STRUCTURED WORLD MODELS FROM LOW-LEVEL OBSERVATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present Structured World Modeling From Low-Level Observations ("SWMPO"), a framework for the unsupervised learning of neural Finite State Machines (FSM) that capture environment structure. Traditional unsupervised world modeling methods for policy optimization rely on unstructured representations, such as neural networks, which do not explicitly represent high-level patterns within the system (e.g., *walking* vs *swimming*). In contrast, SWMPO explicitly models the environment as an FSM, where each state represents a region of the environment's state space with distinct dynamics, exposing the structure of the environment to downstream tasks such as policy optimization. Prior works that synthesize FSMs for this purpose have been limited to discrete spaces, not continuous, high-dimensional spaces. Our FSM synthesis algorithm operates in an unsupervised manner, leveraging low-level features from unprocessed, non-visual data, making it adaptable across various domains. We demonstrate the advantages of SWMPO by benchmarking its environment modeling capabilities in different simulated environments.

1 INTRODUCTION

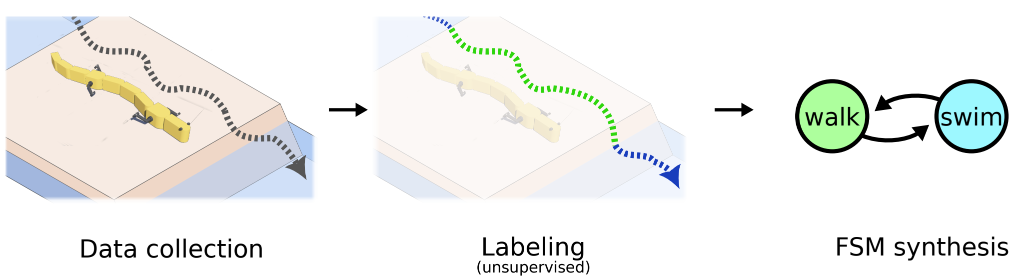


Figure 1: Overview of the proposed method. First, an existing (possibly expensive to run) controller (e.g., a planner) is used to gather data. Then, data is labeled according to the modes of the system in an unsupervised fashion. With this, a model of the environment could be used the form of a state machine is synthesized. In this illustration, the walking mode is green and the swimming mode is blue.

This paper examines 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 the high-level structure of a dynamical system can be used to efficiently solve control problems. Consider, for example, an amphibious robot that must navigate both water and land (see fig. 1). An expert roboticist might approach this problem by breaking down the task into three sub-problems: (1) controlling the robot on water; (2) controlling it on land and; (3) managing the transition between these two modes. With this division, the expert can exploit the fact that the robot moves faster on land than in water, using this knowledge to optimize route planning. Inspired by this strategy, our goal is to develop a method that automatically constructs a representation of the environment and a corresponding fine state machine.

We are interested in extracting structure directly from continuous, low-level, non-visual observations (e.g., LiDAR measurements or joint positions). To this end, we propose Structured World Modeling From Low-Level Observations –SWMPO– a framework where an environment's high-level structure is inferred directly from low-level continuous observations in a fully unsupervised manner (i.e., with no training labels), resulting in an FSM which can then be utilized in downstream tasks such as policy optimization.

The synthesized FSM consists of modes and transitions. Each mode is a neural network that approximates the environment dynamics within a subset of the state space (e.g., the *walking* mode, see fig. 1). Predicates determine when to switch between modes based on observations of the environment. We evaluate SWMPO across a variety of benchmarks and environments with continuous dynamics, including two-and three-dimensional simulations.

Contributions Our contributions are as follows:

1. A novel unsupervised learning algorithm that segments time series data into a discrete set of modes.
2. A state-machine synthesis algorithm that constructs a Finite State Machine (FSM) model directly from continuous low-level observations, enabling interpretable representations of latent dynamics.
3. Empirical testing demonstrating the performance of the state machine across fours test environments.

2 Related work

Automata Synthesis and Symbolic Structure Extraction Hasanbeig et al. (2021) demonstrated that FSMs could be synthesized to model environments, improving performance in RL tasks. However, this method is limited by its reliance 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. To the best of our knowledge, our work is the first to leverage neural world models in the synthesis of FSMs for continuous low-level high-dimensional non-visual observation spaces.

Hidden Markov Models Hidden Markov Models (HMMs) are a standard approach to capturing 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 multimodal or hierarchical tasks (Marturi et al., 2019). Recent advances have extended HMMs using deep neural network architectures (neural HMMs) to handle high-dimensional, continuous observation spaces and to learn more complex transition dynamics (Tran et al., 2016). For instance, 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.

