AXIOM: Learning to Play Games in Minutes with Expanding Object-Centric Models

Conor Heins[1*] **Toon Van de Maele**[1*] **Alexander Tschantz**[1,2*]
Hampus Linander[1] **Dimitrije Markovic**[3] **Tommaso Salvatori**[1] **Corrado Pezzato**[1]
Ozan Catal[1] **Ran Wei**[1] **Magnus Koudahl**[1] **Marco Perin**[1] **Karl Friston**[1,4]
Tim Verbelen[1] **Christopher L Buckley**[1,2]
[1] VERSES AI
[2] University of Sussex, Department of Informatics
[3] Technische Universität Dresden, Faculty of Psychology
[4] University College London, Queen Square Institute of Neurology
{conor.heins,toon.vandemaele,alec.tschantz}@verses.ai

Abstract

Current deep reinforcement learning (DRL) approaches achieve state-of-the-art performance in various domains, but struggle with data efficiency compared to human learning, which leverages core priors about objects and their interactions. Active inference offers a principled framework for integrating sensory information with prior knowledge to learn a world model and quantify the uncertainty of its own beliefs and predictions. However, active inference models are usually crafted for a single task with bespoke knowledge, so they lack the domain flexibility typical of DRL approaches. To bridge this gap, we propose a novel architecture that integrates a minimal yet expressive set of *core priors* about object-centric dynamics and interactions to accelerate learning in low-data regimes. The resulting approach, which we call AXIOM, combines the usual data efficiency and interpretability of Bayesian approaches with the across-task generalization usually associated with DRL. AXIOM represents scenes as compositions of objects, whose dynamics are modeled as piecewise linear trajectories that capture sparse object-object interactions. The structure of the generative model is expanded online by growing and learning mixture models from single events and periodically refined through Bayesian model reduction to induce generalization. AXIOM masters various games within only 10,000 interaction steps, with both a small number of parameters compared to DRL, and without the computational expense of gradient-based optimization.

1 Introduction

Reinforcement learning (RL) has achieved remarkable success as a flexible framework for mastering complex tasks. However, current methods have several drawbacks: they require large amounts of training data, depend on large replay buffers, and focus on maximizing cumulative reward without structured exploration [1]. This contrasts with human learning, which relies on core priors to quickly generalize to novel tasks [2–4]. Core priors represent fundamental organizational principles - or hyperpriors - that shape perception and learning, providing the scaffolding upon which more complex knowledge structures are built. For example, such priors allow humans to intuitively understand that objects follow smooth trajectories unless external forces intervene, and shape our causal reasoning, helping us to grasp action-consequence relationships [5–8]. Describing visual scenes as factorized into objects has shown promise in sample efficiency, generalization, and robustness on various tasks [9–14]. These challenges are naturally addressed by Bayesian agent architectures, such as active inference [15], that provide a principled framework for incorporating prior knowledge into models,

Preprint. Under review.

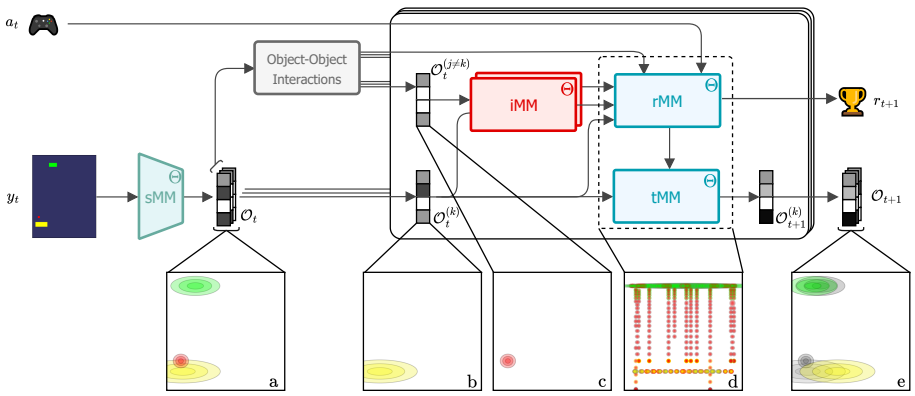


Figure 1: **Inference and prediction flow using AXIOM**: The sMM extracts object-centric representations from pixel inputs. For each object latent and its closest interacting counterpart, a discrete identity token is inferred using the iMM and passed to the rMM, along with the distance and the action, to predict the next reward and the tMM switch. The object latents are then updated using the tMM and the predicted switch to generate the next state for all objects. (a) Projection of the object latents into image space. (b) Projection of the k^{th} latent whose dynamics are being predicted and (c) of its interaction partner. (d) Projection of the rMM in image space; each of the visualized clusters corresponds to a particular linear dynamical system from the tMM. (e) Projection of the predicted latents. The past latents at time t are shown in gray.

supporting continual adaptation without catastrophic forgetting. It has been argued that this approach aligns closely with human cognitive processes [16, 17], where beliefs are updated incrementally as new evidence emerges. Yet, despite these theoretical advantages, applications of active inference have typically been confined to small-scale tasks with carefully designed priors, failing to achieve the versatility that makes deep RL so powerful across diverse domains.

To bridge this gap, we propose a novel active inference architecture that integrates a minimal yet expressive set of core priors about objects and their interactions [9–12, 18]. Specifically, we present AXIOM (**A**ctive e**X**panding **I**nference with **O**bject-centric **M**odels), which employs a object-centric state space model with three key components: (1) a Gaussian mixture model that parses visual input into object-centric representations and automatically expands to accommodate new objects; (2) a transition mixture model that discovers motion prototypes (e.g., falling, sliding, bouncing) [19] and (3) a sparse relational mixture model over multi-object latent features, learning causally relevant interactions as jointly driven by object states, actions, rewards, and dynamical modes. AXIOM's learning algorithm offers three kinds of efficiency: first, it learns sequentially one frame at a time with variational Bayesian updating [20]. This eliminates the need for replay buffers or gradient computations, and enables online adaptation to changes in the data distribution. Second, its mixture architecture facilitates fast structure learning by both adding new mixture components when existing ones cannot explain new data, and merging redundant ones to reduce model complexity [21–24]. Finally, by maintaining posteriors over parameters, AXIOM can augment policy selection with information-seeking objectives and thus uncertainty-aware exploration [15].

To empirically validate our model, we introduce the `Gameworld 10k` benchmark, a new set of environments designed to evaluate how efficiently an agent can play different pixel-based games in 10k interactions. Many existing RL benchmarks, such as the Arcade Learning Environment (ALE) [25] or MuJoCo [26] domains, emphasize long-horizon credit assignment, complex physics, or visual complexity. These factors often obscure core challenges in fast learning and generalization, especially under structured dynamics. To this end, each of the games in `Gameworld 10k` follows a similar, object-focused pattern: multiple objects populating a visual scene, a player object that can be controlled to score points, and objects following continuous trajectories with sparse interaction mechanics. We formulate a set of 10 games with deliberately simplified visual elements (single color sprites of different shapes and sizes) to focus the current work on the representational mechanisms used for modeling dynamics and control, rather than learning an overly-expressive model for object segmentation. The Gameworld environments also enable precise control of game features and

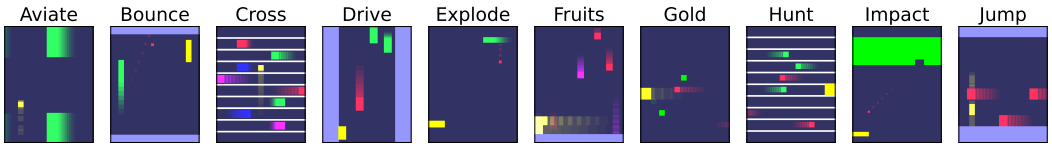


Figure 2: **Gameworld10k**: Visual impression of the 10 games in the `Gameworld 10k` suite. Sequences of ten frames are overlayed with increasing opacity to showcase the game dynamics.

dynamics, which allows testing how systems adapt to sparse interventions to the causal or visual structure of the game, e.g., the shape and color of game objects. On this benchmark, our agent outperforms popular reinforcement learning models in the low-data regime (10,000 interaction steps) without relying on any kind of gradient-based optimization. To conclude, although we have not deployed AXIOM at the scale of complicated control tasks typical of the RL literature, our results represent a meaningful step toward building agents capable of building compact, interpretable world models and exploiting them for rapid decision-making across different domains. Our main contributions are the following:

- We introduce AXIOM, a novel object-centric active inference agent that is learned online, interpretable, sample efficient, adaptable and computationally cheap.[1]
- To demonstrate the efficacy of AXIOM, we introduce a new, modifiable benchmark suite targeting sample-efficient learning in environments with objects and sparse interactions.
- We show that our gradient-free method can outperform state-of-the-art deep learning methods both in terms of sample efficiency and absolute performance, with our online learning scheme showing robustness to environmental perturbations.

2 Methods

AXIOM is formulated in the context of a partially observable Markov decision process (POMDP). At each time step t, the hidden state h_t evolves according to $h_t \sim P\big(h_t \mid h_{t-1}, a_{t-1}\big)$, where a_t is the action taken at time t. The agent does not observe h_t directly but instead receives an observation $y_t \sim P\big(y_t \mid h_t\big)$, and a reward $r_t \sim P\big(r_t \mid h_t, a_t\big)$. AXIOM learns an **object-centric state space model** by maximizing the Bayesian model evidence—equivalently, minimizing (expected) free energy—through active interaction with the environment [15]. The model factorizes perception and dynamics into separate generative blocks: (i) In perception, a slot Mixture Model (sMM) explains pixels with competition between object-centric latent variables $\mathcal{O}_t = \{\mathcal{O}_t^{(1)}, \ldots, \mathcal{O}_t^{(K)}\}$, associating each pixel to one of K slots using the assignment variable $\mathbf{z}_{t,\text{smm}}$; (ii) dynamics are modeled per-object using their object-centric latent descriptions as inputs to a recurrent switching state space model (similar to an rSLDS [19]). We define the full latent sequence as $\mathcal{Z}_{0:T} = \{\mathcal{O}_t, \mathbf{z}_{t,\text{smm}}\}_{t=0}^{T}$. Each slot latent $\mathcal{O}_t^{(k)}$ consists of both continuous $\mathbf{x}_t^{(k)}$ and discrete latent variables. The continuous latents represent properties of an object, such as its position, color and shape. The discrete latents themselves are split into two subtypes: $\mathbf{z}_t^{(k)}$ and $\mathbf{s}_t^{(k)}$. We use $\mathbf{z}_t^{(k)}$ to denote latent descriptors that capture categorical attributes of the slot (e.g., object type), and $\mathbf{s}_t^{(k)}$ to denote a pair of switch states determining the slot's instantaneous trajectory.[2] Model parameters $\tilde{\mathbf{\Theta}}$ are split into module-specific subsets (e.g., $\mathbf{\Theta}_{\text{sMM}}, \mathbf{\Theta}_{\text{tMM}}$). The joint distribution over input sequences $\mathbf{y}_{0:T}$, latent state sequences $\mathcal{Z}_{0:T}$ and parameters $\tilde{\mathbf{\Theta}}$ can be expressed as a hidden Markov model:

$$p(\mathbf{y}_{0:T}, \mathcal{Z}_{0:T}, \tilde{\mathbf{\Theta}}) = p(y_0, \mathcal{Z}_0)p(\tilde{\mathbf{\Theta}}) \prod_{t=1}^{T} \underbrace{p(\mathbf{x}_{t-1}, \mid \mathbf{z}_t, \mathbf{\Theta}_{\text{iMM}})}_{\text{Identity mixture model}} \underbrace{p(\mathbf{x}_{t-1}, \mathbf{z}_t, \mathbf{s}_t, a_{t-1}, r_t \mid \mathbf{\Theta}_{\text{rMM}})}_{\text{Recurrent mixture model}}$$
$$\prod_{k=1}^{K} \underbrace{p(\mathbf{y}_t | \mathbf{x}_t^{(k)}, \mathbf{z}_{t,\text{smm}}, \mathbf{\Theta}_{\text{sMM}})}_{\text{Slot mixture model}} \underbrace{p(\mathbf{x}_t^{(k)} \mid \mathbf{x}_{t-1}^{(k)}, \mathbf{s}_t^{(k)}, \mathbf{\Theta}_{\text{tMM}})}_{\text{Transition mixture model}}, \quad (1)$$

[1]The code for training AXIOM is available at https://github.com/VersesTech/axiom

[2]We use the superscript index k as in $q^{(k)}$ to select only the subset of $q \equiv q^{(1:K)}$ relevant to the k^{th} slot.

where $p(\tilde{\mathbf{\Theta}}) = p(\mathbf{\Theta}_{\text{sMM}})p(\mathbf{\Theta}_{\text{iMM}})p(\mathbf{\Theta}_{\text{rMM}})p(\mathbf{\Theta}_{\text{tMM}})$. The sMM $p(\mathbf{y}_t|\mathbf{x}_t, \mathbf{z}_{t,\text{smm}}, \mathbf{\Theta}_{\text{sMM}})$ is a likelihood model that explains pixel data using mixtures of slot-specific latent states (see schematic in Figure 1). The identity mixture model (iMM) $p(\mathbf{x}_{t-1}|\mathbf{z}_t, \mathbf{\Theta}_{\text{iMM}})$ is a likelihood model that assigns each object-centric latent to one of a set of discrete object types. The transition mixture model (tMM) $p(\mathbf{x}_t^{(k)}|\mathbf{x}_{t-1}^{(k)}, \mathbf{s}_t^{(k)}, \mathbf{\Theta}_{\text{tMM}})$ describes each object's latent dynamics as a piecewise linear function of its own state. Finally, the recurrent mixture model (rMM) $p(\mathbf{x}_{t-1}, \mathbf{z}_t, \mathbf{s}_t, a_{t-1}, r_t \mid \mathbf{\Theta}_{\text{rMM}})$ models the dependencies between multi-object latent states (like the switch states of the transition mixture), other global game states like reward r, action a, and the continuous and discrete features of each object. This module is what allows AXIOM to model sparse interactions between objects (e.g., collisions), while still treating each slot's dynamics as conditionally-independent given the switch states $\mathbf{s}_t^{(k)}$.

Slot Mixture Model (sMM). AXIOM processes sequences of RGB images one frame at a time. Each image is composed of $H \times W$ pixels and is reshaped into $N = HW$ tokens $\{\mathbf{y}_t^n\}_{n=1}^N$. Each token $\mathbf{y}_t^n$ is a vector containing the n^{th} pixel's color in RGB and its image coordinates (normalized to $[-1, +1]$). AXIOM models these tokens at a given time as explained by a mixture of the continuous slot latents; we term this likelihood construction the *Slot Mixture Model* (sMM, see far left side of Figure 1). The K components of this mixture model are Gaussian distributions whose parameters are directly given by the continuous features of each slot latent $\mathbf{x}_t^{(1:K)}$. Associated to this Gaussian mixture is a binary assignment variable $z_{t,k,\text{smm}}^n \in \{0, 1\}$ indicating whether pixel n at time t is driven by slot k, with the constraint that $\sum_k z_{t,k,\text{smm}}^n = 1$. The sMM's likelihood model for a single pixel and timepoint $\mathbf{y}_t^n$ can be expressed as follows (dropping the t subscript for notational clarity):

$$p(\mathbf{y}^n \mid \mathbf{x}^{(k)}, \sigma_{\mathbf{c}}^{(k)}, z_{k,\text{smm}}^n) = \prod_{k=1}^{K} \mathcal{N}(A\mathbf{x}^{(k)}, \text{diag}([B\mathbf{x}^{(k)}, \sigma_{\mathbf{c}}^{(k)}]^\top))^{z_{k,\text{smm}}^n}. \tag{2}$$

The mean of each Gaussian component is given a fixed linear projection A of each object latent, which selects only its position and color features: $A\mathbf{x}^{(k)} = [\mathbf{p}^{(k)}, \mathbf{c}^{(k)}]$. The covariance of each component is a diagonal matrix whose diagonal is a projection of the 2-D shape of the object latent $B\mathbf{x}^{(k)} = \mathbf{e}^{(k)}$ (its spatial extent in the X and Y directions), stacked on top of a fixed variance for each color dimension $\sigma_{\mathbf{c}}^{(k)}$, which are given independent Gamma priors. The latent variables $\mathbf{p}^{(k)}, \mathbf{c}^{(k)}, \mathbf{e}^{(k)}$ are subsets of slot k's full continuous features $\mathbf{x}_t^{(k)}$, and the projection matrices A, B are fixed, unlearned parameters. Each token's slot indicator $\mathbf{z}_{\text{smm}}^n$ is drawn from a Categorical distribution $\mathbf{z}_{\text{smm}}^n \mid \boldsymbol{\pi}_{\text{smm}} \sim \text{Cat}(\boldsymbol{\pi}_{\text{smm}})$ with mixing weights $\boldsymbol{\pi}_{\text{smm}}$. We place a truncated stick-breaking (finite GEM) prior on these weights, which is equivalent to a K-dimensional Dirichlet with concentration vector $(1, \ldots, 1, \alpha_{0,\text{smm}})$, where the first $K-1$ pseudcounts are 1 and the final pseudocount $\alpha_{0,\text{smm}}$ reflects the propensity to add new slots. All subsequent mixture models in AXIOM are equipped with the same sort of truncated stick-breaking priors on the mixing weights [27].

