Neurosymbolic World Models for Sequential Decision Making

Leonardo Hernández Cano [1] **Maxine Perroni-Scharf** [1] **Neil Dhir** [2] **Arun Ramamurthy** [2]
Armando Solar-Lezama [1]

Abstract

We present Structured World Modeling for Policy Optimization (SWMPO), a framework for unsupervised learning of neurosymbolic Finite State Machines (FSM) that capture environmental structure for policy optimization. SWMPO models the environment as a FSM, where each state corresponds to a specific region of the state space with distinct dynamics (e.g., *water* and *land*). This structured representation can be leveraged for tasks like policy optimization. Our proposed FSM synthesis algorithm operates in an unsupervised manner, leveraging low-level features from unprocessed, non-visual data to learn non-linear models, making it adaptable across various domains. The synthesized FSM models are expressive enough to be used in a model-based Reinforcement Learning scheme that leverages offline data to efficiently synthesize environment-specific world models. We demonstrate the advantages of SWMPO by benchmarking its environment modeling capabilities in a number of simulation tasks.

1. Introduction

This work focuses on learned approximations of environment dynamics —known as *world models* (Ha & Schmidhuber, 2018)— in the special case where these models must explicitly encode the high-level structure of the environment dynamics.

We are motivated by the observation that structured-world models may facilitate the reuse of components across different environments. For example, consider modeling a LiDAR-based amphibious robot that must navigate both water and land—which we call *modes*. Across different environments, these modes may appear in different configurations (e.g., one environment may be a lake surrounded by land, another may be mostly land separated by a small stream). Instead of modeling each environment from scratch, a better approach might be to first construct 'primitives' for water and land, and then, given a specific environment, assemble those two primitives into a complete model. This only requires the identification of the boundaries between the two modes relevant to a task, which in principle requires less effort than the construction of the model from scratch.

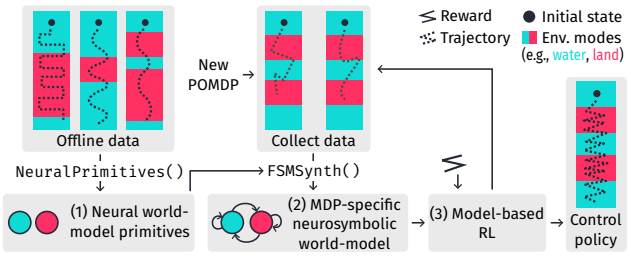


Figure 1. Proposed policy-optimization framework: (1) data is collected from the input POMDP by interacting with the environment in a standard RL loop; (2) a world-model specific to the POMDP is synthesized leveraging neural primitives; (3) the world model and the POMDP's reward function are used to perform model-based optimization. These steps are repeated until a termination condition is met. The neural primitives are learned from data from previous POMDPs. The output is a policy for the new POMPD.

Following this reasoning, our goal is to develop a method that (1) learns a set of world-model primitives in an unsupervised manner from offline data and (2) reuses those primitives in the synthesis of a model specific to a new environment. This model should be accurate enough to be used for model-based policy-optimization. We focus on environments with continuous, low-level and non-visual observations (e.g., LiDAR measurements or joint positions).

To this end, we propose Structured World Modeling for Policy Optimization (SWMPO), a neurosymbolic framework for model-based Reinforcement Learning (RL). SWMPO first uses offline data to learn a set of neural world-model primitives (e.g., *water* and *land*). SWMPO then assembles these primitives into a complete world model specific to a new environment. Finally, SWMPO uses this model for

[1]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute Of Technology, Massachusetts, United States of America [2]SIEMENS, New Jersey, United States of America. Correspondence to: Leonardo Hernández Cano <leohc@mit.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

model-based policy optimization (see Figure 1).

Internally, SWMPO models an environment with a neurosymbolic FSM. The synthesized FSM consists of world-model primitives and transitions between them. Each primitive is a neural network that approximates the environment dynamics within a subset of the state space. The transitions are logical rules that determine when to switch between primitives based on observations of the environment. The modular structure of an FSM allows our system to reuse a set of neural primitives to efficiently synthesize environment-specific FSMs, as then learning the boundaries between the modes is sufficient to assemble a complete world-model.

We evaluate components of SWMPO across various benchmarks and environments with continuous dynamics, including two and three –dimensional simulations.

Contributions Our contributions are as follows:

1. An unsupervised learning algorithm that trains a set of modular world-model primitives (e.g., *water* and *land* models) from a dataset of low-level continuous observations.
2. A state-machine synthesis algorithm that leverages a set of primitives learned from a collection of environments and new data from a specific environment to construct an environment-specific FSM world model.
3. An evaluation that demonstrates these primitives and the FSMs can be used to perform model-based RL in a meta-learning setup.

2. Background

We operate in the standard discrete-time RL framework, where an agent interacts with an environment according to some control policy (Sutton & Barto, 2018). A summary of the notation used in this manuscript can be found in Table 1 in the supplementary material.

Definition 2.1 (Partially Observable Markov Decision Process)**.** A discrete-time Partially Observable Markov Decision Process (POMDP) is a tuple $\mathcal{M} \triangleq \langle \mathcal{S}, \mathcal{A}, T, S_0, \mathcal{O}, O \rangle$, where $\mathcal{S} \subseteq$ is the set of states, $\mathcal{A}$ is a set of actions, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a transition function, S_0 is a distribution of initial states, $\mathcal{O}$ is the set of observations the agent can make, each observation $o \in \mathcal{O}$ made under some state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ with probability $O(o \mid s, a)$. We associate a POMDP with a reward function $R : \mathcal{O} \times \mathcal{A} \times \mathcal{O} \to \mathbb{R}$.

For a given policy $\pi_\theta : \mathcal{O} \to \mathcal{A}$, we use S_t, O_t and A_t to denote the random variables of the state and action at time t respectively. We use lowercase letters to denote values of a random variable.

Definition 2.2 (Trajectory)**.** A trajectory is a time-indexed sequence of transition tuples (o_t, a_t, o_{t+1}). We call o_t and o_{t+1} the source and next observations, respectively.

Definition 2.3 (Finite State Machine World Model)**.** We define a finite state machine world model (FSMWM) to be a tuple $\mathcal{F} \triangleq \langle F, \mathcal{O}, \mathcal{A}, \delta, f_0 \rangle$, where $\mathcal{O}$ and $\mathcal{A}$ are respectively the observation and action spaces of a POMDP; $F = \{f_i : \mathcal{O} \times \mathcal{A} \to \mathcal{O}\}_i$ is a set of models –"modes"– of the environment; and $\delta = \{\delta_{i,j} : \mathcal{O} \times \mathcal{A} \to \{0, 1\}\}_{i,j}$ is a set of mode-transition predicates. The initial mode is f_0.

For a given active mode indexed by i and an observation $o_t \in \mathcal{O}$, the predicted next observation is $o_{t+1} = f_i(o_t, a_t)$. The next active mode index is

$$\delta(o_t, u_t, i) = \begin{cases} \operatorname{argmax}_j \delta_{i,j}(o_t, u_t) & \text{if } \delta_{i,j}(o_t, u_t) > 0 \\ i & \text{otherwise.} \end{cases}$$

We define argmax to choose the first matching index in case of a tie. This definition mirrors previous use of FSMs as policies in the MDP setting (Inala et al., 2020), but here we are using them as world models.