Leveraging Structure in Reinforcement Learning A body of research focuses 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, neural models lack distinct boundaries between the representation of different modes in the environment. Reward machines (Toro Icarte et al., 2019; Icarte et al., 2018) leverage structured models of the reward function to guide the policy optimization process.

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). Recently, Soto et al. (2021) show that automata with affine dynamics can be synthesized from time-series data. In this work, we leverage non-linear neural models to both extract structure and represent dynamics within the automata, making our approach more general.

Finally, 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 these methods focus on formal languages and are not directly applicable to the class of problems we study in this work.

3 Background

We operate in the standard discrete-time RL framework, where an agent interacts with an environment (Sutton & Barto, 2018). A summary of the notation used in this paper can be found in table 1.

3.1 Reinforcement learning

Definition 1 (Partially Observable Markov Decision Process)**.** A discrete-time Partially Observable Markov Decision Process (POMDP) is a tuple $\mathcal{M} \triangleq \langle S, A, T, S_0, \Omega, O \rangle$, where $S \in \mathbb{R}^k$ is the set of states, Ω is the set of observations that the agent can make, A is a set of actions, $T : S \times A \to S$ is a transition function, S_0 is a distribution of initial states, Ω is a set of observations, each observation $o \in \Omega$ made under some state $s \in S$ and action $a \in A$ with probability $O(o \mid s, a)$ given by the set of conditional observation probabilities. We associate a POMDP with a reward function $R : S \times A \times S \to \mathbb{R}$.

Definition 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.

3.2 Finite State Machines

Definition 3 (Finite State Machine World Model)**.** We define a finite state machine world model (FSMWM) to be a tuple $\mathcal{F} \triangleq \langle f, O, A, \delta, f_0 \rangle$, where O and A are respectively the observation and action spaces of a POMDP; $f = \{f_i : O \times A \to O\}_i$ is a set of models of the environment; and $\delta = \{\delta_{i,j} : O \times A \to \{0, 1\}\}_{i,j}$ is a set of mode-transition predicates.

For a given active mode indexed by i and an observation $o_t \in S$, the predicted next observation is $o_{t+1} = f_i(o_t, a_t)$ is the predicted next observation. 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 a FSM as policies in the MDP setting (Inala et al., 2020), but here we are using them as world models.

4 Problem statement

Consider the example of an amphibious robot that must navigate both water and land (see fig. 1), corresponding to the two modes of the system. Our goal is two-fold: (1) to synthesize an FSMWM from low-level observations in an unsupervised manner (i.e., without mode labels) that captures the high-level structure of the environment by moving between modes that correspond to the (unobserved) mode of the system, and (2) to leverage the FSMWM in the RL training loop.

Our fundamental assumption is that the latent categorical variable M_t, which corresponds to the modes, can be characterized by a function $m : (s_{t-1}, a_{t-1}, s_t) \mapsto m_t$. Further assumptions made by our method on the system and M_t are motivated and stated in section 5.1.

5 METHOD

This section outlines the two main components of the SWMPO algorithm: (1) State Machine Synthesis, where we use data from episodes to synthesize an FSM that models the environment's structure and low-level agent behavior, and (2) State-Machine-Guided Policy Optimization, where the synthesized FSM is employed to optimize the policy. Our framework is outlined in algorithm 1.

Algorithm 1 SWMPO

Require: POMDP $\mathcal{M} = (S, A, T, S_0, \Omega, O)$, initial policy π_0, reward function R, mode number m, partition pruning tolerance error ϵ, learning rate γ, intrinsic reward factor η, RL algorithm

1: Collect trajectory dataset D with π_0

2: $\mathcal{F} = \mathsf{synthesizeFSMWM}(D, \pi_0, m, \epsilon, \gamma)$

3: **return** $\mathcal{F}$

As illustrated in fig. 1, the inputs to our proposed framework, SWMPO, are a POMDP $(S, A, T, S_0, \Omega, O)$ with associated reward function R, the number of modes m, and an expert policy π_0 used for initial data collection. The outputs are (1) a state machine with m nodes that approximates T, and (2) a policy that approximately maximizes the reward in the POMDP.

Our method synthesizes an FSM to model the structure of the environment, where each state in the FSM represents a distinct mode of the data (e.g., *swimming* or *walking*). A key challenge is discovering these modes in an unsupervised manner, as only the number of modes is assumed to be known. Ensuring clear separation between modes at different stages of the algorithm is also critical 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:

1. **Labeling**: Divide the transitions in the dataset into different mode subsets.
2. **Pruning**: Simplify the mode-transition dynamics of the partition by removing spurious transitions between modes.
3. **Transition Predicate Synthesis**: Learn when to transition between modes.