Identity Mixture Model (iMM). AXIOM uses an *identity mixture model* (iMM) to infer a discrete identity code $\mathbf{z}_{\text{type}}^{(k)}$ for each object based on its continuous features. These identity codes are used to condition the inference of the recurrent mixture model used for dynamics prediction. Conditioning the dynamics on identity-codes in this way, rather than learning a separate dynamics model for each slot, allows AXIOM to use the same dynamics model across slots. This also enables the model to learn the same dynamics in a *type*-specific, rather than *instance*-specific, manner [28], and to remap identities when e.g., the environment is perturbed and colors change. Concretely, the iMM models the 5-D colors and shapes $\{\mathbf{c}^{(k)}, \mathbf{e}^{(k)}\}_{k=1}^K$ across slots as a mixture of up to V Gaussian components (object types). The slot-level assignment variable $\mathbf{z}_{t,\text{type}}^{(k)}$ indicates which identity is assigned to the k^{th}slot. The generative model for the iMM is (omitting the $t-1$ subscript from latent variables):

$$p([\mathbf{c}^{(k)}, \mathbf{e}^{(k)}]^\top | \mathbf{z}_{\text{type}}^{(k)}, \boldsymbol{\mu}_{1:V,\text{type}}, \boldsymbol{\Sigma}_{1:V,\text{type}}) = \prod_{j=1}^{V} \mathcal{N}(\boldsymbol{\mu}_{j,\text{type}}, \boldsymbol{\Sigma}_{j,\text{type}})^{z_{j,\text{type}}^{(k)}} \tag{3}$$

$$p(\boldsymbol{\mu}_{j,\text{type}}, \boldsymbol{\Sigma}_{j,\text{type}}^{-1}) = \text{NIW}(\mathbf{m}_{j,\text{type}}, \kappa_{j,\text{type}}, \mathbf{U}_{j,\text{type}}, n_{j,\text{type}}) \tag{4}$$

The same type of Categorical likelihood for the type assignments $\mathbf{z}_{\text{type}}^{(k)} \mid \boldsymbol{\pi}_{\text{type}} \sim \text{Cat}(\boldsymbol{\pi}_{\text{type}})$ and truncated stick-breaking prior $\text{Dir}(1, \ldots, 1, \alpha_{0,\text{type}})$ over the mixture weights is used to allow an

arbitrary (up to a maximum of V) number of types to be used to explain the continuous slot features. We equip the prior over the component likelihood parameters with conjugate Normal Inverse Wishart (NIW) priors.

Transition Mixture Model (tMM). The dynamics of each slot are modelled as a mixture of linear functions of the slot's own previous state. To stress the homology between this model and the other modules of AXIOM, we refer to this module as the *transition mixture model* or tMM, but this formulation is more commonly also known as a switching linear dynamical system or SLDS [29]. The tMM's switch variable $\mathbf{s}_{t,\text{tmm}}^{(k)}$ selects a set of linear parameters $\boldsymbol{D}_l, \boldsymbol{b}_l$ to describe the k^{th} slot's trajectory from t to $t+1$. Each linear system captures a distinct rigid motion pattern for a particular object (e.g., "ball in free flight", "paddle moving left").

$$p(\mathbf{x}_t^{(k)} \mid \mathbf{x}_{t-1}^{(k)}, \mathbf{s}_{t,\text{tmm}}^{(k)}, \boldsymbol{D}_{1:L}, \boldsymbol{b}_{1:L}) = \prod_{l=1}^{L} \mathcal{N}(\boldsymbol{D}_l \mathbf{x}_t^{(k)} + \boldsymbol{b}_l, 2I)^{s_{t,l,\text{tmm}}^{(k)}} \tag{5}$$

where we fix the covariance of all L components to be $2I$, and all mixture likelihoods $\boldsymbol{D}_{1:L}, \boldsymbol{b}_{1:L}$ to have uniform priors. The mixing weights $\boldsymbol{\pi}_{\text{tmm}}$ for $\mathbf{s}_{t,\text{tmm}}^{(k)}$ as before are given a truncated stick-breaking prior $\text{Dir}(1, \ldots, 1, \alpha_{0,\text{tmm}})$ enabling the number of linear modes L to be dynamically adjusted to the data by growing the model with propensity $\alpha_{0,\text{tmm}}$. Importantly, the L transition components of the tMM are not slot-dependent, but are shared and thus learned across all K slot latents. The tMM can thus explain and predict the motion of different objects using a shared, expanding set of dynamical motifs. As we will see in the next section, interactions between objects are modelled by conditioning $\mathbf{s}_{t,\text{tmm}}^{(k)}$ on the states of other objects.

Recurrent Mixture Model (rMM). AXIOM employs a *recurrent mixture model* (rMM) to infer the switch states of the transition model directly from current slot-level features. This dependence of switch states on continuous features is the same construct used in the recurrent switching linear dynamical system or rSLDS [19]. However, like the rSLDS, which uses a discriminative mapping to infer the switch state from the continuous state, rMM recovers this dependence *generatively* using a mixture model over mixed continuous–discrete slot states [30]. Concretely, the rMM models the distribution of continuous and discrete variables as a mixture model driven by another per-slot latent assignment variable $\mathbf{s}_{\text{rmm}}^{(k)}$. The rMM's defines a mixture likelihood over continuous and discrete slot-specific information: $(f_{t-1}^{(k)}, d_{t-1}^{(k)})$. The continuous slot features $f_{t-1}^{(k)}$ are a function of of both the k^{th} slot's own continuous state state $\mathbf{x}_{t-1}^{(k)}$ as well as the states of other slots $\mathbf{x}_{t-1}^{(1:K)}$, such as the distance to the closest object. The discrete features include categorical slot features like the identity of the closest object, the switch state associated with the transition mixture model, and the action and reward at the current timestep: $d_{t-1}^{(k)} = (\mathbf{z}_{t-1}^{(k)}, s_{t-1,\text{tmm}}^{(k)}, a_{t-1}, r_t)$. The rMM assignment variable associated to a given slot is a binary vector $\mathbf{s}_{t,\text{rmm}}^{(k)}$ whose m^{th} entry $s_{t,m,\text{rmm}}^{(k)} \in \{0, 1\}$ indicates whether component m explains the current tuple of mixed continuous-discrete data. Each component likelihood selected by $\mathbf{s}_{t,\text{rmm}}^{(k)}$ factorizes into a product of continuous (Gaussian) and discrete (Categorical) likelihoods.

$$f_{t-1}^{(k)} = \left(C\mathbf{x}_{t-1}^{(k)}, g(\mathbf{x}_{t-1}^{(1:K)})\right), \quad d_{t-1}^{(k)} = \left(\mathbf{z}_{t-1}^{(k)}, \mathbf{s}_{t,\text{tmm}}^{(k)}, a_{t-1}, r_t\right) \tag{6}$$

$$p(f_{t-1}^{(k)}, d_{t-1}^{(k)} \mid \mathbf{s}_{t,\text{rmm}}^{(k)}) = \prod_{m=1}^{M} \left[\mathcal{N}(f_{t-1}^{(k)}; \boldsymbol{\mu}_{m,\text{rmm}}, \boldsymbol{\Sigma}_{m,\text{rmm}}) \prod_i \text{Cat}\left(d_{t-1,\,i}; \boldsymbol{\alpha}_{m,i}\right)\right]^{s_{t,m,\text{rmm}}} \tag{7}$$

where the matrix C is a projection matrix that selects a subset of slot k's continuous features are used, and $g(\mathbf{x}_{t-1}^{(1:K)})$ summarizes functions that compute slot-to-slot interaction features, such as the X and Y-displacement to the nearest object, the identity code associated with the nearest object (and other features detailed in Appendix A). As with all the other modules of AXIOM, we equip the mixing weights for $\mathbf{s}_{t,\text{rmm}}^{(k)}$ with a truncated stick-breaking prior whose final M^{th} pseudocount parameter tunes the propensity to add new rMM components. We explored an ablation of the rMM (`fixed_distance`) where the X and Y-displacement vector is not returned by $g(\mathbf{x}_{t-1}^{(1:K)})$; rather, the distance that triggers detection of the nearest interacting object is a fixed hyperparameter of the g function. This hyperparameter can to be tuned to attain higher reward on most environments than the

standard model where the rMM learns the distance online. However, it comes at the cost of having to tune this hyperparamter in an environment-specific fashion (see Figure 3 and Table 1 for the effect of the `fixed_distance` ablation on performance).

Variational inference. AXIOM uses variational inference to perform state inference and parameter learning. Briefly, this requires updating an approximate posterior distribution $q(\mathcal{Z}_{0:T}, \tilde{\mathbf{\Theta}})$ over latent variables and parameters to minimize the variational free energy $\mathcal{F}$, an upper bound on negative log evidence $\mathcal{F} \geq -\log p(\mathbf{y}_{0:T})$. In doing so, the variational posterior approximates the true posterior $p(\mathcal{Z}_{0:T}, \tilde{\mathbf{\Theta}} \mid \mathbf{y}_{0:T})$ from exact but intractable Bayesian inference . We enforce independence assumptions in the variational posterior over several factors: across states and parameters, across the K slot latents, and over time T. This is known as the mean-field approximation:

$$q(\mathcal{Z}_{0:T}, \tilde{\mathbf{\Theta}}) = q(\tilde{\mathbf{\Theta}}) \prod_{t=0}^{T} \left(\prod_{n=1}^{N} q(\mathbf{z}_{t,\text{sMM}}^{n}) \right) \left(\prod_{k=1}^{K} q(\mathcal{O}_t^{(k)}) \right) \tag{8}$$

$$q(\mathcal{O}_t^{(k)}) = q(\mathbf{x}_t^{(k)}) q(\mathbf{z}_t^{(k)}) q(\mathbf{s}_t^{(k)}), \quad q(\tilde{\mathbf{\Theta}}) = q(\mathbf{\Theta}_{\text{sMM}}) q(\mathbf{\Theta}_{\text{iMM}}) q(\mathbf{\Theta}_{\text{tMM}}) q(\mathbf{\Theta}_{\text{rMM}}) \tag{9}$$

Note that the mixture variable of the sMM $\mathbf{z}_{t,\text{smm}}$ is factored out of the other object-centric latents in both the generative model and the posterior because unlike the other discrete latents $\mathbf{z}_t^{(k)}$, it is not independent across K slots.

We update the posterior over latent states $q(\mathcal{Z}_{0:T})$ (i.e., the variational E-step) using a simple form of forward-only filtering and update parameters using coordinate ascent variational inference, using the sufficient statistics of the latents updated during filtering and the data to update the parameters using simple natural parameter updates. These variational E-M updates are run once per timestep, thus implementing a fast, streaming form of coordinate-ascent variational inference [31, 32]. The simple, gradient-free form of these updates inherits from the exponential-family form of all the mixture models used in AXIOM.

2.1 Growing and pruning the model

Fast structure learning. In the spirit of *fast structure learning* [23], AXIOM dynamically *expands* all four mixture modules (sMM, iMM, tMM, rMM) using an online growing heuristic: process each new datapoint sequentially, decide whether it is best explained by an existing component or whether a new component should be created, and then update the selected component's parameters. We fix a maximum number of components $\mathcal{C}_{\max}$ for each mixture model and let $\mathcal{C}_{t-1} \leq \mathcal{C}_{\max}$ be the number currently in use. For each component c we store its variational parameters $\mathbf{\Theta}_c$, where for a particular model this might be a set of Normal Inverse Wishart parameters, e.g. $\mathbf{\Theta}_{c,\text{iMM}} = \{\mathbf{m}_c, \kappa_c, \mathbf{U}_c, n_c\}$.

Upon observing a new input y_t, we compute for each component $c = 1, \ldots, \mathcal{C}_{t-1}$ the variational posterior–predictive log–density $\ell_{t,c} = \mathbb{E}_{q(\mathbf{\Theta}_c)}\big[\log p(y_t \mid \mathbf{\Theta}_c)\big]$. The truncated stick–breaking prior $\boldsymbol{\pi} \sim \text{Dir}(1, \ldots, 1, \alpha)$ then defines a "new–component" threshold $\tau_t = \log p_0(y_t) + \log \alpha$ where p_0 is the prior predictive density under an empty component.We select the component with highest score, $c^* = \arg\max_{c \leq \mathcal{C}_{t-1}} \ell_{t,c}$ and hard-assign y_t to c^* if $\ell_{t,c^*} \geq \tau_t$; otherwise—provided $\mathcal{C}_{t-1} < \mathcal{C}_{\max}$—we instantiate a new component and assign y_t to it. Finally, given the hard assignment $\boldsymbol{z}_t$, we update the chosen component's parameters via a variational M-step (coordinate ascent).The last weight in the Dirichlet absorbs any remaining mass, so $\sum_{c=1}^{\mathcal{C}_{\max}} \pi_c = 1$.) This algorithm is a deterministic, *maximum a posteriori* version of the CRP assignment rule (see Equation (8) of [27]). The expansion threshold τ_t plays the role of the Dirichlet Process concentration α. When α is small the model prefers explaining data with existing slots; larger α makes growth more likely. The procedure is identical for the sMM, iMM, tMM and rMM—only the form of $p(\cdot \mid \Theta_c)$ and the model-specific caps on components $\mathcal{C}_{\max}$ and expansion thresholds τ_t differ.

Bayesian Model Reduction (BMR). Every $\Delta T_{\text{BMR}} = 500$ frames we sample up to $n_{\text{pair}} = 2000$ used rMM components, score their mutual expected log-likelihoods with respect to data generated from the model through ancestral sampling, and greedily test merge candidates. A merge is accepted if it *decreases* the expected free energy of the multinomial distributions over reward and next tMM switch, conditioned on the sampled data for the remaining variables; otherwise it is rolled back. BMR enables AXIOM to generalize dynamics from single events, for example learning that negative reward is obtained when a ball hits the bottom of the screen, by merging multiple single event clusters (see Section 3, Figure 4a).

Table 1: **Cumulative reward over 10k steps for `Gameworld 10k` environments.** Cumulative reward is reported as mean ± std over 10 model seeds. Italic means AXIOM is better than BBF and Dreamer, bold is overall best.

