# Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning


**Rohan Chitnis*, Tom Silver*, Joshua B. Tenenbaum, Tomás Lozano-Pérez, Leslie Pack Kaelbling**

MIT Computer Science and Artificial Intelligence Laboratory

{ronuchit, tslvr, jbt, tlp, lpk}@mit.edu



## Abstract

In robotic domains, learning and planning are complicated by continuous state spaces, continuous action spaces, and long task horizons. In this work, we address these challenges with Neuro-Symbolic Relational Transition Models (NSRTs), a novel class of models that are data-efficient to learn, compatible with powerful robotic planning methods, and generalizable over objects. NSRTs have both symbolic and neural components, enabling a bilevel planning scheme where symbolic AI planning in an outer loop guides continuous planning with neural models in an inner loop. Experiments in four robotic planning domains show that NSRTs can be learned after only tens or hundreds of training episodes, and then used for fast planning in new tasks that require up to 60 actions and involve many more objects than were seen during training.


## 1 Introduction

For robots to plan effectively in the world, they need to contend with continuous state spaces, continuous action spaces, and long task horizons (Figure 1, bottom row). Symbolic AI planning techniques are able to solve tasks with very long horizons, but typically assume discrete, factored spaces (Helmert 2006). Neural network-based approaches have shown promise in continuous spaces, but scaling to long horizons remains challenging (Hafner et al. 2020; Chua et al. 2018). How can symbolic and neural planning methods be combined to overcome the limitations of each?

In this paper, we propose a new model-based approach for learning and planning in deterministic, goal-based, multi-task settings with continuous state and action spaces. Following previous work, we assume that a small number of discrete *predicates* (named relations over objects) are given, having been implemented by a human engineer (Lyu et al. 2019; Illanes et al. 2020; Wang et al. 2021), or learned from previous experience in similar domains. These predicates induce discrete *state abstractions* of the continuous environment state (Sacerdoti 1974; Kokel et al. 2021). For example, HOLDING(`block1`) abstracts away the continuous pose with which `block1` is held. Even when given predicates, the question of *how to make use of them* to learn effective models for planning in continuous state and action spaces is a hard problem that this paper seeks to address.

From the predicates, and through sequential interaction with an environment, we aim to learn: (1) *abstract actions*, which define transitions between abstract states; (2) an *abstract transition model*, with symbolic preconditions and effects akin to AI planning operators; (3) a *neural transition model* over the low-level, continuous state and action spaces; and (4) a set of *neural action samplers*, which define how abstract actions can be refined into continuous actions.

We unify all of these with a new class of models that we term the **N**euro-**S**ymbolic **R**elational **T**ransition Model (NSRT) (pronounced "insert"). NSRTs have both symbolic and neural components; all components are relational, permitting generalization to tasks with any number of objects and allowing sample-efficient learning.

To plan with NSRTs, we borrow techniques from search-then-sample task and motion planning (TAMP) (Garrett et al. 2021), with symbolic AI planning in an outer loop serving as guidance for continuous planning with neural models in an inner loop. This bilevel strategy allows for fast planning in environments with continuous state and action spaces, while avoiding the *downward refinability assumption*, which would assume planning can be decomposed into separate symbolic and continuous planning steps (Bacchus and Yang 1994). When modeling robotic domains symbolically, the predicates are often *lossy*, meaning that downward refinability cannot be assumed (Figure 1, top and middle).

This paper focuses on *how to learn NSRTs* and *how to use NSRTs for planning* in continuous-space, long-horizon tasks. We show in four robotic planning domains, across both the PyBullet (Coumans and Bai 2016) and AI2-THOR (Kolve et al. 2017) simulators, that NSRTs are extremely data-efficient: they can be learned in only tens or hundreds of training episodes. We also show that learned NSRTs allow for fast planning on new tasks, with many more objects than during training and long horizons of up to 60 actions. Baseline comparisons confirm that integrated neuro-symbolic reasoning is key to these successes.

## 2 Related Work

**Model-Based Reinforcement Learning (MBRL).** Our work is related to multi-task MBRL in that we learn and plan with transition models from data collected by environment interaction. Many recent approaches to deep MBRL learn unstructured neural transition models on a

* Equal contribution.