Given an observation and a policy, the FSMWM can simulate a rollout by using the local models to predict the next observation and transitioning between the models accordingly. Unlike a POMDP, the FSMWM operates purely on observations. For conciseness, we write FSM as shorthand for FSMWM.

3. Problem statement

We want to synthesize a world model whose structure corresponds to the high-level structure of a POMDP, and then leverage this model within a RL training loop. With this in mind, we restate our goals as follows:

Goal 1: Use a dataset of trajectories collected by an agent, possibly across several 'offline' POMDPs, to learn a set of world model primitives.

Goal 2: Use a dataset of trajectories from a single 'active' POMDP and a set of world model primitives to synthesize an FSMWM of the active POMDP.

These tasks must be performed in an unsupervised manner (i.e., without mode labels in the input data). Our proposed method is described in Section 4.

Our core assumption is that each of the POMDPs of interest has an (unknown) FSMWM that models it. This implies the existence of a POMDP-specific latent categorical variable M_t, whose value corresponds to the index of the mode of the system at time t — it further implies that M_t can be characterized by a function $(o_{t-1}, a_{t-1}, o_t) \mapsto m_t$. We also assume that the number of modes is known in advance. Further assumptions are motivated and stated in Section 4.1 and Section 4.4. We may use examples of what modes may represent for clarity (e.g., $m_t =$ *water* instead of $m_t = 1$).

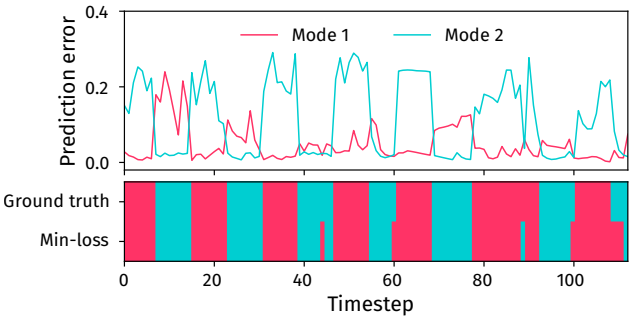


Figure 2. Prediction error of the learned modes (top) and corresponding mode classification (bottom) for a test trajectory from **Point Mass** — an idealized amphibious robot that travels on land (pink) and water (blue) (see Section 6). Each local model is specialized in a specific mode, resulting in low prediction error throughout the entire episode.

4. Structured world models

This section describes our proposed FSM synthesis algorithm. Our method receives a dataset of trajectories and synthesizes an FSM to model the structure of the environment (see Figure 3), where each state in the FSM represents a distinct mode of the data (e.g. *water* or *land*).

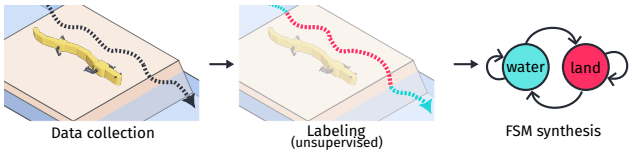


Figure 3. An illustration of the proposed neurosymbolic world-model synthesis method in an environment comprising land and water. First, a controller is used to gather data (left). Then, this data is labeled according to the modes of the system in an unsupervised fashion (middle). These labels are then used to supervise the synthesis of a neural state machine world-model (right), that can be leveraged for policy optimization.

A key challenge is discovering these modes in an unsupervised manner, where only the number of modes is known. It is important to achieve clean separation between modes at different stages of the algorithm to avoid cascading errors from misclassified data, which can progressively degrade the model's performance. To address these challenges, we synthesize our state machine as follows:

Neural world-model primitives: Divide the transitions in a given dataset into different mode subsets and train neural primitives (Section 4.1).

Pruning: Simplify the mode-transition dynamics of the partition by removing spurious transitions between modes (Section 4.2).

Transition predicate synthesis: Learn when to transition between modes (Section 4.3).

4.1. Neural world-model primitives

We first address the problem of decomposing environment dynamics by assigning each transition in a dataset D of trajectories to one of n disjoint subsets, with each subset corresponding to a mode of the POMDP. We assume that n is known in advance. This is a standard task in time-series analysis, although here we focus specifically on discovering the (unobserved) categorical *mode variable* M_t of a system.

We focus on the case where observations conditioned on a state are deterministic, so $o_t = O(s_t)$. Our method revolves around learning M_t as an intermediate computation of a learned first-order model of the form

$$f_{\theta_1}(m_{\theta_2}(o_{t-1}, a_{t-1}, o_t), o_t, a_t) \approx o_{t+1} - o_t,$$

where f_{θ_1} and m_{θ_2} are functions parametrized by $\theta_1, \theta_2 \in \mathbb{R}^k$. In principle, f_{θ_1} could be a standard forward model, but in preliminary experiments we observed that a first order model yields better performance. We hypothesize this happens because a first-order model weakens the dependence on o_t.

Ultimately, we characterize modes as a categorical variable (i.e., the robot is either on *water* or *land*), but here $m_{\theta_2}(o_{t-1}, a_{t-1}, o_t) \in \mathbb{R}^n$ is a continuous approximation of the true latent m_t.

We now describe a series of assumptions about the POMDP and the mode variable that allows us to design an algorithm to predict M_t.

Assumption 4.1 (Mode Identifiability)**.** We assume that M_t can be modeled as a function $m_t \approx m(o_{t-1}, a_{t-1}, o_t)$. Intuitively, this means that the current mode can be identified by observing how the world changes in response to the latest action. Thus, we interpret M_t as an abstraction of the observed system change given a particular action and state.

Assumption 4.2 (Predictability of Change Conditioned on Mode)**.** We assume that the POMDP reduces to a deterministic MDP when conditioned on the mode. More precisely, we assume the existence of a function $T' : M \times \mathcal{A} \times \mathcal{O} \to \mathcal{O}$ such that $T'(m_t, a_t, o_t) = O(T(s_t, a_t))$.

Thus far, our constraints allow the trivial solution $M_t = S_t$, which is not useful. There may be many other variables which satisfy our assumptions. Consequently, we add another assumption that allows us to uniquely identify M_t:

Assumption 4.3 (Minimality of Modes)**.** Let $\mathbf{M}$ be the set of random variables that satisfy the previous assumptions. Then, the mode variable M_t is the unique solution to

$$M_t^* = \arg\min_{M_t \in \mathbf{M}} I(M_t, O_{t+1}),$$

where $I(\cdot, \cdot)$ stands for the mutual information between two random variables.

Under the assumptions so far, we can conclude that if we find a variable M_t that allows us to predict the change in the environment given an action and has minimal mutual information with O_{t+1}, then M_t must be the mode variable. However, our approximation of M_t takes values in a vector space, but we ultimately want to model it as a categorical variable. We therefore add our last assumption:

Assumption 4.4 (Mode Vectors Form Clusters)**.** The mode variable M_t corresponds to a partition of the state space, where, in expectation, the within-subset sum of squares to the centroid of each subset is minimized. That is, we assume a *strict partitioning* and a *centroid model* clustering scheme. Consequently, if k-means is applied to vectors of M_t, the resulting clusters will, in expectation, correspond to the different modes.