5.1 LABELING

Labeling addresses the problem of decomposing environment dynamics by assigning each transition in a dataset D of trajectories to one of m disjoint subsets, with each subset corresponding to a mode of the POMDP. We first state the assumptions of the labeling algorithm, and then describe the algorithm.

Let $\langle S, A, T, S_0, \Omega, O\rangle$ be a POMDP. Let S_t and A_t be the random variables of the state and action at time t respectively, under some fixed policy π. We focus on the case where observations conditioned on a state are deterministic, so $o_t = O(s_t)$. We are interested in modelling the mode variable M_t (e.g., $m_t =$ *walking*), taking values in some set M. Our method revolves around learning M_t as an intermediate computation of a learned first-order model of the form

$$f(m(o_{t-1}, a_{t-1}, o_t), o_t, a_t) \approx o_{t+1} - o_t$$

which we describe in this section. Ultimately we characterize modes as a categorical variable (i.e., robot is either *walking* or *swimming*), but we first approximate M_t as taking values in $\mathbb{R}^n$. We now impose constraints on the POMDP and the mode variable that allows us to design an algorithm to predict M_t. In summary, the assumptions imply that partial observability is a consequence solely of the latent mode variable and that this variable can in principle be predicted from previous observations.

Assumption 1: mode identifiability We start with the assumption that M_t can be modelled as a function $m_t \approx m(o_{t-1}, a_{t-1}, o_t)$. Intuitively, this assumption means that it is possible to identify the current mode by observing how the world changed under the latest action. We thus think of M_t as an abstraction over the observed change of the system under some action and state.

Assumption 2: change can be predicted conditioned on mode The next assumption is that the POMDP becomes a deterministic MDP conditioned on the mode. More precisely, we assume the existence of a function $T' : M \times A \times O \to O$ 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 a constraint that allows us to uniquely identify M_t.

Assumption 3: modes alone have minimal information 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 $M_t = \arg\min_{M_t \in \mathbf{M}} I(M_t, O_{t+1})$.

Under the assumptions so far, we can so far 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: mode vectors form clusters M_t corresponds to a partition of the state space where in expectation the within-subset sum of squares to the centroid of the subset is minimal. That is, we assume a *strict partitioning*, *centroid model* clustering scheme, which means that if k-means is run on vectors of M_t, then in expectation the clusters will correspond to the different modes.

Our 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 $m : O \times A \times O \to M$ and $f : M \times A \times O \to O$ under the joint optimization problem

$$\begin{aligned}\arg\min_{e,d}\ & \mathbb{E}_{(s_{t-1}, a_{t-1}, s_t, a_t, s_{t+1}) \sim \mathcal{M}_\pi} \left[\| f(m(o_{t-1}, a_{t-1}, o_t), a_t) - (T(s_t, a_t) - s_t) \| \right] \\ & - I(m(O_{t-1}, A_t, O_t), O_{t+1}),\end{aligned} \tag{1}$$

where $\|\cdot\|$ 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 eq. (1) implies that $m(\cdot)$ corresponds to the mode variable, which can be clustered with k-means to obtain mode labels for a set of data. In practice, we parametrize $m(\cdot)$ and $f(\cdot)$ with neural networks and approximate the solution 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 At this point it is possible to fit a local model for each cluster of transitions in the dataset, as illustrated in Fig. 2, where each local model has higher performance for a particular mode of the environment. This entire process results in algorithm 2.

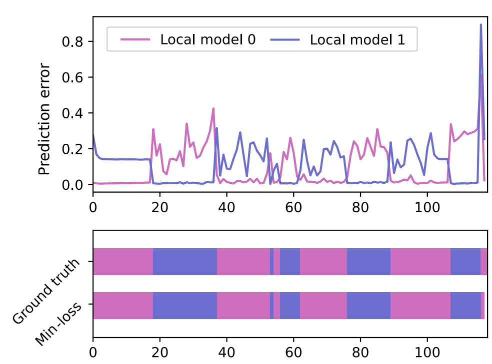


Figure 2: Performance of the specialized models for the *walking* and *swimming* modes of `PointMass`, an idealized version of an amphibious robot (see section 6). Each local model is specialized for a specific mode, leading to a combined low prediction error across the entire episode. The x-axis indicates time.