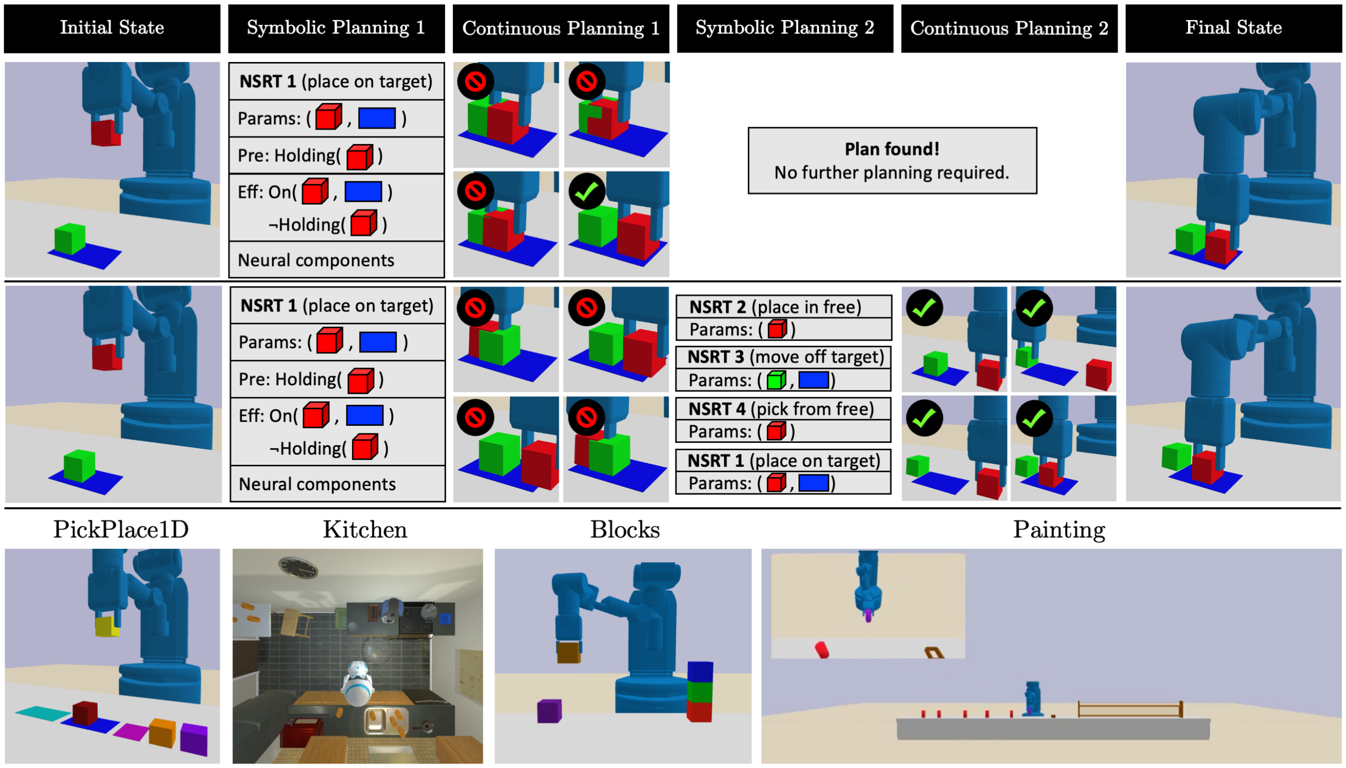


Figure 1: We propose Neuro-Symbolic Relational Transition Models (NSRTs). (Top row) Given the goal of placing the red block completely into the blue target region, we first perform AI planning with the symbolic NSRT components to find a one-step symbolic plan. The Continuous Planning 1 column shows various ways in which the agent attempts to *refine* this one-step symbolic plan into a ground action, using the neural components of (ground) NSRT 1; it finds a collision-free refinement, shown in the Final State column. (Middle row) Here, the green block is initially in a slightly different position, so the red block has no room to be placed into the blue target region. The initial symbolic plan is the same. However, this symbolic plan is not *downward refinable*, so Continuous Planning 1 fails. The agent then continues on to consider a four-step symbolic plan that first moves the green object away (Symbolic Planning 2 column), which is successfully refined in the Continuous Planning 2 column. This example illustrates that in the presence of complex geometric constraints which make symbolic abstractions lossy, integrated symbolic and continuous reasoning is necessary. (Bottom row) Screenshots of our four robotic planning environments. Kitchen uses the AI2-THOR simulator (Kolve et al. 2017); the others use PyBullet (Coumans and Bai 2016).

latent vector-space representation, and therefore must resort to highly undirected planning strategies like the cross-entropy method (Hamrick et al. 2021; Hafner et al. 2020, 2021). Some recent MBRL work has used more powerful planners, such as RRTs (Ichter, Sermanet, and Lynch 2020), LQR (Chebotar et al. 2017), and divide-and-conquer MCTS (Parascandolo et al. 2020). Relational MBRL is a subfield of MBRL that uses relational learning (Džeroski, De Raedt, and Driessens 2001; Tadepalli, Givan, and Driessens 2004) to learn object-centric factored transition models (Battaglia et al. 2016; Chang et al. 2017; Kansky et al. 2017) or to discover STRIPS operator models (Xia et al. 2019; Lang, Toussaint, and Kersting 2012) when given a set of predicates. Our work also learns relational transition models, but with a bilevel structure that allows planning without assuming downward refinability.

**Symbolic AI Planning for RL.** Our work continues a recent line of investigation that seeks to leverage symbolic AI planners for continuous states and actions. For example, previous work learns propositional (Dittadi, Drachmann, and Bolander 2020; Konidaris, Kaelbling, and Lozano-Perez 2018) or lifted (Arora et al. 2018; Chitnis et al. 2021; Asai 2019; Ames, Thackston, and Konidaris 2018; Ahmetoglu et al. 2020) symbolic transition models, and uses them with AI planners (Hoffmann 2001; Helmert 2006). Other related work has used symbolic planners as managers in hierarchical RL, where low-level option policies are learned (Lyu et al. 2019; Sarathy et al. 2020; Illanes et al. 2020; Yang et al. 2018; Kokel et al. 2021). This interface between symbolic planner and low-level policies assumes downward refinability, a critical assumption we do *not* make in this work.

**Learning for Hierarchical Planning.** Reasoning at multiple levels of abstraction is a key theme in hierarchical planning (Bercher, Alford, and Höller 2019). Prior work has considered learning transition models that are compatible with hierarchical planners, including those based on mixed-integer nonlinear programming (Say et al. 2017; Say 2021) or hierarchical task networks (Nejati, Langley, and

Konik 2006; Zhuo et al. 2009). Task and motion planning (TAMP) systems (Garrett et al. 2021) can plan effectively at long horizons (Kaelbling and Lozano-Pérez 2011; Srivastava et al. 2014), but they typically require hand-specified operators and action samplers, and a known low-level transition model. While recent work can learn one of the first two components (Loula et al. 2020; Silver et al. 2021; Wang et al. 2021; Chitnis et al. 2016; Kim, Kaelbling, and Lozano-Pérez 2018), our approach learns all three: the operators, the action samplers, and the low-level transition model.

## 3 Problem Setting

We study a deterministic, goal-based, multi-task setting with continuous object-oriented states, continuous actions, and a fixed, given set of predicates. Formally, we consider an *environment* $\langle \mathcal{T}, d, \mathcal{A}, f, \mathcal{P} \rangle$ and a collection of *tasks*, each of which is a tuple $\langle s_0, g, H \rangle$.

**Environments.** $\mathcal{T}$ is a set of object types, and $d : \mathcal{T} \to \mathbb{N}$ defines the dimensionality of the real-valued attribute (feature) vector of each object type. For example, an object of type `box` might have an attribute vector describing its current pose, side length, and color. An environment state $s$ is a mapping from a set of typed objects $o$ to attribute vectors of dimension $d(o)$, where $d(o)$ is shorthand for the dimension of the attribute vector of the type of object $o$. We use $\mathcal{S}$ to denote this object-oriented state space. The $\mathcal{A} \subseteq \mathbb{R}^m$ is the environment action space. The $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \cup \{fail\}$ is a deterministic transition function mapping a state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ to either a next state in $\mathcal{S}$ or a special failure state *fail*, which can be used, e.g., to capture undesirable behavior such as causing a collision. *The transition function $f$ is unknown; the agent only observes states through online interaction with the environment.*

$\mathcal{P}$ is a set of *predicates* given to the agent. A predicate is a named, binary-valued relation among some number of objects. A *ground atom* applies a predicate to specific objects, such as $\textsc{Above}(o_1, o_2)$, where the predicate is $\textsc{Above}$. A *lifted atom* applies a predicate to typed placeholder variables: $\textsc{Above}(?a, ?b)$. Taken together, the set of ground atoms that hold in a continuous state define a *discrete state abstraction*; let $\textsc{Abstract}(s)$ denote the abstract state for state $s \in \mathcal{S}$, and let $\mathcal{S}^{\uparrow}$ denote the abstract state space. For instance, a state $s$ where objects $o_1$, $o_2$, and $o_3$ are stacked may be represented by the abstract state $\textsc{Abstract}(s) = \{\textsc{On}(o_1, o_2), \textsc{On}(o_2, o_3)\}$; note that this abstract state loses details about the geometry of the scene.

