

STRIPS-WM: Learning Grounded Propositional STRIPS-style World Models from Images

Abhiroop Ajith

Department of Robotics Engineering
Worcester Polytechnic Institute
United States
aajith@wpi.edu

Constantinos Chamzas

Department of Robotics Engineering
Worcester Polytechnic Institute
United States
cchamzas@wpi.edu

Abstract: Robots performing long-horizon visual manipulation observe high-dimensional images, but successful plans depend on action-relevant facts: what can be done now and what changes afterward. Classical task planners exploit this structure through symbolic operators with preconditions and effects, but obtaining such representations from raw visual experience remains challenging. We study a visual task-planning setting in which a robot receives only action-image transitions: the current image, executed high-level action, and resulting image. At test time, given start and goal images, the robot must produce a sequence of high-level actions that reaches the goal. We introduce STRIPS-WM, a framework for learning image-grounded STRIPS-style world models directly from visual transitions. STRIPS-WM first induces a finite abstract transition graph from images, then learns binary predicates and propositional operators that explain this graph. The learned predicates are grounded back into perception through a visual classifier, enabling classical planning directly from novel start and goal images. Experiments on visual rearrangement tasks show that STRIPS-WM improves image-to-plan success over the tested visual rollout, latent graph-search, and latent-symbolic baselines.

Keywords: Learning Logic-Based Representations, Visual Task Planning

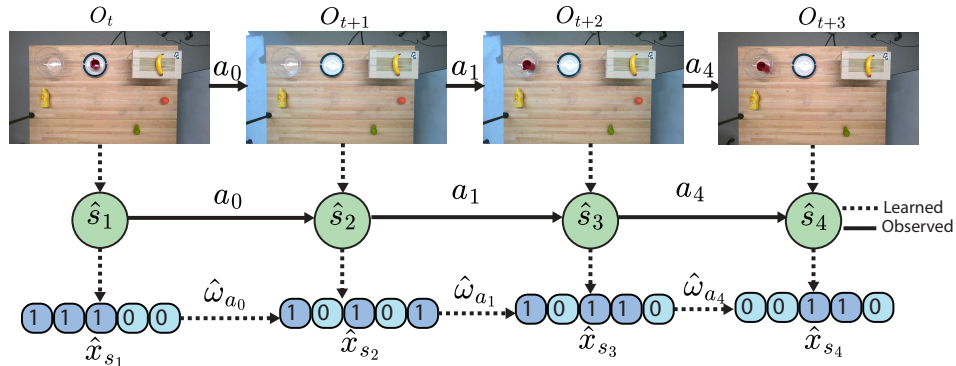


Figure 1: Three planning spaces for the same visual task. **Top:** In image space, the robot observes RGB states connected by high-level action identifiers, $o_t \xrightarrow{a_t} o_{t+1}$; planning here would require reasoning over future images. **Middle:** In the learned task-graph space, observations are compressed into abstract states $\hat{s}$, yielding action-indexed transitions $\hat{s}_t \xrightarrow{a_t} \hat{s}_{t+1}$. This enables graph search, but requires a flat enumeration of abstract states and edges. **Bottom:** In the learned STRIPS space, abstract states receive predicate vectors $\hat{x}_s$, and each action identifier a receives a grounded operator $\hat{w}_a$ with preconditions and effects. This task representation enables classical task planning over predicate states rather than images or enumerated graph edges.

1 Introduction

Robots perceive their surroundings through high-dimensional observations such as RGB images. Yet, long-horizon manipulation tasks often admit much simpler abstract descriptions [1]. In the table-setting example of Fig. 1, the robot must reach a target arrangement by executing a sequence

of high-level actions. Planning directly in image space would require predicting long sequences of future images [2]. However, the pixel level is too fine-grained for task planning: most visual variation is irrelevant to whether an action can be applied or what it will change. What matters is the smaller set of action-relevant conditions that hold in the scene and how actions transform them [3].

Classical task planning formalizes this idea through symbolic states and transition models. In STRIPS planning, states are represented by symbolic facts (predicates), and actions are represented by operators with preconditions and add/delete effects [4]. Given such a model, the remaining search problem is exactly the kind of discrete planning problem that off-the-shelf task planners can solve, rather than predicting future images. This structure is illustrated at the bottom of Fig. 1: instead of raw observations, STRIPS represents the world using symbolic states s with predicate vectors $\mathbf{x}_s$; instead of treating actions only as identifiers a , it represents them as operators ω_a with preconditions and effects. The central challenge is therefore to learn this symbolic world model from data.

We propose STRIPS-WM to address this challenge directly. Given only action-indexed visual transitions $\mathcal{D}_O = \{(o_i, a_i, o'_i)\}_{i=1}^N$, we learn an image-grounded STRIPS world model. As illustrated from the top to the bottom of Fig. 1, this means learning a mapping from observations to predicate states $\hat{\mathbf{x}}_s$, together with one operator $\hat{\omega}_a$ for each action identifier.

STRIPS-WM learns this model in three stages, mirroring the three levels shown in Fig. 1. First, an action-conditioned visual dynamics model induces an image-grounded task graph, converting visual transitions $o_t \xrightarrow{a_t} o_{t+1}$ into abstract transitions $\hat{s}_t \xrightarrow{a_t} \hat{s}_{t+1}$. Second, a CP-SAT solver assigns binary predicate vectors $\hat{\mathbf{x}}_s \in \{0, 1\}^m$ to graph nodes and learns one grounded operator $\hat{\omega}_a$ per action identifier, so that graph edges are explained by STRIPS-style preconditions and add/delete effects. Third, a visual predicate classifier grounds these learned predicate assignments back to images, enabling classical planning directly from novel start and goal observations.

Our contributions are: (i) a framework for learning image-grounded propositional STRIPS world models from action-indexed visual transitions, without object-centric supervision or hand-designed predicates; (ii) a CP-SAT formulation that jointly learns binary predicate assignments for abstract states and grounded STRIPS operators explaining the observed transition graph; and (iii) an image-to-plan pipeline that grounds the learned predicates in perception and enables classical planning from novel start and goal images. Experiments on simulated and real rearrangement tasks show improved long-horizon planning over visual rollout, latent graph-search, and latent-symbolic baselines.

2 Related Work

Reconstruction-based visual planning. Such methods plan from images by learning forward predictive models and searching over candidate action sequences, either by predicting future images [2, 5, 6, 7, 8, 9], or by planning through imagined latent rollouts from pixels [10, 11, 12]. These methods optimize representations for visual prediction, often requiring reconstruction of pixel-level observations, which can preserve details irrelevant to action applicability and effects. STRIPS-WM does not reconstruct or predict images.

Visual planning without decoding. These methods avoid pixel reconstruction by learning continuous latent representations with contrastive, predictive, or goal-conditioned objectives [13, 14, 15, 16, 17, 18]. While these methods discard some irrelevant visual detail, their latent spaces lack the logical structure of symbolic representations. Planning depends on neural forward prediction in the learned latent space, without the guarantees provided by classical planners. STRIPS-WM learns predicate states and grounded operators, yielding a symbolic model that can be searched by classical planners.

Learning symbolic abstractions. A complementary line of work learns symbolic abstractions for planning. This foundational line of work grounds symbols in the preconditions and effects of high-level skills [19, 3], while recent neuro-symbolic methods learn predicates, operators, or relational transition models for planning [20, 21, 22, 23]. These methods show the value of symbolic structure, but typically assume object-centric, relational, or continuous state inputs rather than raw images. Closest to our setting, LatPlan learns propositional STRIPS from unlabeled image pairs [24], and we include it as our main baseline.

3 Problem Formulation

Observation and Action Data. We assume the robot observes the world through RGB images and executes actions from a fixed finite library of high-level primitives. Let $\mathcal{O}$ denote the space of RGB observations, and let $\mathcal{A} = \{1 \dots, L\}$ denote the finite set of executable action identifiers. The action identifiers are opaque: each identifier denotes one reusable primitive, but provides no semantic information about the primitive’s meaning, arguments, applicability, or effects. Additionally, the robot is given a dataset of visual transitions:

$$D_{\mathcal{O}} = \{(o_i, a_i, o'_i)\}_{i=1}^N, \quad o_i, o'_i \in \mathcal{O}, \quad a_i \in \mathcal{A}.$$

Each triplet records the observation before an action, the identifier of the executed action, and the observation after execution. We assume no access to object identities, segmentation masks, object poses, hand-designed predicates, symbolic state labels, or symbolic goals during training.

Action-Relevant Abstraction Assumption. Following the skills-to-symbols perspective of Konidaris et al. [3], we assume that the available high-level primitives induce a finite task-level abstraction of the underlying world. Intuitively, two physical world states belong to the same abstract state if they are interchangeable for planning: the same high-level actions are applicable, executing those actions leads to the same abstract outcomes (deterministic world), and the same task goals are satisfied. Low-level variations within an abstract state may change the RGB image, but they do not affect the feasibility or outcome of high-level action sequences. This assumption is what makes discrete high-level planning well defined from visual observations. We also assume that the dataset $D_{\mathcal{O}}$ sufficiently covers the relevant part of the task space.

Test-Time Planning Objective. At test time, the robot is given a start image o_{start} and a goal image o_{goal} . The objective is to produce a sequence:

$$\pi = (a_1, \dots, a_H), \quad a_h \in \mathcal{A},$$

such that executing π from the world state depicted by o_{start} reaches a world state that is task-equivalent to the one depicted by o_{goal} .

4 Background: STRIPS Propositional Planning

We review STRIPS planning since our method heavily builds on its concepts. STRIPS is a classical formalism for representing deterministic planning problems using symbolic states and action models [4]. In the propositional setting, a vocabulary of m Boolean predicates $\mathcal{P} = \{p_1, \dots, p_m\}$ is given.