Algorithm 2 optimizePartition

Require: list of trajectories D, latent space M, number of modes m

1: Use gradient-descent to find $e : S \times A \to M$ and $d : M \times A \to \hat{S}$ that approximately solve eq. (1)
2: Embed the transitions into mode vectors, $m(D)$
3: Cluster the embeddings into m disjoint subsets using k-means
4: $D_i = \{\tau \in D \mid \text{cluster}(\tau) = i\}$
5: $\bar{D} = \{D_1, \ldots, D_m\}$
6: Fit a local model f_i to approximate the dynamics of D_i.
7: **return** $\bar{D}, f$

The algorithm takes a dataset of trajectories D, and partitions it by solving eq. (1) and clustering the resulting mode vectors. The partition, say $\bar{D} = \{D_1, \ldots, D_m\}$, induces the sequence of modes that the state machine should visit for a given trajectory in the dataset. That is, the state machine should be in state i when processing transition τ if and only if $\tau \in D_i$.

5.2 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, transitions between these models can be pruned with minimal impact on performance.

To address this, we apply 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 and reduce interpretability. We now describe the pruning approach, which optimizes for both accuracy and simplicity.

Pruning Approach We begin 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). Our goal is to remove transitions that have minimal impact on prediction accuracy.

To prune a mode transition, the framework shifts the affected transitions from one subset to another, causing a different model, with equal or greater prediction error, to handle those transitions. If the increase in prediction error 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. There may be multiple ϵ-prunable mode transitions for a given trajectory and partition. Our approach is to greedily prune the first prunable mode transitions with the strategy that results in the smallest prediction error increase (see algorithm 3).

Algorithm 3 greedyPrune

Require: Partition $\bar{D}$ of trajectory dataset D, error tolerance factor ϵ,

1: $\bar{D}_0 = \text{copy}(\bar{D})$
2: **for** $t \in D$ **do**
3:     **while exists** ϵ-prunnable (relative to $\bar{D}_0$) mode transition in t **do**
4:         Prune the first ϵ-prunable (relative to $\bar{D}_0$) mode transition in $\bar{D}$, updating $\bar{D}$
5: **return** $\bar{D}$

5.2.1 Transition predicate synthesis

We describe the mechanism by which the FSM learns when to transition from one mode to another.

Each subset of a partition corresponds to a state of the FSM being synthesized. For each pair of FSM states (f_i, f_j), the core question is: 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? We identify 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 then becomes a standard classification problem, where we find a predicate that outputs True for the positive set and False for the negative set. See algorithm 4. We use scikit-learn Pedregosa et al. (2011) to synthesize these predicates, parametrizing them with small Multi-Layer Perceptrons.

Algorithm 4 synthesizeTransitions

Require: Partition $\bar{D} = \{D_1, \ldots, D_m\}$ and corresponding list of trajectories D

```
for i, j ∈ {1, ..., m} × {1, ..., m} do
    positive = {τ1 ∈ D_i | ∃τ2 ∈ D_j s.t. follows(τ1, τ2)}
    negative = D_i \ {positive}
    δ_{i,j} = synthesizePredicate(positive, negative)
return δ
```

Algorithm 5 synthesizeFSMWM

Require: Dataset D of environment transitions, initial policy π_0, mode number m, partition pruning tolerance error ϵ, learning rate γ, RL algorithm

```
(D̄′, f′) ≜ optimizePartition(D, m, γ)
Sort D̄′ = {D_1, ..., D_m} so that D_1 contains the most initial transitions.
D̄ = greedyPrune(D̄′, ε, D)
δ = synthesizeTransitions(D̄)
F = (f, S, A, δ, f_1)
return F
```

6 Experiments

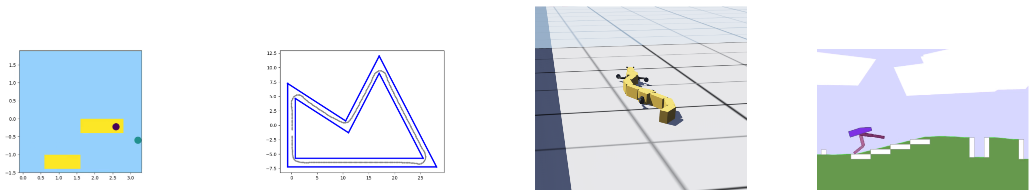
We evaluate SWMPO's ability to identify and approximate the modes of the environment.