**Tasks and Objective.** A task $\langle s_0, g, H \rangle$ is an initial state $s_0 \in \mathcal{S}$, a goal $g$, and a maximum horizon $H$. We will generally denote the set of objects in $s_0$ as $\mathcal{O}$. *This object set $\mathcal{O}$ is fixed within a task, but changes between tasks.* Goals $g$ are sets of ground atoms over the object set $\mathcal{O}$, such as $\{\textsc{On}(o_3, o_2), \textsc{On}(o_2, o_1)\}$. The agent interacts with the environment *episodically*. An episode begins with a task's initial state $s_0$. The agent takes actions sequentially, observing the state at each timestep. If it encounters a state $s$ for which $g \subseteq \textsc{Abstract}(s)$ (i.e., the goal holds), the episode is *solved*. An episode finishes when it is solved, when the failure state *fail* is reached, or after $H$ timesteps. We consider a set of *training tasks* and a set of *test tasks*; test tasks have more objects and longer horizons, and are unknown to the agent during training. The agent's objective is to maximize the number of episodes solved over the test tasks.

**Data Collection.** We focus on the problems of *learning* and *planning*. To isolate these, we use a simple, fixed strategy for data collection that makes use of a *behavior prior* $\pi_0(\cdot \mid s)$, a state-conditioned distribution over $\mathcal{A}$. Recent work has studied learning behavior priors (Ajay et al. 2021); we are assuming it is given, but it could be learned. Data-gathering proceeds by running $\pi_0$ on tasks sampled from the set of training tasks. We do not use $\pi_0$ at test time. Since $\pi_0$ is fixed and given, our setting can be seen as model-based *offline reinforcement learning* (Levine et al. 2020).

## 4 NSRT Representation

The next three sections introduce Neuro-Symbolic Relational Transition Models (NSRTs). In this section, we describe the NSRT representation; in Section 5, we address planning with NSRTs; and in Section 6, we discuss learning NSRTs. Figure 2 illustrates the full pipeline.

We want models that are *learnable*, *plannable*, and *generalizable*. To that end, we propose the following definition:

**Definition 1.** *A* Neuro-Symbolic Relational Transition Model (NSRT) *is a tuple $\langle O, P, E, h, \pi \rangle$, where:*

- *$O = (o_1, \ldots, o_k)$ is an ordered list of* parameters*; each $o_i$ is a variable of some type from type set $\mathcal{T}$.*
- *$P$ is a set of* symbolic preconditions*; each precondition is a lifted atom over parameters $O$.*
- *$E = (E^+, E^-)$ is a tuple of* symbolic effects*. $E^+$ are* add effects*, and $E^-$ are* delete effects*; both are sets of lifted atoms over parameters $O$.*
- *$h : \mathbb{R}^{d(o_1)+\cdots+d(o_k)} \times \mathcal{A} \to \mathbb{R}^{d(o_1)+\cdots+d(o_k)}$ is a* low-level transition model*, a neural network that predicts next attribute values given current ones and an action.*
- *$\pi(a \mid v)$ is an* action sampler*, a neural network defining a conditional distribution over actions $a \in \mathcal{A}$, where $v \in \mathbb{R}^{d(o_1)+\cdots+d(o_k)}$ is a vector of attribute values.*

In this paper, we will learn and plan with a *collection* of NSRTs. Together with the object set $\mathcal{O}$ of a task, a collection of NSRTs jointly defines four things: an *abstract action space* for efficient planning; a (partial) *abstract transition model* over the abstract state space $\mathcal{S}^{\uparrow}$ and the abstract action space; a (partial) *low-level transition model* over environment states and actions; and *action samplers* to refine abstract actions into environment actions. The rest of this section describes how NSRTs define these four components.

First, we define the notion of an NSRT *grounded* with objects, which represents an abstract action for a task:

**Definition 2.** *Given an object set $\mathcal{O}$, a* ground NSRT *is an NSRT whose parameters $o_i \in O$ are replaced by objects from $\mathcal{O}$, following a bijective* substitution *$\sigma$ mapping each $o_i$ to an object. The ground preconditions and effects under $\sigma$ are denoted $P_\sigma$ and $E_\sigma$ respectively.*

Given a set of NSRTs and a task with object set $\mathcal{O}$, the resulting set of *ground* NSRTs defines an *abstract* action space for that task, which we denote as $\mathcal{A}^{\uparrow}$. Therefore, the phrases *abstract action* and *ground NSRT* are interchangeable. For instance, say we wrote an NSRT called STACK with two parameters $?x$ and $?y$; let $\sigma = \{?x \mapsto o_3, ?y \mapsto o_6\}$. Then $\textsc{Stack}(o_3, o_6)$ is an abstract action with substitution $\sigma$.

Working toward a definition of the abstract transition model, we next define ground NSRT *applicability*.

**Definition 3.** *A ground NSRT with preconditions $P_\sigma$ is* applicable *in state $s \in \mathcal{S}$ if $P_\sigma \subseteq \textsc{Abstract}(s)$. It is also* applicable *in abstract state $s^\uparrow \in \mathcal{S}^\uparrow$ if $P_\sigma \subseteq s^\uparrow$.*

In words, applicability simply checks that the ground NSRT's precondition atoms are a subset of the abstract state atoms. A set of ground NSRTs defines a (partial) *abstract transition model* $f^\uparrow : \mathcal{S}^\uparrow \times \mathcal{A}^\uparrow \to \mathcal{S}^\uparrow$, which maps an abstract state and abstract action (ground NSRT) to a next abstract state. The $f^\uparrow(s^\uparrow, a^\uparrow)$ is partial since it is only defined when $a^\uparrow$ is applicable in $s^\uparrow$; when it *is* applicable, we have:

$$f^\uparrow(s^\uparrow, a^\uparrow) = (s^\uparrow \setminus E^-_\sigma) \cup E^+_\sigma, \qquad \text{(Equation 1)}$$

where $E_\sigma = (E^+_\sigma, E^-_\sigma)$ are the effects for $a^\uparrow$. In words, this abstract transition model removes delete effects and includes add effects, as long as the preconditions of the ground NSRT are satisfied. This symbolic representation is akin to operators in classical AI planning (Bonet and Geffner 2001); we use this to our advantage in Section 5.