Game	AXIOM	BBF	DreamerV3	AXIOM (fixed dist.)		AXIOM (no BMR)		AXIOM (no IG)	
Aviate	-90 ± 19	-90 ± 05	-114 ± 20	-76 ± 13		-87 ± 12		$\mathbf{-71 \pm 16}$	
Bounce	*27 ± 13*	-1 ± 15	14 ± 16	$\mathbf{34 \pm 12}$		8 ± 03	↓	8 ± 19	
Cross	-68 ± 36	-48 ± 07	-27 ± 08	-18 ± 21	↑	-34 ± 25		$\mathbf{-7 \pm 03}$	
Drive	-49 ± 04	-37 ± 06	-45 ± 06	$\mathbf{-22 \pm 04}$	↑	-67 ± 03	↓	-32 ± 02	↑
Explode	*180 ± 30*	101 ± 13	35 ± 59	$\mathbf{234 \pm 16}$	↑	165 ± 14		190 ± 16	
Fruits	*182 ± 21*	86 ± 15	60 ± 07	$\mathbf{209 \pm 19}$		141 ± 19	↓	200 ± 20	
Gold	*190 ± 18*	-26 ± 12	-21 ± 10	189 ± 16		45 ± 15	↓	$\mathbf{207 \pm 17}$	
Hunt	*206 ± 20*	4 ± 12	6 ± 09	$\mathbf{231 \pm 28}$		48 ± 13	↓	216 ± 11	
Impact	*189 ± 45*	122 ± 20	168 ± 83	192 ± 09		$\mathbf{197 \pm 21}$		181 ± 72	
Jump	*−55 ± 09*	-96 ± 17	-55 ± 17	$\mathbf{-38 \pm 25}$		-45 ± 05	↑	-43 ± 26	

2.2 Planning

AXIOM uses active inference for planning [33]; it rolls out future trajectories conditioned on different policies (sequences of actions) and then does inference about policies using the expected free energy, where the chosen policy π^* is that which minimizes the expected free energy:

$$\pi^* = \arg\min_{\pi} \sum_{\tau=0}^{H} -\big(\mathbb{E}_{q(\mathcal{O}_\tau|\pi)}[\underbrace{\log p(r_\tau|\mathcal{O}_\tau,\pi)}_{\text{Utility}} - \underbrace{\mathrm{D}_{KL}(q(\boldsymbol{\alpha}_{\text{rmm}}|\mathcal{O}_\tau,\pi) \parallel q(\boldsymbol{\alpha}_{\text{rmm}}))}_{\text{Information gain (IG)}}]\big) \tag{10}$$

The expected per-timestep utility $\mathbb{E}_{q(\mathcal{O}_\tau|\pi)}[\log p(r_\tau|\mathcal{O}_\tau,\pi)]$ is evaluated using the learned model and slot latents at the time of planning, and accumulated over timesteps into the planning horizon. The expected information gain (second term on RHS of Equation (10)) is computed using the posterior Dirichlet counts of the rMM and scores how much information about rMM switch states would be gained by taking the policy under consideration. More details on planning are given in Appendix A.11.

3 Results

To evaluate AXIOM, we compare its performance on Gameworld against two state-of-the-art baselines on sample-efficient, pixel-based deep reinforcement learning: BBF and DreamerV3.

Benchmark. The Gameworld environments are designed to be solvable by human learners within minutes, ensuring that learning does not hinge on brittle exploration or complex credit assignment. The suite includes 10 diverse games generated with the aid of a large language model, drawing

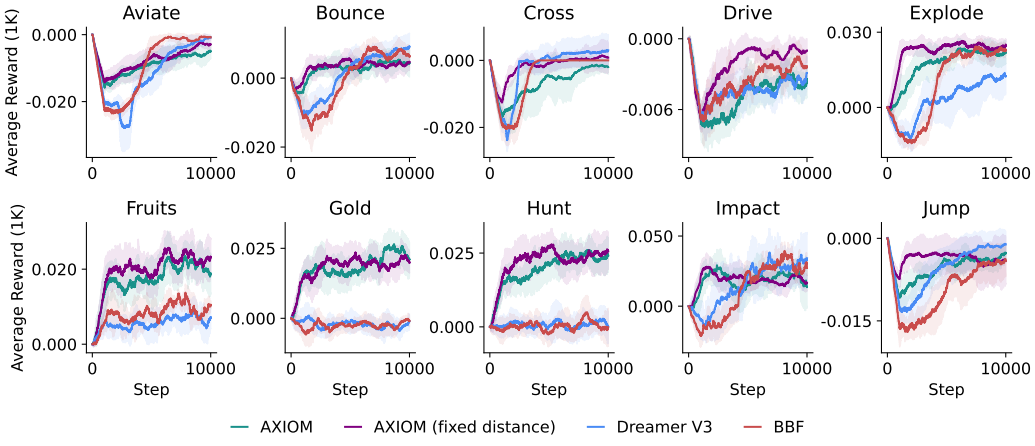


Figure 3: **Online learning performance**. Moving average (1k steps) reward per step during training for AXIOM, BBF and DreamerV3 on `Gameworld 10k` environments. Mean and standard deviation over 10 parameter seeds per model and environment.

inspiration from ALE and classic video games, while maintaining a lightweight and structured design. The Gameworld environments are available at https://github.com/VersesTech/gameworld. Figure 2 illustrates the variety and visual simplicity of the included games. To evaluate robustness, Gameworld 10k supports controlled interventions such as changes in object color or shape, testing an agent's ability to generalize across superficial domain shifts.

Baselines. BBF [34] builds on SR-SPR [35] and represents one of the most sample-efficient model-free approaches. We adapt its preprocessing for the `Gameworld 10k` suite by replacing frame-skip with max-pooling over two consecutive frames; all other published hyperparameters remain unchanged. Second, DreamerV3 [36] is a world-model-based agent with strong performance on games and control tasks with only pixel inputs; we use the published settings but set the train ratio to 1024 at batch size 16 (effective training ratio of 64:1). We chose these baselines because they represent state of the art in sample-efficient learning from raw pixels. Note that for BBF and DreamerV3, we rescale the frames to 84×84 and 96×96 pixels respectively (following the published implementations), whereas AXIOM operates on full 210×160 frames of Gameworld.

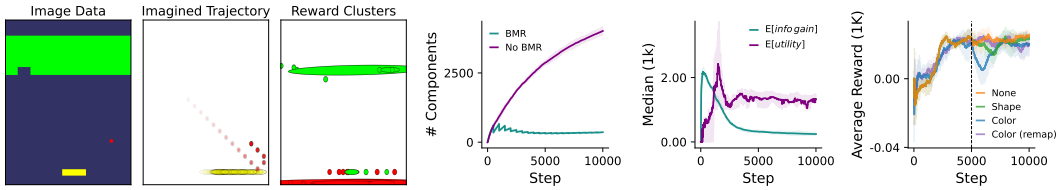
Reward. Figure 3 shows the 1000-step moving average of per-step reward from steps 0 to 10000 on the `Gameworld 10k` suite (mean $\pm$ 1 standard deviation over 10 seeds). Table 1 shows the cumulative reward attained at the end of the 10k interaction steps for AXIOM, BBF and DreamerV3. AXIOM attains higher, or on par, average cumulative reward than BBF and DreamerV3 in every Gameworld environment. Notably, AXIOM not only achieves higher peak scores on several games, but also converges much faster, often reaching most of its final reward within the first 5k steps, whereas BBF and DreamerV3 need nearly the full 10k. For those games where BBF and Dreamer seemed to show no-better-than-random performance at 10k, we confirmed that their performance does eventually improve, ruling out that the games themselves are intrinsically too difficult for these architectures (see Appendix E.1). Taken together, this demonstrates that AXIOM's object-centric world model, in tandem with its fast, online structure learning and inference algorithms, can reduce the number of interactions required to achieve high performance in pixel-based control. Fixing the interaction distance yields higher cumulative reward as the agent doesn't need to spend actions learning it, but doing so requires tuning the interaction distance for each game. This illustrates how having extra knowledge about the domain at hand can be incorporated into a Bayesian model like AXIOM to further improve sample efficiency. Including the information gain term from Equation (10) allows the agent to obtain reward faster in some games (e.g., Bounce), but actually results in a slower increase of the average reward for others (e.g., Gold), as encourages visitation of information-rich but negatively-rewarding states. BMR is crucial for games that need spatial generalization (like Gold and Hunt), but actually hurts performance on Cross, as merging clusters early on discounts the information gain term and discourages exploration. See Appendix E.2 for a more detailed discussion.

Computational costs. Table 2 compares model sizes and per-step training timing (model update and planning) measured on a single A100 GPU. While AXIOM incurs planning overhead due to the use of many model-based rollouts, its model update is substantially more efficient than BBF, yielding favorable trade-offs in wall-clock time per sample. The expanding object-centric model of AXIOM converges to a sufficient complexity given the environment, in contrast to the fixed (and much larger) model sizes of BBF and DreamerV3.

Interpretability. Unlike conventional deep RL methods, AXIOM has a structured, object-centric model whose latent variables and parameters can be directly interpreted in human-readable terms (e.g., shape, color, position). AXIOM's transition mixture model also decomposes complex trajectories into simpler linear sub-sequences. Figure 4a shows imagined trajectories and reward-conditioned clusters

Table 2: **Training time per environment step on `Gameworld 10k`.** Parameter count for AXIOM varies as the model finds a sufficient complexity for each environment. Planning time for AXIOM shows the range for 64 to 512 planning rollouts.

Model	Parameters (M)	Model update (ms/step)	Planning (ms/step)
BBF	6.47	135 ± 36	N/A
DreamerV3	420	221 ± 37	823 ± 93
AXIOM	0.3 - 1.6	18 ± 3	252 - 534



(a) (b) (c) (d)

Figure 4: **Tracking AXIOM's Behavior.** (a) Example frame from Impact at time t (left); imagined trajectory in latent space conditioned on the observation at time t and 32 timesteps into the future, conditioned on an action sequence with high predicted reward (middle); and rMM clusters shown in 2-D space and colored by expected reward (green positive reward, red negative reward aka punishment) (right). (b) Expanding rMM components are pruned over training using Bayesian Model Reduction (BMR) in Explode. (c) Information gain decreases while expected utility increases during training, showing an exploration-exploitation trade-off in Explode. (d) Performance following perturbation at 5k steps shows robustness to changes in game mechanics in Explode.

of the rMM for the Impact game. The imagined trajectories in latent space (middle panel of Figure 4a) are directly readable in terms of the colors and positions of the corresponding object. Because the recurrent mixture model (rMM) conditions switch states on various game- and object-relevant features, we can condition these switch variables on different game features and visualize them to show the rMM's learned associations (e.g., between reward and space). The right-most panel of Figure 4a show the rMM clusters associated with reward (green) and punishment (red) plotted in space. The distribution of these clusters explains AXIOM's beliefs about where in space it expects to encounter rewards, e.g., expecting a punishment when the player misses the ball (red cluster at bottom of the right panel of Figure 4a).

Figure 4b shows the sharp decline in active rMM components during training. By actively merging clusters to minimize the expected free energy associated with the reduced model, Bayesian model reduction (BMR) improves computational efficiency while maintaining or improving performance (see Table 1). The resulting merged components enable interpolation beyond the training data, enhancing generalization. This automatic simplification reveals the minimal set of dynamics necessary for optimal performance, making AXIOM's decision process transparent and robust. Figure 4c demonstrates that, as training progresses, per-step information gain decreases while expected utility rises, reflecting a shift from exploration to exploitation as the world model becomes reliable.

Perturbation Robustness. Finally, we test AXIOM under systematic perturbations of game mechanics. Here, we perform a perturbation to the color or shape of each object at step 5000. Figure 4d shows that AXIOM is resilient to shape perturbations, as it still correctly infers the object type with the iMM. In response to a color perturbation, AXIOM adds new identity types and needs to re-learn their dynamics, resulting in a slight drop in performance and subsequent recovery. Due to the interpretable structure of AXIOM's world model, we can prime it with knowledge about possible color perturbations, and then only use the shape information in the iMM inference step, before remapping the perturbed slots based on shape and rescue performance. For more details, see Appendix E.3.

4 Conclusion

In this work, we introduced AXIOM, a novel and fully Bayesian object-centric agent that learns how to play simple games from raw pixels with improved sample efficiency compared to both model-based and model-free deep RL baselines. Importantly, it does so without relying on neural networks, gradient-based optimization, or replay buffers. By employing mixture models that automatically expand to accommodate environmental complexity, our method demonstrates strong performance within a strict $10,000$-step interaction budget on the `Gameworld 10k` benchmark. Furthermore, AXIOM builds interpretable world models with an order of magnitude fewer parameters than standard models while maintaining competitive performance. To this end, our results suggest that Bayesian methods with structured priors about objects and their interactions have the potential to bridge the gap between the expressiveness of deep RL techniques and the data-efficiency of Bayesian methods with explicit models, suggesting a valuable direction for research.

Limitations and future work. Our work is limited by the fact that the core priors are themselves engineered rather than discovered autonomously. Future work will focus on developing methods to automatically infer such core priors from data, which should allow our approach to be applied to more complex domains like Atari or Minecraft [36], where the underlying generative processes are less transparent but still governed by similar causal principles. We believe this direction represents a crucial step toward building adaptive agents that can rapidly construct structural models of novel environments without explicit engineering of domain-specific knowledge.

Acknowledgements . We would like to thank Jeff Beck, Alex Kiefer, Lancelot Da Costa, and members of the VERSES Machine Learning Foundations and Embodied Intelligence Labs for useful discussions related to the AXIOM architecture.

References

[1] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[2] E. S. Spelke and K. D. Kinzler, "Core knowledge," *Developmental science*, vol. 10, no. 1, pp. 89–96, 2007.

[3] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, p. e253, 2017.

[4] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[5] E. Téglás, E. Vul, V. Girotto, M. Gonzalez, J. B. Tenenbaum, and L. L. Bonatti, "Pure reasoning in 12-month-old infants as probabilistic inference," *science*, vol. 332, no. 6033, pp. 1054–1059, 2011.

[6] E. S. Spelke, R. Kestenbaum, D. J. Simons, and D. Wein, "Spatiotemporal continuity, smoothness of motion and object identity in infancy," *British journal of developmental psychology*, vol. 13, no. 2, pp. 113–142, 1995.

[7] E. S. Spelke, "Principles of object perception," *Cognitive science*, vol. 14, no. 1, pp. 29–56, 1990.

[8] A. M. Leslie and S. Keeble, "Do six-month-old infants perceive causality?," *Cognition*, vol. 25, no. 3, pp. 265–288, 1987.

[9] T. Wiedemer, J. Brady, A. Panfilov, A. Juhos, M. Bethge, and W. Brendel, "Provable compositional generalization for object-centric learning," *arXiv preprint arXiv:2310.05327*, 2023.

[10] F. Kapl, A. M. K. Mamaghan, M. Horn, C. Marr, S. Bauer, and A. Dittadi, "Object-centric representations generalize better compositionally with less compute," in *ICLR 2025 Workshop on World Models: Understanding, Modelling and Scaling*, 2025.

[11] W. Agnew and P. Domingos, "Unsupervised object-level deep reinforcement learning," in *NeurIPS workshop on deep RL*, 2018.

[12] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *International conference on machine learning*, pp. 2688–2697, Pmlr, 2018.

[13] A. Lei, B. Schölkopf, and I. Posner, "Spartan: A sparse transformer learning local causation," *arXiv preprint arXiv:2411.06890*, 2024.

[14] W. Zhang, A. Jelley, T. McInroe, and A. Storkey, "Objects matter: object-centric world models improve reinforcement learning in visually complex environments," *arXiv preprint arXiv:2501.16443*, 2025.

[15] T. Parr, G. Pezzulo, and K. J. Friston, *Active inference: the free energy principle in mind, brain, and behavior*. MIT Press, 2022.

[16] K. Friston, "The free-energy principle: a unified brain theory?," *Nature reviews neuroscience*, vol. 11, no. 2, pp. 127–138, 2010.

[17] D. C. Knill and A. Pouget, "The bayesian brain: the role of uncertainty in neural coding and computation," *TRENDS in Neurosciences*, vol. 27, no. 12, pp. 712–719, 2004.

[18] F. Locatello, D. Weissenborn, and O. Unsupervised, "Object-centric learning with slot attention," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1821–1834, 2020.