6.1 Test Environments

We test SWMPO on four environments of varying complexity (see fig. 3):

1. `PointMass`. These tasks are a simplified version 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. Crucially, the environment is split into terrains with different characteristics: sand with no drag and water with high drag. Additionally, to simulate the need for different policies in different terrains, actions in the sand terrain are inverted. See fig. 3a. We use an MPC controller as the initial expert policy.
2. `LiDAR-Racing`. Adapted from 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. See fig. 3b. We use a pre-trained controller provided by the authors as the expert controller.
3. `Salamander`. A locomotion task in which an amphibious salamander must navigate through water and land. This environment is implemented in the Webots 3D simulator (Michel, 2004), in which the *Salamandra Robotica II* (Crespi et al., 2013) robot is available. See fig. 3c. This environment is a scaled-up version of `PointMass`. For observations, we use the motor positions, the LiDAR readings and the GPS position. We use the controller provided by Webots for this robot as the expert policy. However, to satisfy Assumption 2 we randomly switch the controller's mode, so that the robot sometimes performs swimming actions on the land and viceversa. This is so that the change in the world can only be accurately predicted if the mode variable is extracted.

4. `BipedalWalkerHarcore` (`BipedalWH`). A locomotion task in which a bipedal robot has to locomote over uneven terrain with four different types of obstacles. This is a standard benchmark in the Gymnasium library Towers et al. (2024) and employes the Box2D rigid body simulator Catto (2024). We use a pre-trained controller from the RL Baselines3 Zoo library Raffin (2020) as the expert policy.



(a) `PointMass` (b) `LiDAR-Racing` (c) `Salamander` (d) `BipedalWH`

Figure 3: Benchmark domains.

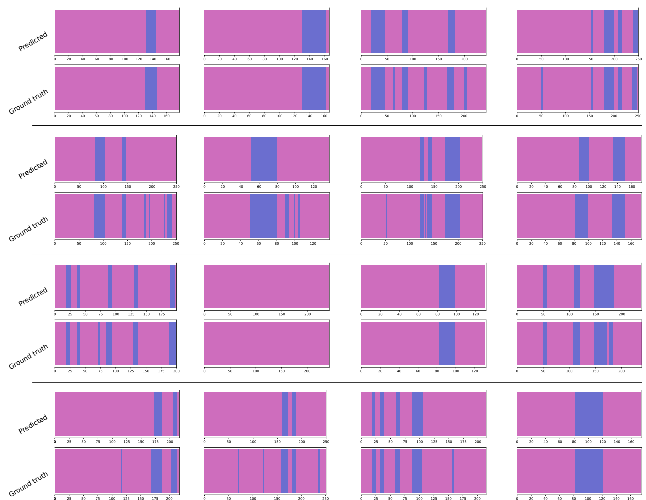
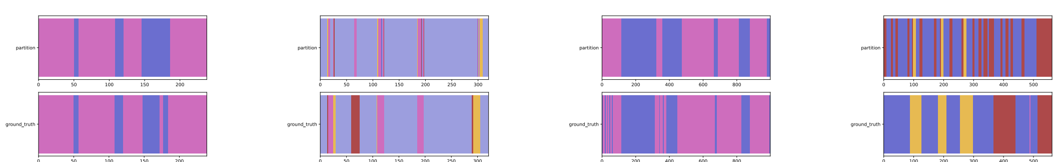
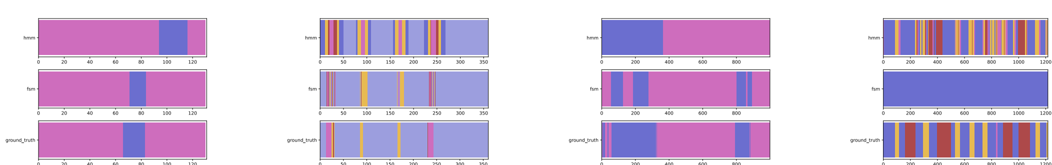


Figure 4: Labels of all the input trajectories from the `PointMass` environment. In each plot, the