What is the connection between the symbolic components of an NSRT ($P$ and $E$) and the environment transitions? To answer this question, we use the following definition:

**Definition 4.** *A ground NSRT $a^\uparrow$ with effects $(E^+_\sigma, E^-_\sigma)$* covers *an environment transition $\tau = (s, a, s')$, denoted $a^\uparrow \models \tau$, if (1) the ground NSRT is applicable in $s$; (2) $E^+_\sigma = \textsc{Abstract}(s') \setminus \textsc{Abstract}(s)$; and (3) $E^-_\sigma = \textsc{Abstract}(s) \setminus \textsc{Abstract}(s')$.*

We assume that the following *weak semantics* connect $P$ and $E$ with the environment: for each ground NSRT $a^\uparrow$, there exists a state $s \in \mathcal{S}$ and there exists an action $a \in \mathcal{A}$ s.t. $a^\uparrow \models (s, a, f(s, a))$. Importantly, this means that the abstraction defined by the NSRTs does not satisfy downward refinability (Marthi, Russell, and Wolfe 2007), which would have required the "there exists a state" to be "for all states." These weak semantics will make learning efficient (Section 6), but will require integrated planning (Section 5).

To plan, it is important to be able to simulate the effects of actions on the continuous environment state. The low-level transition model $h$, which we discuss next, is used for this.

**Definition 5.** *Given a state $s$ and ground NSRT $a^\uparrow$ with substitution $\sigma$, the* context *of $s$ for $a^\uparrow$ is $v_\sigma(s) = s[\sigma(o_1)] \circ \cdots \circ s[\sigma(o_k)]$, where $v_\sigma(s) \in \mathbb{R}^{d(o_1)+\cdots+d(o_k)}$, $s[\cdot]$ looks up an object's attribute vector in $s$, and $\circ$ is vector concatenation.*

In words, the context for a ground NSRT is the subset of a state's attribute vectors that correspond to the ground NSRT's objects, assembled into a vector. The context is the input to the low-level neural transition model $h$:

$$h(v_\sigma(s), a) \approx v_\sigma(f(s, a)),$$

where, recall, $f$ is the *unknown* environment transition model. All objects not in $\sigma$ are predicted to be unchanged.

Finally, the neural action sampler $\pi$ of an NSRT connects the abstract and environment action spaces: it samples continuous actions from the environment action space $\mathcal{A}$ that lead to the NSRT's symbolic effects. Given a state $s$ and applicable ground NSRT with substitution $\sigma$, if $a \sim \pi(\cdot \mid v_\sigma(s))$, then $(s, a, f(s, a))$ should ideally be covered by the ground NSRT. The fact that $\pi$ is stochastic can be useful for planning, where multiple samples may be required to achieve desired effects (see Figure 1, or Wang et al. (2021)).

*There are three key properties of NSRTs to take away from these definitions.* (1) NSRTs are fully relational, i.e., invariant over object identities. This leads to data-efficient learning and generalization to novel tasks and objects. (2) NSRTs do not assume downward refinability, as discussed above. (3) NSRTs are *locally scoped*; all components of a ground NSRT are defined only where it is applicable. This modularity leads to independent learning problems; see Section 6.

## 5 Neuro-Symbolic Planning with NSRTs

We now describe how NSRTs can be used to plan in a given task. Recall that the weak semantics of NSRTs (Section 4) do *not* guarantee downward refinability: abstract actions that achieve a goal cannot necessarily be turned into environment actions achieving that goal. Our strategy will be to perform integrated bilevel planning, with an outer search in the abstract space informing an inner loop producing environment actions. This planning strategy falls under the broad class of search-then-sample TAMP techniques (Garrett et al. 2021). *See Appendix A.1 for pseudocode (Algorithm 1).*

**Symbolic Planning.** We perform an outer A$^*$ search from $\textsc{Abstract}(s_0)$ to $g$, with the abstract transition model of Equation 1 and uniform action costs. For the search heuristic, we use $h_{add}$, a domain-independent heuristic from the symbolic planning literature (Bonet and Geffner 2001) that approximates the state-to-goal distance under a delete relaxation of the abstract model. This A$^*$ search will find candidate *symbolic plans*: sequences of ground NSRTs $a^\uparrow \in \mathcal{A}^\uparrow$.

**Continuous Planning.** For each candidate symbolic plan, an inner loop attempts to refine it into a *plan* — a sequence of actions $a \in \mathcal{A}$ that achieves the goal $g$ — using the neural components of the NSRTs. We use the action sampler $\pi$ and low-level transition model $h$ of each ground NSRT in the symbolic plan to construct an *imagined* state-action trajectory starting from the initial state $s_0$. If the goal $g$ holds in the final imagined state, we are done. If $g$ does not hold, or if any state's abstraction does not equal the expected abstract state according to the A$^*$ search, then we attempt to sample again. After $n_{\text{trials}}$ (a hyperparameter) unsuccessful imagined trajectories, we return control to the A$^*$ search.

**Handling Failures.** Recall that an episode ends if the failure state *fail* is reached. Following Srivastava et al. (2014), we would like to use the presence of a failure state during continuous planning to inform symbolic planning. In Appendix A.2, we describe a simple domain-independent procedure for learning to predict transitions to the *fail* state, and using this learned model during planning. This optimization propagates failure information back to the symbolic level and guides the A$^*$ away from repeating such situations.

