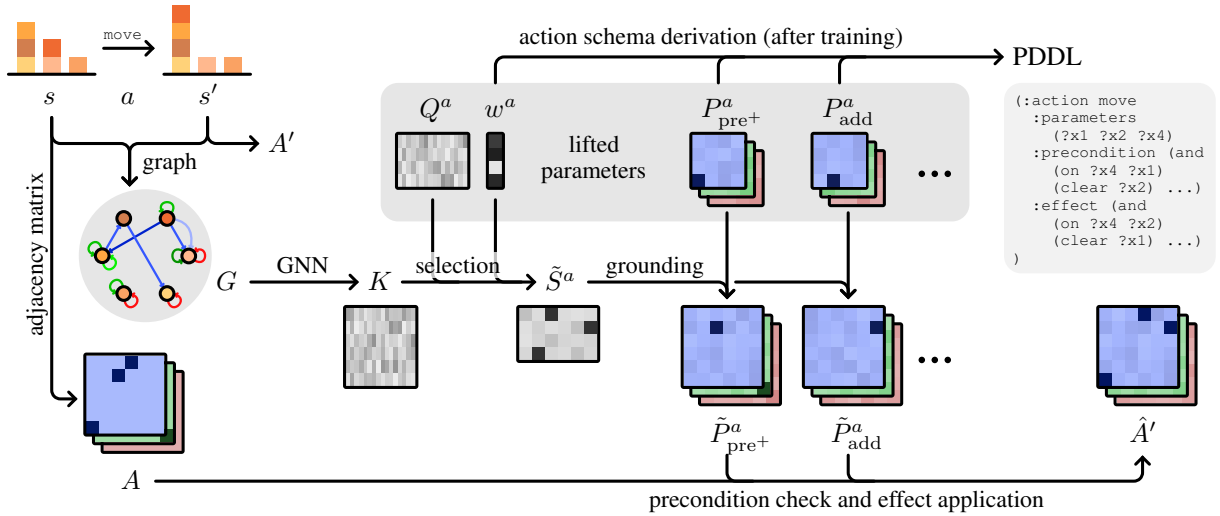


Figure 1: Overview of the DIAS architecture. A transition $\langle s, s' \rangle$ is encoded as a graph G. A GNN computes object embeddings K, which are matched with object queries Q^a for action a. Together with slot weights w^a, this yields an object selection $\tilde{S}^a$. Lifted preconditions and effects $P^a_{\text{pre}^+}$, $P^a_{\text{pre}^-}$, P^a_{add}, and P^a_{del} are then grounded with $\tilde{S}^a$ into adjacency-matrix space as $\tilde{P}^a_{\dots}$. Given the adjacency matrix A of state s, grounded preconditions are checked and effects are applied to predict the successor-state adjacency matrix $\hat{A}'$. Training minimizes a prediction loss between $\hat{A}'$ and A' while additionally maximizing and minimizing lifted preconditions and effects, respectively. After training, the action schema for each action a can be derived from its lifted parameters.

three cases for the action label a: (i) full STRIPS actions, e.g., `move(o3,o7,o4)`, (ii) STRIPS+ actions, e.g., `move(o3,o4)`, or (iii) action names only, e.g., `move`.

Our goal is to learn the lifted preconditions and effects $\langle \text{Pre}(a), \text{Add}(a), \text{Del}(a) \rangle$ of each action schema $a \in \mathcal{A}$ in the hidden lifted domain. A learned domain is considered equivalent to the original domain if, for any state s from any instance I, it generates the same set of successor states s'. This entails soundness, i.e., all generated successors are true successors, and completeness, i.e., all true successors are generated.

Assumptions We do not consider conditional effects, numerical features, or other advanced PDDL features, only pure STRIPS domains with typing and negative preconditions. We also adopt the injective action binding assumption (Juba, Le, and Stern 2021), ruling out domains in which a single object o can be bound to multiple action arguments. Finally, we do not consider domains with predicates of arity other than 1 or 2, and we assume that all domain predicates are known. On the other hand, we do not assume that the action arities are known.

4 Approach

Our goal is to learn lifted preconditions and effects for each action $a(\vec{x})$. We do so by considering individual transitions $\langle s, a, s' \rangle$. DIAS selects the arguments $\vec{x}$ of a, ensures that learned preconditions are met, and applies learned effects to compute a successor-state prediction $\hat{s}'$. All stages use soft and differentiable operations to allow for end-to-end gradient-based optimization. The complete approach is depicted in Figure 1, and the individual stages are introduced below.

4.1 Selection of action arguments

In general, we do not assume knowledge of the objects $\vec{o}$ to assign to the action arguments $\vec{x}$. Instead, we infer them from states s and s' using embeddings computed by a GNN.

The state s is a set of ground atoms $p(\vec{o})$ from which we construct a directed graph $G = \langle V, E \rangle$. The nodes V are given by the set of objects $\mathcal{O}$, and each ground atom $p(o_i, o_j)$ true in s represents a typed edge $\langle p, \langle o_i, o_j \rangle \rangle$ of type p between objects o_i and o_j. We additionally determine $p'_{\text{add}}(\vec{o})$ and $p'_{\text{del}}(\vec{o})$ as the sets of ground atoms added and deleted when transitioning from state s to s'. The respective typed edges $\langle p'_{\text{add}}, \langle o_i, o_j \rangle \rangle$ and $\langle p'_{\text{del}}, \langle o_i, o_j \rangle \rangle$ are added to the set of edges E.

Note that we limit ourselves to unary and binary predicates. Unary atoms $p(o_i)$ are represented by self-loops $\langle p, \langle o_i, o_i \rangle \rangle$. Object types are included analogously as unary atoms $t(o_i)$ and edges $\langle t, \langle o_i, o_i \rangle \rangle$. If any action arguments $x_k = o_i$ are given, we mark them in the graph via self-loops $\langle a^k, \langle o_i, o_i \rangle \rangle$, where a^k denotes an edge type for the k-th argument of action a. Finally, we add the reverse edge $\bar{e}$ for every edge $e \in E$ to facilitate message passing.

The resulting graph G, with random initial embeddings, is passed through a relational graph convolutional network (R-GCN) (Schlichtkrull et al. 2018). We denote the final d-dimensional node feature vectors for the objects as keys $K \in \mathbb{R}^{O \times d}$. To assign objects o_i to action arguments x_k, each lifted action a has a set of d-dimensional learnable query vectors $Q^a \in \mathbb{R}^{M \times d}$. Here, M is the number of

slots, a hyperparameter and upper bound on the action arity that can be represented. We compute how well each object's key K_i matches each slot's query Q_k^a through a correspondence matrix $C^a \in \mathbb{R}^{M\times O}$ akin to scaled dot-product attention (Vaswani et al. 2017) as

$$C^a = \frac{Q^a K^\top}{\sqrt{d}}. \tag{1}$$

We then utilize a rectangular version of the Sinkhorn algorithm (Brun et al. 2022) to compute a soft assignment from objects to mutually exclusive slots. The result $S^a = \mathrm{Sinkhorn\_D1D2}(C^a) \in (0,1)^{M\times O}$ is a row-stochastic matrix with row sums $= 1$ and column sums ≤ 1. Each entry $S^a_{i,k}$ represents the probability that object o_i is assigned to slot k.