Together, these assumptions imply that if a variable M_t is predictive of the change in observed state for any action and has minimal mutual information with O_{t+1}, then that variable corresponds to the mode variable. In other words, consider $f_{\theta_1} : M \times \mathcal{A} \times \mathcal{O} \to \mathcal{O}$ and $m_{\theta_2} : \mathcal{O} \times \mathcal{A} \times \mathcal{O} \to M$ under the joint optimization problem described in Equation (1) — therein $\| \cdot \|$ is the Euclidean norm, $o_{t-1} = O(s_{t-1})$ and $o_t = O(s_t)$. From the assumptions stated above, it follows that the solution to Equation (1) is such that that $m_{\theta_2}(\cdot)$ corresponds to the mode variable, which can be clustered with k-means to obtain mode labels for a set of transitions. In practice, we parametrize $m(\cdot)$ and $f(\cdot)$ with neural networks and approximate the solution of Equation (1) with gradient-based search. To compute the mutual-information $I(\cdot, \cdot)$, we assume independence of features and fit Gaussian distributions to compute a Monte-Carlo approximation.

Fitting local models to the data Once the approximation of the latent mode variable, m_{θ_2}, is trained, the system uses the k-means algorithm to cluster the induced set of latent vectors into disjoint subsets. Then, for each subset, the system fits a local forward model to model the corresponding transitions, as illustrated in Figure 2, where each local model has higher performance for a particular mode of the environment.

The entire process is shown in Algorithm 1. The algorithm takes a dataset of trajectories D, and partitions it by solving Equation (1) and clustering the resulting mode vectors. The resulting partition induces a set of neural primitives, each specialized in modeling the dynamics of a particular mode of the environment.

Algorithm 1 NeuralPrimitives

Require: List of trajectories D, latent space M, number of modes m, and, observation and action spaces $\mathcal{O}$ and $\mathcal{A}$ respectively.

1: Use gradient-descent to find neural networks $f_{\theta_1} : M \times \mathcal{A} \times \mathcal{O} \to \mathcal{O}$ and $m_{\theta_2} : \mathcal{O} \times \mathcal{A} \times \mathcal{O} \to M$ that approximately solve Equation (1).
2: Embed the transitions into mode vectors, $m(D)$.
3: Cluster the embeddings into m disjoint subsets using k-means.
4: Let $D_i \leftarrow \{\tau \in D \mid \text{cluster}(\tau) = i\}$.
5: Assemble partition $\overline{D} \leftarrow \{D_1, \ldots, D_m\}$.
6: Reorder D so that D_1 is the set with the most initial states.
7: Let $f_i \leftarrow \arg\min_f \mathbb{E}_{(o_t, a_t, o_{t+1}) \sim D_i}[\|o'_{t+1} - o_{t+1}\|]$ where $o'_{t+1} \leftarrow f(o_t, a_t)$.
8: Assemble the set of local models $F \leftarrow \{f_1, \ldots, f_m\}$.
9: **return** F

4.2. Pruning

The aforementioned partitioning process can create overly complex transitions. While the FSM globally approximates the environment dynamics, some state regions may have multiple models with similar accuracy, resulting in spurious transitions between states. In such cases, simplifying transition dynamics with minimal impact on forecasting accuracy might be possible.

To address this, we propose a pruning mechanism to eliminate these unwanted transitions. This helps balance the complexity-accuracy trade-off in the state machine search space: while more complex transition patterns can improve accuracy, they also increase the risk of overfitting.

Pruning Approach The system begins by labeling each transition in the dataset with the index of the neural network from the ensemble that best predicts the system's evolution in that state. A mode transition occurs when this label changes between consecutive states. For example, in the sequence `113322`, we transition from mode 1 to 3, then from 3 to 2. Pruning the transition to mode 3 yields two possible sequences: `111122` (forward-prune) or `112222` (backward-prune).

To prune a mode transition, the system relabels the transition so that a different neural network handles it (e.g., we relabel a transition originally marked as *land* to *water*). If the increase in prediction error in the transition is within the user-defined tolerance factor ϵ, the move is considered ϵ-valid relative to the original partition. A mode transition is ϵ-prunable if all the associated moves are ϵ-valid. Note that there may be multiple ϵ-prunable mode transitions for a given trajectory and partition. The system greedily prunes

$$\arg\min_{\theta_1,\theta_2} \mathbb{E}_{(s_{t-1},a_{t-1},s_t,a_t,s_{t+1})\sim\mathcal{M}_\pi} \left[\left\| f_{\theta_1}\left(m_{\theta_2}(o_{t-1},a_{t-1},o_t),o_t,a_t\right) - (o_{t+1}-o_t)\right\|\right] + I\left(m_{\theta_2}(O_{t-1},A_t,O_t),O_{t+1}\right) \tag{1}$$

the first prunable mode transitions with the strategy that results in the smallest prediction error increase, until there are no more ϵ-prunable transitions (this process is summarized in Algorithm 4 in the supplementary material).

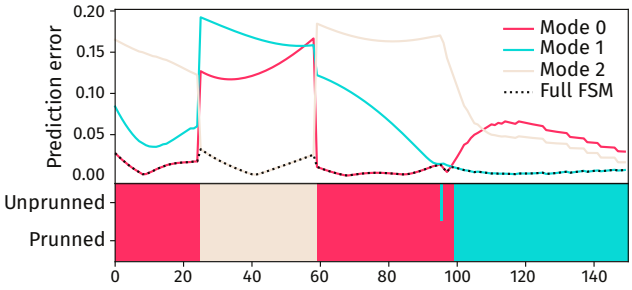


Figure 4. Illustrative comparison of a prunned and an unprunned partition from preliminary experiments. Pruning simplifies the transition dynamics by removing spurious transitions whose removal does not substantially degrade the performance of the resulting FSM. For instance, at timestep 95, model 0 exhibits a lower prediction error than model 2. In the unpruned FSM, there is a brief transition to mode 2 and back, which is removed through pruning.

4.3. Transition predicate synthesis

We now describe the mechanism by which the FSM learns when to transition from one mode to another given a partitioned dataset of trajectories and corresponding primitives.

To construct a POMDP-specific FSM, the system leverages the set of primitives and new POMDP-specific data to identify the subsets of the POMDP where each primitive should be used. Specifically, for each transition (s_t, a_t, s_{t+1}) observed in the new POMDP the FSM must use the primitive which results in the lowest prediction error. That is, given a set of primitives $F = \{f_1, \ldots, f_n\}$, the resulting state machine must use f_i for transition $\tau_t = (o_t, a_t, o_{t+1})$ where

$$f_i = \arg\min_{f\in F} \|o_{t+1} - f(o_t, a_t)\| \tag{2}$$

See Figure 2 for an example of how the primitives lead to accurate labeling.