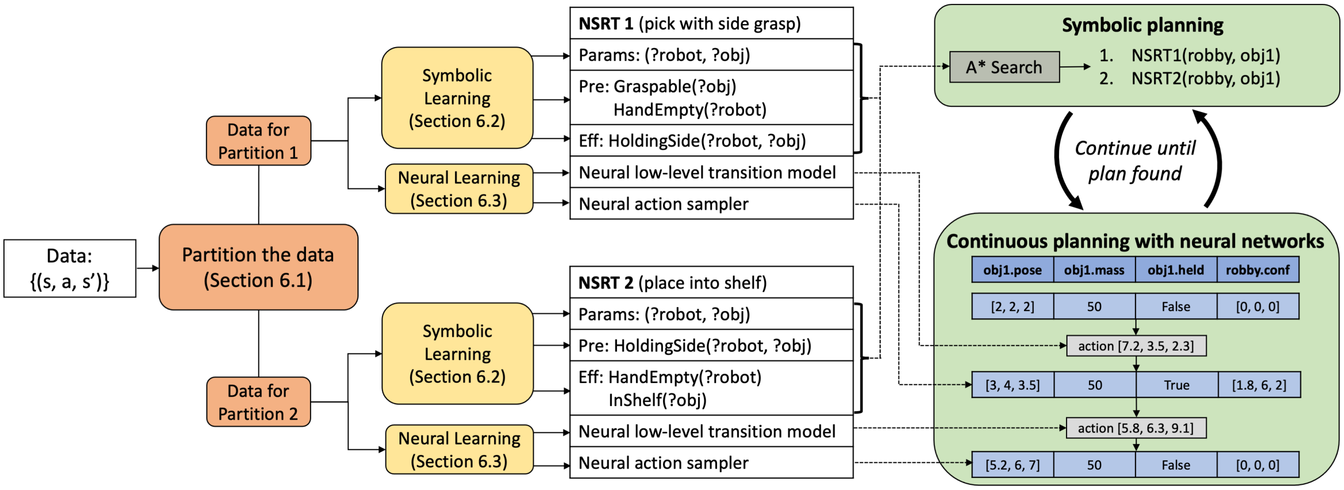


Figure 2: Our pipeline, with a simplified Painting example. An NSRT (Section 4) contains both symbolic components used for A$^*$ search with AI planning heuristics, and neural components used for continuous planning. The example NSRTs shown in the middle require that a robot must be side-grasping an object to place it into a shelf. These NSRTs are *not* ground: their parameters are variables, so these NSRTs can be applied to any objects. We learn NSRTs from transition data (Section 6), and then perform bilevel planning with the learned NSRTs (Section 5). Delete effects are omitted from this figure for visual clarity.

## 6 Learning NSRTs

We now address learning the structure (Section 6.1), the symbolic components (Section 6.2), and the neural components (Section 6.3) of NSRTs. *See Appendix A.1 for pseudocode (Algorithm 2) and an example.*

### 6.1 Partitioning the Transition Data

Recall that data collection (Section 3) gathers a set of samples from the unknown transition model $f$: each sample is a state $s \in \mathcal{S}$, an action $a \in \mathcal{A}$, and either a next state $s' \in \mathcal{S}$ or the failure state *fail*. We will ignore the transitions that led to *fail* here; see Appendix A.2. We begin by partitioning the set of transitions $\tau = (s, a, s')$ so that each partition $\psi \in \Psi$ will correspond to a single NSRT, thus determining the number of learned NSRTs. Two transitions belong to the same partition iff their symbolic effects can be *unified*:

**Definition 6.** *Two transitions $\tau_1$ and $\tau_2$ can be unified if there exists a bijective mapping $\sigma$ from the objects in* $\textsc{Eff}(\tau_1)$ *to the objects in* $\textsc{Eff}(\tau_2)$ *s.t.* $\sigma[\textsc{Eff}(\tau_1)] = \textsc{Eff}(\tau_2)$*, where* $\textsc{Eff}(\tau) = (\textsc{Abstract}(s') \setminus \textsc{Abstract}(s), \textsc{Abstract}(s) \setminus \textsc{Abstract}(s'))$*, and $\sigma[\cdot]$ denotes substitution following $\sigma$.*

These partitions can be computed in time linear in the number of transitions, objects, and atoms per effect set.

### 6.2 Learning the Symbolic Components

We now show how to learn NSRT parameters $O$, symbolic preconditions $P$, and symbolic effects $E$ for each partition $\psi \in \Psi$. First, we define a mapping $\textsc{Ref}$ that maps a transition $\tau$ to a subset of objects in $\tau$ that are "involved" in the transition. In practice, we implement $\textsc{Ref}(\tau)$ by selecting all objects that appear in $\textsc{Eff}(\tau)$.[1] By construction of our partitions, every transition $\tau \in \psi$ will have equivalent $\textsc{Ref}(\tau)$, up to object renaming. We thus introduce NSRT parameters $O$ corresponding to the types of all the objects in any arbitrarily chosen transition's $\textsc{Ref}(\tau)$. For each $\tau \in \psi$, let $\sigma_\tau$ be a bijective mapping from these parameters $O$ to the objects in $\textsc{Ref}(\tau)$. The NSRT symbolic effects follow by construction: $E = \sigma_\tau^{-1}[\textsc{Eff}(\tau)]$ for any arbitrarily chosen $\tau \in \psi$.

To learn the symbolic preconditions $P$ for the NSRT corresponding to partition $\psi$, we use a simple inductive approach (Bonet, Frances, and Geffner 2019) that restricts learning by assuming that for each lifted effect set seen in the data, there is exactly one lifted precondition set.[2] By this assumption, the preconditions follow from an intersection:

$$P = \bigcap_{\tau=(s,\cdot,\cdot)\in\psi} \sigma_\tau^{-1}[\textsc{Project}(\textsc{Abstract}(s))],$$

where $\textsc{Project}$ maps $\textsc{Abstract}(s)$ to the subset of atoms whose objects are all in $\textsc{Ref}(\tau)$. *See the example in Appendix A.1.* Note that by construction, two different learned NSRTs cannot cover (Definition 4) the same transition.

### 6.3 Learning the Neural Components

We now describe how to learn a low-level transition model $h$ and action sampler $\pi$ for each partition's NSRT. The key idea is to use the state projections computed during partitioning to create regression problems. Recalling Definition 5, let $v_\sigma = s[\sigma_\tau(o_1)] \circ \cdots \circ s[\sigma_\tau(o_k)]$ denote the context of state

[1]This suffices for our experiments, but it cannot capture "action at a distance," where some object influences a transition without itself changing; other implementations of $\textsc{Ref}$ could be used.

[2]See Silver et al. (2021) for an alternative method that avoids this assumption, with greater computational cost.

$s$ from transition $\tau$, where $(o_1, o_2, \ldots, o_k)$ are the NSRT parameters. In words, $v_\sigma$ is a vector of the attribute values in state $s$ corresponding to the objects that map the ground atoms $\text{EFF}(\tau)$ of the transition to the lifted effects $E$ of the NSRT. We can do the same to produce $v_{\sigma'}$ for $s'$. Applying this to all transitions in $\psi$ gives us a dataset of $(v_\sigma, a, v_{\sigma'})$.