Additionally, we introduce learnable parameters $w^a \in \mathbb{R}^M$. With a sigmoid activation, they represent the activation probability of each slot. We scale each slot's assignment distribution with the respective probability to obtain the selection matrix

$$\tilde{S}^a = \mathrm{diag}(\sigma(w^a))S^a. \tag{2}$$

4.2 Application of effects

Each atom $p(x_1,\dots,x_k)$ of the lifted action schema $a(\vec{x})$ is either (i) not included in effects, (ii) an add effect, or (iii) a delete effect. We therefore model the lifted effects as a learnable parameter matrix $P^a_{\text{eff}} \in \mathbb{R}^{R\times M\times M\times 3}$ for each action name a, where R is the number of predicates and M the number of slots. For unary predicates, only the diagonal entries are learnable and off-diagonal entries are fixed to $[0 \quad -\infty \quad -\infty]$. Applying the softmax function over the last dimension yields the probabilities for each atom being an add or delete effect as

$$\begin{aligned} P^a_{\text{add},r,i,j} &= \mathrm{softmax}\big(P^a_{\text{eff}}\big)_{r,i,j,2} \\ &= \frac{\exp(P^a_{\text{eff},r,i,j,2})}{\sum_{l=1}^{3}\exp(P^a_{\text{eff},r,i,j,l})} \\ \text{and} \quad P^a_{\text{del}} &= \mathrm{softmax}\big(P^a_{\text{eff}}\big)_{\dots,3}. \end{aligned} \tag{3}$$

Given a selection matrix $\tilde{S}^a$, we then project the lifted effect probabilities P^a_{add} and $P^a_{\text{del}} \in (0,1)^{R\times M\times M}$ onto grounded atoms in adjacency matrix space, yielding $\tilde{P}^a_{\text{add}}$ and $\tilde{P}^a_{\text{del}} \in (0,1)^{R\times O\times O}$ via

$$\tilde{P}^a_{\text{add}} = (\tilde{S}^a)^\top P^a_{\text{add}}\tilde{S}^a \quad \text{and} \quad \tilde{P}^a_{\text{del}} = (\tilde{S}^a)^\top P^a_{\text{del}}\tilde{S}^a. \tag{4}$$

For each grounded atom, these describe the probabilities of it being an add or delete effect. Given the adjacency matrix $A \in \{0,1\}^{R\times O\times O}$ of state s and the projected effect probabilities, we compute a prediction of the successor-state adjacency matrix $\hat{A}' \in (0,1)^{R\times O\times O}$ via

$$\hat{A}' = A + p_{\text{pre}} \cdot \big((1-A)\odot \tilde{P}^a_{\text{add}} - A \odot \tilde{P}^a_{\text{del}}\big). \tag{5}$$

Here, the scalar $p_{\text{pre}} \in (0,1)$ expresses the probability of all preconditions being fulfilled. If they are not fulfilled, no changes will be applied. The computation of p_{pre} is detailed in the next section.

4.3 Evaluation of preconditions

Analogously to effects, each atom $p(x_1,\dots,x_k)$ of the lifted action schema $a(\vec{x})$ is either (i) not included in preconditions, (ii) a positive precondition, or (iii) a negative precondition. We therefore introduce learnable parameters $P^a_{\text{pre}} \in \mathbb{R}^{R\times M\times M\times 3}$ for each action name a and apply a softmax along the last dimension to obtain $P^a_{\text{pre}^+}$ and $P^a_{\text{pre}^-} \in (0,1)^{R\times M\times M}$. We also project them into adjacency matrix space for a given selection $\tilde{S}^a$ with

$$\begin{aligned} \tilde{P}^a_{\text{pre}^+} &= (\tilde{S}^a)^\top P^a_{\text{pre}^+}\tilde{S}^a \quad \text{and} \\ \tilde{P}^a_{\text{pre}^-} &= (\tilde{S}^a)^\top P^a_{\text{pre}^-}\tilde{S}^a. \end{aligned} \tag{6}$$

For each grounded atom, $\tilde{P}^a_{\text{pre}^+}$ and $\tilde{P}^a_{\text{pre}^-}$ describe the probabilities of it being a positive or negative precondition. The failure cases are false atoms for positive preconditions and true atoms for negative preconditions. We multiply their contrary probabilities for each atom and evaluate their conjunction for the complete state's precondition fulfillment probability as

$$\begin{aligned} p_{\text{pre}} = \prod_{r,i,j=1}^{R,O,O} \Big(\big(1-\tilde{P}^a_{\text{pre}^+}\odot(1-A)\big)\odot \\ \big(1-\tilde{P}^a_{\text{pre}^-}\odot A\big)\Big)_{r,i,j}. \end{aligned} \tag{7}$$

In practice, this product is very close to zero for a randomly initialized P^a_{pre}, which prevents any application of learned effects in Equation (5). We therefore begin with the geometric mean and schedule the root degree, with τ exponentially decaying from 1 to 0, so as to approach the product, computing

$$\begin{aligned} p_{\text{pre}} = \Bigg(\prod_{r,i,j=1}^{R,O,O} \Big(\big(1-\tilde{P}^a_{\text{pre}^+}\odot(1-A)\big)\odot \\ \big(1-\tilde{P}^a_{\text{pre}^-}\odot A\big)\Big)_{r,i,j}\Bigg)^{\frac{1}{\tau\cdot R\cdot O^2+(1-\tau)}}. \end{aligned} \tag{8}$$

4.4 Loss formulation

Our primary objective and learning signal is the correct prediction of the successor state s' from state s and action name a. This is done via Equation (5) and therefore includes the selection of action arguments and the evaluation of preconditions through p_{pre}. We compute a binary cross-entropy (BCE) loss with sum aggregation between the predicted and true successor adjacency matrices $\hat{A}'$ and A' as

$$\begin{aligned} L_{\text{adj}} &= \mathrm{BCE}\big(\hat{A}', A'\big) \\ &= \sum_{r,i,j=1}^{R,O,O} A'_{r,i,j}\cdot\log\big(\hat{A}'_{r,i,j}\big) + \\ &\qquad \big(1-A'_{r,i,j}\big)\cdot\log\big(1-\hat{A}'_{r,i,j}\big). \end{aligned} \tag{9}$$

Additionally, we seek the minimal effects that still lead to correct predictions and the most restrictive preconditions

that still keep valid state transitions applicable. To achieve this, we consider the first cases corresponding to inactive effects and preconditions, given by

$$\begin{aligned} P^a_{\text{no\_eff}} &= \text{softmax}\big(P^a_{\text{eff}}\big)_{\dots,1} \quad \text{and} \\ P^a_{\text{no\_pre}} &= \text{softmax}\big(P^a_{\text{pre}}\big)_{\dots,1}. \end{aligned} \tag{10}$$

For minimal effects, we maximize the lifted inactive case $P^a_{\text{no\_eff}}$ through a BCE loss with target 1. For maximal preconditions, we do the same to maximize $(1 - P^a_{\text{no\_pre}})$, but additionally weight each entry by the activation probabilities of its slots. Thus we compute

$$\begin{aligned} L_{\text{eff}} &= \text{BCE}\big(P^a_{\text{no\_eff}}, 1\big) \quad \text{and} \\ L_{\text{pre}} &= \text{BCE}\big(\big(w^a(w^a)^\top\big) \odot \big(1 - P^a_{\text{no\_pre}}\big), 1\big). \end{aligned} \tag{11}$$

Intuitively, minimizing L_{pre} maximizes preconditions, but the respective slots must be active for the maximization to count. Without this weighting, it would be beneficial to deactivate a slot and add arbitrary preconditions instead of keeping the slot active and learning the actual preconditions for its selection.

Notice that we use sum aggregation for the BCE loss terms above. We do not want the importance of individual elements across loss terms to depend on the relative sizes of $\hat{A}'$, $P^a_{\text{no\_eff}}$, and $P^a_{\text{no\_pre}}$. To this end, we introduce a common scaling factor $\frac{1}{N}$ for all loss terms, where N is the total number of elements in $\hat{A}'$, $P^a_{\text{no\_eff}}$, and $P^a_{\text{no\_pre}}$. Excluding off-diagonal entries for unary predicates in $P^a_{\text{no\_eff}}$ and $P^a_{\text{no\_pre}}$, N is given by

$$N = R \cdot O^2 + 2 \cdot \left(\sum_{r=1}^{R} M^{|p_r|} \right). \tag{12}$$

Here, $|p_r|$ is the arity of predicate p_r. Including this scaling factor gives rise to our main loss term L_{main} and an auxiliary loss term L_{aux} as

$$L_{\text{main}} = \frac{1}{N} L_{\text{adj}} \quad \text{and} \quad L_{\text{aux}} = \frac{1}{N}(L_{\text{eff}} + L_{\text{pre}}). \tag{13}$$

Gradient projection Correct successor-state predictions should always take precedence over minimal effects and maximal preconditions. Rather than finding a balance, we optimize L_{aux} only under the condition that L_{main} is optimized. To do this, we separately compute gradients ∇L_{aux} and ∇L_{main} with respect to all model weights and use PCGrad (Yu et al. 2020); see Appendix A.2.2.