Labeling induces a partitioning of a dataset of transitions. Each subset of a partition corresponds to a state of the FSM being synthesized. Each pair of FSM states (f_i, f_j) will be associated with a transition predicate $\delta_{i,j}$ that corresponds to the following question: *given that the state machine was in state f_i, and the agent observed s_t, took action u, and then observed s_{t+1}, should the FSM transition to state f_j?* To synthesize this predicate, the system identifies the subset of D_i containing transitions where the next state is a source state in D_j, referred to as the 'positive' set. The 'negative' set is the complement of the positive set with respect to D_i. The task is reduced to a standard classification problem, where the goal is to find a predicate that outputs True for the positive set and False for the negative set. We use $\text{synthesizePredicate}(\cdot,\cdot)$ to denote the routine that solves this supervised learning problem. In our experiments, we use an off-the-shelf tree-learning algorithm.

Once transition predicates are synthesized, the FSM has been defined. See Algorithm 2 for the entire FSM synthesis algorithm. The computational complexity of training a SWMPO model is linearly slower compared to a standard monolithic world model and depends on the number of modes; see Appendix D for further discussion.

Algorithm 2 FSMSynth

Require: Primitives F (Algorithm 1), dataset of trajectories D, observation and action spaces $\mathcal{O}$ and $\mathcal{A}$ respectively

for $f_i \in F$ **do**
  $D_i \leftarrow \{t \in D \mid f_i = \arg\min_{f\in F} \mathrm{C}(f,t)\}$
end for
for $i, j \in \{1,\ldots,m\} \times \{1,\ldots,m\}$ **do**
  positive $\leftarrow \{\, \tau_1 \in D_i \mid \exists \tau_2 \in D_j$ s.t. $\text{follows}(\tau_1,\tau_2)\,\}$
  negative $\leftarrow D_i \setminus \{\text{positive}\}$
  $\delta_{i,j} \leftarrow \text{synthesizePredicate}(\text{positive}, \text{negative})$
end for
Let $\delta \leftarrow \{\delta_{i,j} \mid 1 \le i,j \le m\}$
$\mathcal{F} \leftarrow \langle F, \mathcal{O}, \mathcal{A}, \delta, f_1\rangle$.
return $\mathcal{F}$

4.4. Neurosymbolic model-based policy optimization

We now describe our approach to integrating the synthesis of FSMWMs into a model-based RL loop. Given a POMDP $\langle \mathcal{S}, \mathcal{A}, T, S_0, \Omega, \mathcal{O}\rangle$, a reward function R, and a horizon length $n \in \mathbb{N}$, our goal is now to find a closed-loop policy $\pi^* : \mathcal{O} \to \mathcal{A}$ that achieves a maximal cumulative reward. That is, the goal is to solve

$$\pi^* = \arg\max_{\pi\in\Pi} \mathbb{E}_{s_0\sim S_0}[R(\pi)], \tag{3}$$

where

$$R(\pi) = \sum_{0\le t\le n} R(s_t, \pi(O(s_t))).$$

Our algorithm should reuse offline data from similar 'offline' POMDPs, while only using relatively little data specific of

'active' POMDP to optimize a policy. This approach aims to reduce the number of agent-environment interactions that must occur within the POMDP to find an optimal policy by alternating optimization in the model and in the POMDP. Specifically, the offline data is leveraged to synthesize a neural set of primitives which will be used in the POMDP-specific FSM.

At the core of our approach is the following assumption:

Assumption 4.5 (Global Modes)**.** The modes in the active POMDP must be a subset of the modes in the offline POMDPs.

For example, consider two offline POMDPs: one with modes 'A' and 'B' and one with modes 'C' and 'D'; the active POMDP could consist of modes 'A' and 'C' which we argue is not very restrictive. In the example of the amphibious robot, Assumption 4.5 is satisfied because each POMDP corresponds to a different terrain configuration. Thus, transitions between land and water dynamics change between POMDPs, but the dynamics of each mode are constant across all POMDPs.

Thus, given a new POMDP and offline data from similar POMDPs, our proposed algorithm proceeds as follows: first, a set of neural primitives is trained from the offline data in an unsupervised manner. Then, a short standard RL loop (observe, act, receive reward) is run over the POMDP to collect data. The new data and the set of primitives are used to synthesize a POMDP-specific FSM, which is used to perform model-based RL. These steps are iterated until a termination criterion is met. The algorithm is described in Algorithm 3.

Algorithm 3 SWMPO

Require: POMDP and reward function $(\mathcal{M}, R)$, trajectory datasets from previous POMDPs $D_1, \ldots, D_n$, observation, and action spaces $\mathcal{O}$ and $\mathcal{A}$ respectively, latent space M, number of modes m, T^1_{session}, T^2_{session} the length of real- and model-based RL phases, respectively, off-the-shelf RL algorithm RL, step budget T_{total}.

Initialize parametric policy π_θ.
$D = D_1 \cup \ldots \cup D_n$
$F =$ **NeuralPrimitives**$(D, M, m, \mathcal{O}, \mathcal{A})$
$D_{\mathcal{M}} \leftarrow \emptyset$; $t \leftarrow 0$
while $t \leq T_{\text{total}}$ **do**
  $\pi_\theta, D'_{\mathcal{M}} \leftarrow \text{RL}(\pi_\theta, \mathcal{M}, R, T^1_{\text{session}})$
  $t \leftarrow t + T^1_{\text{session}}$
  $D_{\mathcal{M}} \leftarrow D_{\mathcal{M}} \cup D'_{\mathcal{M}}$
  $\mathcal{F} \leftarrow$ **FSMSynth**$(F, D_{\mathcal{M}}, \mathcal{O}, \mathcal{A})$
  $\pi_\theta, \_ \leftarrow \text{RL}(\pi_\theta, \mathcal{F}, R, T^2_{\text{session}})$
end while
return π_θ

5. Related work

Graphical Models Hidden Markov models (HMMs) are a standard approach to capture temporal dependencies and mode-switching behavior in sequential data (Li & Biswas, 2002; Bouguila et al., 2022). In robotics, HMMs have been leveraged to segment trajectories into discrete modes (Goh et al., 2012) and used during policy learning for multi-modal or hierarchical tasks (Marturi et al., 2019). Recent advances have extended HMMs using deep neural network architectures (neural HMMs) to handle continuous and high-dimensional observation spaces and to learn more complex transition dynamics (Tran et al., 2016). For example, neural HMMs have been used in unsupervised settings to model complex sensory streams for trajectory clustering (Vakanski et al., 2012) and to predict latent modes during task execution (Wu et al., 2019). However, HMMs suffer from the fundamental limitation that the transition between modes is determined by a probability distribution that is only conditioned on the latent state, this means that the observed evolution of the system itself is only indirectly used to update the active mode. State of the art graphical models include Hidden Parameter recurrent State Space Models (Shaj et al., 2023), and recurrent Switching Dynamical Systems Models (Glaser et al., 2020). Generalizing these classes of models to include non-linear dynamics makes inference computationally expensive (Frigola et al., 2013).