Recall that we want to learn $h$ such that $h(v_\sigma(s), a) \approx v_\sigma(f(s, a))$. With the dataset above, this learning problem now reduces to regression, with $v_\sigma$ and $a$ being the inputs and $v_{\sigma'}$ being the output. We use a fully connected neural network (FCN) as the regressor, trained to minimize mean-squared error. Learning $\pi$ requires *distribution* regression, where we fit $P(a \mid v_\sigma)$ to the transitions $(v_\sigma, a, \cdot)$. We use an FCN that takes $v_\sigma$ as input and predicts the mean $\mu$ and covariance matrix $\Sigma$ of a Gaussian. This FCN is trained to maximize the likelihood of action $a$ under $\mathcal{N}(\mu, \Sigma)$.[3] Since Gaussians have limited expressivity, we also learn an *applicability classifier* that maps pairs $(v_\sigma, a)$ to 0 or 1, implemented as an FCN with binary cross-entropy loss. To sample from $\pi$, we then rejection sample from the Gaussian.[4]

# 7 Experiments

Our empirical evaluations address the following key questions: **(Q1)** Can NSRTs be learned data-efficiently? **(Q2)** Can learned NSRTs be used to plan to long horizons, especially in tasks involving new and more objects than were seen during training? **(Q3)** Is bilevel planning efficient and effective, and are both levels needed? **(Q4)** To what extent are learned action samplers useful for planning?

## 7.1 Experimental Setup

We evaluate Q1-Q4 by running eight methods on four environments. All experiments were run on Ubuntu 18.04 using 4 CPU cores of an Intel Xeon Platinum 8260 processor.

**Environments.** In this section, we describe our four environments at a high level, with details in Appendix A.3. The environments are illustrated in Figure 1 (bottom row). Each environment has three sets of tasks: training, "easy" test, and "hard" test. "Hard" test tasks require generalization to more objects. In all environments, we transition to the failure state *fail* whenever a geometric collision occurs.

- *Environment 1:* In "PickPlace1D," a robot must pick blocks and place them into designated target regions on a table. All poses are 1D. Some placements are obstructed by movable objects; none of the predicates capture obstructions, leading to a lack of downward refinability.
- *Environment 2:* In "Kitchen," a robot waiter in 3D must pick cups, fill them with water, wine, or coffee, and serve them to customers. Some cups are too heavy to be lifted; the cup masses are not represented by the predicates, leading to a lack of downward refinability.
- *Environment 3:* In "Blocks," a robot in 3D must stack blocks on a table to make towers. In this environment only, the downward refinability assumption holds.
- *Environment 4:* In "Painting," a robot in 3D must pick, wash, dry, paint, and place widgets into a box or shelf. Placing into the box (resp. shelf) requires picking with a top (resp. side) grasp. All widgets must be painted a particular color before being placed, which first requires washing/drying if the widget starts off dirty or wet. The box has a lid that may obstruct placements; whether the lid will obstruct a placement is not represented symbolically, leading to a lack of downward refinability.

**Methods Evaluated.** We evaluate the following methods. See Appendix A.4 for additional details.

- *Ours: Bilevel planning with NSRTs.* This is our main approach. Plans are executed open-loop.
- *B1: Symbolic planning only.* This baseline performs symbolic planning using the symbolic components of the learned NSRTs. When a symbolic plan is found that reaches the goal, it is immediately executed by calling the learned action samplers for the corresponding ground NSRTs in sequence, open-loop. The low-level transition models are not used. This baseline ablates away our integrated planner and assumes downward refinability.
- *B2: Neural planning only with forward shooting.* This baseline randomly samples $H$-length sequences of ground NSRTs and uses their neural components to imagine a trajectory, repeating until it finds a trajectory where the final state satisfies the goal. This baseline does not use the symbolic components of the NSRTs, and thus can be seen as an ablation of the symbolic planning.
- *B3: Neural planning only with hill climbing.* This baseline performs local search over full plans. At each iteration, a random plan step is resampled using the learned action sampler of a random NSRT. The new plan is rejected unless it improves the number of goal atoms satisfied in the final imagined state. As in B2, the symbolic components of the NSRTs are not used.
- *B4: GNN action-value function learning.* This "model-free" baseline trains a goal-conditioned graph neural network (GNN) action-value function using fitted Q-iteration. The GNN takes as input a continuous low-level state, the corresponding abstract state, and a continuous action; it outputs expected discounted future returns. At evaluation time, given a state, we draw several candidate actions from the behavior prior $\pi_0$, and take the best one.
- *B5: Behavior prior only.* This baseline takes actions that are directly sampled from the prior $\pi_0$.
- *B6: Bilevel planning with prior.* This baseline is an ablation of our main approach that does not use the learned NSRT action samplers $\pi$. Instead, actions are selected by rejection sampling from the prior $\pi_0$, with rejections determined by checking the learned applicability classifier.
- *B7: Forward shooting with prior.* This baseline uses the forward shooting of B2 with rejection sampling from $\pi_0$, like B6. Only the low-level transition models $h$ are used.

All methods except B5 (non-learning) receive the same data.

## 7.2 Results and Discussion

See Figure 3 for learning curves. The main observation is that in all environments, our method quickly learns to solve tasks within the allotted 3-second timeout. Thus, **Q1** and **Q2**

[3] Here, we are assuming that the desired action distribution has nonzero measure. In practice, $\Sigma$ can be arbitrarily small.

[4] If this fails the applicability classifier enough times (10 in experiments), we terminate the inner loop and continue the A* search.