4.5 Derivation of action schemas

After training, we can infer the complete action schema for each action name a from the learned parameters P^a_{eff}, P^a_{pre}, and w^a. First, we apply a threshold to the slot activation probabilities to keep only clearly active slots, yielding the mask $\bar{w}^a = (\sigma(w^a) > \frac{1}{2})$. The number of active slots defines the learned arity of action a. For example, with $\bar{w}^a = [1 \quad 1 \quad 0 \quad 1 \quad 0]$, we get a 3-ary action $a(x_1, x_2, x_4)$.

We then compute P^a_{add}, P^a_{del}, $P^a_{\text{pre}^+}$, and $P^a_{\text{pre}^-}$ as in Equation (3) above. The sets in the lifted action schema $\langle \text{Add}(a(\vec{x})), \text{Del}(a(\vec{x})), \text{Pre}^+(a(\vec{x})), \text{Pre}^-(a(\vec{x})) \rangle$ can then be derived by applying a high threshold, analogously to

$$\begin{aligned} \text{Add}(a(\vec{x})) = \Big\{ p_r(x_i, x_j) \;\Big|\; & \big(\big(\bar{w}^a(\bar{w}^a)^\top\big) P^a_{\text{add}}\big)_{r,i,j} > \frac{1}{2}, \\ & r \in \mathcal{P}_2,\ \ i, j \in 1, \dots, M \Big\} \\ \cup \Big\{ p_r(x_i) \;\Big|\; & \big(\big(\bar{w}^a(\bar{w}^a)^\top\big) P^a_{\text{add}}\big)_{r,i,i} > \frac{1}{2}, \\ & r \in \mathcal{P}_1,\ \ i \in 1, \dots, M \Big\}. \end{aligned} \tag{14}$$

5 Experiments

In the following, we first describe the training and evaluation setup for DIAS. We then report and discuss the main results for three action-observation variants, compare them to related work, and provide further analyses of noise robustness and an ablation on state-dependent effects.

5.1 Setup

We evaluate on 13 standard IPC domains from the *PDDL Generators* repository (Seipp, Torralba, and Hoffmann 2022) or minor variations; see Table 1. Training data is collected by a random walk on a single small problem instance using the ground-truth PDDL domain. We stop once at least $N_{\min}$ samples have been collected for each action name. During training, we draw action-balanced batches of transitions and conduct 10 runs with different random seeds per configuration. As stated in Section 3, we consider three cases for the action label a: (i) full STRIPS actions, (ii) STRIPS+ actions, or (iii) action names only. Full STRIPS actions do not require the argument-selection step from Section 4.1. For the other two cases, we fix the number of slots M, i.e., the maximum action arity, to five. For STRIPS+ actions, the selection of known arguments is forced. Domain properties, sample counts, and model and scheduling parameters are listed in Appendices A.2 and A.3.

After training, we derive action schemas following Section 4.5. Since schema arities and argument orderings may differ from the ground truth, direct comparison is not meaningful. Instead, for each test state s, we compare the sets of reachable successor states $\mathcal{S}'_{\text{true}}(s)$ and $\mathcal{S}'_{\text{pred}}(s)$ under the ground-truth and learned domains. Aggregated over all test states, this yields true positives TP, false positives FP, and false negatives FN, from which we report precision $\frac{\text{TP}}{\text{TP+FP}}$ and recall $\frac{\text{TP}}{\text{TP+FN}}$ as measures of soundness and completeness. We also report N_s for the number of runs achieving soundness with full precision, N_c for runs achieving completeness with full recall, and N_{sc} for runs that are both sound and complete. Evaluation is performed on three held-out problem instances.

5.2 Main results

Table 1 lists the main results on 13 domains. We first note that with full STRIPS action labels, e.g., `move(o3,o7,o4)`, all domains are learned perfectly in every run. This demonstrates that the optimization objective is suitable when the argument selection is given.

	Full STRIPS actions					STRIPS+ actions					Action names only				
Domain	Pr.	Re.	N_s	N_c	N_{sc}	Pr.	Re.	N_s	N_c	N_{sc}	Pr.	Re.	N_s	N_c	N_{sc}
Blocks-3	1.00	1.00	10	10	10	1.00	1.00	10	10	10	1.00	1.00	10	10	10
Delivery	1.00	1.00	10	10	10	1.00	1.00	10	10	10	1.00	1.00	10	10	10
Driverlog	1.00	1.00	10	10	10	1.00	1.00	10	10	10	0.80	0.91	8	8	8
Gripper	1.00	1.00	10	10	10	1.00	1.00	10	10	10	1.00	1.00	10	10	10
Hanoi	1.00	1.00	10	10	10	1.00	1.00	10	10	10	0.81	0.99	8	8	8
Logistics	1.00	1.00	10	10	10	0.82	1.00	7	10	7	0.94	0.97	9	8	7
Miconic	1.00	1.00	10	10	10	1.00	1.00	10	10	10	1.00	1.00	10	10	10
N-Puzzle	1.00	1.00	10	10	10	1.00	1.00	10	10	10	1.00	1.00	10	10	10
Satellite	1.00	1.00	10	10	10	1.00	0.99	1	3	0	0.74	1.00	0	10	0
Sokoban	1.00	1.00	10	10	10	1.00	1.00	10	6	6	1.00	1.00	10	6	6
Sokoban-pull	1.00	1.00	10	10	10	1.00	1.00	10	10	10	1.00	1.00	10	10	10
Spanner	1.00	1.00	10	10	10	1.00	1.00	10	10	10	1.00	1.00	10	10	10
Visitall	1.00	1.00	10	10	10	1.00	1.00	10	10	10	1.00	1.00	10	10	10

Table 1: Main results on IPC domains over 10 runs for three action observation variants. We report the average precision (Pr.) and recall (Re.), and the number of learned domains that are sound, complete, or sound and complete as N_s, N_c, and N_{sc}. Eight domains are learned perfectly in every setting. Metrics with only some or zero successful runs are highlighted in orange or red, respectively.

The variants with STRIPS+ actions, e.g., `move(o3,o4)`, or action names only, e.g., `move`, both depend on learning the argument selection via a GNN, as described in Section 4.1. Nevertheless, in 8 of the 13 domains, we learn all action schemas perfectly in every training run. Apart from *Satellite*, which is discussed below, the remaining domains are learned perfectly in at least 6 out of 10 runs. This shows that DIAS can learn correct lifted symbolic models via gradient descent.

Analyzing failures In *Sokoban*, the imperfect runs (4 out of 10) are due to dead ends, much as in symbolic methods (Gösgens, Jansen, and Geffner 2025). Dead ends such as pushing a box into a corner not only prevent reaching the goal, but also block entire regions of the state space. Such domains are problematic for learning with random walks because the resulting sampling is not informative enough. Indeed, *Sokoban-pull* extends *Sokoban* with an extra action schema that undoes pushes, and this domain, although more complex because it contains one additional action schema, is learned perfectly. *Hanoi* is learned perfectly when some action arguments are given, but it fails in 2 of the 10 runs in the action-names-only setting. The likely reason is the same: the state space of *Hanoi* resembles a Sierpinski triangle and is poorly covered by a random walk. *Driverlog* and *Logistics* feature preconditions on variables that are not part of the effects. Learning such preconditions requires identifying the variables that appear in the effects and that determine the values of the precondition variables through the appropriate atoms. This appears to explain the failures to recover the exact models in 2 and 3 of the 10 runs, respectively. *Satellite* is learned almost perfectly in the STRIPS+ setting, but not when all action arguments must be inferred. The main issue is that `take_image` is not modeled as a well-formed STRIPS action in which add and delete effects are always effective. Indeed, in the random walks, 75 % of `take_image` transitions do not change the state at all because the add and delete effects are already satisfied when the action is executed. In addition, some state transitions can be accounted for by multiple groundings of the same action.