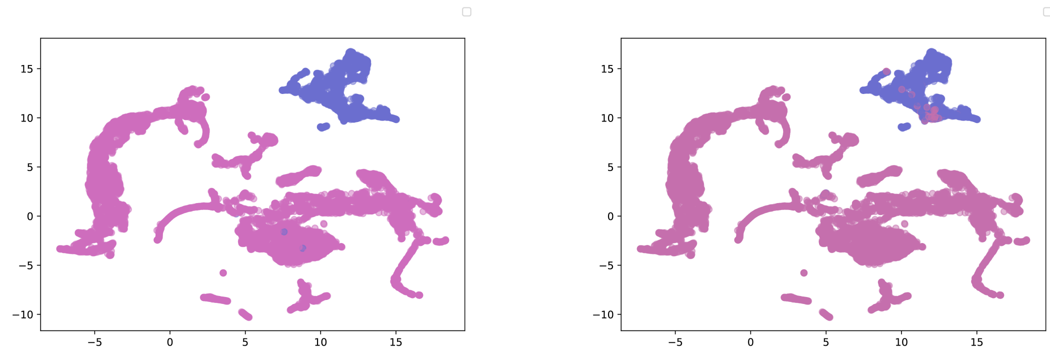
(a) `PointMass` (b) `LiDAR-Racing` (c) `Salamander` (d) `BipedalWH`

Figure 5: Example unsupervised labeling outputs.



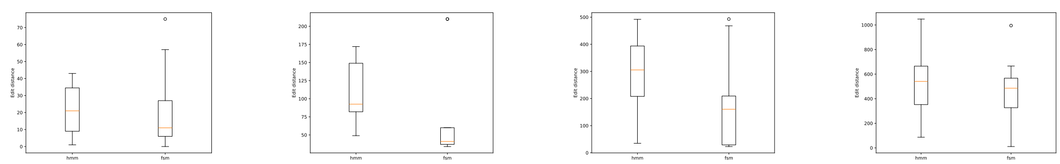
(a) `PointMass` (b) `LiDAR-Racing` (c) `Salamander` (d) `BipedalWH`

Figure 6: Example mode tracking on unseen data.



(a) `Ground truth` (b) `Learned partition`

Figure 7: Mode vectors learned through SWMPO in the `PointMass` environment. Each point represents a transition encoded with the learned $m(\cdot)$ (after dimensionality reduction through UMAP). In the left plot, the ground truth labels are used to color the vectors. In the right, the learned partition is used to assign colors.



(a) `PointMass` (b) `LiDAR-Racing` (c) `Salamander` (d) `BipedalWH`

Figure 8: We compare the performance of our method against HMMs across four different environments. The box plots illustrate the Levenshtein distance between FSM-predicted and ground truth labels for each environment, with SWMPO results shown on the right and HMM results on the left.

The algorithm to approximate the solution to eq. (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 Scikit-learn (Pedregosa et al., 2011).

6.2 Learned Representation of the Mode Variable

We present a plot of the data in the `PointMass` environment, where each transition is colored according to both the ground truth and the learned partitions, shown in fig. 7. The results indicate that the data forms clusters that align with the ground truth labels, demonstrating that the different modes are separated. There is a high level of correspondence between the ground truth and learned labels, although a few transitions are mislabeled.

6.3 FSM synthesis

We evaluate the performance of our FSM synthesis algorithm across all four environments.

For each environment, we use the expert policy to generate input data. We then use SWMPO to partition the transitions in the input data into the number of modes for that environment. For illustration purposes, we include all the labeled data for the `PointMass` environment (see fig. 5). We then synthesize the FSMWM. We compare the states visited by the synthesized FSMWM in unseen data against the ground truth states, as well as the visited states predicted by a hidden Markov model with Gaussian emissions fitted to the training data. See fig. 6). We calculate the accuracy of each partition with the Levenshtein distance to ground-truth labels (see fig. 8).

In `PointMass`, SWMPO outperforms HMMs and in `LiDAR-Racing` and `Salamander`, SWMPO significantly outperforms HMM. In `Bipedal Walker`, SWMPO marginally beats

HMM, however, where the models struggle to capture the underlying dynamics of the agent in its environment.

7 LIMITATIONS

The main limitations of the framework are stated formally as assumptions in section 5.1. The main assumption is that the partial observability of the environment is a consequence solely of the mode variable. Another limitation is that the mode variable must be approximated from a single transition; generalizing this to allow for modes that require multiple steps to be identified is left for future work.

8 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. Our analysis shows that the synthesized FSMWMs effectively capture the underlying structure of the environment by mapping latent modes to discrete states. Additionally, our algorithm matches or surpasses the performance of a Hidden Markov Model baseline on challenging dynamical systems. The implementation of the framework and all the code necessary to replicate the experiments, including hyperparameters, are attached to this manuscript, and are open sourced.

REFERENCES

R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, February 1995. ISSN 0304-3975. doi: 10.1016/0304-3975(94)00202-T. URL `https://www.sciencedirect.com/science/article/pii/030439759400202T`.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular Multitask Reinforcement Learning with Policy Sketches. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 166–175. PMLR, July 2017. URL `https://proceedings.mlr.press/v70/andreas17a.html`. ISSN: 2640-3498.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. 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, April 2024. doi: 10.1145/3620665.3640366. URL `https://pytorch.org/assets/pytorch2-2.pdf`.