A symbolic state σ is represented by a binary vector $\mathbf{x}_{\sigma} \in \{0, 1\}^m$, where $\mathbf{x}_{\sigma,i} = 1$ means that p_i is true in state σ , and $\mathbf{x}_{\sigma,i} = 0$ means that it is false. Each action a is represented by an operator:

$$\omega_a = \langle \mathbf{pre}_a^+, \mathbf{pre}_a^-, \mathbf{add}_a, \mathbf{del}_a \rangle,$$

where $\mathbf{pre}_a^+, \mathbf{pre}_a^-, \mathbf{add}_a, \mathbf{del}_a \in \{0, 1\}^m$ are binary masks over the predicate vocabulary. The masks $\mathbf{pre}_a^+$ and $\mathbf{pre}_a^-$ specify which predicates must be true or false for action a to be applicable. In state σ , action a is applicable if

$$\mathbf{pre}_a^+ \leq \mathbf{x}_{\sigma}, \quad \mathbf{pre}_a^- \leq \mathbf{1} - \mathbf{x}_{\sigma},$$

where inequalities are interpreted elementwise. If the action is applicable, it produces a successor state $\mathbf{x}_{\sigma_1}$ by setting predicates in $\mathbf{add}_a$ to true, setting predicates in $\mathbf{del}_a$ to false, and leaving all other predicates unchanged. Equivalently, each operator ω_a induces a partial transition function τ_{ω_a} , defined on applicable states as:

$$\mathbf{x}_{\sigma_1} = \tau_{\omega_a}(\mathbf{x}_{\sigma_0}) = \mathbf{add}_a \vee (\mathbf{x}_{\sigma_0} \wedge \neg \mathbf{del}_a).$$

The add and delete masks are disjoint, and the effects are state-independent: whenever the preconditions of a hold, the same add/delete masks are applied. This yields a compact transition model for search over symbolic states.

A STRIPS planning problem consists of an initial symbolic state, a goal condition, and a set of operators. In our setting, neither the predicate vocabulary $\mathcal{P}$ nor the operator masks are given; both must be learned from visual transition data.

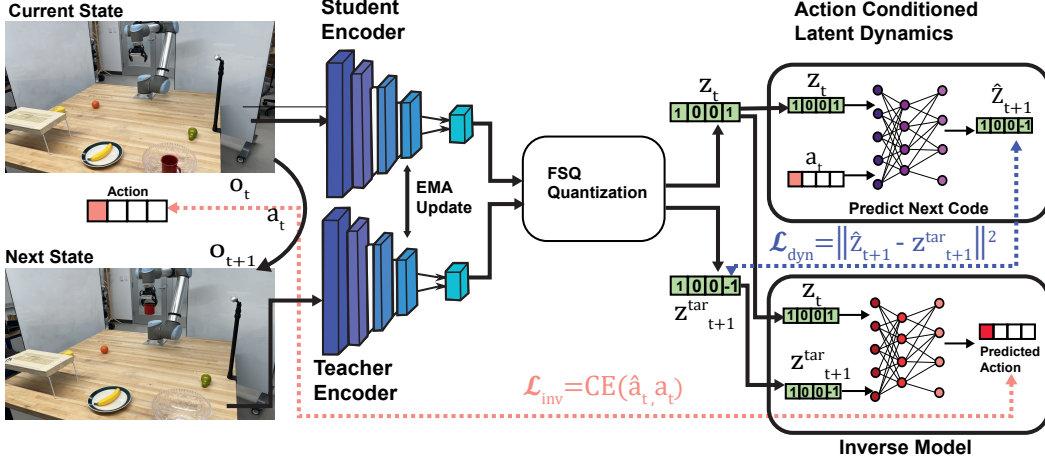


Figure 2: **Learning the image-grounded task graph:** A student encoder maps the current observation to a quantized FSQ code, while an EMA teacher encoder provides the target successor code. An action-conditioned dynamics model predicts the next code from the current code and action identifier, and an inverse model predicts the action from current and successor codes. Unique quantized codes define visual task-graph nodes, and observed transitions define labeled graph edges.

5 Methodology

5.1 Image-Grounded Task Graph

The goal of this stage is to use the visual transition dataset $D_{\mathcal{O}} = \{(o_i, a_i, o'_i)\}_{i=1}^N$ to learn an image-grounded task graph $\hat{G} = (\hat{S}, \hat{E})$. We do this by learning a discrete state assignment $\alpha : \mathcal{O} \rightarrow \hat{S}$ that maps each observation to a learned symbolic-state identity $\hat{s} = \alpha(o)$. Ideally, observations that are interchangeable for high-level planning receive the same state identity, even if they differ in irrelevant visual details.

Fig. 2 shows the architecture used to learn this assignment. A student encoder maps the current observation o_t to a latent representation, which is discretized with finite scalar quantization to produce a code z_t . A slowly updated teacher encoder maps the successor observation o_{t+1} to a target code z_{t+1}^{tar} . Given z_t and the opaque action identifier a_t , an action-conditioned predictor is trained to predict the successor code. This gives a JEPA-style [16] objective: prediction happens in latent code space rather than through pixel reconstruction.

To encourage the representation to retain action-relevant information, we also train an inverse dynamics head that predicts the executed action identifier from the current and successor latent codes. This auxiliary loss discourages collapsed codes that explain the visual data but fail to distinguish transitions produced by different actions.

After training, each observation is assigned an abstract-state identifier $\hat{s}$ from its quantized code: $\hat{s} = \alpha(o)$. We apply this map to every visual transition in $D_{\mathcal{O}}$. For each transition (o_i, a_i, o'_i) , we obtain the abstract transition $\hat{t}_i = (\alpha(o_i), a_i, \alpha(o'_i))$. Collecting these transformed transitions yields the learned action task graph.

$$\hat{G} = (\hat{S}, \hat{E}), \quad \hat{E} = \{(\hat{s}_i, a_i, \hat{s}'_i)\}_{i=1}^N.$$

The nodes of this graph are grounded in images, and the edges record observed effects of the opaque action identifiers. However, the graph is not yet a symbolic planning model: it has no predicates, preconditions, or reusable add/delete effects. The next stage learns those symbolic structures.

5.2 Learning STRIPS Operators and Predicates from the Task Graph

The learned task graph $\hat{G}$ from Sec. 5.1 gives us a discrete model of the visual experience, but it is still not a symbolic planning model. Its nodes are opaque abstract states, and its edges are labeled only by opaque action identifiers. To perform STRIPS-style planning, we must obtain a binary predicate

vector $\hat{\mathbf{x}}_{\hat{s}} \in \{0, 1\}^m$ for each graph node $\hat{s} \in \hat{\mathcal{S}}$, together with one grounded propositional operator $\hat{\omega}_a$ for each action identifier $a \in \mathcal{A}$.

We cast this as a combinatorial search over predicate assignments and operator masks, and encode the search as a binary CP-SAT problem [25]. Similar in spirit to Bonet and Geffner, who learn symbolic planning representations from state-space graphs using a SAT-based formulation [26], we use graph structure as supervision. Our setting differs in that the graph nodes are induced from raw images, and the learned operators are grounded propositional operators rather than lifted first-order schemas.

We formulate the CP-SAT problem as:

$$\begin{aligned} \text{find } & \hat{X} = \{\hat{\mathbf{x}}_{\hat{s}}\}_{\hat{s} \in \hat{\mathcal{S}}}, \quad \hat{\Omega} = \{\hat{\omega}_a\}_{a \in \mathcal{A}} \\ \text{such that } & \hat{\omega}_a \text{ is applicable in } \hat{\mathbf{x}}_{\hat{s}_0}, \quad \forall (\hat{s}_0, a, \hat{s}_1) \in \hat{E}, \end{aligned} \quad (\text{C1})$$

$$\tau_{\hat{\omega}_a}(\hat{\mathbf{x}}_{\hat{s}_0}) = \hat{\mathbf{x}}_{\hat{s}_1}, \quad \forall (\hat{s}_0, a, \hat{s}_1) \in \hat{E}, \quad (\text{C2})$$

$$\hat{\omega}_a \text{ is not applicable in } \hat{\mathbf{x}}_{\hat{s}}, \quad \forall (\hat{s}, a) \in \hat{\mathcal{N}}, \quad (\text{C3}).$$

Here, $\hat{X}$ assigns a learned symbolic predicate vector to every learned abstract state, $\hat{\Omega}$ assigns one learned grounded STRIPS-style operator to every action identifier, and $\tau_{\hat{\omega}_a}$ denotes the STRIPS transition induced by operator $\hat{\omega}_a$. The set $\hat{\mathcal{N}}$ contains the trusted missing state-action pairs used for negative evidence.

Let $\hat{\mathcal{S}}_{\text{trust}} \subseteq \hat{\mathcal{S}}$ denote the subset of graph states on which we apply the closed-world negative-evidence assumption. These are graph states on which we make a local closed-world assumption over outgoing action labels. Missing action labels from states outside $\hat{\mathcal{S}}_{\text{trust}}$ are ignored. We define the trusted missing state-action pairs as

$$\hat{\mathcal{N}} = \left\{ (\hat{s}, a) \mid \hat{s} \in \hat{\mathcal{S}}_{\text{trust}}, \nexists \hat{s}' \in \hat{\mathcal{S}} \text{ such that } (\hat{s}, a, \hat{s}') \in \hat{E} \right\}.$$

Constraints (C1) and (C2) say that **observed edges provide positive evidence**. If $(\hat{s}_0, a, \hat{s}_1) \in \hat{E}$, then action a must be applicable in $\hat{\mathbf{x}}_{\hat{s}_0}$, and its effects must map the predicate vector of $\hat{s}_0$ to the predicate vector of $\hat{s}_1$.

Constraint (C3) says that trusted missing state-action pairs provide negative evidence: if $(\hat{s}, a) \in \hat{\mathcal{N}}$, then action a was not observed from trusted state $\hat{s}$ and is not applicable in $\hat{\mathbf{x}}_{\hat{s}}$, unless applicability slack is used.

Observed transitions provide positive evidence. First we encode the positive requirements (C1), (C2) as binary constraints. Each observed edge $(\hat{s}_0, a, \hat{s}_1) \in \hat{E}$ is treated as a successful execution of action a . Thus (C1) requires the learned preconditions $\mathbf{pre}^+_a, \mathbf{pre}^-_a$ to hold in $\hat{\mathbf{x}}_{\hat{s}_0}$:

$$\hat{x}_{\hat{s}_0, p} \geq \mathbf{pre}^+_{a, p}, \quad \hat{x}_{\hat{s}_0, p} + \mathbf{pre}^-_{a, p} \leq 1, \quad \forall (\hat{s}_0, a, \hat{s}_1) \in \hat{E}, p \in [m].$$

Thus, if predicate p is a positive precondition of a , it must be true in every observed source state of a ; if it is a negative precondition, it must be false.