[19] S. W. Linderman, A. C. Miller, R. P. Adams, D. M. Blei, L. Paninski, and M. J. Johnson, "Recurrent switching linear dynamical systems," *arXiv preprint arXiv:1610.08466*, 2016.

[20] C. Heins, H. Wu, D. Markovic, A. Tschantz, J. Beck, and C. Buckley, "Gradient-free variational learning with conditional mixture networks," *arXiv preprint arXiv:2408.16429*, 2024.

[21] K. J. Friston, V. Litvak, A. Oswal, A. Razi, K. E. Stephan, B. C. Van Wijk, G. Ziegler, and P. Zeidman, "Bayesian model reduction and empirical bayes for group (dcm) studies," *Neuroimage*, vol. 128, pp. 413–431, 2016.

[22] K. Friston, T. Parr, and P. Zeidman, "Bayesian model reduction," *arXiv preprint arXiv:1805.07092*, 2018.

[23] K. Friston, C. Heins, T. Verbelen, L. Da Costa, T. Salvatori, D. Markovic, A. Tschantz, M. Koudahl, C. Buckley, and T. Parr, "From pixels to planning: scale-free active inference," *arXiv preprint arXiv:2407.20292*, 2024.

[24] K. J. Friston, L. Da Costa, A. Tschantz, A. Kiefer, T. Salvatori, V. Neacsu, M. Koudahl, C. Heins, N. Sajid, D. Markovic, *et al.*, "Supervised structure learning," *Biological Psychology*, vol. 193, p. 108891, 2024.

[25] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.

[26] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[27] H. Ishwaran and L. F. James, "Gibbs sampling methods for stick-breaking priors," *Journal of the American statistical Association*, vol. 96, no. 453, pp. 161–173, 2001.

[28] J. Beck and M. J. Ramstead, "Dynamic markov blanket detection for macroscopic physics discovery," *arXiv preprint arXiv:2502.21217*, 2025.

[29] Z. Ghahramani and G. E. Hinton, "Switching state-space models," *University of Toronto Technical Report CRG-TR-96-3, Department of Computer Science*, 1996.

[30] C. Bishop and J. Lasserre, "Generative or discriminative? getting the best of both worlds," *Bayesian statistics*, vol. 8, no. 3, pp. 3–24, 2007.

[31] M. J. Wainwright, M. I. Jordan, *et al.*, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[32] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *the Journal of machine Learning research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[33] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, and G. Pezzulo, "Active inference: a process theory," *Neural computation*, vol. 29, no. 1, pp. 1–49, 2017.

[34] M. Schwarzer, J. S. O. Ceron, A. Courville, M. G. Bellemare, R. Agarwal, and P. S. Castro, "Bigger, better, faster: Human-level atari with human-level efficiency," in *International Conference on Machine Learning*, pp. 30365–30380, PMLR, 2023.

[35] P. D'Oro, M. Schwarzer, E. Nikishin, P.-L. Bacon, M. G. Bellemare, and A. Courville, "Sample-efficient reinforcement learning by breaking the replay ratio barrier," in *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

[36] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse control tasks through world models," *Nature*, pp. 1–7, 2025.

[37] T. P. Minka, "Expectation propagation for approximate bayesian inference," *arXiv preprint arXiv:1301.2294*, 2013.

[38] M. Okada and T. Taniguchi, "Variational inference mpc for bayesian model-based reinforcement learning," in *Conference on Robot Learning*, 2019.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[40] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao, "Mastering atari games with limited data," *Advances in neural information processing systems*, vol. 34, pp. 25476–25488, 2021.

[41] S. Wang, S. Liu, W. Ye, J. You, and Y. Gao, "Efficientzero v2: Mastering discrete and continuous control with limited data," *arXiv preprint arXiv:2403.00564*, 2024.

[42] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

[43] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," *arXiv preprint arXiv:2010.02193*, 2020.

[44] K. Greff, R. L. Kaufman, R. Kabra, N. Watters, C. Burgess, D. Zoran, L. Matthey, M. Botvinick, and A. Lerchner, "Multi-object representation learning with iterative variational inference," in *International conference on machine learning*, pp. 2424–2433, PMLR, 2019.

[45] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf, "Object-centric learning with slot attention," *Advances in neural information processing systems*, vol. 33, pp. 11525–11538, 2020.

[46] R. Singh and C. L. Buckley, "Attention as implicit structural inference," *Advances in Neural Information Processing Systems*, vol. 36, pp. 24929–24946, 2023.

[47] D. Kirilenko, V. Vorobyov, A. K. Kovalev, and A. I. Panov, "Object-centric learning with slot mixture module," *arXiv preprint arXiv:2311.04640*, 2023.

[48] J. Jiang, F. Deng, G. Singh, M. Lee, and S. Ahn, "Slot state space models," *arXiv preprint arXiv:2406.12272*, 2024.

[49] S. Ferraro, P. Mazzaglia, T. Verbelen, and B. Dhoedt, "Focus: Object-centric world models for robotic manipulation," *Frontiers in Neurorobotics*, vol. 19, p. 1585386, 2025.

[50] J. Collu, R. Majellaro, A. Plaat, and T. M. Moerland, "Slot structured world models," *arXiv preprint arXiv:2402.03326*, 2024.

[51] C. Rasmussen, "The infinite gaussian mixture model," *Advances in neural information processing systems*, vol. 12, 1999.

[52] T. Champion, M. Grześ, and H. Bowman, "Structure learning with temporal gaussian mixture for model-based reinforcement learning," *arXiv preprint arXiv:2411.11511*, 2024.

[53] Z. Ghahramani and G. E. Hinton, "Variational learning for switching state-space models," *Neural computation*, vol. 12, no. 4, pp. 831–864, 2000.

[54] V. Geadah, J. W. Pillow, *et al.*, "Parsing neural dynamics with infinite recurrent switching linear dynamical systems," in *The Twelfth International Conference on Learning Representations*, 2024.

[55] S. Linderman, M. J. Johnson, and R. P. Adams, "Dependent multinomial models made easy: Stick-breaking with the pólya-gamma augmentation," *Advances in neural information processing systems*, vol. 28, 2015.

A Full Model Details

AXIOM's world model is a hidden Markov model with an object-centric latent state space. The model itself has two main components: 1) an object-centric, slot-attention-like [18] likelihood model; and 2) a recurrent switching state space model [19]. The recurrent switching state space model is applied to each object or slot identified by the likelihood model, and models the dynamics of each object with piecewise-linear trajectories. Unlike most other latent state-space models, including other object-centric ones, AXIOM is further distinguished by its adaptable complexity – it grows and prunes its model online through iterative expansion routines (see Algorithm 1) and reduction (see Algorithm 2) to match the structure of the world it's interacting with. This includes automatically inferring the number of objects in the scene as well as the number of dynamical modes needed to describe the motion of all objects. This is inspired by the recent *fast structure learning* approach [23] developed to automatically learn a hierarchical generative model of a dataset from scratch.

Preface on notation Capital bold symbols denote collections of matrix- or vector-valued random variables and lowercase bold symbols denote multivariate variables.

A.1 Generative model

The model factorizes perception and dynamics into separate generative blocks: (i) In perception, a slot Mixture Model (sMM) models pixels $\mathbf{y}_t$ as competitively explained by continuous latent variables factorized across slots or objects: $\mathbf{x}_t = \{\mathbf{x}_t^{(1)}, \ldots, \mathbf{x}_t^{(K)}\}$. Each pixel is assigned to one of (up to) K slots using the assignment variable $\mathbf{z}_{t,\text{smm}}$; (ii) dynamics are modeled per-object using their object-centric latent descriptions as inputs to a recurrent switching state space model (similar to an rSLDS [19]). We define the full latent sequence as $\mathcal{Z}_{0:T} = \{\mathcal{O}_t, \mathbf{z}_{t,\text{smm}}\}_{t=0}^{T}$. Each slot latent $\mathcal{O}_t^{(k)}$ consists of both continuous $\mathbf{x}_t^{(k)}$ and discrete latent variables. The continuous latents represent continuous properties of an object, such as its position, color and shape. The discrete latents are themselves split into two subtypes: $\mathbf{z}_t^{(k)}$ and $\mathbf{s}_t^{(k)}$. We use $\mathbf{z}_t^{(k)}$ to denote four latent descriptors that capture categorical attributes of the slot (e.g., object type), and $\mathbf{s}_t^{(k)}$ to denote a pair of switch states determining the slot's instantaneous trajectory.[3] Model parameters $\tilde{\mathbf{\Theta}}$ are split into module-specific subsets (e.g., $\mathbf{\Theta}_{\text{sMM}}, \mathbf{\Theta}_{\text{tMM}}$). The joint distribution over input sequences $\mathbf{y}_{0:T}$, latent state sequences $\mathcal{Z}_{0:T}$ and parameters $\tilde{\mathbf{\Theta}}$ can be expressed as a hidden Markov model:

$$p(\mathbf{y}_{0:T}, \mathcal{Z}_{0:T}, \tilde{\mathbf{\Theta}}) = p(y_0, \mathcal{Z}_0)p(\tilde{\mathbf{\Theta}}) \prod_{t=1}^{T} \underbrace{p(\mathbf{x}_{t-1}, \mid \mathbf{z}_t, \mathbf{\Theta}_{\text{iMM}})}_{\text{Identity mixture model}} \underbrace{p(\mathbf{x}_{t-1}, \mathbf{z}_t, \mathbf{s}_t, a_{t-1}, r_t \mid \mathbf{\Theta}_{\text{rMM}})}_{\text{Recurrent mixture model}}$$
$$\prod_{k=1}^{K} \underbrace{p(\mathbf{y}_t | \mathbf{x}_t^{(k)}, \mathbf{z}_{t,\text{smm}}, \mathbf{\Theta}_{\text{sMM}})}_{\text{Slot mixture model}} \underbrace{p(\mathbf{x}_t^{(k)} \mid \mathbf{x}_{t-1}^{(k)}, \mathbf{s}_t^{(k)}, \mathbf{\Theta}_{\text{tMM}})}_{\text{Transition mixture model}}, \quad (11)$$

We intentionally exclude the slot mixture assignment variable $\mathbf{z}_{t,\text{smm}}$ from the other object-centric discrete latents $\{\mathbf{z}_t^{(k)}\}_{k=1}^{K}$ because the sMM assignment variable is importantly not factorized over slots, since it is a categorical distribution over K-dimensional one-hot vectors, $\mathbf{z}_{t,\text{smm}} \in \{0,1\}^K$.

Latent object states. At a given time–step $t \in 0, \ldots, T$ each object $k \in 1, \ldots, K$ is described by sets of both continuous and discrete variables (we reserve k for 'slot index' everywhere):

$$\mathcal{O}_t = \{\mathbf{x}_t^{(k)}, \mathbf{z}_t^{(k)}, \mathbf{s}_t^{(k)}\}_{k=1}^{K},$$

The continuous state $\mathbf{x}_t^{(k)}$ summarize latent features or descriptors associated with the k^{th} object, including its 2-D position $\mathbf{p}_t^{(k)} = \{p_{t,x}^{(k)}, p_{t,y}^{(k)}\}$, a corresponding 2-D velocity $\mathbf{v}_t^{(k)} = \{v_{t,x}^{(k)}, v_{t,y}^{(k)}\}$, its color $\mathbf{c}_t^{(k)} = \{c_{t,r}^{(k)}, c_{t,g}^{(k)}, c_{t,b}^{(k)}\}$ its 2-D shape encoded as its extent along the X and Y directions

[3] We use the superscript index k as in $q^{(k)}$ to select only the subset of $q \equiv q^{(1:K)}$ relevant to the k^{th} slot.

$\mathbf{e}_t^{(k)} = \{e_{t,x}^{(k)}, e_{t,y}^{(k)}\}$, and an 'unused counter' $u_t^{(k)}$, which tracks how long the k^{th} object has gone undetected:

$$\mathbf{x}_t^{(k)} = \left[\mathbf{p}_t^{(k)}, \mathbf{c}_t^{(k)}, \mathbf{v}_t^{(k)}, u_t^{(k)}, \mathbf{e}_t^{(k)}\right]^\top \in \mathbb{R}^{10}.$$

In addition to the continuous latents, each object is also characterized by two sets of discrete variables: $\mathbf{z}_t^{(k)}$ and $\mathbf{s}_t^{(k)}$. The first set $\mathbf{z}_t^{(k)}$ captures categorical information about the object that is relevant to predicting its instantaneous dynamics. This includes a latent object 'type' $\mathbf{z}_{t,\text{type}}^{(k)}$ (used to identify dynamics across object instances based on their shared continuous properties, e.g., objects that have the same shape and color are expected to behave similarly); the object type index of the nearest other object that slot k is interacting with (or if it isn't interacting with anything) $\mathbf{z}_{t,\text{interacting}}^{(k)}$, its presence/absence in the current frame $\mathbf{z}_{t,\text{presence}}^{(k)}$, and whether the object is moving or not $\mathbf{z}_{t,\text{moving}}^{(k)}$. We define each of these discrete variables in 'one-hot' vector format, so as vectors whose entries $z_{t,m,\text{name}}^{(k)}$ are either 0 or 1, with the constraint that $\sum_m z_{t,m,\text{name}}^{(k)} = 1$. We can thus write the full discrete $\mathbf{z}_t^{(k)}$ latent as follows:

$$\begin{aligned}
\mathbf{z}_t^{(k)} &= \left[\mathbf{z}_{t,\text{type}}^{(k)}, \mathbf{z}_{t,\text{interacting}}^{(k)}, \mathbf{z}_{t,\text{presence}}^{(k)}, \mathbf{z}_{t,\text{moving}}^{(k)}\right] \\
&\in \{0,1\}^{\mathcal{C}_{\text{type}}+\mathcal{C}_{\text{interacting}}+\mathcal{C}_{\text{presence}}+\mathcal{C}_{\text{moving}}} \\
&\text{with} \quad \begin{cases} \mathcal{C}_{\text{type}} = V \leq V_{\max} \\ \mathcal{C}_{\text{interacting}} = V + 1 \\ \mathcal{C}_{\text{presence}} = 2 \\ \mathcal{C}_{\text{moving}} = 2 \end{cases}
\end{aligned}$$

where V is the number of object types inferred by the identity mixture model or iMM, subject to a maximum value of $V_{\max}$ (see Appendix A.6). Note that $\mathcal{C}_{\text{interacting}}$ has maximum index $V+1$ because it includes an extra index for the state of 'not interacting with any other object.'

The second set of discrete variables $\mathbf{s}_t^{(k)}$ form a pair of concurrent switch states that jointly select one of an expanding set of linear dynamical systems to predict the object's future motion:

$$\begin{aligned}
\mathbf{s}_t^{(k)} &= \left[\mathbf{s}_{t,\text{tmm}}^{(k)}, \mathbf{s}_{t,\text{rmm}}^{(k)}\right] \\
&\in \{0,1\}^{\mathcal{S}_{\text{tmm}}+\mathcal{S}_{\text{rmm}}} \\
&\text{with} \quad \begin{cases} \mathcal{S}_{\text{tmm}} \leq L \\ \mathcal{S}_{\text{rmm}} \leq M \end{cases}
\end{aligned}$$

The first variable $\mathbf{s}_{t,\text{tmm}}^{(k)}$ is the switching variable of the transition mixture model or tMM, whereas the second switch variable $\mathbf{s}_{t,\text{rmm}}^{(k)}$ is the assignment variable of another mixture model – the recurrent mixture model or rMM – which furnishes a likelihood over $\mathbf{s}_{\text{tmm}}$ as well as other continuous and discrete features of each object.

In the sections that follow we will detail each of the components in the full AXIOM world model and give their descriptions in terms of generative models. These descriptions will show how the latent variables $\mathcal{Z}_{0:T} = \{\mathcal{O}_t, \mathbf{z}_{t,\text{smm}}\}_{t=0}^T$ and component-specific parameters (e.g, $\boldsymbol{\Theta}_{\text{rMM}}$ relate to observations and other latent variables.