Leveraging structure in RL Many prior works focus on leveraging structure to solve control problems with RL (Mohan et al., 2024). Hierarchical RL (Xu & Fekri, 2021; Botvinick, 2012; Li et al., 2006) and modular RL (Simpkins & Isbell, 2019; Andreas et al., 2017; Devin et al., 2017) encode structure directly into the policy architecture, here we instead consider the synthesis of a structured model. Model-based RL approaches leverage neural world models to optimize policies more efficiently (Moerland et al., 2023; Ha & Schmidhuber, 2018). However, traditional neural models lack distinct boundaries between the representation of different modes in the environment; this lack of modularity makes reuse challenging. Reward machines (Toro Icarte et al., 2019; Icarte et al., 2018) leverage structured models of the reward function to guide the policy optimization process. Large Language Models have been used to synthesize programmatic world models (Tang et al., 2024), but not in continuous systems with low-level observations. Hasanbeig et al. (2021) demonstrate that FSMs can be synthesized to model discrete environments, improving performance in RL tasks. However, this method is reliant on fully symbolic representations obtained from pre-trained vision models in grid-world settings. In contrast, our approach extracts structure from continuous, low-level, non-visual observations.

Structure induction and hybrid systems In the broader field of hybrid systems, modeling environments as a collection of modes with distinct dynamics is standard practice (Alur et al., 1995; Paoletti et al., 2007; Ferrari-Trecate et al., 2003; Devin et al., 2017; Camacho et al., 2010). Soto et al. (2021) have shown that automata with affine dynamics can be synthesized from time-series data. Our approach instead is more general, using non-linear neural models to both extract structure and represent dynamics within the automata.

Other approaches to discovering structure which are not directly applicable to time-series data include the use of graph neural networks (Cranmer et al., 2020) and sparse networks (Gupta et al., 2024). Similarly, methods that leverage recurrent neural networks (RNNs) and FSMs to model linguistic structures (Kolen, 1993; Koul et al., 2018; Jacobsson, 2005) share a conceptual foundation with our work, but focus on formal languages and are not directly applicable to the class of problems we study in this work.

6. Experiments

Our experiments aim to answer two research questions: (1) *how effectively does SWMPO leverage offline data in the synthesis of an environment-specific FSM?*; (2) *is the resulting FSM accurate enough for model-based RL?*

The algorithm[1] to approximate the solution to Equation (1) is written with Pytorch (Ansel et al., 2024) using the Adam optimizer (Kingma & Ba, 2017). We use multi-layer perceptrons with ReLU activations for all the neural networks involved in the algorithm. Predicate synthesis is performed with the tree-learning algorithm implemented in Scikit-learn (Pedregosa et al., 2011). The underlying RL algorithm is Soft-Actor Critic, as implemented in StableBaselines3 (Raffin et al., 2021). We manually tune the hyperparameters of all algorithms, including baselines. We test SWMPO on four simulated environments of varying complexity. Hyperparameters for all experiments can be found in Appendix C.

6.1. Simulation environments

All of our experiments are performed in simulation on the following environments:

Point Mass These tasks are an idealized model of the amphibious robot running example, and consist of applying a sequence of thrusts to a two-dimensional point mass to take it to a target position from an initial position (see Figure 6). Crucially, the environments consist of randomly generated terrains with different characteristics: land and water, which correspond to the modes. To simulate the different dynamics in different terrains, actions in the land terrain are inverted. We use an MPC controller to gather offline data.

LiDAR Racing Adapted from the work by Ivanov et al. (2021). Tasks in this environment consist of driving a two-dimensional vehicle with bicycle dynamics and LiDAR sensors through a track randomly assembled from pieces of five different types, which correspond to the modes. We use a pre-trained controller provided by the authors to gather offline data.

Salamander A locomotion task in which an amphibious salamander must navigate through water and land, which correspond to the modes. This environment is implemented in the Webots 3D simulator (Michel, 2004), in which the *Salamandra Robotica II* (Crespi et al., 2013) robot is available. This environment is a scaled-up version of **Point Mass**. For observations, we use the motor positions, the LiDAR readings and the GPS position. We use the controller provided in Webots to generate offline data.

Bipedal Walker (**BipedalWH**) A locomotion task in which a bipedal robot has to loco mote over uneven terrain with four different types of obstacles, which correspond to the modes (see Figure 6). This is a standard benchmark in the Gymnasium library (Towers et al., 2024) and employs the Box2D rigid body simulator (Catto, 2024). We use a pre-trained controller from the RL Baselines3 Zoo library (Raffin et al., 2021) to generate offline data.

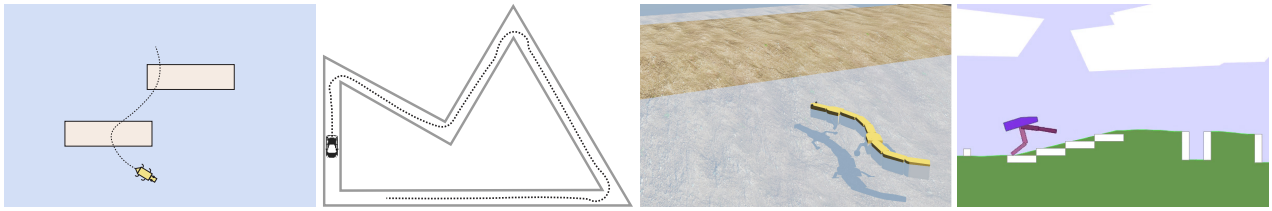
See Figure 5 for an illustration of each environment.

6.2. FSM synthesis

We evaluate the performance of our FSM synthesis algorithm across all four environments. As described before, our goal is to train a set of neural primitives from a dataset of 'offline' POMDPs, and finally assemble the primitives into an FSM specific to an 'active' POMDP. Crucially, we want the resulting FSMWM to accurately track the mode variable (e.g., the FSMWM should have modes that correspond to *water* and *land* and accurately switch between them as the robot locomotes through the 'active' POMDP).

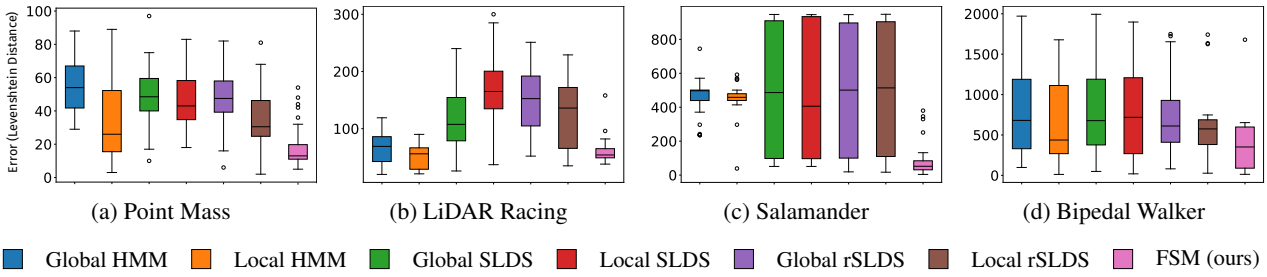
To test the accuracy of the structure of the FSMWM, we compare the states visited by the synthesized FSMWM in unseen data from the 'active' POMDP against ground truth mode labels, as well as the visited states predicted by baseline models. The baseline models are (1) a Hidden Markov model (HMM) with Gaussian emissions (Baum & Petrie, 1966) implemented in the HMM Learn library (Learn, 2024), (2) a Switching Linear Dynamical System (SLDS) (Ackerson & Fu, 1970) and (3) a recurrent SLDS (rSLDS) (Glaser

[1] Implementation details can be found at: https://gitlab.com/da_doomer/swmpo



(a) Point Mass (b) LiDAR Racing (c) Salamander (d) Bipedal Walker

Figure 5. The four simulated evaluation environments.