Constraint (C2) requires the learned effects of $\mathbf{add}_a, \mathbf{del}_a$ to produce the successor state $\hat{s}_1$. Expanding the STRIPS transition $\tau_{\hat{\omega}_a}$ to linear constraints for every $(\hat{s}_0, a, \hat{s}_1) \in \hat{E}$ and p :

$$\begin{aligned} \hat{x}_{\hat{s}_1, p} &\geq \mathbf{add}_{a, p}, & \hat{x}_{\hat{s}_1, p} + \mathbf{del}_{a, p} &\leq 1, \\ \hat{x}_{\hat{s}_1, p} - \hat{x}_{\hat{s}_0, p} &\leq \mathbf{add}_{a, p} + \mathbf{del}_{a, p}, & \hat{x}_{\hat{s}_0, p} - \hat{x}_{\hat{s}_1, p} &\leq \mathbf{add}_{a, p} + \mathbf{del}_{a, p}. \end{aligned}$$

The first inequality forces added predicates to be true in the successor, while the second forces deleted predicates to be false. The last two inequalities encode inertia: if action a neither adds nor deletes predicate p , then the truth value of p must be unchanged from $\hat{s}_0$ to $\hat{s}_1$.

Finally, we enforce the usual consistency constraints

$$\mathbf{add}_{a, p} + \mathbf{del}_{a, p} \leq 1, \quad \mathbf{pre}^+_{a, p} + \mathbf{pre}^-_{a, p} \leq 1, \quad \forall a \in \mathcal{A}, p \in [m],$$

so an operator cannot both add and delete the same predicate, or require the same predicate to be both true and false.

In practice, we keep the observed-applicability constraints in C1 hard since every observed edge is a successful execution, and soften only the effect/inertia constraints in (C2) with transition slack. Let $\hat{e} = (\hat{s}_0, a, \hat{s}_1)$ index an observed edge. We introduce a slack variable $\xi_{\hat{e},p}$ for each observed edge–predicate pair, which allows isolated violations when the visual abstraction aliases states or contains noise. The exact softened linear constraints are given in Appendix B.1.

Trusted missing state-action pairs provide negative evidence. We now encode the negative-applicability requirement (C3). Observed executions alone do not force informative preconditions: an operator with empty preconditions can explain all observed executions, but it would also make the action applicable from every learned state. To learn when actions are not applicable, we use the trusted missing pairs $\hat{\mathcal{N}}$ as negative evidence [27].

For each trusted missing pair $(\hat{s}, a) \in \hat{\mathcal{N}}$, the learned preconditions of action a should be violated in state $\hat{s}$. Thus we require:

$$\sum_{p=1}^m (pre_{a,p}^+(1 - \hat{x}_{\hat{s},p}) + pre_{a,p}^- \hat{x}_{\hat{s},p}) + \eta_{\hat{s},a} \geq 1, \quad \forall (\hat{s}, a) \in \hat{\mathcal{N}}.$$

The first term counts violated positive preconditions, and the second counts violated negative preconditions. Thus, every trusted missing action must either violate at least one learned precondition or pay applicability slack $\eta_{\hat{s},a}$. This prevents the precondition model from collapsing to the empty set while making exceptions to the closed-world assumption explicit.

The products above are between binary decision variables. In the CP-SAT encoding, we linearize them with auxiliary violation variables, so that the constraint simply enforces that at least one precondition of a is incompatible with the learned predicate assignment of $\hat{s}$, unless slack is used. The exact linearized constraints are given in Sec. B.1.

Lexicographic model selection. Many predicate/operator assignments may satisfy the semantic constraints. We therefore choose among feasible models using a lexicographic objective:

$$\min_{\text{lex}} \left(\sum_{\hat{e} \in \hat{E}} \sum_{p=1}^m \xi_{\hat{e},p}, \sum_{(\hat{s},a) \in \hat{\mathcal{N}}} \eta_{\hat{s},a}, \sum_{a \in \mathcal{A}} \sum_{p=1}^m (add_{a,p} + del_{a,p}), \sum_{a \in \mathcal{A}} \sum_{p=1}^m (pre_{a,p}^+ + pre_{a,p}^-) \right) \quad (1)$$

The priority order reflects what we want from the learned model. First, we minimize transition slack, because observed executions should be explained whenever possible. Second, we minimize applicability slack, so that trusted missing state-action pairs are ruled out by learned preconditions rather than ignored. Third, we prefer sparse effects, since actions in manipulation domains usually change only a small number of task-relevant facts. Finally, among equally good explanations, we prefer sparse preconditions.

Guarantees. In Appendix B.2 we provide formal guarantees for STRIPS-WM. The key point is that the constraints define a space of grounded propositional STRIPS models. Proposition 7 shows that every feasible assignment induces well-formed STRIPS operators, regardless of whether slack is used. In the zero-slack regime, Theorem 10 establishes soundness and completeness of the encoding: feasible solutions correspond exactly to m -predicate grounded STRIPS models that reproduce the observed transitions and rule out the selected trusted missing state-action pairs. The complete CP-SAT encoding is given in Appendix B.1.

5.3 Visual Predicate Classification and Test-Time Planning

The previous stage produces a predicate target $\hat{x}_{\hat{s}} \in \{0, 1\}^m$ for each learned state $\hat{s} \in \hat{\mathcal{S}}$. These targets are obtained offline by discrete optimization and cannot be directly applied to a new image. We therefore train a visual predicate classifier $b_\phi : \mathcal{O} \rightarrow [0, 1]^m$ to predict the learned predicate vector from pixels.

Let $\mathcal{D}_{\text{pred}} = \{(o_j, \hat{s}_j)\}_{j=1}^{N_{\text{pred}}}$ denote the set of training observations paired with their learned state identity from the visual task graph. For each pair $(o_j, \hat{s}_j)$, we have the target predicate vector $\hat{x}_{\hat{s}_j}$. We then train b_ϕ with a multi-label binary prediction loss: $\mathcal{L}_{\text{pred}} = \sum_{(o_j, \hat{s}_j) \in \mathcal{D}_{\text{pred}}} \text{BCE}(b_\phi(o_j), \hat{x}_{\hat{s}_j})$.

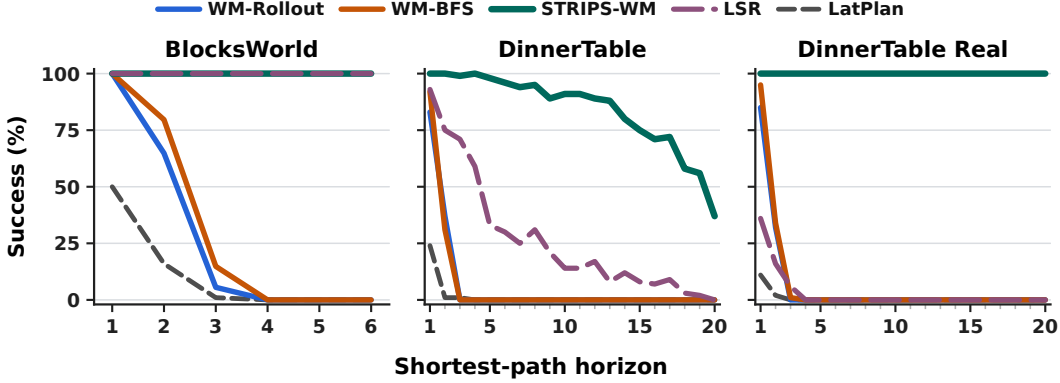


Figure 3: Planning success as a function of shortest-path horizon. Each point is evaluated by executing the returned action sequence in the ground-truth transition graph. Success therefore requires reaching the true goal state, not merely the goal predicted by the learned model.

At test time, the input is a start image o_{start} and a goal image o_{goal} . The visual predicate classifier maps them to predicate probabilities, which we binarize as $\hat{\mathbf{x}}_{\text{start}} = \text{bin}(b_\phi(o_{\text{start}}))$ and $\hat{\mathbf{x}}_{\text{goal}} = \text{bin}(b_\phi(o_{\text{goal}}))$. We then solve a classical planning problem in the learned predicate space with initial state $\hat{\mathbf{x}}_{\text{start}}$, goal state $\hat{\mathbf{x}}_{\text{goal}}$, and operator set $\hat{\Omega}$. During search, an action identifier $a \in \mathcal{A}$ is considered only if the learned operator $\hat{\omega}_a$ is applicable in the current predicate state x ; the successor is computed by the learned STRIPS transition $x' = \tau_{\hat{\omega}_a}(x)$. Search terminates when the goal predicate code is reached and returns the corresponding action sequence $\pi = (a_1, \dots, a_H)$.

6 Experiments & Results

Visual Planning Domains: We evaluate the performance of STRIPS-WM in three image-based rearrangement domains. **BlocksWorld** consists of three identical blocks and three stack-

ing regions as in [28]. **DinnerTable** is a simulated tabletop rearrangement problem with five distinct objects and seven possible placements. **DinnerTable Real** uses real images from the same tabletop rearrangement setting, with additional static distractor objects such as a plate, bowl, and shelf, as shown in Fig. 1. For evaluation purposes, we manually create ground-truth transition task graphs $G^* = (\mathcal{S}^*, \mathcal{E}^*)$, for each planning domain that explain $D_{\mathcal{O}}$. The statistics for each domain are reported in Table 1. We note that for **DinnerTable Real** and **DinnerTable**, we explored a subset of the potential task space with the collected dataset $D_{\mathcal{O}}$.

Visual Planning Baselines. We compare STRIPS-WM against four baselines.

WM-Rollout: rolls out candidate action sequences with the learned action-conditioned FSQ dynamics model and selects the sequence whose predicted code is closest to the goal. This follows the spirit of rollout-based visual MPC and latent world-model planning [2, 6, 12].

WM-BFS: uses the same learned dynamics model, but replaces sequence rollout with discrete graph search in the FSQ space: from the start code, it applies each action identifier, enqueues unseen successor codes, and stops when the goal code is reached. This follows latent graph-search approaches to visual planning [29].

LSR [9] learns a continuous visual latent space from image transitions, clusters latent observations into roadmap nodes, and adds directed edges from observed transitions. It can be viewed as learning a latent task graph $\hat{G} = (\hat{\mathcal{S}}, \hat{\mathcal{E}})$, with planning performed by graph search over the roadmap.

LatPlan-AMA3 [24] is the most similar work to STRIPS-WM. It learns STRIPS-style predicates, operators, binary latent image codes, and an action-conditioned neural transition model with add/delete/no-op style bit effects. We adapted the AMA3 model to use our provided action identifiers for a fair comparison.

Table 1: Domain and learned abstraction statistics.