Matthew Michael Botvinick. Hierarchical reinforcement learning and decision making. *Current Opinion in Neurobiology*, 22(6):956–962, December 2012. ISSN 0959-4388. doi: 10.1016/j.conb.2012.05.008. URL `https://www.sciencedirect.com/science/article/pii/S0959438812000876`.

Nizar Bouguila, Wentao Fan, and Manar Amayri. *Hidden Markov models and applications*. Springer, 2022.

E. F. Camacho, D. R. Ramirez, D. Limon, D. Muñoz de la Peña, and T. Alamo. Model predictive control techniques for hybrid systems. *Annual Reviews in Control*, 34(1):21–31, April 2010. ISSN 1367-5788. doi: 10.1016/j.arcontrol.2010.02.002. URL `https://www.sciencedirect.com/science/article/pii/S1367578810000040`.

Erin Catto. erincatto/box2d, September 2024. URL https://github.com/erincatto/box2d. original-date: 2015-03-14T16:52:46Z.

Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering Symbolic Models from Deep Learning with Inductive Biases. In *Advances in Neural Information Processing Systems*, volume 33, pp. 17429–17442. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c9f2f917078bd2db12f23c3b413d9cba-Abstract.html.

Alessandro Crespi, Konstantinos Karakasiliotis, André Guignard, and Auke Jan Ijspeert. Salamandra Robotica II: An Amphibious Robot to Study Salamander-Like Swimming and Walking Gaits. *IEEE Transactions on Robotics*, 29(2):308–320, April 2013. ISSN 1941-0468. doi: 10.1109/TRO.2012.2234311. URL https://ieeexplore.ieee.org/document/6416074. Conference Name: IEEE Transactions on Robotics.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2169–2176, May 2017. doi: 10.1109/ICRA.2017.7989250.

Giancarlo Ferrari-Trecate, Marco Muselli, Diego Liberati, and Manfred Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39(2):205–217, February 2003. ISSN 0005-1098. doi: 10.1016/S0005-1098(02)00224-8. URL https://www.sciencedirect.com/science/article/pii/S0005109802002248.

Chong Yang Goh, Justin Dauwels, Nikola Mitrovic, Muhammad Tayyab Asif, Ali Oran, and Patrick Jaillet. Online map-matching based on hidden markov model for real-time traffic sensing applications. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 776–781. IEEE, 2012.

Kavi Gupta, Chenxi Yang, Kayla McCue, Osbert Bastani, Phillip A. Sharp, Christopher B. Burge, and Armando Solar-Lezama. Improved modeling of RNA-binding protein motifs in an interpretable neural model of RNA splicing. *Genome Biology*, 25(1):23, January 2024. ISSN 1474-760X. doi: 10.1186/s13059-023-03162-x. URL https://doi.org/10.1186/s13059-023-03162-x.

David Ha and Jürgen Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html.

Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9): 7647–7656, May 2021. ISSN 2374-3468. doi: 10.1609/aaai.v35i9.16935. URL https://ojs.aaai.org/index.php/AAAI/article/view/16935. Number: 9.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 2107–2116. PMLR, July 2018. URL https://proceedings.mlr.press/v80/icarte18a.html. ISSN: 2640-3498.

Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. 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. URL https://openreview.net/forum?id=S1l8oANFDH.

Radoslav Ivanov, Kishor Jothimurugan, Steve Hsu, Shaan Vaidya, Rajeev Alur, and Osbert Bastani. Compositional Learning and Verification of Neural Network Controllers. *ACM Transactions on Embedded Computing Systems*, 20(5s):92:1–92:26, September 2021. ISSN 1539-9087. doi: 10.1145/3477023. URL https://dl.acm.org/doi/10.1145/3477023.

Henrik Jacobsson. Rule Extraction from Recurrent Neural Networks: ATaxonomy and Review. *Neural Computation*, 17(6):1223–1263, June 2005. ISSN 0899-7667. doi: 10.1162/0899766053630350. Conference Name: Neural Computation.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

John Kolen. Fool' s Gold: Extracting Finite State Machines from Recurrent Network Dynamics. In *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993. URL https://proceedings.neurips.cc/paper/1993/hash/470e7a4f017a5476afb7eeb3f8b96f9b-Abstract.html.