A.2 Slot Mixture Model (sMM)

AXIOM processes sequences of RGB images one frame at a time. Each image is composed of $H \times W$ pixels and is reshaped into $N = HW$ tokens $\{\mathbf{y}_t^n\}_{n=1}^N$. Each token $\mathbf{y}_t^n$ is a vector containing the n^{th} pixel's color in RGB and its image coordinates (normalized to $[-1, +1]$). AXIOM models these tokens at a given time as explained by a mixture of the continuous slot latents; we term this likelihood construction the *Slot Mixture Model* (sMM, see far left side of Figure 1). The K components of this

mixture model are Gaussian distributions whose parameters are directly given by the continuous features of each slot latent $\mathbf{x}_t^{(1:K)}$. Associated to this Gaussian mixture is a binary assignment variable $z^n_{t,k,\text{smm}} \in \{0,1\}$ indicating whether pixel n at time t is driven by slot k, with the constraint that $\sum_k z^n_{t,k,\text{smm}} = 1$. The sMM's likelihood model for a single pixel and timepoint $\mathbf{y}_t^n$ can be expressed as follows (dropping the t subscript for notational clarity):

$$p(\mathbf{y}^n \mid \mathbf{x}^{(1:K)}, \sigma_{\mathbf{c}}^{(1:K)}, \mathbf{z}^n_{\text{smm}}) = \prod_{k=1}^{K} \mathcal{N}(A\mathbf{x}^{(k)}, \text{diag}(\left[B\mathbf{x}^{(k)}, \sigma_{\mathbf{c}}^{(k)}\right]^\top))^{z^n_{k,\text{smm}}} \tag{12}$$

$$A = \begin{bmatrix} I_5 & \mathbf{0}_{5\times 5} \end{bmatrix}, \quad B = \begin{bmatrix} \mathbf{0}_{2\times 8} & I_2 \end{bmatrix} \tag{13}$$

$$p(\mathbf{z}^n_{\text{smm}} \mid \boldsymbol{\pi}_{\text{smm}}) = \text{Cat}(\boldsymbol{\pi}_{\text{smm}}),\ p(\boldsymbol{\pi}_{\text{smm}}) = \text{Dir}(\underbrace{1, \ldots, 1}_{K-1 \text{ times}}, \alpha_{0,\text{smm}}) \tag{14}$$

$$p(\sigma_{\mathbf{c}}^{(k)}) = \prod_{j \in \text{R, G, B}} \Gamma(\gamma_{0,j}, 1) \tag{15}$$

The mean of each Gaussian component is given a fixed linear projection A of each object latent, which selects only its position and color features: $A\mathbf{x}^{(k)} = \left[\mathbf{p}^{(k)}, \mathbf{c}^{(k)}\right]$. The covariance of each component is a diagonal matrix whose diagonal is a projection of the 2-D shape of the object latent $B\mathbf{x}^{(k)} = \mathbf{e}^{(k)}$ (its spatial extent in the X and Y directions), stacked on top of a fixed variance for each color dimension $\sigma_{\mathbf{c}}^{(k)}$, which are given independent Gamma priors. The latent variables $\mathbf{p}^{(k)}, \mathbf{c}^{(k)}, \mathbf{e}^{(k)}$ are subsets of slot k's full continuous features $\mathbf{x}_t^{(k)}$, and the projection matrices A, B remain fixed and unlearnable. Each token's slot indicator $\mathbf{z}^n_{\text{smm}}$ is drawn from a Categorical distribution with mixing weights $\boldsymbol{\pi}_{\text{smm}}$. We place a truncated stick-breaking (finite GEM) prior on these weights, which is equivalent to a K-dimensional Dirichlet with concentration vector $(1, \ldots, 1, \alpha_{0,\text{smm}})$, where the first $K-1$ pseudocounts are 1 and the final pseudocount $\alpha_{0,\text{smm}}$ reflects the propensity to add new slots. All subsequent mixture models in AXIOM are equipped with the same sort of truncated stick-breaking priors on the mixing weights [27]. We can collect parameters of the sMM together into $\boldsymbol{\Theta}_{\text{sMM}} = \{\boldsymbol{\pi}_{\text{smm}}, \sigma_{\mathbf{c}}^{(1:K)}, A, B\}$. The priors over these parameters can be written as a product of the truncated stick-breaking prior and the Gamma priors for each color variance, and the priors over A and B can be thought of as Dirac delta functions, rendering them unlearnable.

A.3 Moving and presence latent variables

Within the discrete variables that describe each slot $\mathbf{z}_t^{(k)}, \mathbf{s}_t^{(k)}$, there are two one-hot-encoded Bernoulli variables: $\mathbf{z}^{(k)}_{t,\text{moving}} \in \{0,1\}^2$ and $\mathbf{z}^{(k)}_{t,\text{present}} \in \{0,1\}^2$. These represent whether the object associated to the k^{th} slot is moving and present on the screen, respectively. These variables have a particular relationship to the generative model, particularly the dynamics model, which allows inference over them to act as a 'pre-processing step' or filter for slot latents, before passing their continuous features further down the inference chain for use by the identity model, recurrent mixture model, and transition model. The way this gating is functionally defined is detailed in the subsection **Gated dynamics learning** below.

The latent state sequences $\mathbf{z}^{(k)}_{0:T,\text{presence}}$ and $\mathbf{z}^{(k)}_{0:T,\text{moving}}$ variables are modelled as evolving according to discrete, object-factorized markov chains which concurrently emit observations via a proxy presence variable $o_t^{(k)}$ (see below for details on how this is computed). This conditional hidden Markov model can be written as follows for the two variables:

$$p(\mathbf{z}^{(k)}_{0:T,\text{presence}}) = \prod_{t=0}^{T} p(o_t^{(k)} \mid \mathbf{z}^{(k)}_{t,\text{presence}}) p(\mathbf{z}^{(k)}_{t,\text{presence}}) \mid \mathbf{z}^{(k)}_{t-1,\text{presence}}, \theta_{\text{presence}})$$

$$p(\mathbf{z}^{(k)}_{0:T,\text{moving}}) = \prod_{t=1}^{T} p(\mathbf{z}^{(k)}_{t,\text{moving}} \mid \mathbf{z}^{(k)}_{t-1,\text{moving}}, o_t^{(k)}, \mathbf{v}^{(k)}_{t-1}, \theta_{\text{moving}}) \tag{16}$$

Presence latent $\mathbf{z}_{t,\text{presence}}$ The presence chain uses a *time-homogeneous* 2×2 transition matrix parametrised by

$$\theta_{\text{presence}} = \{\phi_{\text{NP}\to\text{P}},\ \phi_{\text{P}\to\text{NP}}\}, \qquad 0 \le \phi_{\text{NP}\to\text{P}}, \phi_{\text{P}\to\text{NP}} \le 1.$$

where the subscripts for the 'not present' and 'present' states of $\mathbf{z}^{(k)}_{t,\text{presence}}$ are abbreviated as NP and P, respectively. Writing $\phi_{i\to\text{not present}} = 1 - \phi_{i\to\text{present}}$, the transition matrix is

$$T_{\text{presence}} = \begin{pmatrix} \phi_{\text{NP}\to\text{NP}} & \phi_{\text{NP}\to\text{P}} \\ \phi_{1\to\text{NP}} & \phi_{\text{P}\to\text{P}} \end{pmatrix} = \begin{pmatrix} 1-\phi_{\text{NP}\to\text{P}} & \phi_{\text{NP}\to\text{P}} \\ \phi_{\text{P}\to\text{NP}} & 1-\phi_{\text{P}\to\text{NP}} \end{pmatrix}.$$

For all experiments we *fix* $\phi_{0\to1} = 0,\ \phi_{1\to0} = 0.01$, encoding the prior that an absent slot cannot spontaneously re-appear while a present one "dies out" with probability 0.01 each frame.

Proxy observation. We define the assignment-count indicator $o_t^{(k)}$ as a variable that indicates whether any pixels $1, 2, \ldots, N$ were assigned to slot k at time t. This can be expressed as an element-wise product of all pixel-specific entries of the row of sMM assignment variable corresponding to slot k's assignments: $\mathbf{z}^{(1:N)}_{t,k,\text{smm}}$:

$$o_t^{(k)} = \prod_{n=1}^{N} z^n_{t,k,\text{smm}}$$

Finally, the relationship between the count-assignments-indicator $o_t^{(k)}$ and the presence variable $\mathbf{z}_{t,\text{presence}}$ is a Bernoulli likelihood with the following form:

$$p(o_t|\mathbf{z}_{t,\text{presence}}) = o_t^{z_{t,P,\text{presence}}}(1-o_t)^{z_{t,A,\text{presence}}} \tag{17}$$

which means that presence and the assignment count $o_t^{(k)}$ are expected to be 'on' simultaneously.

The subscripts P and A refer to the indices of $\mathbf{z}^{(k)}_{t,\text{presence}}$ that correspond to the 'present' and 'absent' indicators, respectively.

Moving latent $\mathbf{z}_{t,\text{moving}}$ We define the speed of slot k as follows (leaving out the k superscript to avoid overloaded superscripts):

$$\psi_t = \sqrt{v_{t,x}^2 + v_{t,y}^2} \tag{18}$$

The transition likelihood for the moving latent $\mathbf{z}^{(k)}_{t,\text{moving}}$ depends on the presence indicator $o_t^{(k)}$ and the slot speed ψ_{t-1}; this forces inference of the moving indicator $\mathbf{z}^{(k)}_{t,\text{moving}}$ to be driven by the inferred speed and presence of the k-th slot. The form of this dependence is encoded in the 2×2 transition matrix T_{moving} with parameters θ_{moving} as follows:

$$\theta_{\text{moving}} = \{\lambda, \beta\}, \qquad 0 \le \lambda \le 1,\ 0 \le \beta \le 1,\ \lambda + \beta \le 1. \tag{19}$$

The parameters λ and β determine the two conditional probabilities $\phi_{i\to\text{M}}(o_t^{(k)}, \psi_{t-1})$ and $\phi_{i\to\text{NM}}(o_t^{(k)}, \psi_{t-1})$. We use the subscripts NM, M to indicate the 'not-moving' and 'moving' states of $\mathbf{z}^{(k)}_{t,\text{moving}}$, respectively. The dependence of the conditional probabilities on the λ, β hyperparameters can be written as follows:

$$\phi_{i\to\text{M}}(o_t^{(k)}, \psi_{t-1}) = \begin{cases} \lambda\, i + \beta\, \psi_{t-1}, & o_t^{(k)} = 1, \\ i, & o_t^{(k)} = 0, \end{cases} \qquad \phi_{i\to\text{NM}}(o_t^{(k)}, \psi_{t-1}) = 1 - \phi_{i\to1}(o_t^{(k)}, \psi_{t-1}).$$

The full transition matrix T_{moving} can be written:

$$T_{\text{moving}}\big(o_t^{(k)}, \psi_{t-1}; \theta_{\text{moving}}\big) = \begin{pmatrix} \phi_{\text{NM}\to\text{NM}} & \phi_{\text{NM}\to\text{M}} \\ \phi_{\text{M}\to\text{NM}} & \phi_{\text{M}\to\text{M}} \end{pmatrix}. \tag{20}$$

This sort of parameterization results in the following interpretation: if the slot is inferred to be absent (i.e., no pixels are assigned to it and $o_t^{(k)} = 0$), $\mathbf{z}_{t,\text{moving}}^{(k)}$ stays in its previous state. However, if the slot is inferred to be present ($o_t^{(k)} = 1$), then the previous "moving" probability is shrunk by λ and nudged upward by $\beta\psi_{t-1}$.

In all experiments we set $\lambda = 0.99,\ \beta = 0.01$, but they remain exposed hyperparameters.

Gated dynamics learning The presence and moving latents $\mathbf{z}_{t,\text{presence}}, \mathbf{z}_{t,\text{moving}}$ exist in order to filter which slots get fit by the rMM. In order to achieve this selective routing of only active, moving slots, we introduce an auxiliary gate variable that is connected to the moving- and presence-latents via a multiplication factor that parameterizes a Bernoulli likelihood over the two values ('ON' and 'OFF') of the gate variable $\mathcal{G}_t^{(k)}$:

$$p(\mathcal{G}_t^{(k)} \mid \mathbf{z}_{t,\text{moving}}^{(k)} \mathbf{z}_{t,\text{present}}^{(k)}) = \text{Bernoulli}\big(p_{\text{gate}}\big) \qquad (21)$$
$$\text{where} \quad p_{\text{gate}} = z_{t,\text{M,moving}}^{(k)} z_{t,\text{P,present}}^{(k)}$$

This binary gate variable then modulates the input precision of the various likelihoods associated with the identity model (iMM), transition mixture model (tMM), and recurrent mixture model (rMM) to effectively 'mask' the learning of these models on untracked or absent slots. The end effect is that slots which are inferred to be *moving* and *present* keep full precision, while any other combination deflates the slot-specific input covariance to 0, removing the influence of their sufficient statistics from parameter learning.

A.4 Interaction variable

We also associate each object with a discrete latent variable $\mathbf{z}_{t,\text{interacting}}^{(k)}$ which indicates the type of the closest object interacting with the focal object (i.e., that indexed by k). In practice, we infer this interaction variable by finding the object whose position variable is closest to the focal slot, i.e. $\arg\min_{j \in \text{nearest}} \|\mathbf{p}_j - \mathbf{p}_l\|$, within some constrained set of 'nearest' objects whose position latents are within a predefined interaction radius of the focal object (determined by a fixed parameter r_{min}). This interaction radius can be tuned on a game specific basis – see the main text results in Section 3 for how fixing this parameter affects the results. We then perform inference on the identity model using the continuous features of that nearest-interacting slot: $\left[\mathbf{c}_t^{(j)}, \mathbf{e}_t^{(j)}\right]^\top$. The inferred identity of the resulting j^{th} slot is then converted into a one-hot vector representing the type of the 'interacting' latent $\mathbf{z}_{t,\text{interacting}}^{(k)}$, which can then be fed as input into the recurrent mixture model or rMM as described in the following section.

A.5 Unused counter

To keep track of how long a slot has remained inactive we introduce a non-negative-integer latent that is treated as a continuous variable in $\mathbb{R}$: $u_t^{(k)}$ or the 'unused counter'. This allows the model to predict the respawning of objects after they go off-screen. We couple the unused counter again to the proxy assignment-count variable $o_t^{(k)}$ using an exponentially–decaying Bernoulli likelihood (identical in spirit to the EP–style presence likelihood):

$$P\big(o_t^{(k)} = 1 \mid u_t^{(k)}\big) = 1 - \exp\big\{-\xi\, e^{-\gamma_u\, u_t^{(k)}}\big\},$$
$$P\big(o_t^{(k)} = 0 \mid u_t^{(k)}\big) = \exp\big\{-\xi\, e^{-\gamma_u\, u_t^{(k)}}\big\}, \qquad \xi \in (0,1],\ \gamma_u > 0. \qquad (22)$$

When $u_t^{(k)} = 0$ the slot is present with probability $1 - e^{-\xi} \simeq \xi$ (for typical $\xi \gtrsim 0.8$). Each increment by ν_u multiplies that probability by $\exp\big(-\xi\, \gamma_u\, \nu_u\big)$, i.e. it decays roughly one e-fold per unused step when $\gamma_u \simeq \nu_u^{-1}$.