Domain	Domain Statistics			STRIPS-WM Statistics				
	$ \mathcal{A} $	$ \mathcal{D}_{\mathcal{O}} $	$ \mathcal{S}^* $	$ \hat{\mathcal{S}} $	$ \hat{\mathcal{X}} $	m	$\sum \xi$	$\sum \eta$
BlocksWorld	18	5,000	16	16	16	9	0	0
DinnerTable	70	12,000	101	101	101	35	0	9
DinnerTable Real	64	3,000	71	111	71	35	0	13

RQ1: Does STRIPS-WM learn the correct abstraction? The data in Table 1 show how the learned abstraction compares to the ground-truth task graph. Let $\hat{\mathcal{X}}$ denote the set of distinct predicate states produced by operator learning. In **BlocksWorld**, STRIPS-WM recovers the abstraction exactly: $|\hat{\mathcal{S}}| = |\mathcal{S}^*| = |\hat{\mathcal{X}}| = 16$. The model uses $m = 9$ predicates and has zero transition and applicability slack ($\sum \xi = 0, \sum \eta = 0$). Thus, BlocksWorld lies in the exact zero-slack regime of Theorem 10: the learned STRIPS model is sound and complete with respect to the learned graph and the selected trusted missing state-action pairs. In **DinnerTable**, STRIPS-WM again recovers the ground-truth abstraction, with $|\hat{\mathcal{S}}| = |\mathcal{S}^*| = |\hat{\mathcal{X}}| = 101$, and has zero transition slack ($\sum \xi = 0$) and a small amount of applicability slack ($\sum \eta = 9$), indicating residual applicability errors in the learned precondition model. In **DinnerTable Real**, the visual abstraction over-splits the task space, producing $|\hat{\mathcal{S}}| = 111$ visual nodes for only $|\mathcal{S}^*| = 71$ task states. The CP-SAT solver collapses these into $|\hat{\mathcal{X}}| = 71$ distinct predicate states, again with zero transition slack ($\sum \xi = 0$). The remaining applicability slack ($\sum \eta = 13$) is therefore best interpreted as a fragmentation-induced negative-evidence conflict, rather than a failure to model the observed transition dynamics.

RQ2: How well does STRIPS-WM scale to long-horizon tasks? We evaluate image-to-plan performance using the ground-truth transition graph $G^* = (\mathcal{S}^*, \mathcal{E}^*)$. For each domain and shortest-path horizon ℓ , we sample 200 start-goal image queries whose endpoints are distance ℓ in G^* . Each method receives only the start and goal images and returns either an action sequence or no plan. Returned plans are executed in G^* , and success is the fraction of queries whose plans reach the ground-truth goal state. Thus, reaching the goal only in the learned model is not sufficient.

The results in Fig. 3 show that STRIPS-WM achieves 100% success in BlocksWorld, consistent with the zero-slack abstraction in Table 1 and the exact regime of Theorem 10. In DinnerTable Real, STRIPS-WM also achieves 100% success despite using slack: here, slack reflects the collapse of an over-split visual graph into the correct task-level STRIPS abstraction. In simulated DinnerTable, however, the introduced slack corresponds to genuine errors in the learned applicability model, which appear as failures at longer horizons.

The baselines degrade quickly over longer horizons. WM-Rollout compounds latent prediction errors, and WM-BFS searches the same FSQ space without explicit preconditions or persistent effects. LSR solves BlocksWorld but relies on flat roadmap connectivity, which is less reliable in larger long-horizon domains. LatPlan-AMA3 learns a STRIPS-style latent model, but does not recover reliable predicates and operators in these visual settings.

RQ3: Are predicates better than monolithic graph search? We compare STRIPS-WM to direct graph search over the learned task graph $\hat{G} = (\hat{\mathcal{S}}, \hat{\mathcal{E}})$. As shown in Fig. 4, the learned STRIPS model achieves a higher success rate because the CP-SAT layer can collapse over-split but action-equivalent visual nodes and repair inconsistencies in the flat graph.

Furthermore, the predicate representation is more compact: instead of classifying among monolithic graph nodes $\hat{s} \in \hat{\mathcal{S}}$, STRIPS-WM predicts an m -dimensional binary predicate vector $\hat{\mathbf{x}}_{\hat{s}}$, with $m \ll |\hat{\mathcal{S}}|$. This reduces the perception problem in Sec. 5.3 and enables planning with a few symbolic operators rather than memorized graph edges.

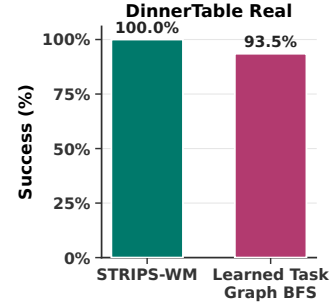


Figure 4: Planning success on DinnerTable Real.

7 Limitations

STRIPS-WM inherits three limitations. First, it assumes deterministic STRIPS dynamics with fixed preconditions and add/delete effects, and therefore does not model stochastic outcomes or hidden state. Second, the learned model depends on the quality and coverage of $\hat{G}$: visual aliasing or incomplete outgoing action coverage can produce incorrect preconditions or unreliable negative evidence. Third, STRIPS-WM learns grounded propositional operators rather than lifted relational schemas, so it does not generalize compositionally to new object sets or action arguments without additional data.

References

- [1] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.
- [2] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.
- [3] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61:215–289, Jan. 2018. ISSN 1076-9757. doi:10.1613/jair.5575. URL <https://www.jair.org/index.php/jair/article/view/11175>.
- [4] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.
- [5] L. Yen-Chen, M. Bauza, and P. Isola. Experience-embedded visual foresight. In *Conference on Robot Learning*, pages 1015–1024. PMLR, 2020.
- [6] B. Ichter and M. Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019.
- [7] C. Paxton, Y. Barnoy, K. Katyal, R. Arora, and G. D. Hager. Visual Robot Task Planning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8832–8838, May 2019. doi:10.1109/ICRA.2019.8793736. URL <https://ieeexplore.ieee.org/document/8793736/>.
- [8] D. Driess, J.-S. Ha, and M. Toussaint. Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image. In *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, July 2020. ISBN 978-0-9923747-6-1. doi:10.15607/RSS.2020.XVI.003. URL <http://www.roboticsproceedings.org/rss16/p003.pdf>.
- [9] M. Lippi, P. Poklukar, M. C. Welle, A. Varava, H. Yin, A. Marino, and D. Kragic. Enabling Visual Action Planning for Object Manipulation Through Latent Space Roadmap. *IEEE Transactions on Robotics*, 39(1):57–75, Feb. 2023. ISSN 1941-0468. doi:10.1109/TRO.2022.3188163. URL <https://ieeexplore.ieee.org/document/9833914/>.
- [10] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning Latent Dynamics for Planning from Pixels, June 2019. URL <http://arxiv.org/abs/1811.04551>. arXiv:1811.04551 [cs].
- [11] D. Hafner, K.-H. Lee, I. Fischer, and P. Abbeel. Deep hierarchical planning from pixels. *Advances in Neural Information Processing Systems*, 35:26091–26104, 2022.
- [12] P. Wu, A. Escontrela, D. Hafner, P. Abbeel, and K. Goldberg. Daydreamer: World models for physical robot learning. In *6th Annual Conference on Robot Learning*, 2022. URL <https://openreview.net/forum?id=3RBY8fKjHeu>.
- [13] G. Zhou, H. Pan, Y. LeCun, and L. Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *Forty-second International Conference on Machine Learning*, 2025. URL <https://openreview.net/forum?id=D5RNAC0ZEI>.
- [14] C. Chamzas, M. Lippi, M. C. Welle, A. Varava, L. E. Kavraki, and D. Kragic. Comparing reconstruction-and contrastive-based models for visual task planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 12550–12557, Oct. 2022. doi:10.1109/IROS47612.2022.9981533. URL <https://doi.org/10.1109/IROS47612.2022.9981533>.
- [15] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. Universal planning networks, 2018. URL <https://arxiv.org/abs/1804.00645>.

- [16] M. Assran, Q. Duval, I. Misra, P. Bojanowski, P. Vincent, M. G. Rabbat, Y. LeCun, and N. Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *CVPR*, pages 15619–15629, 2023. URL <https://doi.org/10.1109/CVPR52729.2023.01499>.
- [17] T. Kipf, E. van der Pol, and M. Welling. Contrastive learning of structured world models, 2020. URL <https://arxiv.org/abs/1911.12247>.
- [18] M. Laskin, A. Srinivas, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International conference on machine learning*, pages 5639–5650. PMLR, 2020.
- [19] G. Konidaris, L. Kaelbling, and T. Lozano-Perez. Constructing Symbolic Representations for High-Level Planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), June 2014. ISSN 2374-3468. doi:10.1609/aaai.v28i1.9004. URL <https://ojs.aaai.org/index.php/AAAI/article/view/9004>.
- [20] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez. Learning Symbolic Operators for Task and Motion Planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3182–3189, Sept. 2021. doi:10.1109/IROS51168.2021.9635941. URL <https://ieeexplore.ieee.org/document/9635941/>. ISSN: 2153-0866.
- [21] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning Neuro-Symbolic Skills for Bilevel Planning. In *Proceedings of The 6th Conference on Robot Learning*, pages 701–714. PMLR, Mar. 2023. URL <https://proceedings.mlr.press/v205/silver23a.html>.
- [22] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4166–4173, Oct. 2022. doi:10.1109/IROS47612.2022.9981440. URL <https://ieeexplore.ieee.org/document/9981440/>. ISSN: 2153-0866.
- [23] N. Shah, J. Nagpal, P. Verma, and S. Srivastava. From reals to logic and back: Inventing symbolic vocabularies, actions and models for planning from raw data. *arXiv preprint arXiv:2402.11871*, 2:A14, 2024.
- [24] M. Asai, H. Kajino, A. Fukunaga, and C. Muise. Classical planning in deep latent space. *J. Artif. Int. Res.*, 74, Sept. 2022. ISSN 1076-9757. doi:10.1613/jair.1.13768. URL <https://doi.org/10.1613/jair.1.13768>.
- [25] L. Perron and F. Didier. Cp-sat, 2025. URL https://developers.google.com/optimization/cp/cp_solver/.
- [26] B. Bonet and H. Geffner. Learning first-order symbolic representations for planning from the structure of the state space, 2020. URL <https://arxiv.org/abs/1909.05546>.
- [27] D. Aineto and E. Scala. Action model learning with guarantees. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR '24*, 2024. ISBN 978-1-956792-05-8. doi:10.24963/kr.2024/75. URL <https://doi.org/10.24963/kr.2024/75>.
- [28] A. Ajith and C. Chamzas. Learning discrete abstractions for visual rearrangement tasks using vision-guided graph coloring, 2026. URL <https://arxiv.org/abs/2509.14460>.
- [29] M. Bagatella, M. Olšák, M. Rolínek, and G. Martius. Planning from pixels in environments with combinatorially hard search spaces. *Advances in Neural Information Processing Systems*, 34:24707–24718, 2021.