Comparison to a symbolic method The only other approach that can learn STRIPS action schemas from traces consisting of full states and action names only, without even knowing the action arities, is the L1 algorithm by Balyo et al. (2024), which is based on multiple SAT calls. While the code for L1 is not publicly available, the authors tested it on our traces for the 13 domains above. The full results are listed in Appendix A.4.3. L1 recovers perfect models for 6 of the 13 domains: *Gripper*, *Hanoi*, *Miconic*, *Visitall*, *Delivery*, and *N-Puzzle*. On the other 7 domains, L1 misses some preconditions. As confirmed by the authors, this is due to limitations of L1, which, for example, cannot account for precondition variables that do not appear in action effects. Other approximations made to tame the resulting SAT problems explain the failures in *Blocks*, *Sokoban*, and *Sokoban-pull*.

5.3 Robustness to noise

Table 2 shows the performance of DIAS in the presence of observation noise. We consider four noise levels, where on average between one and eight randomly chosen atoms flip their observed truth value in each state transition. The noise model and the training-side adjustments specific to this setting are described in Appendix A.4.2. Some tolerance to noise is required if fully symbolic states are to be replaced by other observations, such as images, that must be mapped to symbolic states. Most domains can still be learned reasonably well up to a noise level of two flips per transition on average. At four flips, the average number of true effects per transition is exceeded for all domains, meaning that the GNN sees more noise-induced effects than real ones. Nevertheless, *Blocks* is still learned perfectly, and *Gripper* and *Miconic* recover sound domains. At eight flips per state transi-

	1 flip		2 flips		4 flips		8 flips	
Domain	Pr.	Re.	Pr.	Re.	Pr.	Re.	Pr.	Re.
Blocks-3	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.93
Delivery	0.67	0.36	1.00	0.72	0	0	0	0
Driverlog	1.00	0.43	0.39	0.39	0	0	0	0
Gripper	1.00	0.64	1.00	0.64	1.00	0.68	0.67	0.03
Hanoi	1.00	0.89	1.00	0.88	0	0	0	0
Logistics	0.42	0.99	0.42	0.96	0.32	0.82	0	0
Miconic	1.00	0.16	1.00	0.70	1.00	0.60	0	0
N-Puzzle	1.00	1.00	1.00	1.00	0.67	0.58	0	0
Satellite	0.67	0.35	0.01	0.00	0	0	0	0
Sokoban	0.96	1.00	0.79	0.97	0	0	0	0
Sokoban-pull	1.00	1.00	1.00	1.00	0.33	0.33	0	0
Spanner	1.00	1.00	0.01	1.00	0.02	0.38	0	0
Visitall	1.00	0.28	0.34	0.26	0	0	0	0

Table 2: Results under observation noise for three runs with action names only. We report the average precision (Pr.) and recall (Re.) for four noise levels from one to eight randomly flipped atoms per transition, in expectation. Dashes indicate a timeout during evaluation.

Domain	Pr.	Re.	N_s	N_c	N_{sc}
Blocks-3	1.00	1.00	10	10	10
Delivery	0.00	0.00	0	0	0
Driverlog	1.00	0.99	9	0	0
Gripper	1.00	1.00	10	10	10
Hanoi	0.00	0.00	0	0	0
Logistics	0.40	1.00	0	10	0
Miconic	1.00	1.00	10	10	10
N-Puzzle	1.00	1.00	10	10	10
Satellite	0.00	0.00	0	0	0
Sokoban	0.00	0.00	0	0	0
Sokoban-pull	0.00	0.00	0	0	0
Spanner	0.03	1.00	0	10	0
Visitall	1.00	1.00	10	10	10

Table 3: Results of the MLP variation for action names only. We report average precision (Pr.) and recall (Re.) over 10 runs. N_s, N_c, and N_{sc} denote the number of runs that are sound, complete, or both sound and complete.

tion on average, no domain is learned correctly. Robustness to noise varies across domains, likely due to differences in the structure of the action schemas. Schemas with variables that appear in preconditions but not in effects, for example, are already more challenging to learn exactly in the noise-free setting and may become even harder to learn in the presence of random flips in almost every state transition.

5.4 MLP effects vs. STRIPS effects

We also test an MLP-effects ablation against the STRIPS model. To do so, we replace the component that learns state-independent STRIPS effects, P^a_{eff}, with an MLP. The MLP uses the k objects selected by the GNN to map a substate $\hat{s}$ of s to a substate $\hat{s}'$ of the successor state s'. These substates consist of the truth values, in s and s' respectively, of all atoms defined over the selected objects. All other atoms keep their truth values in the transition from s to s'. The MLP thus learns localized effects, as in STRIPS models and slot-based dynamic models (Goyal et al. 2021). Unlike the latter, however, this dynamic model generalizes to larger numbers of objects because the MLP always sees a set of atoms, i.e., boolean variables, of the same size in both the input and the output. This type of generalization is not compatible with slot- or variable-based models for domains such as *Blocks*, where more objects mean more state variables and more possible values per variable. As shown in Table 3, although this ablation does not strictly outperform the main model, it achieves strong performance across several domains. Notably, these models were trained with the exact hyperparameters optimized for the STRIPS model and without any additional tuning; see Section 5.2. Table 3 therefore demonstrates that this architecture can, in principle, learn STRIPS effects, although not as reliably. At the same time, it can also handle state-dependent effects that would require conditional effects in the planning setting. Yet, this architecture does not yield symbolic action schemas that can be used efficiently for planning. Further details about the architecture and the experiments are given in Appendix A.4.1.

6 Conclusions

We have developed a novel neural architecture for learning STRIPS action schemas from traces in which states are fully observed but action arguments are hidden. The main challenge is to learn the action schemas while simultaneously identifying the action arguments from observed state changes, especially when preconditions involve variables that do not appear in the action effects. We evaluated the proposed architecture, DIAS, on 13 classical planning domains and found that it learns the ground-truth schemas in 8 domains in every run and in 4 more domains in most runs. We also analyzed the failure cases and robustness to noise, and compared DIAS to both a symbolic baseline and a slot-based dynamic-model variant that uses an MLP instead of STRIPS effects. In future work, we aim to use DIAS as a differentiable component for learning action schemas reliably from traces of state images and action names.

Acknowledgements

We thank Jonas Gösgens, Michael Aichmüller, and especially Niklas Jansen for insightful comments and valuable discussions. Furthermore, we are grateful to Tomáš Balyo for his cooperation in providing results of his method on our domains. The research has been supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry for Education and Research. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 885107). This project was also funded by the German Federal Ministry of Education and Research (BMBF) and the Ministry of Culture and Science of the German State of North Rhine-Westphalia (MKW) under the Excellence Strategy of the Federal Government and the Länder.

A Supplementary material and technical appendices

This appendix collects additional background, implementation details, data statistics, and additional results that complement the main paper.

A.1 Related work

The domain learning problem considered here has been studied across symbolic and neural lines of work, under different assumptions about observability, action labels, and noise, and structured representations. We discuss four related research threads and position our approach relative to them.

Symbolic model learning in classical planning The problem of learning lifted STRIPS models from state-action traces has received considerable attention (Zhuo and Kambhampati 2013; Aineto, Celorrio, and Onaindia 2019; Lamanna et al. 2021; Verma, Marpally, and Srivastava 2021; Le, Juba, and Stern 2024; Bachor and Behnke 2024; Aineto and Scala 2024; Lamanna et al. 2025). While observability of the states can be partial or noisy, in almost all cases the observations are assumed to reveal all predicates and all action arguments. Exceptions include the SAT-based approaches of Bonet and Geffner (2020) and Rodriguez et al. (2021), where only action names and state equalities need to be observed (i.e., whether two states are the same without knowing their contents). More recently, SIFT learns lifted models from action traces alone, assuming that the actions are full STRIPS actions, as in the earlier LOCM systems (Cresswell and Gregory 2011; Cresswell, McCluskey, and West 2013). Domain predicates are learned along the way, and the algorithm is provably correct and efficient. The limitation is that assuming all STRIPS action arguments are observed is often unrealistic. The recent SYNTH algorithm (Jansen, Gösgens, and Geffner 2025) instead assumes that states and non-redundant action arguments are observable, while Balyo et al. (2024) introduce a SAT-based formulation where only states and action names are observed, as in our setting.