A.6 Identity mixture model

AXIOM uses an *identity mixture model* (iMM) to infer a discrete identity code $\mathbf{z}_{\text{type}}^{(k)}$ for each object based on its continuous features. These identity codes are used to condition the inference of the recurrent mixture model used for dynamics prediction. Conditioning the dynamics on identity-codes in this way, rather than learning a separate dynamics model for each slot, allows AXIOM to use the same dynamics model across slots. This also enables the model to learn the same dynamics in a *type*-specific, rather than *instance*-specific, manner, and to remap identities when e.g., the environment is perturbed and colors change. Concretely, the iMM models the 5-D colors and shapes $\{\mathbf{c}^{(k)}, \mathbf{e}^{(k)}\}_{k=1}^{K}$ across slots as a mixture of up to V Gaussian components (object types). The slot-level assignment variable $\mathbf{z}_{t,\text{type}}^{(k)}$ indicates which identity is assigned to the k^{th}slot. The generative model for the iMM is (omitting the $t-1$ subscript from object latents):

$$p(\left[\mathbf{c}^{(k)}, \mathbf{e}^{(k)}\right]^{\top} | \mathbf{z}_{\text{type}}^{(k)}, \boldsymbol{\mu}_{1:V,\text{type}}, \boldsymbol{\Sigma}_{1:V,\text{type}}) = \prod_{j=1}^{V} \mathcal{N}(\boldsymbol{\mu}_{j,\text{type}}, \left(\mathcal{G}_t^{(k)}\right)^{-1} \boldsymbol{\Sigma}_{j,\text{type}})^{z_{j,\text{type}}^{(k)}} \tag{23}$$

$$p(\boldsymbol{\mu}_{j,\text{type}}, \boldsymbol{\Sigma}_{j,\text{type}}^{-1}) = \text{NIW}(\mathbf{m}_{0,j,\text{type}}, \kappa_{0,j,\text{type}}, \mathbf{U}_{0,j,\text{type}}, n_{0,j,\text{type}}) \tag{24}$$

$$p(\mathbf{z}_{\text{type}}^{(k)} \mid \boldsymbol{\pi}_{\text{type}}) = \text{Cat}(\boldsymbol{\pi}_{\text{type}}),\ p(\boldsymbol{\pi}_{\text{type}}) = \text{Dir}(\underbrace{1, \ldots, 1}_{V-1 \text{ times}}, \alpha_{0,\text{type}}) \tag{25}$$

The $\left(\mathcal{G}_t^{(k)}\right)^{-1} \mathbf{X}$ notation represents element-wise broadcasting of the reciprocal of $\mathcal{G}_t^{(k)}$ across all elements of the matrix or vector $\mathbf{X}$. When applied as a mask to a covariance matrix, for instance, the effect is that slots that are inferred to be moving and present do not have their covariance affected $\left(\mathcal{G}_t^{(k)}\right)^{-1} = 1$, whereas those slots that are inferred to be either not present or not moving will 'inflate' the covariance of the mixture model, due to $\left(\mathcal{G}_t^{(k)}\right)^{-1} \to \infty$. The same type of Categorical likelihood for the type assignments and truncated stick-breaking prior over the mixture weights is used to allow an arbitrary (up to a maximum of V) number of types to be used to explain the continuous slot features. We equip the prior over the component likelihood parameters with conjugate Normal Inverse Wishart (NIW) priors.

A.7 Transition Mixture Model

The dynamics of each slot are modelled as a mixture of linear functions of the slot's own previous state. To stress the homology between this model and the other modules of AXIOM, we refer to this module as the *transition mixture model* or tMM, but this formulation is more commonly also known as a switching linear dynamical system or SLDS [29]. The tMM's switch variable $\mathbf{s}_{t,\text{tmm}}^{(k)}$ selects a set of linear parameters $\boldsymbol{D}_l, \boldsymbol{b}_l$ to describe the k^{th} slot's trajectory from t to $t+1$. Each linear system captures a distinct rigid motion pattern for a particular object:

$$p(\mathbf{x}_t^{(k)} \mid \mathbf{x}_{t-1}^{(k)}, \mathbf{s}_{t,\text{tmm}}^{(k)}, \boldsymbol{D}_{1:L}, \boldsymbol{b}_{1:L}) = \prod_{l=1}^{L} \mathcal{N}(\boldsymbol{D}_l \mathbf{x}_t^{(k)} + \boldsymbol{b}_l, \left(\mathcal{G}_t^{(k)}\right)^{-1} 2I)^{s_{t,l,\text{tmm}}^{(k)}} \tag{26}$$

$$p(\mathbf{s}_{t,\text{tmm}}^{(k)} \mid \boldsymbol{\pi}_{\text{tmm}}) = \text{Cat}(\boldsymbol{\pi}_{\text{tmm}}),\ p(\boldsymbol{\pi}_{\text{tmm}}) = \text{Dir}(\underbrace{1, \ldots, 1}_{L-1 \text{ times}}, \alpha_{0,\text{tmm}}) \tag{27}$$

$$\tag{28}$$

where we fix the covariance of all L components to be $2I$, and all mixture likelihoods $\boldsymbol{D}_{1:L}, \boldsymbol{b}_{1:L}$ to have uniform priors. Note that we gate the covariance once again using the reciprocal of $\mathcal{G}_t^{(k)}$ to filter the tMM to only model objects that are moving and present. The truncated stick-breaking prior $\text{Dir}(1, \ldots, 1, \alpha_{0,\text{tmm}})$ over the component mixing weights enables the number of linear modes L to be dynamically adjusted to the data by growing the model with propensity $\alpha_{0,\text{tmm}}$. Importantly, the L transition components of the tMM are not slot-dependent, but are shared and thus learned using the

data from all K slot latents. The tMM can thus explain and predict the motion of different objects using a shared, expanding set of (up to L distinct) linear dynamical systems.

A.8 Recurrent Mixture Model

The *recurrent mixture model* (rMM) is used to infer the switch states of the transition model directly from current slot-level features. This dependence of switch states on continuous features is the same construct used in the recurrent switching linear dynamical system or rSLDS [19]. However, in contrast to the rSLDS, which uses a discriminative mapping to infer the switch state from the continuous state (usually a softmax or stick-breaking parameterization thereof), the rMM recovers this dependence *generatively* using a mixture model over mixed continuous–discrete slot states [30]. In this way, 'selecting' the switch state used for conditioning the tMM actually emerges from *inference* over discrete latent variables, which have a particular conditional relationship (in this context, a joint mixture likelihood relationship) to other latent and observed variables. Concretely, the rMM models the distribution of continuous and discrete variables as a mixture model driven by another per-slot latent assignment variable $\mathbf{s}_{\text{rmm}}^{(k)}$. The rMM defines a mixture likelihood over a tuple of continuous and discrete slot-specific information. We use the notation $f_{t-1}^{(k)}$ to collect the continuous features that the rMM parameterizes a density over: they include a subset of the k^{th} slot's own continuous state $\mathbf{x}_{t-1}^{(k)}$, as well as a nonlinear transformation g applied to other slots' features, that computes the 2-D distance vector pointing from the inferred position of the focal object (i.e., slot k) to the nearest interacting slot j. We detail how these features are computed below:

Continuous features for the rMM The continuous latent dimensions of $\mathbf{x}_{t-1}^{(k)}$ used for the rMM include the following: $\mathbf{p}_{t-1}^{(k)}, \mathbf{v}_{t-1}^{(k)}, u_{t-1}^{(k)}$. We represent extracting this subset as a sparse linear projection applied to the full continuous latents $C\mathbf{x}_{t-1}^{(k)}$. In addition, the rMM models the distribution of the 2-D vectors pointing from the focal object's position (the slot k in consideration) to the position of the nearest interacting object, i.e. $\Delta\mathbf{p}_{t-1}^{(k)} \equiv \mathbf{p}_{t-1}^{(j)} - \mathbf{p}_{t-1}^{(k)}$. The nearest interacting object with index j is the one whose inferred position is the closest to the focal object's, while only considering neighbors within some interaction zone with radius r_{min}. This is the same interaction distance used to compute nearest-neighbors when populating the $\mathbf{z}_{\text{interacting}}^{(k)}$ latent, see Appendix A.4 for details. If no object is within the interaction radius, then we set $\Delta\mathbf{p}_{t-1}^{(k)}$ to a random sample from a 2-D uniform distribution with mean $\left[1.2, 1.2\right]$ and lower/upper bounds of $\left[1.198, 1.198\right]$, $\left[1.202, 1.202\right]$. We summarize this computation with a generic nonlinear function applied to the latent states of all slots $g(\mathbf{x}_{t-1}^{(1:K)})$, to intimate possible generalizations where different sorts of (possibly learnable) neighborhood relationships could be inserted here.

Discrete features for the rMM The discrete features include the following: $\mathbf{z}_{t-1,\text{type}}^{(k)}$, $\mathbf{z}_{t-1,\text{interacting}}^{(k)}$, the assignment-count indicator variable $o_{t-1}^{(k)}$, the switch state of the transition mixture model $\mathbf{s}_{t,\text{tmm}}^{(k)}$, the action at timestep $t-1$ and reward at the current timestep r_t. We refer to this discrete collection as $d_{t-1}^{(k)}$. The inclusion of $\mathbf{s}_{t,\text{tmm}}^{(k)}$ in the discrete inputs to the rMM is a critical ingredient of this recurrent dynamics formulation – it allows inference on this the switching state of the tMM to be driven by high-dimensional configurations of continuous and discrete variables relevant for predicting motion.

rMM description The rMM assignment variable associated to a given slot is a binary vector $\mathbf{s}_{t,\text{rmm}}^{(k)}$ whose m^{th} entry $s_{t,m,\text{rmm}}^{(k)} \in \{0, 1\}$ indicates whether component m explains the current tuple of mixed continuous-discrete data. Each component likelihood selected by $\mathbf{s}_{t,\text{rmm}}^{(k)}$ factorizes into a product of continuous (Gaussian) and discrete (Categorical) likelihoods.

$$f_{t-1}^{(k)} = \left(C\mathbf{x}_{t-1}^{(k)}, g(\mathbf{x}_{t-1}^{(1:K)})\right), \quad d_{t-1}^{(k)} = \left(\mathbf{z}_{t-1,\text{type}}^{(k)}, \mathbf{z}_{t-1,\text{interacting}}^{(k)}, o_{t-1}^{(k)}, \mathbf{s}_{t,\text{tmm}}^{(k)}, a_{t-1}, r_t\right) \tag{29}$$

$$p(f_{t-1}^{(k)}, d_{t-1}^{(k)} \mid \mathbf{s}_{t,\text{rmm}}^{(k)}) = \prod_{m=1}^{M} \left[\mathcal{N}(f_{t-1}^{(k)}; \boldsymbol{\mu}_{m,\text{rmm}}, \left(\mathcal{G}_t^{(k)}\right)^{-1} \boldsymbol{\Sigma}_{m,\text{rmm}}) \prod_i \text{Cat}\left(d_{t-1,\,i}; \mathcal{G}_t^{(k)} \mathbf{a}_{m,i}\right) \right]^{s_{t,m,\text{rmm}}} \tag{30}$$

$$p(\boldsymbol{\mu}_{m,\text{rmm}}, \boldsymbol{\Sigma}_{m,\text{rmm}}^{-1}) = \text{NIW}(\mathbf{m}_{0,m,\text{rmm}}, \kappa_{0,m,\text{rmm}}, \mathbf{U}_{0,m,\text{rmm}}, n_{0,m,\text{rmm}}),\ p(\mathbf{a}_{m,i}) = \text{Dir}(\mathbf{a}_{0,m,i}) \tag{31}$$

$$p(\mathbf{s}_{t,\text{rmm}}^{(k)} \mid \boldsymbol{\pi}_{\text{rmm}}) = \text{Cat}(\boldsymbol{\pi}_{\text{rmm}}),\ p(\boldsymbol{\pi}_{\text{rmm}}) = \text{Dir}(\underbrace{1, \ldots, 1}_{M-1 \text{ times}}, \alpha_{0,\text{rmm}}) \tag{32}$$

The parameters of the multivariate normal components are equipped with NIW priors and those of the discrete Categorical likelihoods with Dirichlet priors. As with all the other modules of AXIOM, we equip the mixing weights for $\mathbf{s}_{t,\text{rmm}}^{(k)}$ with a truncated stick-breaking prior whose final M^{th} pseudocount parameter tunes the propensity to add new rMM components. Note also the use of the gate variable $\mathcal{G}_t^{(k)}$ to filter slots for dynamics learning by inflating the covariance associated with any slot inputs not inferred moving and present.

Fixed distance variant. We explored a variant of the rMM (`fixed_distance`) where the displacement vector $\Delta\mathbf{p}_{t-1}^{(k)}$ is not returned by $g(\mathbf{x}_{t-1}^{(1:K)})$ and therefore not included as one of the continuous input features for the rMM. In this case, the entry of $\mathbf{z}_{t-1,\text{interacting}}^{(k)}$ that corresponds to the nearest interacting object is still determined by r_{min}, however. In this case, the choice of r_{min} matters more for performance because the rMM cannot learn to nuance its dynamic predictions based on precise distances. In general, this means that r_{min} requires more game-specific tuning. See Section 3 for the results of tuning r_{min} compared to learning it directly by providing $\Delta\mathbf{p}_{t-1}^{(k)}$ as input to the rMM.

A.9 Variational inference and learning

To perform inference and learning within our proposed model, we employ a variational Bayesian approach. The core idea is to approximate the true posterior distribution over latent variables and parameters with a more tractable factorized distribution, $q(\mathcal{Z}_{0:T}, \tilde{\boldsymbol{\Theta}})$. This is achieved by optimizing the variational free-energy functional, $\mathcal{F}(q)$, which establishes an upper-bound on the negative log-marginal likelihood of the observed data $\mathbf{y}_{0:T}$:

$$\mathcal{F}(q) = \mathbb{E}_q\Big[\log q(\mathcal{Z}_{0:T}, \tilde{\boldsymbol{\Theta}}) - \log p(\mathbf{y}_{0:T}, \mathcal{Z}_{0:T}, \tilde{\boldsymbol{\Theta}})\Big], \quad \text{such that} \quad \mathcal{F}(q) \geq -\log p(\mathbf{y}_{0:T}). \tag{33}$$

Minimizing this functional $\mathcal{F}(q)$ with respect to q is equivalent to maximizing the Evidence Lower Bound (ELBO), thereby driving the approximate posterior $q(\mathcal{Z}_{0:T}, \tilde{\boldsymbol{\Theta}})$ to more closely resemble the true posterior $p(\mathcal{Z}_{0:T}, \tilde{\boldsymbol{\Theta}} \mid \mathbf{y}_{0:T})$.

We assume a mean-field factorization for the approximate posterior, which decomposes over the global parameters $\tilde{\boldsymbol{\Theta}}$ and the sequence of latent variables $\mathcal{Z}_{0:T}$:

$$q(\mathcal{Z}_{0:T}, \tilde{\boldsymbol{\Theta}}) = q(\tilde{\boldsymbol{\Theta}}) \prod_{t=0}^{T} q(\mathcal{Z}_t), \tag{34}$$

where $q(\mathcal{Z}_t)$ further factorizes across individual latent variables for frame t:

$$q(\mathcal{Z}_t) = \left(\prod_{n=1}^{N} q\big(\mathbf{z}_{t,\text{smm}}^{n}\big) \right) \prod_{k=1}^{K} \left(q\big(\mathbf{x}_t^{(k)}\big)\, q\big(\mathbf{z}_t^{(k)}\big)\, q\big(\mathbf{s}_t^{(k)}\big) \right). \tag{35}$$

The variational distribution over the global parameters $\tilde{\boldsymbol{\Theta}}$ is also assumed to factorize according to the distinct components of our model:

$$q(\tilde{\boldsymbol{\Theta}}) = q(\boldsymbol{\Theta}_{\text{sMM}})\, q(\boldsymbol{\Theta}_{\text{iMM}})\, q(\boldsymbol{\Theta}_{\text{tMM}})\, q(\boldsymbol{\Theta}_{\text{rMM}}). \tag{36}$$

Note that the pixel-to-slot assignment variables $\mathbf{z}_{t,\text{smm}}^{n}$ are specific to each pixel n but are not factorized across slots for a given pixel, as they represent a single categorical choice from K slots. Other latent variables, such as the continuous state $\mathbf{x}_t^{(1:K)}$ and discrete attributes $\mathbf{z}_t^{(1:K)}, \mathbf{s}_t^{(1:K)}$, are factorized per slot k.

The inference and learning procedure for each new frame t involves an iterative alternation between an E-step, where local latent variable posteriors are updated, and an M-step, where global parameter posteriors are refined.