Appendix

A Architecture and Implementation Details

This section gives additional implementation details for the three learned components of STRIPS-WM: the image-grounded task graph in Sec. 5.1, the operator-learning stage in Sec. 5.2, and the visual predicate classifier in Sec. 5.3.

A.1 Image-grounded task graph model

To construct the image-grounded task graph $\hat{G} = (\hat{\mathcal{S}}, \hat{E})$, we learn a discrete state assignment $\alpha : \mathcal{O} \rightarrow \hat{\mathcal{S}}$. As shown in Fig. 2, the model has a student image encoder, an EMA teacher encoder, a finite-scalar-quantized latent code, an action-conditioned latent dynamics model, and an inverse dynamics head. The student encoder f_θ maps the current observation o_t to a continuous latent vector. Finite scalar quantization then maps this vector to a discrete code

$$z_t = q(f_\theta(o_t)).$$

The successor observation o_{t+1} is encoded by an exponential-moving average teacher encoder $f_{\bar{\theta}}$. The teacher parameters are updated after each optimization step as

$$\bar{\theta} \leftarrow \mu \bar{\theta} + (1 - \mu)\theta,$$

with $\mu = 0.996$. The teacher output is quantized with the same FSQ quantizer to produce the target successor code

$$z_{t+1}^{\text{tar}} = q(f_{\bar{\theta}}(o_{t+1})).$$

The action-conditioned latent dynamics model predicts a successor code $\hat{z}_{t+1}$ from z_t and the action identifier a_t . The inverse model predicts a_t from the current and successor codes, which discourages collapsed representations that ignore action-relevant visual differences.

We train this architecture with the following objective:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{dyn}} + \lambda_{\text{inv}} \mathcal{L}_{\text{inv}} + \lambda_{\text{com}} \mathcal{L}_{\text{com}} + \lambda_{\text{aug}} \mathcal{L}_{\text{aug}} + \lambda_{\text{sep}} \mathcal{L}_{\text{sep}}.$$

The dynamics loss is

$$\mathcal{L}_{\text{dyn}} = \|\hat{z}_{t+1} - z_{t+1}^{\text{tar}}\|_2^2,$$

and trains the latent dynamics model to predict the teacher successor code. The inverse loss is

$$\mathcal{L}_{\text{inv}} = \text{CE}(\hat{a}_t, a_t),$$

where $\hat{a}_t$ is the inverse model prediction. This term encourages the code to preserve information that distinguishes which action caused a transition.

The commitment loss stabilizes the quantized representation. Let $\tilde{z}_t$ denote the continuous latent value immediately before straight-through rounding, and let z_t denote the corresponding quantized code. We use

$$\mathcal{L}_{\text{com}} = \|\tilde{z}_t - \text{stopgrad}(z_t)\|_2^2.$$

This pulls the encoder output toward the discrete FSQ grid point it is assigned to, reducing oscillation near quantization boundaries while leaving the quantized target fixed.

For real-image training, we also used mild same-image augmentation consistency. If $\text{aug}(o_t)$ is an augmented version of the same observation, then $\mathcal{L}_{\text{aug}}$ encourages o_t and $\text{aug}(o_t)$ to receive the same latent code. This term is used as a visual invariance regularizer.

Latent nondeterminism. We also monitor the determinism of the learned latent transition graph. Given the encoded transitions

$$(z_i, a_i, z'_i),$$

we say that a latent state-action pair (z, a) is *latent-nondeterministic* if it has more than one observed successor code:

$$|\{z' \mid (z, a, z') \text{ is observed}\}| > 1.$$

This diagnostic uses only the learned codes and the action identifiers in $D_{\mathcal{O}}$. It does not use ground-truth symbolic states, object metadata, or manually labeled predicates. Low latent nondeterminism is desirable because the downstream CP-SAT stage learns deterministic STRIPS operators. If a single learned code z with the same action label a leads to multiple successor codes, then the visual abstraction has either aliased different task states or introduced transition noise.

The predictive separation term is designed to reduce this failure mode. When two examples have the same current code and the same action label but different successor codes, $\mathcal{L}_{\text{sep}}$ encourages their pre-quantized current latents to move apart. This makes it easier for the quantizer to assign distinct codes to states that are visually similar but action-relevantly different.

Latent sweep and checkpoint selection. For each domain, we swept the FSQ latent dimensionality while keeping a centered five-level scalar grid in each active dimension. The candidate latent spaces were

$$[5]^5, \quad [5]^6, \quad [5]^7,$$

corresponding to $B \in \{5, 6, 7\}$ FSQ dimensions with five levels per dimension. We selected checkpoints using the total training loss and latent-nondeterminism diagnostics, preferring lower latent nondeterminism among comparable low-loss checkpoints.

Setting	BlocksWorld	DinnerTable	DinnerTable Real
Encoder backbone	ConvNeXt-Tiny	ConvNeXt-Tiny	ConvNeXt-Tiny
Image resolution	228×228	228×228	228×228
FSQ latent sweep	$[5]^5, [5]^6, [5]^7$	$[5]^5, [5]^6, [5]^7$	$[5]^5, [5]^6, [5]^7$
Selected FSQ levels	$[5]^5$	$[5]^7$	$[5]^7$
Optimizer	AdamW	AdamW	AdamW
Learning rate	8×10^{-4}	8×10^{-4}	8×10^{-4}
Batch size	64	64	64
Training epochs	170	170	170
Teacher EMA decay μ	0.996	0.996	0.996
Inverse loss weight λ_{inv}	5.0	5.0	5.0
Commitment loss weight λ_{com}	0.05	0.05	0.05
Predictive separation weight λ_{sep}	0.03	0.03	0.03

Table 2: Image-grounded task-graph training hyperparameters. We swept the FSQ latent dimensionality for each domain and selected the final checkpoint using training loss and latent-graph diagnostics.

A.2 Operator-learning hyperparameters

Given the learned task graph $\hat{G} = (\hat{\mathcal{S}}, \hat{E})$, the operator-learning stage assigns a binary predicate vector $\hat{\mathbf{x}}_{\hat{s}} \in \{0, 1\}^m$ to each learned graph node $\hat{s} \in \hat{\mathcal{S}}$, and learns one grounded propositional operator $\hat{\omega}_a$ for each action identifier $a \in \mathcal{A}$. We use the CP-SAT formulation described in [Sec. 5.2](#), with the exact linearized constraints given in [Sec. B.1](#).

The main hyperparameter in this stage is the predicate dimension m . For each domain, we swept m and selected the smallest predicate dimension that produced a compact STRIPS explanation within the solver budget. In particular, we prioritized models that reproduce the observed graph transitions with zero transition slack, and then used the lexicographic objective in [Equation 1](#) to trade off applicability slack, effect sparsity, and precondition sparsity. Each fixed- m CP-SAT run was given a total wall-clock budget of 300 seconds. This budget is applied to the complete solver run for that candidate predicate dimension, rather than to each individual objective term. The slack variables have different interpretations. Transition slack is represented by variables $\xi_{\hat{e}, p}$, one for each observed edge $\hat{e} = (\hat{s}, a, \hat{s}') \in \hat{E}$ and predicate $p \in [m]$. These variables relax the requirement that the learned add/delete/inertia effects exactly reproduce the observed successor predicate value. The aggregate transition slack reported in the experiments is

$$\sum \xi = \sum_{\hat{e} \in \hat{E}} \sum_{p=1}^m \xi_{\hat{e}, p}.$$

Thus, $\sum \xi = 0$ means that every observed edge in the learned graph is reproduced exactly in predicate space. Applicability slack is represented by variables $\eta_{\hat{s},a}$, one for each trusted missing state-action pair $(\hat{s}, a) \in \hat{\mathcal{N}}$. The aggregate applicability slack is

$$\sum \eta = \sum_{(\hat{s},a) \in \hat{\mathcal{N}}} \eta_{\hat{s},a}.$$

Thus, $\sum \eta > 0$ does not mean that an observed edge is modeled incorrectly; it means that some trusted missing action remained applicable under the learned precondition model.

Setting	BlocksWorld	DinnerTable	DinnerTable Real
Solver backend	CP-SAT	CP-SAT	CP-SAT
Wall-clock budget per fixed- m run	300s	300s	300s
Selected predicate dimension m	9	35	35
Transition slack in selected solution $\sum \xi$	0	0	0
Applicability slack in selected solution $\sum \eta$	0	9	13

Table 3: Operator-learning settings and selected CP-SAT solutions.

The formal guarantees in [Sec. B.2](#) apply directly to these learned models. In particular, [Proposition 7](#) shows that every feasible CP-SAT assignment induces well-formed grounded propositional STRIPS operators, regardless of the slack values. When both transition slack and applicability slack are zero, [Theorem 10](#) gives the exact soundness and completeness statement for the learned graph and the selected trusted missing state-action pairs.

A.3 Visual predicate classifier

The operator-learning stage assigns a learned predicate vector $\hat{x}_{\hat{s}} \in \{0, 1\}^m$ to each learned graph node $\hat{s} \in \hat{\mathcal{S}}$. These predicate assignments are defined only on the offline learned task graph, so they cannot be directly applied to a new start or goal image. We therefore train a visual predicate classifier $b_{\phi} : \mathcal{O} \rightarrow [0, 1]^m$, as described in [Sec. 5.3](#). The classifier grounds the learned predicates back into perception by predicting the predicate vector associated with the graph node assigned to an image.

Let

$$g_{\phi}(o) \in \mathbb{R}^m$$

denote the classifier logits, and let

$$b_{\phi}(o) = \sigma(g_{\phi}(o))$$

denote the corresponding predicate probabilities.

For each visual transition $(o_i, a_i, o'_i) \in D_{\mathcal{O}}$, the learned state assignment maps the images to graph nodes:

$$\hat{s}_i = \alpha(o_i), \quad \hat{s}'_i = \alpha(o'_i).$$

We use the corresponding learned predicate assignments as binary supervision vectors:

$$y_i = \hat{x}_{\hat{s}_i}, \quad y'_i = \hat{x}_{\hat{s}'_i}.$$

Thus, the predicate classifier is trained only from predicate assignments learned by STRIPS-WM.