Neural model learning in classical planning Works addressing STRIPS model learning via gradient descent include LatPlan (Asai and Fukunaga 2018; Asai et al. 2022), a transformer-based architecture (Núñez-Molina, Gómez, and Geffner), and the ROSAME 1 and 2 systems (Xi, Gould, and Thiébaux 2024, 2026). The first two learn propositional STRIPS models, while ROSAME 1 learns lifted STRIPS models from traces made up of images and full STRIPS actions, and ROSAME 2 learns from traces made up of images and action names only. The latter is closest to our work, although it addresses the more challenging setting in which only the initial and final symbolic states of the traces are given, while the intermediate symbolic states are replaced by images. The accuracy of the learned STRIPS models, however, remains limited. We aim instead at the simpler problem where all symbolic states in the traces are given, possibly with noise, with the goal of solving it nearly perfectly.

Model-based reinforcement learning Model-based reinforcement learning algorithms learn controllers by also learning (stochastic) MDP models, without making assumptions about their internal structure (Sutton and Barto 2018). In the tabular setting, they result in flat state models with transition probabilities obtained from simple counts (Brafman and Tennenholtz 2003). In some cases, a first-order state language is assumed, but the state predicates are given (Diuk, Cohen, and Littman 2008; Zettlemoyer, Pasula, and Kaelbling 2005). In more recent approaches, the learned dynamics is represented not in compact languages such as STRIPS or probabilistic PDDL (Younes et al. 2005), but in terms of deep neural networks (Micheli, Alonso, and Fleuret 2023; Hafner et al. 2021; Burchi and Timofte 2025). These models are useful for control, but they are not lifted and thus do not generalize naturally to new objects.

Slot-based dynamic model learning Slot-based dynamic models in deep learning, such as RIMs (Goyal et al. 2021) and related approaches, reduce states to the values of a fixed number of slots that play the role of state variables. It is further assumed that only a few slots change value at any time point, and that these changes are a function of their previous values. These models define an implicit representation language over state variables, but the main evaluation focus is usually predictive performance rather than whether the intended symbolic structure is actually recovered. Our architecture differs in that it does not select a subset of state variables at each point, but a subset of objects, which in turn define a fixed set of atoms or Boolean variables. This distinction matters for generalization to more objects, because then both the number of state variables and the number of values they can take may change. This limitation does not arise in our setting, where the learned lifted representations generalize to an arbitrary number of objects.

A.2 Model and training details

This subsection gathers implementation and training details that support reproducibility but are not central to the main narrative.

A.2.1 Rectangular Sinkhorn normalisation in log space

The argument-selection step in Section 4.1 relies on a rectangular variant of the Sinkhorn algorithm introduced by (Brun et al. 2022). In the following, we discuss two design choices in our implementation.

No slack column The correspondence matrix $C^a \in \mathbb{R}^{M\times O}$ is rectangular: the number of slots M is a fixed hyperparameter, while the number of objects O varies across domains. Standard Sinkhorn normalisation targets a doubly-stochastic matrix, which is necessarily square, and does not directly apply here. Brun et al. (2022) address this via the Linear Sum Assignment Problem with Edition (LSAPE), expanding the assignment matrix with both a slack row and a slack column to encode the likelihood of leaving an element unmatched. We adopt only the slack row, extending C^a to $\hat{C}^a \in \mathbb{R}^{(M+1)\times O}$ with the new row initialized to zero. The slack-column role of leaving slots unmatched is instead taken by the learned slot activations w^a in Section 4.1, which softly suppress the selected objects in a slot via $\sigma(w^a) \to 0$.

During alternating normalisation, the slack row is exempt from row normalisation, so it acts as a free absorber for the residual column mass of objects not matched to any real slot. After convergence, the slack row is discarded. The remaining M rows form the soft assignment $S^a \in (0,1)^{M\times O}$, which has row sums $= 1$ and column sums ≤ 1. Each real slot is therefore guaranteed to have a distribution over objects, while objects may remain unselected.

Log-space iterations Naive Sinkhorn operates on probability values that quickly underflow to zero in single precision for large matrices or strong attention scores. We instead perform all iterations in log-space. Let $u \in \mathbb{R}^{M+1}$ and $v \in \mathbb{R}^O$ denote log-domain row and column scaling vectors, i.e., $\exp(u_i)$ scales row i and $\exp(v_j)$ scales column j. We initialise both to $\mathbf{0}$, which is equivalent to no initial scaling ($\exp(\mathbf{0}) = \mathbf{1}$ in probability space). One Sinkhorn iteration reads

$$u_i \leftarrow -\operatorname{logsumexp}_j\big(\hat{C}^a_{ij} + v_j\big), \quad i = 1,\dots,M, \tag{15}$$

$$u_{M+1} \leftarrow 0, \tag{16}$$

$$v_j \leftarrow -\operatorname{logsumexp}_i\big(\hat{C}^a_{ij} + u_i\big), \quad j = 1,\dots,O. \tag{17}$$

We iterate until the ℓ_∞-change of both u and v falls below a tolerance ε, and recover the assignment as

$$S^a_{ij} = \exp\big(u_i + \hat{C}^a_{ij} + v_j\big), \quad i = 1,\dots,M,\ \ j = 1,\dots,O. \tag{18}$$

The complete procedure is summarized in Algorithm 1.

Algorithm 1: Log-space rectangular Sinkhorn with slack row.

Require: logit matrix $\hat{C}^a \in \mathbb{R}^{(M+1)\times O}$, tolerance ε, max iterations $T_{\max}$

$u \leftarrow \mathbf{0} \in \mathbb{R}^{M+1}, \quad v \leftarrow \mathbf{0} \in \mathbb{R}^O$
for $t = 1,\dots,T_{\max}$ **do**
  $u_{\text{prev}} \leftarrow u, \quad v_{\text{prev}} \leftarrow v$
  for $i = 1,\dots,M$ **do** ▷ real slot rows
    $u_i \leftarrow -\operatorname{logsumexp}_j\big(\hat{C}^a_{ij} + v_j\big)$
  end for
  $u_{M+1} \leftarrow 0$ ▷ slack row: dual pinned to zero
  for $j = 1,\dots,O$ **do** ▷ all columns
    $v_j \leftarrow -\operatorname{logsumexp}_i\big(\hat{C}^a_{ij} + u_i\big)$
  end for
  if $\max\big(\|u - u_{\text{prev}}\|_\infty,\ \|v - v_{\text{prev}}\|_\infty\big) < \varepsilon$ **then**
    break
  end if
end for
return $S^a_{ij} = \exp\big(u_i + \hat{C}^a_{ij} + v_j\big), \quad i = 1,\dots,M,\ j = 1,\dots,O$

All Sinkhorn computations are run in `float32` regardless of the surrounding mixed-precision training context, since `float16` overflows for the logit magnitudes typical in this setting.

A.2.2 Shared model and training hyperparameters

Shared hyperparameters All runs share the same model and optimiser configuration. The object-embedding dimension is $d = 32$ and the number of slots is $M = 5$ (for the full-action-knowledge variant, M is set per action to its ground-truth arity). We use AdamW (Loshchilov and Hutter 2019) with learning rate $\eta = 5 \times 10^{-3}$, batch size 200, and 10 000 gradient steps with stratified action-balanced batches. The Sinkhorn temperature is fixed to 1. The schedule for τ in the precondition aggregation decays exponentially, with $\tau = 1$ at step 0 and $\tau = 0.1$ at step 500, and continues to decay toward 0 for the remainder of training. We start at $\tau = 1$ (the geometric mean) because, with randomly initialized precondition logits, the true product across $R \cdot O^2$ entries is almost zero and provides no useful gradient. The geometric mean keeps p_{pre} in a usable range early on, and the gradual decay toward 0 recovers $\prod_{r,i,j}$ once the parameters approach an informative schema. The nodes of the R-GCN are initialized randomly: we set the first half of the $d = 32$ dimensions to $0.1 \cdot \mathcal{N}(0,1)$ noise and leave the remaining half at zero, so the R-GCN can break symmetry between objects while retaining a well-defined zero anchor.