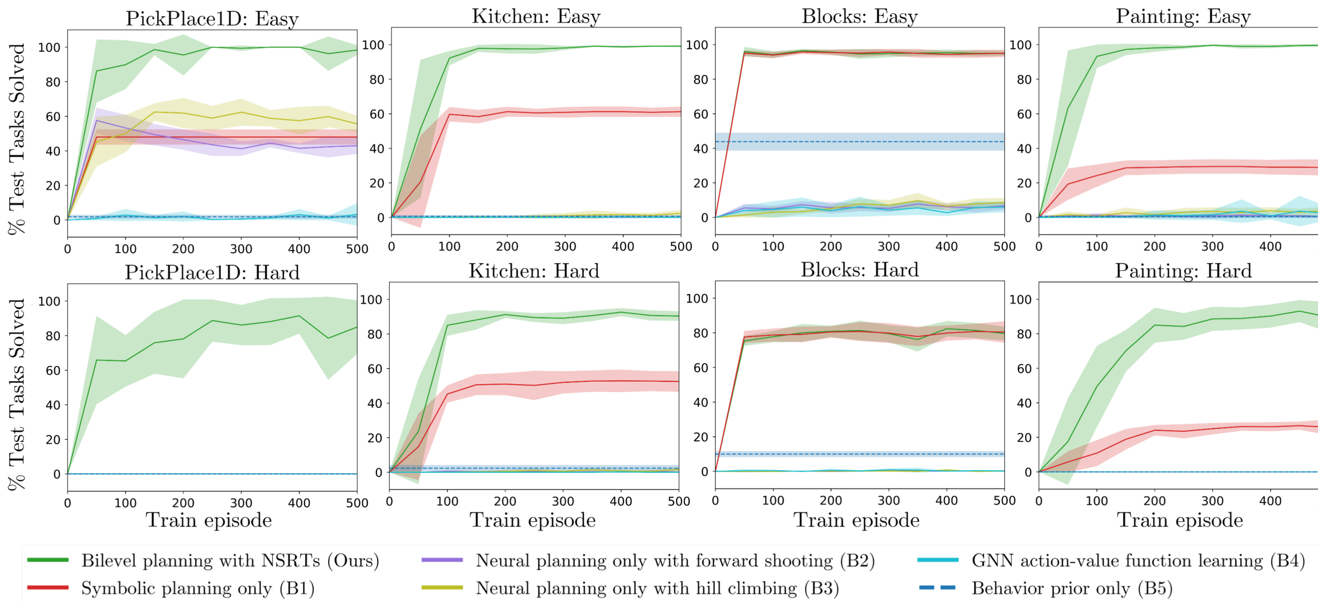


Figure 3: Learning curves for every environment, showing the percentage of 100 randomly generated test tasks (top row: easy tasks; bottom row: hard tasks) solved as a function of the number of training episodes. Each curve depicts a mean over 8 seeds, with standard deviation shaded. All methods are given a timeout of 3 seconds per task. We can see that our method (green) quickly learns to solve many more tasks than all the baselines, especially in the hard tasks of each environment.

| | PickPlace1D | | Kitchen | | Blocks | | Painting | |
|---|---|---|---|---|---|---|---|---|
| *Methods* | Easy | Hard | Easy | Hard | Easy | Hard | Easy | Hard |
| Bilevel planning with NSRTs (Ours) | **98.4** | **85.0** | **99.1** | **90.4** | **95.0** | **79.6** | **99.6** | **89.6** |
| Bilevel planning with prior (B6) | **95.9** | 46.4 | 71.9 | 32.6 | 89.9 | 53.4 | 84.5 | 0.1 |
| Forward shooting with prior (B7) | 71.1 | 0.0 | 0.0 | 1.5 | 62.9 | 8.6 | 5.4 | 0.0 |

Table 1: Percentage of 100 randomly generated test tasks solved after 500 episodes of training. Each number is a mean over 8 seeds; bold results are within one standard deviation of best (Appendix A.5).

can be answered affirmatively. Turning to **Q3**, we can study whether bilevel planning is effective by comparing Ours, B1, and B2. The gap between Ours and B1 shows the importance of integrated bilevel planning. B1 will not be effective in any environment where downward refinability does not hold — only Blocks is downward refinable, which explains the identical performance of Ours and B1 there. B2 fails in most cases, confirming the usefulness of the learned abstractions.

Both B3 and B4 are generally ineffective. B3 performs local search, which is much weaker than our directed A$^*$. B4 is model-free, forgoing planning in favor of learning an action-value function directly; such strategies are known to be more data-hungry (Moerland, Broekens, and Jonker 2020). B5 does not require any training, and is just included to illustrate the performance of the behavior prior alone.

To evaluate **Q4**, we turn to an ablation study. Table 1 compares our method with B6 and B7, both of which rejection sample from the generic behavior prior $\pi_0$ rather than using our learned NSRT action samplers. First, comparing B6 and B7, bilevel planning is much better than shooting, which speaks to the benefits of using the symbolic components of the NSRTs to guide the continuous planning; this conclusion was also supported by Figure 3. Second, comparing Ours and B6, the learned action samplers help substantially versus rejection sampling from $\pi_0$. This is because the behavior prior is highly generic, not targeted toward any specific set of effects like NSRT action samplers are, so rejection sampling can take many tries to pass the applicability classifier.

## 8 Conclusion, Limitations, and Future Work

We proposed NSRTs for long-horizon, goal-based, object-oriented planning tasks. We showed that their neuro-symbolic structure affords fast bilevel planning, and found experimentally that they are data-efficient to learn and effective at generalization, outperforming several baselines.

Key limitations of the current work include: (1) that predicates are given; (2) that a behavior prior is given for data collection; (3) that environments are deterministic and fully observable. To address (1), NSRTs could be combined with work on learning predicates from high-dimensional inputs (Asai 2019). (2) could be addressed using skill prior learning techniques (Ajay et al. 2021). For (3), we hope to draw on TAMP techniques for handling stochasticity and partial observability (Hadfield-Menell et al. 2015; Garrett et al. 2020).

## References


Ahmetoglu, A.; Seker, M. Y.; Sayin, A.; Bugur, S.; Piater, J.; Oztop, E.; and Ugur, E. 2020. DeepSym: Deep Symbol Generation and Rule Learning from Unsupervised Continuous Robot Interaction for Planning. *arXiv preprint arXiv:2012.02532*.

Ajay, A.; Kumar, A.; Agrawal, P.; Levine, S.; and Nachum, O. 2021. OPAL: Offline Primitive Discovery for Accelerating Offline Reinforcement Learning. In *International Conference on Learning Representations*.

Alkhazraji, Y.; Frorath, M.; Grützner, M.; Helmert, M.; Liebetraut, T.; Mattmüller, R.; Ortlieb, M.; Seipp, J.; Springenberg, T.; Stahl, P.; and Wülfing, J. 2020. Pyperplan.

Ames, B.; Thackston, A.; and Konidaris, G. 2018. Learning symbolic representations for planning with parameterized skills. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 526–533. IEEE.

Arora, A.; Fiorino, H.; Pellier, D.; Etivier, M.; and Pesty, S. 2018. A review of learning planning action models. *Knowledge Engineering Review*, 33.

Asai, M. 2019. Unsupervised grounding of plannable first-order logic representation from images. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 583–591.