(a) Point Mass (b) LiDAR Racing (c) Salamander (d) Bipedal Walker

Figure 6. Bottom row: Box plots comparing the Levenshtein distance between predicted and ground truth labels for unseen trajectories (lower is better) for each evaluation environment. Systems that accurately track the latent mode variable of a dynamical system more accurately match the ground-truth labels. The box-plots show that SWMPO (right) outperforms baselines (left) in all environments in tracking the latent mode variable.

et al., 2020), the latter two implemented in the SSM library (Linderman, 2020).

Since the FSM undergoes two training stages—one using data from the 'offline' POMDPs multiple terrains to synthesize primitives, and another on the 'active' POMDP to synthesize transitions—we train two versions of each baseline: a 'local' version, trained exclusively on trajectories from the 'active' POMDP (excluding the test trajectories), and a 'global' version, trained data from both the 'offline' and 'active' POMDPs.

To evaluate each model, we compute the Levenshtein distance between the predicted labels and the ground-truth labels of unseen test trajectories. The final errors are aggregated across four test trajectories for each of eight different terrains (see Figure 6). To synthesize the FSMWM, we use Algorithm 1 to train the neural primitives from the 'offline' data. Together with this set of neural primitives, data from the 'active' POMDP is used to synthesize the POMDP-specific FSMWM with Algorithm 2. We use an expert policy to generate all data.

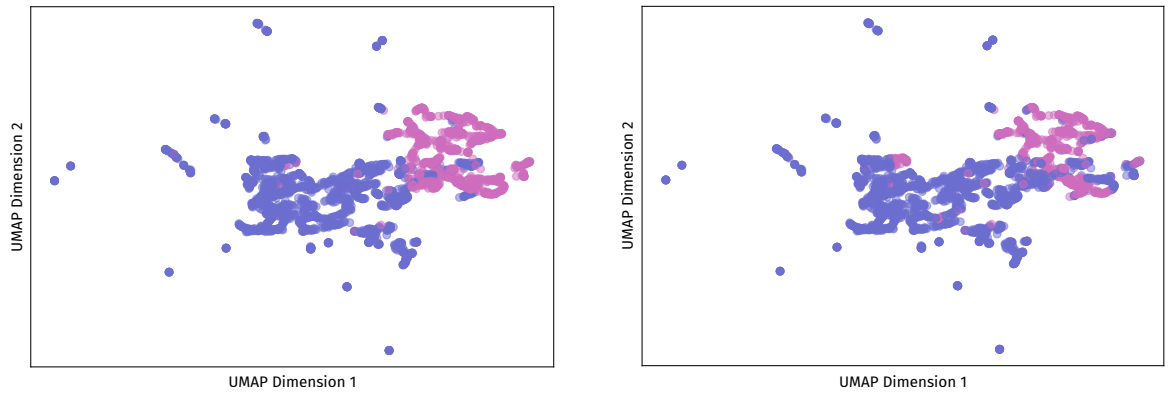
In **Point Mass** and **Salamander**, SWMPO outperforms all baselines. In **LiDAR Racing**, SWMPO significantly outperforms SLDS and rSLDS and only marginally outperforms HMM. In **BipedalWH**, SWMPO marginally beats the baselines, but all models struggle to capture the structure and dynamics of the environment.

We plot the latent mode variable and qualitatively compare the induced partition with the ground-truth partition – see Figure 7.

6.3. Reinforcement Learning

We evaluate the performance of the FSM models within the model-based RL loop described in Algorithm 3, shown in Figure 8. We use the data from the 'offline' POMDPs (described in Section 6.2) as the input. We use Soft-Actor Critic (Haarnoja et al., 2018) as the RL routine. We compare with the following baselines: (2) standard model-free RL, and (2) mode-based RL using a neural forward-model of the environment synthesized from the 'active' POMDP data as the environment is explored. The model-based RL baseline is the same as Algorithm 3 except that it learns online a monolithic neural network instead of refining the FSMWM transitions. Note that, for the model-based RL baseline, we do not pre-train the neural network on the 'offline' data because the data comes from other POMDPs with different dynamics. We augment the observations with a time variable and normalized them with a running mean wrapper. Due to resource constraints, we benchmark only on **Point Mass**.

The model-based rollout logic for both SWMPO and the model-based baseline consist of sampling a random observation from an environment trajectory and performing a shorter (30 steps) model-based rollout. This is instead of relying on the models for long trajectories (150 steps), which can lead to compounding errors. This reduces reliance on



(a) Ground Truth (b) Predicted

Figure 7. UMAP-processed plot of the ground truth (top) and approximation (bottom) of the latent mode variable $m_{\theta_2}(o_t, a_t, o_{t+1})$ across a collection of trajectories in the **Point Mass** environment.

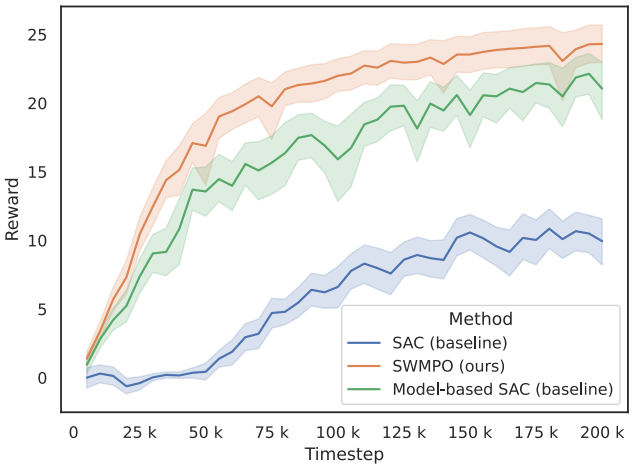


Figure 8. Reward curves in the **Point Mass** environment for SWMPO (ours), baseline model-free RL, and baseline model-based RL with a neural feed-forward model learned online. Shaded area are 95% confidence intervals computed over 64 different random seeds.

the model for long-horizon prediction, improving sample efficiency and stability for both methods. Note that the performance of the low-level neural models for long horizons is orthogonal to our claims.

7. Conclusion

We presented a novel framework for synthesizing Finite State Machine World Models (FSMWMs) in an unsupervised manner using low-level, non-visual continuous observations. We outlined the key assumptions underpinning our approach and demonstrated its applicability. The main limitations of the framework are stated explicitly as assumptions throughout the text.

Our experiments indicate that the synthesized FSMWMs effectively capture the underlying structure of the simulated evaluation environments, as they match or surpass the performance of the baselines on modeling the latent variable of the tested dynamical systems.

In the RL experiment, our model-based RL approach outperformed both the model-free and the model-based monolithic neural baselines, the latter of which learned the environment model online. We highlight that SWMPO did not train any neural model online, instead using the online data to synthesize transitions between the neural primitives trained from unlabeled offline data.