Gradient projection and auxiliary loss weighting As described in Section 4.4, we use PCGrad (Yu et al. 2020) to prevent L_{aux} from interfering with the main training task. PCGrad splits ∇L_{aux} into two components: (i) $\nabla L_{\text{aux}\parallel\text{main}}$ parallel to ∇L_{main} and (ii) $\nabla L_{\text{aux}\perp\text{main}}$ orthogonal to ∇L_{main}. If the parallel gradient component opposes the main gradient, we remove it, effectively projecting ∇L_{aux} onto the hyperplane orthogonal to ∇L_{main}. This gives a projected auxiliary gradient

$$\tilde{\nabla} L_{\text{aux}} = \begin{cases} \nabla L_{\text{aux}\perp\text{main}} & \text{if } \nabla L_{\text{aux}\parallel\text{main}} \cdot \nabla L_{\text{main}} < 0 \\ \nabla L_{\text{aux}} & \text{else} \end{cases}. \tag{19}$$

We then cap its norm by the norm of the main gradient and compute the total gradient with a weighting factor α as

$$\nabla L_{\text{total}} = \nabla L_{\text{main}} + \alpha \cdot \min\Big(\frac{\|\nabla L_{\text{main}}\|}{\|\tilde{\nabla} L_{\text{aux}}\|}, 1\Big) \cdot \tilde{\nabla} L_{\text{aux}}. \tag{20}$$

The norm cap is motivated as follows: when L_{main} is near a minimum, ∇L_{main} can become noisy and very small compared to ∇L_{aux}, so without capping the resulting gradient may push the parameters away from the found minimum. The scalar α thus effectively controls the relative magnitude of ∇L_{aux} to ∇L_{main}. We have found empirically that the model is very sensitive to this parameter. We therefore tune α per domain on validation data and report the final per-domain values in Appendix A.3.3.

A.2.3 On state-dependent preconditions

In the state-dependent effects ablation (Section 5.4), preconditions remain static learnable parameters P^a_{pre} rather than being predicted by an MLP based on the state. The reason is that state-dependent precondition parameters admit a trivial solution in the absence of negative samples.

Concretely, suppose P^a_{pre} is replaced with an MLP. The precondition fulfillment probability p_{pre} from Equation (5) is maximized when all atoms in s are declared positive preconditions: the model simply copies the state into the preconditions, ensuring they are always trivially satisfied ($p_{\text{pre}} \to 1$). This is consistent with every observed transition, because all training transitions are applicable by construction.

The fix is to include negative samples (transitions where the action is inapplicable), so the model is penalized for incorrectly predicting $p_{\text{pre}} > 0$. We leave this extension to future work.

A.2.4 Compute requirements

All training runs were executed on our institute's compute cluster, on a single NVIDIA A10 (24 GB) or NVIDIA L40S (48 GB) GPU, with 8 CPU cores and up to 128 GB of host RAM per run (typically 64 GB). Depending on the experiment, we either dedicated one GPU per run or packed several runs onto the same GPU in parallel. Wall-clock time was capped at 4 hours per Slurm task, but most individual runs completed within 2 hours. Beyond the runs reported in this paper, the total compute consumed during the project is substantially larger due to hyperparameter tuning, architectural exploration, and failed experiments that did not make it into the paper.

A.3 Training and evaluation data

This subsection details how we collect training, validation, and test transitions, and lists the per-domain configuration in Appendix A.3.3.

A.3.1 Training data

For each domain, we collect training and validation transitions via a random walk in the first problem instance listed in our code repository (see Appendix A.3.3 for object counts). The random walk continues until at least $N^{\text{tr}}_{\text{min}}$ transitions per action name are collected for the training split and $N^{\text{val}}_{\text{min}}$ for the validation split, with at most $N^{\text{tr}}_{\text{max}}$ and $N^{\text{val}}_{\text{max}}$ kept per action. The per-domain bounds are listed in Appendix A.3.3.

A.3.2 Test data

For every domain, we evaluate on three larger held-out problem instances (problems 2–4 in the lists in our code repository), from which we sample 1500 test states in total via BFS. Precision and recall are accumulated over all test states. The number of objects in each of the three test instances is reported alongside the training instance in Appendix A.3.3.

A.3.3 Per-domain hyperparameters and statistics

Tables 4 to 16 list, for each of the 13 domains, the action and predicate signatures of the original PDDL domain (each annotated with its arity), the number of objects in the training and the three test problem instances, the per-action sample bounds, and the auxiliary-loss weight α.

Actions	`stack/2`, `newtower/2`, `move/3`
Predicates	`clear/1`, `on-table/1`, `on/2`, `eq/2`
Objects (tr, te 1–3)	5, 10, 20, 40
N^{tr}	$N_{\text{min}} = 100$, $N_{\text{max}} = 1000$
N^{val}	$N_{\text{min}} = 50$, $N_{\text{max}} = 500$
α	1.0

Table 4: Domain *Blocks-3*.

Actions	`move-up/3`, `move-down/3`, `move-left/3`, `move-right/3`
Predicates	`tile/1`, `cell/1`, `at/2`, `blank/1`, `above/2`, `right/2`
Objects (tr, te 1–3)	17, 31, 49, 71
N^{tr}	$N_{\text{min}} = 100$, $N_{\text{max}} = 1000$
N^{val}	$N_{\text{min}} = 50$, $N_{\text{max}} = 500$
α	1.0

Table 5: Domain *N-Puzzle*.

Actions	`pick-package/3`, `drop-package/3`, `move/3`
Predicates	`at/2`, `carrying/2`, `empty/1`, `adjacent/2`
Objects (tr, te 1–3)	15, 21, 45, 81
N^{tr}	$N_{\text{min}} = 1000$, $N_{\text{max}} = 2000$
N^{val}	$N_{\text{min}} = 500$, $N_{\text{max}} = 1000$
α	0.1

Table 6: Domain *Delivery*.

Actions	`LOAD-TRUCK/3`, `UNLOAD-TRUCK/3`, `BOARD-TRUCK/3`, `DISEMBARK-TRUCK/3`, `DRIVE-TRUCK/4`, `WALK/3`
Predicates	`at/2`, `in/2`, `driving/2`, `link/2`, `path/2`, `empty/1`
Objects (tr, te 1–3)	16, 24, 48, 89
N^{tr}	$N_{\text{min}} = 2000$, $N_{\text{max}} = 3000$
N^{val}	$N_{\text{min}} = 1000$, $N_{\text{max}} = 1500$
α	0.01

Table 7: Domain *Driverlog*.

Actions	`move/2`, `pick/3`, `drop/3`
Predicates	`room/1`, `ball/1`, `gripper/1`, `at-robby/1`, `at/2`, `free/1`, `carry/2`, `eq/2`
Objects (tr, te 1–3)	10, 12, 22, 42
N^{tr}	$N_{\text{min}} = 200$, $N_{\text{max}} = 1000$
N^{val}	$N_{\text{min}} = 100$, $N_{\text{max}} = 500$
α	1.0

Table 8: Domain *Gripper*.

Actions	`move/3`
Predicates	`clear/1`, `on/2`, `smaller/2`
Objects (tr, te 1–3)	9, 15, 27, 51
N^{tr}	$N_{\text{min}} = 1000$, $N_{\text{max}} = 1000$
N^{val}	$N_{\text{min}} = 500$, $N_{\text{max}} = 500$
α	0.3

Table 9: Domain *Hanoi*.

Actions	`load/3`, `unload/3`, `drive/4`, `fly/3`
Predicates	`object/1`, `truck/1`, `airplane/1`, `vehicle/1`, `location/1`, `airport/1`, `city/1`, `loc/2`, `at/2`, `in/2`
Objects (tr, te 1–3)	15, 24, 36, 48
N^{tr}	$N_{\min} = 100$, $N_{\max} = 1000$
N^{val}	$N_{\min} = 50$, $N_{\max} = 500$
α	0.3

Table 10: Domain *Logistics*.