We train the classifier with binary cross-entropy on both the current and successor images:

$$\mathcal{L}_{\text{BCE}} = \frac{1}{2} [\text{BCE}(g_{\phi}(o_i), y_i) + \text{BCE}(g_{\phi}(o'_i), y'_i)].$$

Here BCE denotes binary cross-entropy with logits. We also use a small confidence regularizer on the logits:

$$\mathcal{L}_{\text{margin}} = \frac{1}{2} [\text{ReLU}(\gamma - |g_{\phi}(o_i)|) + \text{ReLU}(\gamma - |g_{\phi}(o'_i)|)],$$

averaged over predicates. This term encourages predictions to move away from the binarization threshold while the binary cross-entropy term anchors them to the learned predicate assignments. The reported predicate-classifier objective is

$$\mathcal{L}_{\text{pred}} = \mathcal{L}_{\text{BCE}} + \lambda_{\text{margin}} \mathcal{L}_{\text{margin}}.$$

At test time, the classifier maps a start image and goal image to predicate probabilities. We binarize each predicate independently:

$$\hat{x}_{\text{start}} = \text{bin}(b_\phi(o_{\text{start}})), \quad \hat{x}_{\text{goal}} = \text{bin}(b_\phi(o_{\text{goal}})),$$

where

$$\text{bin}(b)_p = \begin{cases} 1, & b_p \geq 0.5, \\ 0, & b_p < 0.5. \end{cases}$$

Planning then uses $\hat{x}_{\text{start}}$, $\hat{x}_{\text{goal}}$, and the learned operator set $\hat{\Omega}$ exactly as described in [Sec. 5.3](#).

Setting	BlocksWorld	DinnerTable	DinnerTable Real
Classifier backbone	ConvNeXt-Tiny	ConvNeXt-Tiny	ConvNeXt-Tiny
Input resolution	228×228	228×228	228×228
Output dimension	$m = 9$	$m = 35$	$m = 35$
Optimizer	Adam	Adam	Adam
Learning rate	10^{-3}	10^{-3}	10^{-3}
Batch size	64	64	64
Training epochs	50	150	150
Margin γ	1.0	1.0	1.0
Margin weight λ_{margin}	0.1	0.1	0.1
Binarization threshold	0.5	0.5	0.5

Table 4: Visual predicate classifier hyperparameters. The output dimension is the selected predicate dimension m from [Table 1](#).

B Additional Theoretical Details

B.1 Exact CP-SAT Encoding for Operator Learning

This section gives the exact binary optimization used for the operator-learning stage in [Sec. 5.2](#). The input is the learned task graph $\hat{G} = (\hat{S}, \hat{E})$, together with the trusted missing state-action pairs $\hat{\mathcal{N}}$. For a fixed predicate dimension m , the CP-SAT solver assigns predicate vectors to learned graph nodes and learns one grounded propositional STRIPS operator for each action identifier.

For every learned graph node $\hat{s} \in \hat{S}$ and predicate $p \in [m]$, we introduce a binary predicate variable

$$\hat{x}_{\hat{s},p} \in \{0, 1\}.$$

The full predicate vector for $\hat{s}$ is

$$\hat{\mathbf{x}}_{\hat{s}} = (\hat{x}_{\hat{s},1}, \dots, \hat{x}_{\hat{s},m}) \in \{0, 1\}^m.$$

For every action identifier $a \in \mathcal{A}$ and predicate $p \in [m]$, we introduce binary operator-mask variables

$$\text{add}_{a,p}, \text{del}_{a,p} \in \{0, 1\}, \quad \text{pre}_{a,p}^+, \text{pre}_{a,p}^- \in \{0, 1\}.$$

We also use transition slack variables $\xi_{\hat{e},p} \in \{0, 1\}$. Here $\hat{e} = (\hat{s}, a, \hat{s}') \in \hat{E}$ denotes an observed edge in the learned task graph, and $p \in [m]$ denotes a predicate. Thus $\xi_{\hat{e},p}$ is the per-edge/per-predicate form of the aggregate transition slack $\sum \xi$ reported in the main text. We use applicability slack variables $\eta_{\hat{s},a} \in \{0, 1\}$ for trusted missing pairs $(\hat{s}, a) \in \hat{\mathcal{N}}$, and auxiliary violation and distinctness variables introduced below.

Observed applications satisfy learned preconditions. Each observed edge

$$\hat{e} = (\hat{s}, a, \hat{s}') \in \hat{E}$$

records a successful execution of action identifier a from $\hat{s}$ to $\hat{s}'$. The action identifier a is the edge label. Therefore, the learned preconditions of a must hold in the predicate vector of the source node $\hat{s}$. For every $\hat{e} = (\hat{s}, a, \hat{s}') \in \hat{E}$ and every $p \in [m]$, we impose

$$\hat{x}_{\hat{s},p} \geq \text{pre}_{a,p}^+, \tag{2}$$

$$\hat{x}_{\hat{s},p} + \text{pre}_{a,p}^- \leq 1. \tag{3}$$

Thus, if predicate p is a positive precondition of a , it must be true in every observed source node for a . If p is a negative precondition, it must be false in every observed source node for a .

Observed edges follow STRIPS effects with inertia. For each observed edge $\hat{e} = (\hat{s}, a, \hat{s}') \in \hat{E}$ and each predicate $p \in [m]$, we impose

$$\hat{x}_{\hat{s}',p} + \xi_{\hat{e},p} \geq add_{a,p}, \quad (4)$$

$$\hat{x}_{\hat{s}',p} + del_{a,p} \leq 1 + \xi_{\hat{e},p}, \quad (5)$$

$$\hat{x}_{\hat{s}',p} - \hat{x}_{\hat{s},p} \leq add_{a,p} + del_{a,p} + \xi_{\hat{e},p}, \quad (6)$$

$$\hat{x}_{\hat{s},p} - \hat{x}_{\hat{s}',p} \leq add_{a,p} + del_{a,p} + \xi_{\hat{e},p}. \quad (7)$$

The add/delete masks and positive/negative precondition masks are disjoint:

$$add_{a,p} + del_{a,p} \leq 1, \quad pre_{a,p}^+ + pre_{a,p}^- \leq 1, \quad (8)$$

for all $a \in \mathcal{A}$ and $p \in [m]$. When $\xi_{\hat{e},p} = 0$, these constraints implement the standard bitwise STRIPS update:

$$\hat{x}_{\hat{s}',p} = \begin{cases} 1, & add_{a,p} = 1, \\ 0, & del_{a,p} = 1, \\ \hat{x}_{\hat{s},p}, & add_{a,p} = del_{a,p} = 0. \end{cases}$$

Thus, add effects set predicates to true, delete effects set predicates to false, and all other predicates obey inertia.

Negative evidence from trusted missing pairs. The trusted missing state-action pairs are

$$\hat{\mathcal{N}} = \{(\hat{s}, a) \mid \hat{s} \in \hat{\mathcal{S}}_{\text{trust}}, \nexists \hat{s}' \text{ such that } (\hat{s}, a, \hat{s}') \in \hat{E}\}.$$

For each $(\hat{s}, a) \in \hat{\mathcal{N}}$, the learned preconditions of action a should be violated in $\hat{\mathbf{x}}_{\hat{s}}$, unless applicability slack is used. We linearize these violations using binary variables

$$v_{\hat{s},a,p}^+, v_{\hat{s},a,p}^- \in \{0, 1\}.$$

For every $(\hat{s}, a) \in \hat{\mathcal{N}}$ and $p \in [m]$, we impose

$$v_{\hat{s},a,p}^+ \leq pre_{a,p}^+, \quad v_{\hat{s},a,p}^+ \leq 1 - \hat{x}_{\hat{s},p}, \quad v_{\hat{s},a,p}^+ \geq pre_{a,p}^+ - \hat{x}_{\hat{s},p}, \quad (9)$$

$$v_{\hat{s},a,p}^- \leq pre_{a,p}^-, \quad v_{\hat{s},a,p}^- \leq \hat{x}_{\hat{s},p}, \quad v_{\hat{s},a,p}^- \geq pre_{a,p}^- + \hat{x}_{\hat{s},p} - 1. \quad (10)$$

These constraints encode

$$v_{\hat{s},a,p}^+ = pre_{a,p}^+ \wedge (1 - \hat{x}_{\hat{s},p}), \quad v_{\hat{s},a,p}^- = pre_{a,p}^- \wedge \hat{x}_{\hat{s},p}.$$

We then require

$$\sum_{p=1}^m (v_{\hat{s},a,p}^+ + v_{\hat{s},a,p}^-) + \eta_{\hat{s},a} \geq 1, \quad \forall (\hat{s}, a) \in \hat{\mathcal{N}}. \quad (11)$$

Thus, every trusted missing state-action pair must either violate at least one learned precondition or pay applicability slack.

Optional distinctness constraints. The solver may optionally require selected learned graph nodes to receive different predicate vectors. Let $\mathcal{P}_{\text{dist}}$ denote the set of unordered node pairs on which this constraint is imposed. Full distinctness corresponds to including every pair $\{\hat{s}, \hat{r}\} \subseteq \hat{\mathcal{S}}$, while the relaxed settings used in some experiments impose fewer or no distinctness constraints.

For each $\{\hat{s}, \hat{r}\} \in \mathcal{P}_{\text{dist}}$ and predicate $p \in [m]$, we introduce a binary difference variable $d_{\hat{s},\hat{r},p} \in \{0, 1\}$ and impose

$$d_{\hat{s},\hat{r},p} \geq \hat{x}_{\hat{s},p} - \hat{x}_{\hat{r},p}, \quad (12)$$

$$d_{\hat{s},\hat{r},p} \geq \hat{x}_{\hat{r},p} - \hat{x}_{\hat{s},p}, \quad (13)$$

$$d_{\hat{s},\hat{r},p} \leq \hat{x}_{\hat{s},p} + \hat{x}_{\hat{r},p}, \quad (14)$$

$$d_{\hat{s},\hat{r},p} \leq 2 - \hat{x}_{\hat{s},p} - \hat{x}_{\hat{r},p}, \quad (15)$$

$$\sum_{p=1}^m d_{\hat{s},\hat{r},p} \geq 1. \quad (16)$$

Together, these constraints enforce

$$\hat{\mathbf{x}}_{\hat{s}} \neq \hat{\mathbf{x}}_{\hat{r}}$$

for each selected pair $\{\hat{s}, \hat{r}\} \in \mathcal{P}_{\text{dist}}$.