E-step In the Expectation-step (E-step), we hold the variational posteriors of the global parameters $q(\mathbf{\Theta})$ fixed. We then update the variational posteriors for each local latent variable factor within $q(\mathcal{Z}_t)$, such as $q(\mathbf{z}^n_{t,\text{smm}})$, $q(\mathbf{x}^{(k)}_t)$, $q(\mathbf{z}^{(k)}_t)$, and $q(\mathbf{s}^{(k)}_t)$. These updates are derived by optimizing the ELBO with respect to each factor in turn and often result in closed-form coordinate-ascent updates due to conjugacy between the likelihood terms and the chosen forms of the variational distributions.

M-step In the Maximization-step (M-step), we update the variational posteriors for the global parameters $q(\mathbf{\Theta}_\mu)$ associated with each model component $\mu \in \{\text{sMM}, \text{iMM}, \text{tMM}, \text{rMM}\}$. Each $q(\mathbf{\Theta}_\mu)$ is assumed to belong to the exponential family, characterized by natural parameters η_μ:

$$q(\mathbf{\Theta}_\mu) = h(\mathbf{\Theta}_\mu)\, \exp\big\{\eta_\mu^\top T(\mathbf{\Theta}_\mu) \;-\; A(\eta_\mu)\big\}, \tag{37}$$

where $T(\mathbf{\Theta}_\mu)$ represents the sufficient statistics for $\mathbf{\Theta}_\mu$, and $A(\eta_\mu)$ is the log-partition function (or log-normalizer). The M-step update proceeds in two stages for each component:

1. First, we compute the *expected sufficient statistics* $\widehat{T}_\mu$ using the current posteriors over the latent variables $q(\mathcal{Z}_t)$ obtained from the t-th E-step:

$$\widehat{T}_\mu = \mathbb{E}_{q(\mathcal{Z}_t)}\big[T\big(\mathbf{\Theta}_\mu, \mathcal{Z}_t\big)\big]. \tag{38}$$

   These expected statistics are then combined with prior natural parameters $\eta_{\mu,0}$ to form the target natural parameters for the update: $\widehat{\eta}_\mu = \widehat{T}_\mu + \eta_{\mu,0}$.

2. Second, we update the current natural parameters $\eta_\mu^{(t-1)}$ using a *natural-gradient* step, which acts as a stochastic update blending the previous parameters with the new target parameters, controlled by a learning rate schedule ρ_t:

$$\eta_\mu^{(t)} \;\leftarrow\; (1-\rho_t)\,\eta_\mu^{(t-1)} \;+\; \rho_t\,\widehat{\eta}_\mu, \qquad \text{where } 0 < \rho_t \leq 1. \tag{39}$$

Slot Mixture Model (sMM)

The Slot Mixture Model (sMM) provides a likelihood for the observed pixel data $\mathbf{y}_t$ by modeling each pixel as originating from one of K object slots. The variational approximation involves posteriors over pixel-to-slot assignments $\mathbf{z}^n_{t,\text{smm}}$, slot mixing weights $\boldsymbol{\pi}_{\text{smm}}$, slot-specific color variances $\sigma^{(k)}_{\text{c},j}$, and the continuous latent states of slots $\mathbf{x}^{(k)}_t$. Specifically, for each pixel n at time t, the posterior probability that it belongs to slot k is $q(z^n_{t,k,\text{smm}} = 1) = r^n_{t,k}$, ensuring that $\sum_{k=1}^K r^n_{t,k} = 1$. The posterior over the slot-mixing probabilities $\boldsymbol{\pi}_{\text{smm}}$ is a Dirichlet distribution: $q(\pi_{\text{smm}}) = \text{Dir}(\pi_{\text{smm}} \mid \alpha_{1,\text{smm}}, \ldots, \alpha_{K,\text{smm}})$, parameterized by concentrations $\alpha_{k,\text{smm}} > 0$. For each slot k and color channel $j \in \{r, g, b\}$, the posterior over the color variance $\sigma^{(k)}_{\text{c},j}$ is a Gamma distribution: $q(\sigma^{(k)}_{\text{c},j}) = \text{Gamma}(\sigma^{(k)}_{\text{c},j} \mid \gamma_{k,j}, b_{k,j})$, with shape $\gamma_{k,j}$ and rate $b_{k,j}$. Finally, each slot's continuous latent state $\mathbf{x}^{(k)}_t \in \mathbb{R}^{10}$ (encompassing position, color, velocity, shape, and the unused counter) is modeled by a Gaussian distribution: $q(\mathbf{x}^{(k)}_t) = \mathcal{N}(\mathbf{x}^{(k)}_t \mid \mu^{(k)}_t, \Sigma^{(k)}_t)$. The precision matrix is denoted $\Lambda = \Sigma^{-1}$, and the precision-weighted mean is $h^{(k)}_t = \Lambda^{(k)}_t\, \mu^{(k)}_t$.

E-step Updates for sMM During the E-step for the sMM at frame t, the variational distributions for local latent variables $\mathbf{z}^n_{t,\text{smm}}$ (represented by the responsibilities $r^n_{t,k}$) and $\mathbf{x}^{(k)}_t$ (represented by $\mu^{(k)}_t, \Sigma^{(k)}_t$) are updated, while the global sMM parameters $q(\mathbf{\Theta}_{\text{sMM}})$ are held fixed.

1. The pixel responsibilities $r^n_{t,k}$, representing $q(z^n_{t,k,\text{sMM}} = 1)$, are updated using the standard mixture model update:

$$r^n_{t,k} = \frac{\exp\big(\mathbb{E}_q[\log \pi_{k,\text{smm}}] + \mathbb{E}_q[\log \mathcal{N}(\mathbf{y}^n_t;\, A\mathbf{x}^{(k)}_t,\, \mathbf{\Sigma}^{(k)})]\big)}{\sum_{j=1}^K \exp\big(\mathbb{E}_q[\log \pi_{j,\text{smm}}] + \mathbb{E}_q[\log \mathcal{N}(\mathbf{y}^n_t;\, A\mathbf{x}^{(j)}_t,\, \mathbf{\Sigma}^{(j)})]\big)}. \tag{40}$$

The per-slot observation covariance $\mathbf{\Sigma}^{(k)}$ is constructed as $\mathbf{\Sigma}^{(k)} = \mathrm{diag}\big(B\,\mathbb{E}_q[\mathbf{x}_t^{(k)}],\ \mathbb{E}_q[\sigma_{\mathbf{c}}^{(k)}]\big)$, consistent with the generative model where $B\mathbf{x}^{(k)}$ provides variance related to shape and $\sigma_{\mathbf{c}}^{(k)}$ provides color channel variances (see Equation (12) for the sMM likelihood equations).

2. The parameters of the Gaussian posterior $q(\mathbf{x}_t^{(k)})$ are updated by incorporating evidence from the pixels assigned to slot k. This involves updating its natural parameters (precision $\Lambda_t^{(k)}$ and precision-adjusted mean $h_t^{(k)}$):

$$\begin{aligned}\Lambda_t^{(k)} &= \Lambda_{t|t-1}^{(k)} + \sum_{n=1}^{N} r_{t,k}^n\, A^\top \big(\Sigma^{(k)}\big)^{-1} A,\\ h_t^{(k)} &= h_{t|t-1}^{(k)} + \sum_{n=1}^{N} r_{t,k}^n\, A^\top \big(\Sigma^{(k)}\big)^{-1} \mathbf{y}_t^n.\end{aligned} \tag{41}$$

The terms $\Lambda_{t|t-1}^{(k)}$ and $h_{t|t-1}^{(k)}$ are the natural parameters of the predictive distribution for $\mathbf{x}_t^{(k)}$. The standard parameters are then $\Sigma_t^{(k)} = \left(\Lambda_t^{(k)}\right)^{-1}$ and $\mu_t^{(k)} = \Sigma_t^{(k)}\, h_t^{(k)}$.

M-step Updates for sMM In the M-step, the global parameters of the sMM, which are the Dirichlet parameters $\alpha_{k,\mathrm{smm}}$ for mixing weights and the Gamma parameters $(\gamma_{k,j}, b_{k,j})$ for color variances, are updated. This begins by accumulating the expected sufficient statistics from the E-step:

$$\begin{aligned}N_{t,k} &= \sum_{n=1}^{N} r_{t,k}^n,\\ S_{1,t,k}^{\mathbf{y}} &= \sum_{n=1}^{N} r_{t,k}^n\, \mathbf{y}_t^n,\\ S_{2,t,k}^{\mathbf{y}} &= \sum_{n=1}^{N} r_{t,k}^n\, \mathbf{y}_t^n\, (\mathbf{y}_t^n)^\top.\end{aligned} \tag{42}$$

The Dirichlet concentration parameters are updated as (now using the t index to represent the current vs. last settings of the posterior parameters):

$$\begin{aligned}\alpha_{t,k,\mathrm{smm}} &= (1-\rho_t)\,\alpha_{t-1,k,\mathrm{smm}} + \rho_t\,(\alpha_{0,k,\mathrm{smm}} + N_{t,k}),\\ \gamma_{t,k,j} &= (1-\rho_t)\,\gamma_{t-1,k,j} + \rho_t\,\left(\gamma_{0,j} + \frac{N_{t,k}}{2}\right),\\ b_{t,k,j} &= (1-\rho_t)\,b_{t-1,k,j} + \rho_t\,\left(1 + \frac{1}{2}\sum_{n=1}^{N} r_{t,k}^n (\mathbf{y}_{t,\text{color }j}^n - (A\mathbb{E}_q[\mathbf{x}_t^{(k)}])_{\text{color }j})^2\right).\end{aligned} \tag{43}$$

Here, $\alpha_{0,k,\mathrm{smm}}$ represents the prior concentration for the Dirichlet distribution (e.g., 1 for the first $K-1$ components and $\alpha_{0,\mathrm{smm}}$ for the K-th, if using a truncated stick-breaking prior). For the Gamma parameters, $\gamma_{0,j}$ is the prior shape and 1 is the prior rate (or related prior parameters). The projection matrices A and B are considered fixed and are not learned.

Presence, Motion, and Unused Counter Dynamics

The model includes latent variables for each slot k that track its presence $\mathbf{z}_{t,\text{presence}}^{(k)}$, motion $\mathbf{z}_{t,\text{moving}}^{(k)}$, and an unused counter $u_t^{(k)}$.

Inference over the presence latent The presence state is informed by an 'assignment-count-indicator' $o_t^{(k)}$. This indicator is set to 1 if the slot is actively explaining pixels (e.g., if the sum of its responsibilities $\sum_n r_{t,k}^n$ exceeds a small threshold ϵ_{active}), and 0 otherwise.

Recall the Bernoulli likelihood over $o_t^{(k)}$ that links it to the $\mathbf{z}_{t,\text{presence}}^{(k)}$ latent as follows (cf. Equation (17)):

$$p(o_t|\mathbf{z}_{t,\text{presence}}) = o_t^{z_{t,P,\text{presence}}}(1-o_t)^{z_{t,A,\text{presence}}} \tag{44}$$

There is an implied superscript k on both $o_t^{(k)}$ and the presence variable $z_{t,P,\text{presence}}^{(k)}$, which are left out to avoid visual clutter. This linkage is incorporated into $q(\mathbf{z}_{t,\text{presence}}^{(k)})$ using an Expectation Propagation (EP) style update via a pseudo-likelihood [37]:

$$\tilde{\ell}\big(\mathbf{z}_{t,\text{presence}}^{(k)}\big) = \Big[(o_t^{(k)})^{z_{t,P,\text{presence}}^{(k)}}\big(1-o_t^{(k)}\big)^{z_{t,A,\text{presence}}^{(k)}}\Big]^{\zeta}, \tag{45}$$

where ζ is a damping factor. This update increases posterior evidence for presence if $o_t^{(k)}=1$, and for absence if $o_t^{(k)}=0$. The first-order effect of this pseudo-likelihood when updating the posterior over $\mathbf{z}_{t,\text{presence}}^{(k)}$ is

$$q(z_{t,\text{P,presence}}^{(k)}) \approx (1-\zeta)q(z_{t-1,\text{P,presence}}^{(k)}) + \zeta\, o_t^{(k)}. \tag{46}$$

Recall that the A, P subscripts refer to the indices of $\mathbf{z}_{t-1,\text{presence}}^{(k)}$ that signify the 'is-absent' and 'is-present' states, respectively.

Inference over the moving latent Similar EP updates apply for inferring $q(z_{t,M,\text{moving}}^{(k)})$ based on its specific likelihoods involving velocity and $o_t^{(k)}$. The gate $\mathcal{G}_t^{(k)} = q(z_{t,P,\text{present}}^{(k)}=1)\cdot q(z_{t,\text{M,moving}}^{(k)}=1)$ is then formed from these inferred probabilities.

Inference over the unused counter The 'unused counter' $u_t^{(k)}$ tracks how long slot k has been inactive. Appendix A.5 of the full model details describes a generative likelihood $P(o_t^{(k)}=1 \mid u_t^{(k)}) = 1-\exp(-\xi e^{-\gamma_u u_t^{(k)}})$, for which damped EP updates for $q(u_t^{(k)})$ can be derived and would remain in closed form. However, a specific case, by choosing hyperparameters such that a hard constraint $o_t^{(k)}=1 \iff u_t^{(k)}=0$ is effectively enforced (e.g., by taking $\gamma_u \to \infty$ and $\xi=1$ in the generative likelihood), leads to a simplified, deterministic update for the posterior mean $\mu_{t,u}^{(k)} = \mathbb{E}_q[u_t^{(k)}]$:

$$\mu_{t,u}^{(k)} = \big(1-o_t^{(k)}\big)\,\big(\mu_{t-1,u}^{(k)} + \nu_u\big), \qquad \nu_u = 0.05. \tag{47}$$

In this simplified regime, the counter is reset to 0 if $o_t^{(k)}=1$; otherwise, it increments by a fixed amount ν_u.

Identity Mixture Model (iMM)

The Identity Mixture Model (iMM) assigns one of V possible discrete identities to each active slot, based on its continuous features. This allows for shared characteristics and dynamics across instances of the same object type. The variational approximation targets posteriors over these slot-to-identity assignments $\mathbf{z}_{t,\text{type}}^{(k)}$ (which is a component of $\mathbf{z}_t^{(k)}$), identity mixing weights $\boldsymbol{\pi}_{\text{type}}$, and the parameters $(\boldsymbol{\mu}_{j,\text{type}}, \boldsymbol{\Sigma}_{j,\text{type}})$ for each identity's feature distribution. The features $\mathbf{y}_{t,\text{imm}}^{(k)}$ utilized by the iMM for slot k are its color $\mathbf{c}_t^{(k)}$ and shape $\mathbf{e}_t^{(k)}$, thus $\mathbf{y}_{t,\text{imm}}^{(k)} = [\mathbf{c}_t^{(k)}, \mathbf{e}_t^{(k)}]^\top$. For each slot k at time t where the activity gate $\mathcal{G}_t^{(k)} \approx 1$, the posterior probability that it belongs to identity v is $q(z_{t,v,\text{imm}}^{(k)}=1) = \gamma_{t,v}^{(k)}$, satisfying $\sum_{v=1}^{V}\gamma_{t,v}^{(k)} = 1$. For slots where $\mathcal{G}_t^{(k)} \approx 0$, these responsibilities are effectively null or uniform, contributing negligibly to parameter updates. The posterior over identity-mixing probabilities $\pi_{1:V,\text{iMM}}$ is a Dirichlet distribution: $q(\boldsymbol{\pi}_{1:V,\text{type}}) = \text{Dir}(\boldsymbol{\pi}_{1:V,\text{type}} \mid \alpha_{1,\text{type}}, \ldots, \alpha_{V,\text{type}})$, with $\alpha_{v,\text{type}} > 0$. Each identity v is characterized by a mean $\boldsymbol{\mu}_{v,\text{type}}$ and covariance $\boldsymbol{\Sigma}_{v,\text{type}}$. The variational posterior over these parameters is a Normal–Inverse-Wishart (NIW) distribution: $q(\boldsymbol{\mu}_{v,\text{type}}, \boldsymbol{\Sigma}_{v,\text{type}}) = \text{NIW}(\boldsymbol{\mu}_{v,\text{type}}, \boldsymbol{\Sigma}_{v,\text{type}} \mid \boldsymbol{m}_{v,\text{type}}, \kappa_{v,\text{type}}, \mathbf{U}_{v,\text{type}}, n_{v,\text{type}})$.