Actions	`unboard/2`, `board/2`, `move_up/2`, `move_down/2`
Predicates	`person/1`, `floor/1`, `lift_pos/1`, `in_lift/1`, `in_floor/2`, `above/2`
Objects (tr, te 1–3)	10, 10, 20, 40
N^{tr}	$N_{\min} = 100$, $N_{\max} = 1000$
N^{val}	$N_{\min} = 50$, $N_{\max} = 500$
α	1.0

Table 11: Domain *Miconic*.

Actions	`turn_to/3`, `switch_on/2`, `switch_off/2`, `calibrate/3`, `take_image/4`
Predicates	`on_board/2`, `supports/2`, `pointing/2`, `power_avail/1`, `power_on/1`, `calibrated/1`, `have_image/2`, `calibration_target/2`
Objects (tr, te 1–3)	36, 72, 80, 89
N^{tr}	$N_{\min} = 100$, $N_{\max} = 1000$
N^{val}	$N_{\min} = 50$, $N_{\max} = 500$
α	1.0

Table 12: Domain *Satellite*.

Actions	`move-player/2`, `push-box/3`
Predicates	`has_player/1`, `has_box/1`, `adjacent/2`, `adjacent_2/2`
Objects (tr, te 1–3)	16, 36, 49, 81
N^{tr}	$N_{\min} = 20$, $N_{\max} = 1000$
N^{val}	$N_{\min} = 10$, $N_{\max} = 500$
α	1.0

Table 13: Domain *Sokoban*.

Actions	`move-player/2`, `push-box/3`, `pull-box/3`
Predicates	`has_player/1`, `has_box/1`, `adjacent/2`, `adjacent_2/2`
Objects (tr, te 1–3)	16, 36, 49, 81
N^{tr}	$N_{\min} = 100$, $N_{\max} = 1000$
N^{val}	$N_{\min} = 50$, $N_{\max} = 500$
α	1.0

Table 14: Domain *Sokoban-pull*.

Actions	`walk/3`, `pickup_spanner/3`, `tighten_nut/4`
Predicates	`at/2`, `carrying/2`, `useable/1`, `link/2`, `tightened/1`, `loose/1`
Objects (tr, te 1–3)	28, 35, 67, 88
N^{tr}	$N_{\min} = 200$, $N_{\max} = 1000$
N^{val}	$N_{\min} = 100$, $N_{\max} = 500$
α	0.1

Table 15: Domain *Spanner*.

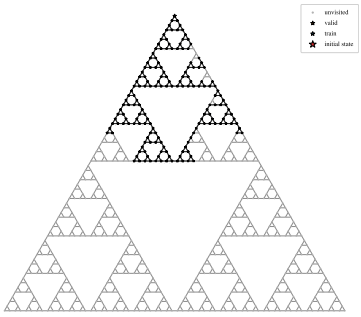
Actions	`move/2`
Predicates	`connected/2`, `at-robot/1`, `visited/1`
Objects (tr, te 1–3)	25, 36, 49, 64
N^{tr}	$N_{\min} = 100$, $N_{\max} = 1000$
N^{val}	$N_{\min} = 50$, $N_{\max} = 500$
α	1.0

Table 16: Domain *Visitall*.

A.3.4 Hanoi state-space coverage

The state space of *Hanoi* has the structure of a Sierpiński triangle: every legal configuration of n discs corresponds to a node, and the three subtrees rooted at moving the largest disc connect only at their corner states. This self-similar layout means that a random walk tends to remain inside one of the recursive sub-triangles for long stretches and crosses the narrow corner connections only rarely. Figure 2 illustrates this for the `hanoi-3-6` instance with $3^6 = 729$ states: a balanced random walk of 1500 steps visits only 174 unique states out of 729, leaving large parts of the state space unobserved.

Figure 2: State space of *Hanoi* with 6 disks (729 unique states). The balanced random walk with 1500 steps visited only 174 of them. The recursive Sierpiński structure causes the walk to be trapped in sub-triangles, leaving the rest of the state space unobserved.



A.4 Ablations

This subsection provides additional details for the ablations summarized in the main paper.

A.4.1 MLP-effects ablation: architecture and training

This section provides the technical details of the MLP-effects ablation discussed in Section 5.4. We replace the learned static effect parameters P^a_{eff} with a shared MLP that produces a per-action readout, while preconditions remain modeled as static learnable parameters P^a_{pre} (see Appendix A.2.3 for why preconditions cannot also be made state-dependent in our setting).

Training The R-GCN input is unchanged: states s and s' are embedded into a single graph and the resulting keys K are used to compute the selection matrix $\tilde{S}^a$ via the same Sinkhorn mechanism as in the main model. We then extract the lifted state adjacency matrix for the selected objects and compute the state-dependent effect logits as

$$\begin{aligned} \tilde{A}^a &= \tilde{S}^a A (\tilde{S}^a)^\top \in (0,1)^{R\times M\times M} \quad \text{and} \\ P^a_{\text{eff}} &= \text{MLP}^a\big(\tilde{A}^a\big) \in \mathbb{R}^{R\times M\times M\times 3}, \end{aligned} \tag{21}$$

where MLP^a denotes the output slice of a single shared MLP corresponding to action a. The effect probabilities P^a_{add} and P^a_{del} are derived as in Equation (3), and the projection and state transition proceed as in Equation (5).

Evaluation At evaluation time, the R-GCN is not used, as the successor state s' is unknown. Instead, the learned static preconditions P^a_{pre} are used to identify all applicable grounded actions for a given test state. For each valid grounding, we construct a hard selection matrix from the object-to-slot bindings and query the MLP to predict the successor state.

Hyperparameters The MLP-effects models are trained with the exact hyperparameters used for the main STRIPS-effects model (Appendix A.2.2), without any additional tuning.

A.4.2 Noise model and training adjustments

This section provides the technical details of the noise-robustness experiments in Section 5.3.

Noise model We perturb the training states' adjacency matrices by randomly deleting and adding atoms. The amount of noise is controlled by the expected number of changed atoms per transition $\langle s, s'\rangle$. Within each relation, additions and removals are balanced in expectation, and changes are concentrated on relations where both true and false atoms are available, so very sparse or very dense relations are perturbed less.

Training adjustments For these experiments we increase the smallest per-action minimum sample count to $N_{\min} = 200$ and reduce the batch size to 50 to limit the effect of gradient projection (see Appendix A.2.2). Since the true product from Equation (8) is too strict as a precondition check under noise, we additionally raise the floor of the scheduling parameter τ to 0.1. Also, we raise the threshold for action schema derivation in Equation (14) to 0.75.

A.4.3 Detailed comparison with L1

We compare against the L1 algorithm of Balyo et al. (2024), evaluating it in the same setting as DIAS but for one random trace only. The resulting precision and recall values are summarized in Table 17.

Domain	Precision	Recall
Blocks-3	0.8636	1.0000
Delivery	1.0000	1.0000
Driverlog	0.1924	1.0000
Gripper	1.0000	1.0000
Hanoi	1.0000	1.0000
Logistics	0.3396	1.0000
Miconic	1.0000	1.0000
N-Puzzle	1.0000	1.0000
Satellite	0.6686	1.0000
Sokoban	0.8962	1.0000
Sokoban-pull	0.8618	1.0000
Spanner	0.0049	1.0000
Visitall	1.0000	1.0000

Table 17: Results for L1 (Balyo et al. 2024). Precision and recall per domain, aggregated across problem instances 1–3.

A.5 Qualitative example: learned *Blocks-3* operators

Table 18 compares the ground-truth blocksworld operators with the operators we learn on *Blocks-3*. Variables have been re-aligned across the two columns so that name differences (e.g. `?bm` vs. `?x1`) do not appear as differences and only literals that are genuinely missing on one side are highlighted.

References

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

Aineto, D.; and Scala, E. 2024. Action Model Learning with Guarantees. *arXiv preprint arXiv:2404.09631*.

Asai, M.; and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI*.

Asai, M.; Kajino, H.; Fukunaga, A.; and Muise, C. 2022. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, 74: 1599–1686.