Avenues for future work broadly include (1) weakening the limitations of the framework, (2) leveraging neural primitives in a different class of models (i.e., instead of FSMs), (3) explicitly encoding priors (e.g., through Physics Informed Neural Networks), and (4) leveraging the structure of the FSM explicitly during policy optimization (e.g., by using the 'product MDP' as done by Hasanbeig et al. (2021)).

Impact Statement

Given the relatively narrow scope of our experiments, we do not believe there are any direct societal consequences of this work that must be specifically highlighted.

References

Ackerson, G. and Fu, K. On state estimation in switching environments. *IEEE Transactions on Automatic Control*, 15(1), 1970.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P. H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1), 1995.

Andreas, J., Klein, D., and Levine, S. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2017.

Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., Hirsh, B., Huang, S., Kalambarkar, K., Kirsch, L., Lazos, M., Lezcano, M., Liang, Y., Liang, J., Lu, Y., Luk, C., Maher, B., Pan, Y., Puhrsch, C., Reso, M., Saroufim, M., Siraichi, M. Y., Suk, H., Suo, M., Tillet, P., Wang, E., Wang, X., Wen, W., Zhang, S., Zhao, X., Zhou, K., Zou, R., Mathews, A., Chanan, G., Wu, P., and Chintala, S. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, 2024.

Baum, L. E. and Petrie, T. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6), 1966.

Botvinick, M. M. Hierarchical reinforcement learning and decision making. *Current Opinion in Neurobiology*, 22 (6), 2012.

Bouguila, N., Fan, W., and Amayri, M. *Hidden Markov models and applications*. Springer, 2022.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

Camacho, E. F., Ramirez, D. R., Limon, D., Muñoz de la Peña, D., and Alamo, T. Model predictive control techniques for hybrid systems. *Annual Reviews in Control*, 34(1), 2010.

Catto, E. erincatto/box2d: a 2D physics engine for games, 2024. URL https://github.com/erincatto/box2d. Accessed: 2025-01-31.

Cranmer, M., Sanchez Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. Discovering Symbolic Models from Deep Learning with Inductive Biases. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

Crespi, A., Karakasiliotis, K., Guignard, A., and Ijspeert, A. J. Salamandra Robotica II: An Amphibious Robot to Study Salamander-Like Swimming and Walking Gaits. *IEEE Transactions on Robotics*, 29(2), 2013.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

Ferrari-Trecate, G., Muselli, M., Liberati, D., and Morari, M. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39(2), 2003.

Frigola, R., Lindsten, F., Schön, T. B., and Rasmussen, C. E. Bayesian inference and learning in gaussian process state-space models with particle mcmc. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

Glaser, J., Whiteway, M., Cunningham, J. P., Paninski, L., and Linderman, S. Recurrent Switching Dynamical Systems Models for Multiple Interacting Neural Populations. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc., 2020.

Goh, C. Y., Dauwels, J., Mitrovic, N., Asif, M. T., Oran, A., and Jaillet, P. Online map-matching based on hidden markov model for real-time traffic sensing applications. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2012.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT Press, 2016.

Gupta, K., Yang, C., McCue, K., Bastani, O., Sharp, P. A., Burge, C. B., and Solar-Lezama, A. Improved modeling of RNA-binding protein motifs in an interpretable neural model of RNA splicing. *Genome Biology*, 25(1), 2024.

Ha, D. and Schmidhuber, J. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 2018.

Hasanbeig, M., Jeppu, N. Y., Abate, A., Melham, T., and Kroening, D. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9), 2021.

Icarte, R. T., Klassen, T., Valenzano, R., and McIlraith, S. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018.

Inala, J. P., Bastani, O., Tavares, Z., and Solar-Lezama, A. Synthesizing Programmatic Policies that Inductively Generalize. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Ivanov, R., Jothimurugan, K., Hsu, S., Vaidya, S., Alur, R., and Bastani, O. Compositional Learning and Verification of Neural Network Controllers. *ACM Transactions on Embedded Computing Systems*, 20(5s), 2021.

Jacobsson, H. Rule Extraction from Recurrent Neural Networks: ATaxonomy and Review. *Neural Computation*, 17 (6), 2005.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization, 2017.

Klusowski, J. M., , and Tian, P. M. Large Scale Prediction with Decision Trees. *Journal of the American Statistical Association*, 119(545), 2024. Publisher: ASA Website.

Kolen, J. Fool' s Gold: Extracting Finite State Machines from Recurrent Network Dynamics. In *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993.

Koul, A., Greydanus, S., and Fern, A. Learning Finite State Representations of Recurrent Policy Networks, 2018.

Learn, H. hmmlearn/hmmlearn: Unsupervised learning and inference of Hidden Markov Models, 2024. URL https://github.com/hmmlearn/hmmlearn. Accessed: 2025-01-31.

Li, C. and Biswas, G. Applying the hidden Markov model methodology for unsupervised learning of temporal data. *International Journal of Knowledge Based Intelligent Engineering Systems*, 6(3), 2002.

Li, L., Walsh, T. J., and Littman, M. L. Towards a Unified Theory of State Abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*, 2006.

Linderman, L. lindermanlab/ssm: Bayesian learning and inference for state space models, 2020. URL https://github.com/lindermanlab/ssm. Accessed: 2025-01-31.

Marturi, N., Kopicki, M., Rastegarpanah, A., Rajasekaran, V., Adjigble, M., Stolkin, R., Leonardis, A., and Bekiroglu, Y. Dynamic grasp and trajectory planning for moving objects. *Autonomous Robots*, 43, 2019.

Michel, O. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems*, 1(1), 2004.

Moerland, T. M., Broekens, J., Plaat, A., and Jonker, C. M. Model-based Reinforcement Learning: A Survey. *Foundations and Trends® in Machine Learning*, 16(1), 2023.

Mohan, A., Zhang, A., and Lindauer, M. Structure in Deep Reinforcement Learning: A Survey and Open Problems. *J. Artif. Int. Res.*, 79, 2024.

Paoletti, S., Juloski, A. L., Ferrari-Trecate, G., and Vidal, R. Identification of Hybrid Systems A Tutorial. *European Journal of Control*, 13(2), 2007.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2011.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268), 2021.

Shaj, V., Buchler, D., Sonker, R., Becker, P., and Neumann, G. Hidden Parameter Recurrent State Space Models For Changing Dynamics Scenarios, 2023.

Simpkins, C. and Isbell, C. Composable Modular Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01), 2019.

Soto, M. G., Henzinger, T. A., and Schilling, C. Synthesis of hybrid automata with affine dynamics from time-series data. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, HSCC '21. Association for Computing Machinery, 2021.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction, 2nd ed*. Reinforcement learning: An introduction, 2nd ed. The MIT Press, 2018.

Tang, H., Key, D., and Ellis, K. WorldCoder, a Model-Based LLM Agent: Building World Models by Writing Code and Interacting with the Environment. *CoRR*, abs/2402.12275, 2024.

Toro Icarte, R., Waldie, E., Klassen, T., Valenzano, R., Castro, M., and McIlraith, S. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, A. J. S., and Younis, O. G. Gymnasium, 2024. URL https://zenodo.org/records/11232524.