Bacchus, F.; and Yang, Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71(1): 43–100.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Battaglia, P. W.; Pascanu, R.; Lai, M.; Rezende, D. J.; and Kavukcuoglu, K. 2016. Interaction Networks for Learning about Objects, Relations and Physics. In *Advances in Neural Information Processing Systems*.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning-One Abstract Idea, Many Concrete Realizations. In *IJCAI*, 6267–6275.

Bonet, B.; Frances, G.; and Geffner, H. 2019. Learning features and abstract actions for computing generalized plans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2703–2710.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Chang, M. B.; Ullman, T.; Torralba, A.; and Tenenbaum, J. B. 2017. A compositional object-based approach to learning physical dynamics. In *International Conference on Learning Representations*.

Chebotar, Y.; Hausman, K.; Zhang, M.; Sukhatme, G.; Schaal, S.; and Levine, S. 2017. Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 703–711.

Chitnis, R.; Hadfield-Menell, D.; Gupta, A.; Srivastava, S.; Groshev, E.; Lin, C.; and Abbeel, P. 2016. Guided search for task and motion plans using learned heuristics. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 447–454. IEEE.

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

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In *Advances in Neural Information Processing Systems*, volume 31.

Clevert, D.; Unterthiner, T.; and Hochreiter, S. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *4th International Conference on Learning Representations*.

Coumans, E.; and Bai, Y. 2016. PyBullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*.

Dittadi, A.; Drachmann, F. K.; and Bolander, T. 2020. Planning From Pixels in Atari With Learned Symbolic Representations. *arXiv preprint arXiv:2012.09126*.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine learning*, 43(1): 7–52.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4: 265–293.

Garrett, C. R.; Paxton, C.; Lozano-Pérez, T.; Kaelbling, L. P.; and Fox, D. 2020. Online replanning in belief space for partially observable task and motion problems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 5678–5684. IEEE.

Hadfield-Menell, D.; Groshev, E.; Chitnis, R.; and Abbeel, P. 2015. Modular task and motion planning in belief space. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4991–4998. IEEE.

Hafner, D.; Lillicrap, T.; Ba, J.; and Norouzi, M. 2020. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*.

Hafner, D.; Lillicrap, T. P.; Norouzi, M.; and Ba, J. 2021. Mastering Atari with Discrete World Models. In *International Conference on Learning Representations*.

Hamrick, J. B.; Friesen, A. L.; Behbahani, F.; Guez, A.; Viola, F.; Witherspoon, S.; Anthony, T.; Buesing, L. H.; Veličković, P.; and Weber, T. 2021. On the role of planning in model-based deep reinforcement learning. In *International Conference on Learning Representations*.

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

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine*, 22(3): 57–57.

Ichter, B.; Sermanet, P.; and Lynch, C. 2020. Broadly-Exploring, Local-Policy Trees for Long-Horizon Task Planning. *arXiv preprint arXiv:2010.06491*.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 540–550.

Kaelbling, L. P.; and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 1470–1477. IEEE.

Kansky, K.; Silver, T.; Mély, D. A.; Eldawy, M.; Lázaro-Gredilla, M.; Lou, X.; Dorfman, N.; Sidor, S.; Phoenix, S.; and George, D. 2017. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *International Conference on Machine Learning*, 1809–1818.

Kim, B.; Kaelbling, L. P.; and Lozano-Pérez, T. 2018. Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kokel, H.; Manoharan, A.; Natarajan, S.; Balaraman, R.; and Tadepalli, P. 2021. RePReL: Integrating Relational Planning and Reinforcement Learning for Effective Abstraction. In *Thirty First International Conference on Automated Planning and Scheduling (ICAPS)*.

Kolve, E.; Mottaghi, R.; Han, W.; VanderBilt, E.; Weihs, L.; Herrasti, A.; Gordon, D.; Zhu, Y.; Gupta, A.; and Farhadi, A. 2017. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Lang, T.; Toussaint, M.; and Kersting, K. 2012. Exploration in relational domains for model-based reinforcement learning. *The Journal of Machine Learning Research*, 13(1): 3725–3768.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Loula, J.; Allen, K.; Silver, T.; and Tenenbaum, J. 2020. Learning constraint-based planning models from demonstrations. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2970–2977.

Marthi, B.; Russell, S. J.; and Wolfe, J. A. 2007. Angelic Semantics for High-Level Actions. In *ICAPS*, 232–239.

Moerland, T. M.; Broekens, J.; and Jonker, C. M. 2020. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*.

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *Proceedings of the 23rd international conference on Machine learning*, 665–672.

Parascandolo, G.; Buesing, L.; Merel, J.; Hasenclever, L.; Aslanides, J.; Hamrick, J. B.; Heess, N.; Neitz, A.; and Weber, T. 2020. Divide-and-conquer monte carlo tree search for goal-directed planning. *arXiv preprint arXiv:2004.11410*.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2): 115–135.

Sarathy, V.; Kasenberg, D.; Goel, S.; Sinapov, J.; and Scheutz, M. 2020. SPOTTER: Extending Symbolic Planning Operators through Targeted Reinforcement Learning. *arXiv preprint arXiv:2012.13037*.

Say, B. 2021. A Unified Framework for Planning with Learned Neural Network Transition Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 5016–5024.

Say, B.; Wu, G.; Zhou, Y. Q.; and Sanner, S. 2017. Nonlinear Hybrid Planning with Deep Net Learned Transition Models and Mixed-Integer Linear Programming. In *IJCAI*, 750–756.

Silver, T.; Chitnis, R.; Tenenbaum, J.; Kaelbling, L. P.; and Lozano-Perez, T. 2021. Learning Symbolic Operators for Task and Motion Planning. *arXiv preprint arXiv:2103.00589*.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 639–646. IEEE.

Tadepalli, P.; Givan, R.; and Driessens, K. 2004. Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 workshop on relational reinforcement learning*, 1–9.

Wang, Z.; Garrett, C. R.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7): 866–894.

Xia, V.; Wang, Z.; Allen, K.; Silver, T.; and Kaelbling, L. P. 2019. Learning sparse relational transition models. In *International Conference on Learning Representations*.

Yang, F.; Lyu, D.; Liu, B.; and Gustafson, S. 2018. PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making. In *International Joint Conference on Artificial Intelligence*.

Zhuo, H. H.; Hu, D. H.; Hogg, C.; Yang, Q.; and Munoz-Avila, H. 2009. Learning HTN method preconditions and action models from partial observations. In *Twenty-First International Joint Conference on Artificial Intelligence*.