Solver-side regularization. The constraints above are the semantic core of the formulation. In the implementation, we also use solver-side choices such as predicate-frequency symmetry breaking, anti-collapse constraints, and relaxed distinctness over selected node pairs. These choices affect which feasible solution is preferred or how efficiently it is found, but they do not change the STRIPS form of the learned operators.

B.2 Formal Guarantees for Operator Learning

The guarantees below are stated directly for the learned task graph $\hat{G} = (\hat{\mathcal{S}}, \hat{E})$ and the selected trusted missing pairs $\hat{\mathcal{N}}$. They do not assume access to the full physical state space. Slack variables determine how strictly the learned model must agree with $\hat{E}$ and $\hat{\mathcal{N}}$, but they do not change the form of the learned operators.

Trusted missing-pair assumption. All statements involving $\hat{\mathcal{N}}$ are relative to the trusted set of missing state-action pairs selected by the method. Thus, the negative-evidence guarantees below should be read as guarantees about pairs in $\hat{\mathcal{N}}$ itself. When $\hat{\mathcal{S}}_{\text{trust}}$ corresponds to graph nodes with complete outgoing action-label information, these guarantees have the standard interpretation that missing labels imply inapplicability.

For any feasible assignment, define

$$\hat{\mathbf{x}}_{\hat{s}} = (\hat{x}_{\hat{s},1}, \dots, \hat{x}_{\hat{s},m}) \in \{0, 1\}^m$$

for each $\hat{s} \in \hat{\mathcal{S}}$, and define

$$\begin{aligned} \text{Add}(a) &= \{p \in [m] \mid \text{add}_{a,p} = 1\}, & \text{Del}(a) &= \{p \in [m] \mid \text{del}_{a,p} = 1\}, \\ \text{Pre}^+(a) &= \{p \in [m] \mid \text{pre}_{a,p}^+ = 1\}, & \text{Pre}^-(a) &= \{p \in [m] \mid \text{pre}_{a,p}^- = 1\}. \end{aligned}$$

The induced grounded propositional STRIPS operator for action identifier a is

$$\hat{\omega}_a = \langle \text{Pre}^+(a), \text{Pre}^-(a), \text{Add}(a), \text{Del}(a) \rangle.$$

Its successor function is

$$\tau_{\hat{\omega}_a}(\mathbf{x})_p = \begin{cases} 1, & p \in \text{Add}(a), \\ 0, & p \in \text{Del}(a), \\ x_p, & \text{otherwise.} \end{cases}$$

Action a is applicable in $\mathbf{x}$ if $x_p = 1$ for all $p \in \text{Pre}^+(a)$ and $x_p = 0$ for all $p \in \text{Pre}^-(a)$.

Lemma 1 (Well-formed operators). *For every feasible assignment and every action identifier $a \in \mathcal{A}$,*

$$\text{Add}(a) \cap \text{Del}(a) = \emptyset \quad \text{and} \quad \text{Pre}^+(a) \cap \text{Pre}^-(a) = \emptyset.$$

Hence each induced operator $\hat{\omega}_a$ is a well-formed grounded propositional STRIPS operator.

Proof. This follows directly from Equation 8, which enforces

$$\text{add}_{a,p} + \text{del}_{a,p} \leq 1 \quad \text{and} \quad \text{pre}_{a,p}^+ + \text{pre}_{a,p}^- \leq 1$$

for all $a \in \mathcal{A}$ and $p \in [m]$. □

Lemma 2 (Bitwise STRIPS semantics under zero transition slack). *Fix an observed edge $\hat{e} = (\hat{s}, a, \hat{s}')$ and a predicate $p \in [m]$. If $\xi_{\hat{e},p} = 0$, then*

$$\hat{x}_{\hat{s}',p} = \tau_{\hat{\omega}_a}(\hat{\mathbf{x}}_{\hat{s}})_p.$$

Equivalently, when the transition slack for this edge-predicate pair is zero, the learned effects obey exact STRIPS add/delete/inertia semantics for predicate p .

Proof. By Equation 8, the pair $(\text{add}_{a,p}, \text{del}_{a,p})$ can take only one of three forms.

If $\text{add}_{a,p} = 1$, then $\text{del}_{a,p} = 0$, and Equation 4 gives

$$\hat{x}_{\hat{s}',p} \geq 1.$$

Thus $\hat{x}_{\hat{s}',p} = 1$.

If $\text{del}_{a,p} = 1$, then $\text{add}_{a,p} = 0$, and Equation 5 gives

$$\hat{x}_{\hat{s}',p} + 1 \leq 1.$$

Thus $\hat{x}_{\hat{s}',p} = 0$.

If $\text{add}_{a,p} = \text{del}_{a,p} = 0$, then Equation 6–Equation 7 reduce to

$$\hat{x}_{\hat{s}',p} - \hat{x}_{\hat{s},p} \leq 0 \quad \text{and} \quad \hat{x}_{\hat{s},p} - \hat{x}_{\hat{s}',p} \leq 0,$$

which imply $\hat{x}_{\hat{s}',p} = \hat{x}_{\hat{s},p}$. These are exactly the three cases in the definition of $\tau_{\hat{\omega}_a}(\hat{\mathbf{x}}_{\hat{s}})_p$. $\square$

Corollary 3 (Exact reproduction when transition slack is zero). *If $\xi_{\hat{e},p} = 0$ for all $p \in [m]$ on an observed edge $\hat{e} = (\hat{s}, a, \hat{s}')$, then*

$$\hat{\mathbf{x}}_{\hat{s}'} = \tau_{\hat{\omega}_a}(\hat{\mathbf{x}}_{\hat{s}}).$$

In particular, if $\sum \xi = 0$, then every observed edge in $\hat{E}$ is reproduced exactly in predicate space.

Proof. Apply Lemma 2 component-wise for all $p \in [m]$. $\square$

Lemma 4 (Observed edges satisfy learned preconditions). *For every observed edge $(\hat{s}, a, \hat{s}') \in \hat{E}$, action a is applicable in $\hat{\mathbf{x}}_{\hat{s}}$.*

Proof. Fix $(\hat{s}, a, \hat{s}') \in \hat{E}$ and $p \in [m]$. If $\text{pre}_{a,p}^+ = 1$, then Equation 2 gives

$$\hat{x}_{\hat{s},p} \geq 1,$$

so $\hat{x}_{\hat{s},p} = 1$. If $\text{pre}_{a,p}^- = 1$, then Equation 3 gives

$$\hat{x}_{\hat{s},p} + 1 \leq 1,$$

so $\hat{x}_{\hat{s},p} = 0$. Therefore all positive and negative preconditions of a hold in $\hat{\mathbf{x}}_{\hat{s}}$, and a is applicable. $\square$

Lemma 5 (Violation indicators are exact). *For every trusted missing pair $(\hat{s}, a) \in \hat{\mathcal{N}}$ and predicate $p \in [m]$, Equation 9–Equation 10 enforce*

$$\begin{aligned} v_{\hat{s},a,p}^+ &= 1 \iff (\text{pre}_{a,p}^+ = 1 \text{ and } \hat{x}_{\hat{s},p} = 0), \\ v_{\hat{s},a,p}^- &= 1 \iff (\text{pre}_{a,p}^- = 1 \text{ and } \hat{x}_{\hat{s},p} = 1). \end{aligned}$$

Thus $v_{\hat{s},a,p}^+$ and $v_{\hat{s},a,p}^-$ are exact indicators of positive- and negative-precondition violations.

Proof. We prove the claim for $v_{\hat{s},a,p}^+$; the argument for $v_{\hat{s},a,p}^-$ is analogous. Since all variables are binary, there are three relevant cases.

If $\text{pre}_{a,p}^+ = 0$, then Equation 9 gives $v_{\hat{s},a,p}^+ \leq 0$, so $v_{\hat{s},a,p}^+ = 0$. If $\text{pre}_{a,p}^+ = 1$ and $\hat{x}_{\hat{s},p} = 1$, then Equation 9 gives $v_{\hat{s},a,p}^+ \leq 1 - \hat{x}_{\hat{s},p} = 0$, so $v_{\hat{s},a,p}^+ = 0$. Finally, if $\text{pre}_{a,p}^+ = 1$ and $\hat{x}_{\hat{s},p} = 0$, then Equation 9 gives

$$v_{\hat{s},a,p}^+ \geq \text{pre}_{a,p}^+ - \hat{x}_{\hat{s},p} = 1,$$

so $v_{\hat{s},a,p}^+ = 1$. Therefore $v_{\hat{s},a,p}^+ = 1$ exactly when a positive precondition is violated. The proof for $v_{\hat{s},a,p}^-$ is identical, using Equation 10. $\square$

Lemma 6 (Trusted missing pairs are excluded when applicability slack is zero). *For every trusted missing pair $(\hat{s}, a) \in \hat{\mathcal{N}}$, if $\eta_{\hat{s},a} = 0$, then action a is not applicable in $\hat{\mathbf{x}}_{\hat{s}}$.*

Proof. By Equation 11, if $\eta_{\hat{s},a} = 0$, then

$$\sum_{p=1}^m (v_{\hat{s},a,p}^+ + v_{\hat{s},a,p}^-) \geq 1.$$

Hence, for some predicate p , either $v_{\hat{s},a,p}^+ = 1$ or $v_{\hat{s},a,p}^- = 1$. By Lemma 5, this means that either a positive precondition of a is false in $\hat{\mathbf{x}}_{\hat{s}}$, or a negative precondition of a is true in $\hat{\mathbf{x}}_{\hat{s}}$. In either case, a is not applicable in $\hat{\mathbf{x}}_{\hat{s}}$. $\square$

Proposition 7 (Every feasible solution is STRIPS-compatible). *Any feasible assignment to Equation 2–Equation 11, together with Equation 8, induces a family of well-formed grounded propositional STRIPS operators $\{\hat{\omega}_a\}_{a \in \mathcal{A}}$. This statement holds regardless of the values of the transition slack variables $\xi_{\hat{e},p}$ and applicability slack variables $\eta_{\hat{s},a}$.*

Proof. Well-formedness follows from Lemma 1. Transition slack only relaxes agreement with observed edges, and applicability slack only relaxes exclusion of trusted missing pairs. Neither changes $\text{Add}(a)$, $\text{Del}(a)$, $\text{Pre}^+(a)$, or $\text{Pre}^-(a)$, and therefore neither changes the STRIPS form of the induced operators. $\square$

Proposition 8 (Lexicographic optimization preserves semantics). *The lexicographic objective in Equation 1 does not change the semantic form of the learned model; it only ranks feasible STRIPS models according to slack and sparsity.*

Proof. The STRIPS semantics are determined by the structural constraints in the formulation. The objective in Equation 1 selects among assignments that already satisfy those constraints. Therefore any optimizer remains a feasible STRIPS model and inherits the properties proved above. $\square$

Corollary 9 (Node-level nondeterminism is incompatible with full distinctness and zero transition slack). *Assume full state distinctness, so that $\hat{\mathbf{x}}_{\hat{s}_1} \neq \hat{\mathbf{x}}_{\hat{s}_2}$ is required whenever $\hat{s}_1 \neq \hat{s}_2$. Also assume zero transition slack. If the learned graph contains*

$$(\hat{s}, a, \hat{s}_1) \in \hat{E} \quad \text{and} \quad (\hat{s}, a, \hat{s}_2) \in \hat{E}$$

with $\hat{s}_1 \neq \hat{s}_2$, then the formulation is infeasible.