Anurag Koul, Sam Greydanus, and Alan Fern. Learning Finite State Representations of Recurrent Policy Networks, November 2018. URL http://arxiv.org/abs/1811.12530. arXiv:1811.12530 [cs, stat].

Cen Li and Gautam Biswas. Applying the hidden Markov model methodology for unsupervised learning of temporal data. *International Journal of Knowledge Based Intelligent Engineering Systems*, 6(3):152–160, 2002. Publisher: UNKNOWN.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. 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. URL http://anytime.cs.umass.edu/aimath06/proceedings/P21.pdf.

Naresh Marturi, Marek Kopicki, Alireza Rastegarpanah, Vijaykumar Rajasekaran, Maxime Adjigble, Rustam Stolkin, Aleš Leonardis, and Yasemin Bekiroglu. Dynamic grasp and trajectory planning for moving objects. *Autonomous Robots*, 43:1241–1256, 2019. Publisher: Springer.

O. Michel. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004. URL http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf.

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based Reinforcement Learning: A Survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, January 2023. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000086. URL https://www.nowpublishers.com/article/Details/MAL-086. Publisher: Now Publishers, Inc.

Aditya Mohan, Amy Zhang, and Marius Lindauer. Structure in Deep Reinforcement Learning: A Survey and Open Problems. *J. Artif. Int. Res.*, 79, April 2024. ISSN 1076-9757. doi: 10.1613/jair.1.15703. URL https://dl.acm.org/doi/10.1613/jair.1.15703.

Simone Paoletti, Aleksandar Lj. Juloski, Giancarlo Ferrari-Trecate, and René Vidal. Identification of Hybrid Systems A Tutorial. *European Journal of Control*, 13(2):242–260, January 2007. ISSN 0947-3580. doi: 10.3166/ejc.13.242-260. URL https://www.sciencedirect.com/science/article/pii/S0947358007708221.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Antonin Raffin. RL Baselines3 Zoo, 2020. URL https://github.com/DLR-RM/rl-baselines3-zoo. Publication Title: GitHub repository.

Christopher Simpkins and Charles Isbell. Composable Modular Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4975–4982, July 2019. ISSN 2374-3468. doi: 10.1609/aaai.v33i01.33014975. URL https://ojs.aaai.org/index.php/AAAI/article/view/4428. Number: 01.

Miriam García Soto, Thomas A. Henzinger, and Christian Schilling. 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, pp. 1–11, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8339-4. doi: 10.1145/3447928.3456704. URL https://dl.acm.org/doi/10.1145/3447928.3456704.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction, 2nd ed.* Reinforcement learning: An introduction, 2nd ed. The MIT Press, Cambridge, MA, US, 2018. ISBN 978-0-262-03924-6. Pages: xxii, 526.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/532435c44bec236b471a47a88d63513d-Abstract.html.

Mark Towers, Jordan K Terry, Ariel Kwiatkowski, John U. Balis, Gianluca Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Jin Shen Tan, and Omar G. Younis. Gymnasium, May 2024. URL https://zenodo.org/records/11232524.

Ke Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. Unsupervised neural hidden Markov models. *arXiv preprint arXiv:1609.09007*, 2016.

Aleksandar Vakanski, Iraj Mantegh, Andrew Irish, and Farrokh Janabi-Sharifi. 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):1039–1052, 2012. Publisher: IEEE.

Hongmin Wu, Yisheng Guan, and Juan Rojas. A latent state-based multimodal execution monitor with anomaly detection and classification for robot introspection. *Applied Sciences*, 9(6):1072, 2019. Publisher: MDPI.

Duo Xu and Faramarz Fekri. Interpretable Model-based Hierarchical Reinforcement Learning using Inductive Logic Programming, June 2021. URL http://arxiv.org/abs/2106.11417. arXiv:2106.11417 [cs].

A NOTATION TABLE

Table 1: Notation Table

Symbol	Meaning
$\mathcal{M}$	Markov Decision Process (MDP)
$S \in \mathbb{R}^k$	Set of observations
A	Set of actions
$T : S \times A \rightarrow S$	Transition function
S_0	Distribution of initial states
$R : S \times A \times S \rightarrow \mathbb{R}$	Reward function
$\pi_\theta : S \rightarrow A$	Policy parametrized by θ
$V_{\pi_\theta}(s)$	Value function for state s
s_t	Source state 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
$(\mathcal{M} \otimes \mathcal{F})$	Product MDP
$S^\otimes$	Augmented state space
$S_0^\otimes$	Augmented initial state distribution
$T^\otimes$	Transition function for product MDP