Tran, K., Bisk, Y., Vaswani, A., Marcu, D., and Knight, K. Unsupervised neural hidden Markov models. *arXiv preprint arXiv:1609.09007*, 2016.

Vakanski, A., Mantegh, I., Irish, A., and Janabi-Sharifi, F. Trajectory learning for robot programming by demonstration using hidden Markov model and dynamic time warping. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(4), 2012.

Wu, H., Guan, Y., and Rojas, J. A latent state-based multimodal execution monitor with anomaly detection and classification for robot introspection. *Applied Sciences*, 9 (6), 2019.

Xu, D. and Fekri, F. Interpretable Model-based Hierarchical Reinforcement Learning using Inductive Logic Programming, 2021.

A. Notation

Table 1. Notation Table

Symbol	Description
$\mathcal{M}$	Markov Decision Process (MDP)
$S \in \mathbb{R}^k$	Set of observations
A	Set of actions
$T : S \times A \to S$	Transition function
S_0	Distribution of initial states
$R : S \times A \times S \to \mathbb{R}$	Reward function
$\pi_\theta : S \to A$	Policy parametrized by θ
$V_{\pi_\theta}(s)$	Value function for state s under policy π_θ
s_t	Source state at time t
a_t	Action at time t
s_{t+1}	Next state after taking action
$\mathcal{F}$	Finite State Machine World Model (FSMWM)
f	Set of environment models
δ	Set of mode-transition predicates
$s_{t+1} = f_i(s_t, u_t)$	Predicted next observation
$\delta(s_t, u_t, i)$	Mode transition function

B. Partition prunning

Algorithm 4 greedyPrune

Require: Partition $\overline{D} = \{D_1, \ldots, D_m\}$ of trajectory dataset D, error tolerance factor ϵ,
1: $\overline{D}_0 = \text{copy}(\overline{D})$.
2: **for** $t \in D$ **do**
3: **while exists** ϵ-prunnable (relative to $\overline{D}_0$) mode transition in t **do**
4: Prune the first ϵ-prunable (relative to $\overline{D}_0$) mode transition in $\overline{D}$, updating $\overline{D}$.
5: **end while**
6: **end for**
7: **return** $\overline{D}$

C. Hyperparameters

Table 2. Parameters for Point Mass.

Parameter	Value
hidden_sizes	32 32 32
learning_rate	0.0001
partition_size	2
batch_size	1000
min_island_size	4
autoencoder_latent_size	8
prunning_error_tolerance	0.04
model_mode_iter_n	1024
cluster_dimensionality_reduce	0
information_content_regularization_scale	1e-05
mutual_information_regularization_scale	0.00001
mutual_information_mini_batch_size	32
predicate_hyperparameters.max_depth	8
local_model_hyperparameters.hidden_layer_sizes	[32, 32]
local_model_hyperparameters.max_iter	500

Table 3. Parameters for Autonomous Driving.

Parameter	Value
hidden_sizes	32 32 32
learning_rate	0.01
partition_size	2
batch_size	1024
min_island_size	4
autoencoder_latent_size	8
prunning_error_tolerance	0.04
model_mode_iter_n	1000
cluster_dimensionality_reduce	0
information_content_regularization_scale	1e-05
mutual_information_regularization_scale	0.00001
mutual_information_mini_batch_size	32
predicate_hyperparameters.max_depth	8
local_model_hyperparameters.hidden_layer_sizes	[32, 32]
local_model_hyperparameters.max_iter	500

Table 4. Parameters for Salamander.

Parameter	Value
hidden_sizes	32 32 32
learning_rate	0.001
partition_size	2
batch_size	1024
min_island_size	4
autoencoder_latent_size	8
prunning_error_tolerance	0.04
model_mode_iter_n	1000
cluster_dimensionality_reduce	0
information_content_regularization_scale	1e-05
mutual_information_regularization_scale	0.00001
mutual_information_mini_batch_size	32
predicate_hyperparameters.max_depth	8
local_model_hyperparameters.hidden_layer_sizes	[32, 32]
local_model_hyperparameters.max_iter	500

Table 5. Parameters for Bipedal Walker.

Parameter	Value
machine_hidden_sizes	32 32 32
machine_learning_rate	0.001
machine_partition_size	2
machine_batch_size	1024
machine_min_island_size	4
machine_autoencoder_latent_size	8
machine_prunning_error_tolerance	0.04
machine_model_mode_iter_n	1000
machine_cluster_dimensionality_reduce	0
machine_information_content_regularization_scale	1e-05
machine_mutual_information_regularization_scale	0.00001
machine_mutual_information_mini_batch_size	32
machine_predicate_hyperparameters.max_depth	8
machine_local_model_hyperparameters.hidden_layer_sizes	[32, 32]
machine_local_model_hyperparameters.max_iter	500

D. Complexity

Let n be the number of experiences (o_t, a_t, o_{t+1}), each of dimension d, and let m the number of modes.

The computational complexity of training is driven by these steps:

- **Solving Eq. 1**: To solve Eq. 1 we use SGD, which is $\mathcal{O}(G_1 n d K_1)$, where G is the number of SGD steps, and K_1 accounts for the architecture of the model (see work by Goodfellow et al. (2016) for further discussion). The variables corresponding to the amount of data, the number of iterations and the model complexity needed for training, are not independent. E.g., a system with a high number of modes might need more data. However, a monolithic model would also require more data because it too must implicitly learn the same complex dynamics. Thus, we do not expect this step to present significant more overhead than a monolithic approach.
- Training the neural primitives: We assume datasets will have in expectation $\mathcal{O}(\frac{n}{m})$ elements. If models are trained sequentially, this is $\mathcal{O}\left(m G_2 \frac{n}{m} d K_2\right)$, where G_2 and K_2 account for the number of SGD steps and the architecture of the primitives. The caveats in 1. regarding the non-independence of features apply, but each step is at most as expensive as training a monolithic model. Primitives can be trained in parallel if wall-clock performance is critical.

- Transition Predicate Synthesis: This step trains $2m^2$ decision trees with CART (Breiman et al., 1984; Klusowski et al., 2024). For the average case, we assume each dataset will be of size $\mathcal{O}(\frac{n}{m})$. Therefore, the overall cost in expectation is $\mathcal{O}\left(m^3 d \frac{n}{m} \log^2\left(\frac{n}{m}\right)\right)$.

Applying dimensionality reduction techniques may help with high-dimensional spaces. In some high-dimensional domains (e.g., visual domains, which are not the focus of our work), we would expect a different class of models (e.g., CNNs) to be more effective.

Simplifying, the average complexity of training a SWMPO model is

$$\mathcal{O}\left(GnmdK + nmd + nmd \log^2\left(\frac{n}{m}\right)\right)$$

where $G = \max(G_1, G_2)$ and $K = \max(K_1, K_2)$. Thus, the expected overhead of training in SWMPO compared to a monolithic model is only a linear factor on m.

Then, during each model-based RL step, SWMPO evaluates the predicates of the current mode, which are small functions that could be evaluated in parallel, and the active mode's neural network. Thus, the expected run time is only an added small constant factor slower than a traditional monolithic world model.