Proof. By Corollary 3,

$$\hat{\mathbf{x}}_{\hat{s}_1} = \tau_{\hat{\omega}_a}(\hat{\mathbf{x}}_{\hat{s}}) = \hat{\mathbf{x}}_{\hat{s}_2}.$$

This contradicts the full distinctness requirement $\hat{\mathbf{x}}_{\hat{s}_1} \neq \hat{\mathbf{x}}_{\hat{s}_2}$. $\square$

Theorem 10 (Soundness and completeness of the exact encoding). *Fix the predicate dimension m , the trusted missing-pair set $\hat{\mathcal{N}}$, and the selected distinctness constraints. Consider the exact regime in which*

$$\sum_{\hat{e} \in \hat{E}} \sum_{p=1}^m \xi_{\hat{e},p} = 0 \quad \text{and} \quad \sum_{(\hat{s},a) \in \hat{\mathcal{N}}} \eta_{\hat{s},a} = 0.$$

Equivalently, all transition slack variables and all applicability slack variables are fixed to zero. Then the formulation consisting of Equation 2–Equation 11, together with Equation 8 and any selected distinctness constraints from Equation 12–Equation 16, is feasible if and only if there exist:

1. *predicate vectors $\hat{\mathbf{x}}_{\hat{s}} \in \{0,1\}^m$ for all $\hat{s} \in \hat{\mathcal{S}}$, and*
2. *grounded propositional STRIPS operators $\{\hat{\omega}_a\}_{a \in \mathcal{A}}$ over these m predicates,*

such that:

1. *every observed edge $(\hat{s}, a, \hat{s}') \in \hat{E}$ satisfies the preconditions of $\hat{\omega}_a$ at $\hat{\mathbf{x}}_{\hat{s}}$;*
2. *every observed edge $(\hat{s}, a, \hat{s}') \in \hat{E}$ satisfies*

$$\tau_{\hat{\omega}_a}(\hat{\mathbf{x}}_{\hat{s}}) = \hat{\mathbf{x}}_{\hat{s}'};$$

3. *every trusted missing pair $(\hat{s}, a) \in \hat{\mathcal{N}}$ is inapplicable in $\hat{\mathbf{x}}_{\hat{s}}$;*
4. *every selected distinctness constraint is satisfied.*

Proof. For the forward direction, suppose the formulation is feasible in the exact regime. By [Proposition 7](#), the induced operators are well-formed grounded propositional STRIPS operators. By [Lemma 4](#), every observed edge satisfies the learned preconditions. By [Corollary 3](#), every observed edge is reproduced exactly in predicate space. By [Lemma 6](#), every trusted missing pair is inapplicable. The selected distinctness constraints hold by construction whenever they are included.

For the reverse direction, suppose such predicate vectors and grounded propositional STRIPS operators exist. Assign $\hat{x}_{\hat{s},p}$ according to the given predicate vectors. Set $add_{a,p}$, $del_{a,p}$, $pre_{a,p}^+$, and $pre_{a,p}^-$ according to $Add(a)$, $Del(a)$, $Pre^+(a)$, and $Pre^-(a)$. Because the operators are well-formed, [Equation 8](#) holds.

For each observed edge $\hat{e} = (\hat{s}, a, \hat{s}') \in \hat{E}$, the assumed precondition satisfaction gives [Equation 2](#)–[Equation 3](#). The assumed successor equality

$$\tau_{\hat{\omega}_a}(\hat{\mathbf{x}}_{\hat{s}}) = \hat{\mathbf{x}}_{\hat{s}'}$$

gives [Equation 4](#)–[Equation 7](#) with $\xi_{\hat{e},p} = 0$ for all $p \in [m]$.

For each trusted missing pair $(\hat{s}, a) \in \hat{\mathcal{N}}$, set $v_{\hat{s},a,p}^+ = 1$ exactly when a positive precondition is violated at predicate p , and set $v_{\hat{s},a,p}^- = 1$ exactly when a negative precondition is violated. Then [Equation 9](#)–[Equation 10](#) hold. Since $(\hat{s}, a)$ is inapplicable, at least one learned precondition is violated, so [Equation 11](#) holds with $\eta_{\hat{s},a} = 0$. Finally, the selected distinctness constraints hold by assumption. Hence the CP-SAT formulation is feasible. $\square$

Interpretation. [Proposition 7](#) is the unconditional structural guarantee: every feasible assignment induces grounded propositional STRIPS operators. Slack affects agreement with the learned graph, not the form of the operators. When $\sum \xi = 0$, [Corollary 3](#) shows that every observed edge in $\hat{E}$ is reproduced exactly in predicate space. When $\sum \eta = 0$ as well, [Lemma 6](#) shows that every trusted missing pair in $\hat{\mathcal{N}}$ is ruled out by at least one learned precondition. The exact zero-slack correspondence is summarized by [Theorem 10](#).

C Extended Experiments and Results

RQ4: Is negative evidence necessary for learning useful preconditions? The main operator-learning formulation uses trusted missing state-action pairs $\hat{\mathcal{N}}$ as negative evidence. We ablate this design choice on DinnerTable and DinnerTable Real to test whether observed transitions alone are sufficient to learn useful preconditions.

We compare three conditions. The first is the standard STRIPS-WM setting used in the main experiments, which includes negative evidence. The second keeps the same solver setting but removes the negative-evidence constraints from $\hat{\mathcal{N}}$. The third also removes negative evidence, but adds a minimum-unique-state constraint

$$|\hat{X}| \geq |S^*|,$$

where S^* is the ground-truth task-state set used only for evaluation. This last condition tests whether simply forcing the solver to keep enough distinct predicate states can replace negative evidence.

[Fig. 5](#) shows that negative evidence is essential for planning. Without negative evidence, both domains collapse to two distinct predicate vectors under the standard relaxed-distinctness setting, and planning success drops to 0%. Adding the minimum-unique-state constraint prevents this two-state collapse, but it does not recover useful preconditions: planning remains poor because the learned model still permits thousands of unobserved state-action pairs.

The operator-learning statistics in [Table 5](#) show the same failure mode. With negative evidence, STRIPS-WM learns compact preconditions: DinnerTable has only 9 false-positive applicable state-action pairs, and DinnerTable Real has 13. Removing negative evidence increases this number to 6864 and 6893, respectively. The min-unique constraint does not fix the issue. It can force a larger predicate-state set, but it does not tell the solver which actions should be inapplicable in which states.

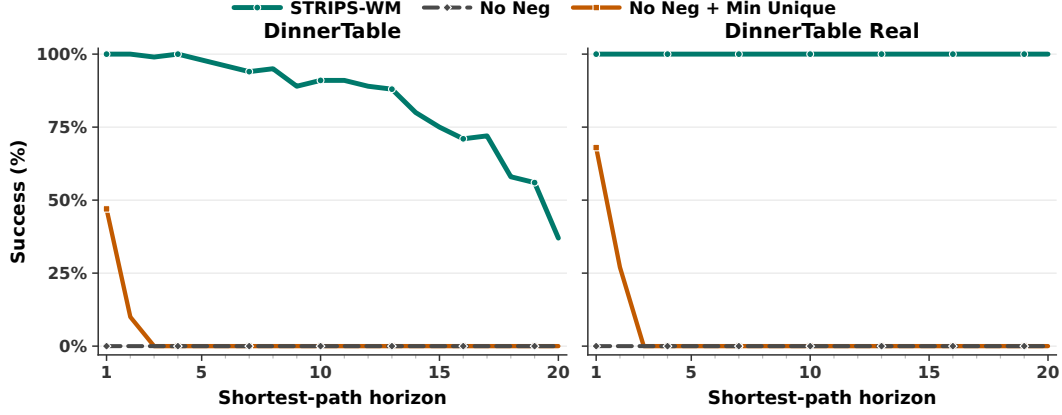


Figure 5: Negative-evidence ablation on DinnerTable and DinnerTable Real. Removing negative evidence makes planning fail, even when the solver is guided with the true minimum number of task states.

Domain	Condition	$ S^* $	$\sum \xi$	$\sum \eta$	False positives	$ \hat{X} $
DinnerTable	STRIPS-WM	101	0	9	9	101
DinnerTable	No neg.	101	0	0	6864	2
DinnerTable	No neg. + min-unique	101	0	0	6864	101
DinnerTable Real	STRIPS-WM	71	0	13	13	71
DinnerTable Real	No neg.	71	0	0	6893	2
DinnerTable Real	No neg. + min-unique	71	32	0	6893	83

Table 5: Negative-evidence ablation for operator learning. False positives count applicable state-action pairs that are not observed in the learned task graph. The min-unique condition adds the constraint $|\hat{X}| \geq |S^*|$, using the ground-truth task-state count only as a diagnostic guide.

The ablation highlights a failure mode that is hidden if we only look at whether observed transitions can be explained. Without negative evidence, the solver can fit the observed executions while leaving many unobserved actions applicable. The min-unique constraint changes the size of the predicate-state set, but it does not resolve this applicability ambiguity: DinnerTable still has 6864 false-positive state-action pairs, and DinnerTable Real has 6893. The trusted missing pairs $\hat{\mathcal{N}}$ therefore provide the signal that rules out these spurious applications, which is what makes the learned operators useful for long-horizon planning.