Bachor, P.; and Behnke, G. 2024. Learning Planning Domains from Non-Redundant Fully-Observed Traces: Theoretical Foundations and Complexity Analysis. In *Proc. AAAI*, 20028–20035.

Balyo, T.; Suda, M.; Chrpa, L.; Šafránek, D.; Gocht, S.; Dvořák, F.; Barták, R.; and Youngblood, G. M. 2024. Planning domain model acquisition from state traces without action parameters. *arXiv preprint arXiv:2402.10726*.

Bonet, B.; and Geffner, H. 2020. Learning first-order symbolic representations for planning from the structure of the state space. In *Proc. ECAI*.

Brafman, R.; and Tennenholtz, M. 2003. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3: 213–231.

Brun, L.; Gaüzère, B.; Renton, G.; Bougleux, S.; and Yger, F. 2022. A Differentiable Approximation for the Linear Sum Assignment Problem with Edition. In *2022 26th International Conference on Pattern Recognition (ICPR)*, 3822–3828.

Original	Learned
(:action stack :parameters (?bm ?bt) :precondition (and (clear ?bm) (clear ?bt) (on-table ?bm) (not (eq ?bm ?bt))) :effect (and (not (clear ?bt)) (not (on-table ?bm)) (on ?bm ?bt)))	(:action learned_stack :parameters (?x1 ?x3) :precondition (and (clear ?x1) (clear ?x3) (on-table ?x1) (not (eq ?x1 ?x3)) (eq ?x1 ?x1) (eq ?x3 ?x3) (not (eq ?x3 ?x1)) (not (on ?x1 ?x1)) (not (on ?x1 ?x3)) (not (on ?x3 ?x1)) (not (on ?x3 ?x3))) :effect (and (not (clear ?x3)) (not (on-table ?x1)) (on ?x1 ?x3)))
(:action newtower :parameters (?bm ?bf) :precondition (and (clear ?bm) (on ?bm ?bf) (not (eq ?bm ?bf))) :effect (and (not (on ?bm ?bf)) (on-table ?bm) (clear ?bf)))	(:action learned_newtower :parameters (?x2 ?x3) :precondition (and (clear ?x3) (on ?x3 ?x2) (not (eq ?x3 ?x2)) (eq ?x2 ?x2) (eq ?x3 ?x3) (not (clear ?x2)) (not (eq ?x2 ?x3)) (not (on ?x2 ?x2)) (not (on ?x2 ?x3)) (not (on ?x3 ?x3)) (not (on-table ?x3))) :effect (and (not (on ?x3 ?x2)) (on-table ?x3) (clear ?x2)))
(:action move :parameters (?bm ?bf ?bt) :precondition (and (clear ?bm) (clear ?bt) (on ?bm ?bf) (not (eq ?bm ?bt)) (not (eq ?bm ?bf)) (not (eq ?bf ?bt))) :effect (and (not (clear ?bt)) (not (on ?bm ?bf)) (on ?bm ?bt) (clear ?bf)))	(:action learned_move :parameters (?x1 ?x2 ?x4) :precondition (and (clear ?x1) (clear ?x2) (on ?x1 ?x4) (not (eq ?x1 ?x2)) (not (eq ?x1 ?x4)) (not (eq ?x4 ?x2)) (eq ?x1 ?x1) (eq ?x2 ?x2) (eq ?x4 ?x4) (not (clear ?x4)) (not (eq ?x2 ?x1)) (not (eq ?x2 ?x4)) (not (eq ?x4 ?x1)) (not (on ?x1 ?x1)) (not (on ?x1 ?x2)) (not (on ?x2 ?x1)) (not (on ?x2 ?x2)) (not (on ?x2 ?x4)) (not (on ?x4 ?x1)) (not (on ?x4 ?x2)) (not (on ?x4 ?x4)) (not (on-table ?x1))) :effect (and (not (clear ?x2)) (not (on ?x1 ?x4)) (on ?x1 ?x2) (clear ?x4)))

Table 18: Original *Blocks-3* operators (left) versus the operators learned by our model (right). Common literals are shown in black. Literals that appear only on one side are highlighted.

Burchi, M.; and Timofte, R. 2025. Learning Transformer-based World Models with Contrastive Predictive Coding. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

Cresswell, S.; and Gregory, P. 2011. Generalised Domain Model Acquisition from Action Traces. *Proc. 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.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, 240–247.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham: Springer International Publishing. ISBN 978-3-031-00436-0 978-3-031-01564-9.

Ghallab, M.; Nau, D.; and Traverso, P. 2025. *Acting, Planning, and Learning*. Cambridge University Press. ISBN 9781009579384.

Gösgens, J.; Jansen, N.; and Geffner, H. 2025. Learning Lifted STRIPS Models from Action Traces Alone: A Simple, General, and Scalable Solution. In *Proc. ICAPS*.

Goyal, A.; and Bengio, Y. 2022. Inductive Biases for Deep Learning of Higher-Level Cognition. *Proceedings of the Royal Society A*, 478(2266): 20210068.

Goyal, A.; Lamb, A.; Hoffmann, J.; Sodhani, S.; Levine, S.; Bengio, Y.; and Schölkopf, B. 2021. Recurrent Independent Mechanisms. In *International Conference on Learning Representations*.

Hafner, D.; Lillicrap, T.; Norouzi, M.; and Ba, J. 2021. Mastering atari with discrete world models. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

Jansen, N.; Gösgens, J.; and Geffner, H. 2025. Learning Lifted Action Models from Traces of Incomplete Actions and States. *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 22(1): 832–842.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning*, 379–389. Hanoii, Vietnam: International Joint Conferences on Artificial Intelligence Organization. ISBN 978-1-956792-99-7.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; Traverso, P.; et al. 2021. Online Learning of Action Models for PDDL Planning. In *IJCAI*, 4112–4118.

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

Lamb, A.; He, D.; Goyal, A.; Ke, G.; Liao, C.-F.; Ravanelli, M.; and Bengio, Y. 2021. Transformers with Competitive Ensembles of Independent Mechanisms. *arXiv preprint arXiv:2103.00336*.

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

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Madan, K.; Ke, N. R.; Goyal, A.; Schölkopf, B.; and Bengio, Y. 2021. Fast and Slow Learning of Recurrent Independent Mechanisms. In *International Conference on Learning Representations*.

Micheli, V.; Alonso, E.; and Fleuret, F. 2023. Transformers are Sample-Efficient World Models. In *Int. Conf. on Learning Representations (ICLR)*.

Núñez-Molina, C.; Gómez, V.; and Geffner, H. ???? From Next Token Prediction to (STRIPS) World Models. In *Proc. KR 2026*.

Rodriguez, I. D.; Bonet, B.; Romero, J.; and Geffner, H. 2021. Learning First-Order Representations for Planning from Black Box States: New Results. In *Proc. KR*, 539–548.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; van den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling Relational Data with Graph Convolutional Networks. In Gangemi, A.; Navigli, R.; Vidal, M.-E.; Hitzler, P.; Troncy, R.; Hollink, L.; Tordai, A.; and Alam, M., eds., *The Semantic Web*, 593–607. Cham: Springer International Publishing. ISBN 978-3-319-93417-4.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL Generators. Zenodo.

Sutton, R. S.; and Barto, A. 2018. *Reinforcement learning: an introduction*. The MIT Press. 2nd edition.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; ukasz Kaiser, Ł.; and Polosukhin, I. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the right questions: Learning interpretable action models through query answering. In *Proc. AAAI*, 12024–12033.

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

Xi, K.; Gould, S.; and Thiébaux, S. 2026. Learning Lifted Action Models from Unsupervised Visual Traces.

Younes, H.; Littman, M.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, 24: 851–887.

Yu, T.; Kumar, S.; Gupta, A.; Levine, S.; Hausman, K.; and Finn, C. 2020. Gradient Surgery for Multi-Task Learning. In *Advances in Neural Information Processing Systems*, volume 33, 5824–5836. Curran Associates, Inc.

Zettlemoyer, L. S.; Pasula, H.; and Kaelbling, L. P. 2005. Learning planning rules in noisy stochastic worlds. In *AAAI*, 911–918.

Zhuo, H. H.; and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proc. IJCAI*.