E-step Updates for iMM In the E-step for the iMM, the local assignment probabilities $\gamma_{t,v}^{(k)}$ for each slot k are updated. This update is primarily driven by slots where $\mathcal{G}_t^{(k)} \approx 1$:

$$\gamma_{t,v}^{(k)} \;\propto\; \exp\big(\mathbb{E}_q[\log \pi_{v,\text{type}}]\big) \;\times\; \exp\big(\mathbb{E}_q[\log \mathcal{N}(\mathbf{y}_{t,\text{iMM}}^{(k)};\, \boldsymbol{\mu}_{v,\text{type}}, \boldsymbol{\Sigma}_{v,\text{type}})]\big). \tag{48}$$

These responsibilities are normalized such that $\sum_{v=1}^{V} \gamma_{t,v}^{(k)} = 1$ for each active slot.

M-step Updates for iMM During the M-step, the global parameters of the iMM are updated. Sufficient statistics are accumulated, weighted by the gate $\mathcal{G}_t^{(k)}$ to ensure that only actively moving slots contribute significantly to the updates:

$$\begin{aligned} N_{t,v} &= \sum_{k=1}^{K} \mathcal{G}_t^{(k)}\, \gamma_{t,v}^{(k)}, \\ S_{1,t,v} &= \sum_{k=1}^{K} \mathcal{G}_t^{(k)}\, \gamma_{t,v}^{(k)}\, \mathbb{E}_q[\mathbf{y}_{t,\text{imm}}^{(k)}], \\ S_{2,t,v} &= \sum_{k=1}^{K} \mathcal{G}_t^{(k)}\, \gamma_{t,v}^{(k)}\, \mathbb{E}_q[\mathbf{y}_{t,\text{iMM}}^{(k)} (\mathbf{y}_{t,\text{iMM}}^{(k)})^\top]. \end{aligned} \tag{49}$$

The Dirichlet parameters $\alpha_{t,v,\text{type}}$ are updated using $N_{t,v}$ and prior parameters $\alpha_{0,v,\text{type}}$ (which, due to the stick-breaking priors are all 1's except for the final count $\alpha_{0,V,\text{type}}$):

$$\alpha_{t,v,\text{type}} = (1-\rho_t)\, \alpha_{t-1,v,\text{type}} + \rho_t \left(\alpha_{0,v,\text{type}} + N_{t,v}\right). \tag{50}$$

The NIW parameters $(\mathbf{m}_{t,v,\text{type}},\, \kappa_{t,v,\text{type}},\, \mathbf{U}_{t,v,\text{type}},\, n_{t,v,\text{type}})$ are updated by blending their natural parameter representations. The target natural parameters $\widehat{\eta}_{t,v}^{\text{NIW}}$ are derived from the current sufficient statistics $\{N_{t,v}, S_{1,t,v}, S_{2,t,v}\}$ and the NIW prior parameters:

$$(\text{NatParams}_{t,v}^{\text{NIW}}) \;\leftarrow\; (1-\rho_t)\, (\text{NatParams}_{t-1,v}^{\text{NIW}}) \;+\; \rho_t\, \widehat{\eta}_{t,v}^{\text{NIW}}. \tag{51}$$

Recurrent Mixture Model (rMM)

The Recurrent Mixture Model (rMM) provides a generative model for a collection of slot-specific features, and importantly, it furnishes the distribution over the switch state $\mathbf{s}_{t,\text{tmm}}^{(k)}$ that governs the Transition Mixture Model (tMM). The rMM itself is a mixture model with M components, and its own slot-specific assignment variable is $\mathbf{s}_{t,\text{rmm}}^{(k)}$. The variational factors include the posterior probability of assignment to rMM component m, $q(s_{t,m,\text{rmm}}^{(k)} = 1) = \rho_{t,m}^{(k)}$ (which sums to one over m), a Dirichlet posterior $q(\boldsymbol{\pi}_{\text{rmm}}) = \text{Dir}(\boldsymbol{\pi}_{\text{rmm}} \mid \alpha_{1,\text{rmm}}, \ldots, \alpha_{M,\text{rmm}})$ for its mixing weights, NIW posteriors $q(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ for continuous features $f_{t-1}^{(k)}$ modeled by each component m, and Dirichlet posteriors $q(\mathbf{a}_{i,m})$ for the parameters of categorical distributions over various discrete features $d_i^{(k)}$ also modeled by component m. These discrete features $d_i^{(k)}$ encompass inputs like $\mathbf{z}_{t-1,\text{type}}^{(k)}$, $\mathbf{z}_{t-1,\text{interacting}}^{(k)}$, as well as the tMM switch state $\mathbf{s}_{t,\text{tmm}}^{(k)}$ which the rMM models generatively.

E-step Updates for rMM In the rMM E-step, for each slot k considered active (i.e., $\mathcal{G}_t^{(k)} \approx 1$), the responsibilities $\rho_{t,m}^{(k)}$ for its M components are updated. These updates depend on the likelihood of the slot's input features under each rMM component. The input features include continuous aspects $f_{t-1}^{(k)}$ (a subset of $\mathbf{x}_{t-1}^{(k)}$ such as position and velocity, and interaction features like $\Delta\mathbf{p}_{t-1}^{(k)}$) and a set of discrete features $d_{t-1,\text{inputs}}^{(k)}$ (e.g., type from the previous step $\mathbf{z}_{t-1,\text{type}}^{(k)}$).

$$\begin{aligned} \rho_{t,m}^{(k)} \;\propto\; &\exp\big(\mathbb{E}_q[\log \pi_{m,\text{rmm}}]\big) \;\times\; \exp\big(\mathbb{E}_q[\log \mathcal{N}(f_{t-1}^{(k)}; \boldsymbol{\mu}_{m,\text{rmm}}, \boldsymbol{\Sigma}_{m,\text{rmm}})]\big) \\ &\times \prod_{i \in \text{input discrete features}} \exp\big(\mathbb{E}_q[\log\, \text{Cat}(d_{t-1,i}^{(k)}; \mathbf{a}_{i,m})]\big). \end{aligned} \tag{52}$$

These responsibilities are normalized to sum to one for each active slot k. During rollouts used in planning, the predicted posterior distribution for the tMM switch state $\mathbf{s}_{t,\text{tmm}}^{(k)}$ is then determined as a mixture of the output distributions from the rMM components, weighted by $\rho_{t,m}^{(k)}$:

$$q\big(s_{t,l,\text{tmm}}^{(k)}=1\big)=\sum_{m=1}^{M}\rho_{t,m}^{(k)}\,\mathbb{E}_q\big[\mathbf{a}_{\text{tmm\_switch},m,l}\big],$$

where $\mathbf{a}_{\text{tmm\_switch},m,l}$ is the probability of tMM switch state l under the learned parameters of rMM component m.

M-step Updates for rMM In the rMM M-step, its global parameters are updated, with contributions from slots weighted by the gate $\mathcal{G}_t^{(k)}$. The expected sufficient statistics are accumulated. For the mixing weights:

$$N_{t,m}=\sum_{k=1}^{K}\mathcal{G}_t^{(k)}\,\rho_{t,m}^{(k)}. \tag{53}$$

For the continuous feature distributions (NIW parameters):

$$\begin{aligned}S_{1,t,m}^{f}&=\sum_{k=1}^{K}\mathcal{G}_t^{(k)}\,\rho_{t,m}^{(k)}\,\mathbb{E}_q[f_{t-1}^{(k)}],\\ S_{2,t,m}^{f}&=\sum_{k=1}^{K}\mathcal{G}_t^{(k)}\,\rho_{t,m}^{(k)}\,\mathbb{E}_q[f_{t-1}^{(k)}(f_{t-1}^{(k)})^{\top}].\end{aligned} \tag{54}$$

For each discrete feature d_i modeled by the rMM (this includes input features $d_{t-1,i}^{(k)}$ and the output tMM switch state $s_{t,\text{tmm}}^{(k)}$), and its category ℓ:

$$N_{t,m,i,\ell}=\sum_{k=1}^{K}\mathcal{G}_t^{(k)}\,\rho_{t,m}^{(k)}\begin{cases}\mathbb{I}[d_{t-1,i}^{(k)}=\ell] & \text{if } d_i \text{ is an input from } t-1\\ q(s_{t,\text{tmm}}^{(k)}=\ell) & \text{if } d_i \text{ is } s_{t,\text{tmm}}^{(k)}\end{cases}. \tag{55}$$

The parameters are then updated using these statistics. Dirichlet parameters for rMM mixing weights $\alpha_{t,m,\text{rmm}}$ (from $N_{t,m}$), NIW parameters for continuous features $(\mathbf{m}_{t,m,\text{rmm}},\,\kappa_{t,m,\text{rmm}},\,\mathbf{U}_{t,m,\text{rmm}},\,n_{t,m,\text{rmm}})$ (from $N_{t,m}, S_{1,t,m}^{f}, S_{2,t,m}^{f}$), and Dirichlet parameters $\mathbf{a}_{t,i,m,\ell}$ for all discrete features (from $N_{t,m,i,\ell}$) are updated via natural gradient blending:

$$\begin{aligned}\alpha_{t,m,\text{rmm}}&=(1-\rho_t)\,\alpha_{t-1,m,\text{rmm}}+\rho_t\,\big(\alpha_{0,m,\text{rmm}}+N_{t,m}\big),\\ (\text{NatParams}_{t,m}^{\text{NIW}})\;&\leftarrow\;(1-\rho_t)\,(\text{NatParams}_{t-1,m}^{\text{NIW}})+\rho_t\,\widehat{\eta}_{t,m}^{\text{NIW}},\\ \mathbf{a}_{t,i,m,\ell}&=(1-\rho_t)\,\mathbf{a}_{t-1,i,m,\ell}+\rho_t\,\big(\mathbf{a}_{0,i,m,\ell}+N_{t,m,i,\ell}\big).\end{aligned} \tag{56}$$

Transition Mixture Model (tMM)

The Transition Mixture Model (tMM) describes the dynamics of slot states $\mathbf{x}_t^{(k)}$ using a mixture of L linear transitions. We approximate posteriors over transition assignments and mixing weights with variational factors. Transition responsibilities for slot k using transition l are $q(s_{t,l,\text{tmm}}^{(k)}=1)=\xi_{t,l}^{(k)}$, satisfying $\sum_{l=1}^{L}\xi_{t,l}^{(k)}=1$. The mixing-weight distribution over the L transitions $\boldsymbol{\pi}_{\text{tmm}}$ is $q(\boldsymbol{\pi}_{\text{tmm}})=\text{Dir}(\boldsymbol{\pi}_{\text{tmm}}\mid\alpha_{1,\text{tmm}},\ldots,\alpha_{L,\text{tmm}})$.

E-step Updates In the tMM E-step, for each slot k, the responsibilities $\xi_{t,l}^{(k)}$ for each transition l are updated based on how well that transition explains the observed change from $\mathbf{x}_{t-1}^{(k)}$ to $\mathbf{x}_t^{(k)}$:

$$\xi_{t,l}^{(k)}=\frac{\exp\big(\mathbb{E}[\log\pi_l]\big)\,\mathcal{N}\big(\mathbf{x}_t^{(k)};\,D_l\,\mathbf{x}_{t-1}^{(k)}+b_l,\;\mathcal{G}_{t-1}^{(k)}\,2I\big)}{\sum_{u=1}^{L}\exp\big(\mathbb{E}[\log\pi_u]\big)\,\mathcal{N}\big(\mathbf{x}_t^{(k)};\,D_u\,\mathbf{x}_{t-1}^{(k)}+b_u,\;\mathcal{G}_{t-1}^{(k)}\,2I\big)}$$

for $l=1,\ldots,L$. The term $\mathcal{G}_{t-1}^{(k)}$ indicates if the slot was active at $t-1$, and $2I$ is the process noise covariance, assumed fixed or shared.

Algorithm 1 Mixture Model Expansion Algorithm

Input	**Output**	**Hyperparameters / Settings**
$\mathbf{Y} \in \mathbb{R}^{N \times d}$: Matrix whose i^{th} row ($i = 1, \ldots, N$) is a d-dimensional token (e.g., pixel).	$\boldsymbol{\theta}^*_{1:K_{t+1}}$: Updated posterior NIW parameters (K_{t+1} components where $K_{t+1} \geq K_t$)	τ: expansion threshold
$\boldsymbol{\theta}^*_{1:K_t} = (\mathbf{m}_{1:K_t}, \kappa_{1:K_t}, \mathbf{U}_{1:K_t}, n_{K_t})$: Initial posterior NIW parameters (K_t components).		$\mathcal{E}$: maximum expansion steps

Initialise $K \leftarrow K_t$ and NIW parameters $\boldsymbol{\theta}_k = \big(\mathbf{m}_k, \kappa_k, \mathbf{U}_k, n_k\big)$ for $k = 1 : K$
for $g = 1$ **to** $\mathcal{E}$ **do** ▷ outer "expand-or-stop" loop
// E-step
  for $i = 1$ **to** N **do**
    $\ell_{ik} \leftarrow \mathbb{E}_{q(\theta_k)}\big[\log p(y_i \mid \theta_k)\big],\ k = 1{:}K$
    $r_{ik} \leftarrow \exp(\ell_{ik}) \Big/ \sum_{j=1}^{K} \exp(\ell_{ij})$
  $\ell_i^{\max} \leftarrow \max_{k \leq K} \ell_{ik},\ \ i = 1{:}N$
  if $\min_i \ell_i^{\max} > \tau$ **then**
    break ▷ all tokens well explained
  $i^* \leftarrow \arg\min_i \ell_i^{\max}$ ▷ worst-explained token
  $K \leftarrow K + 1$ ▷ instantiate new component
  Hard-assign $r_{i^*,k} \leftarrow 0\ (k < K), r_{i^*,K} \leftarrow 1$
  Initialise component K**:** $\kappa_K \leftarrow 1,\ \nu_K \leftarrow d + 2,\ \mu_K \leftarrow y_{i^*},\ \Psi_K \leftarrow \Psi_0$
// M-step (natural–gradient update)
  for $k = 1$ **to** K **do**
    $\widehat{\eta}_k \ \leftarrow \ \underbrace{\sum_{i=1}^{N} r_{ik}\, T(y_i)}_{\widehat{T}_k} \ + \eta_{k,0}$ ▷ compute target natural parameters
    $\eta_k^{(t)} \ \leftarrow \ (1 - \rho_t)\, \eta_k^{(t-1)} \ + \ \rho_t\, \widehat{\eta}_k$ ▷ natural-gradient update with rate ρ_t
    Unpack $\eta_k^{(t)} \rightarrow (m_k, \kappa_k, U_k, n_k)$ ▷ recover NIW hyperparams
return $\boldsymbol{\theta}^*_{1:K_{t+1}}$

M-step Updates The tMM does not use an explicit M-step. Instead, parameters are fixed to their initial values identified during the expansion algorithm. In other words, once we identify a new dynamics mode in the expansion algorithm, these parameters are added as a new component for the tMM and remain fixed.

A.10 Bayesian model reduction

Growing new clusters ensures plasticity, but left unchecked it leads to over-parameterisation and over-fitting. To enable generalization, every $\Delta T_{\text{BMR}} = 500$ frames we therefore run *Bayesian model reduction* on the rMM, merging pairs of components whenever doing so *increases* the expected evidence lower bound (ELBO) of the multinomial distributions over the next reward and SLDS switch. The ELBO is computed with respect to generated data from the model through ancestral sampling. Given two candidate components k_1, k_2 with posterior-sufficient statistics (η_{k_1}, η_{k_2}), their merged statistics are $\eta_{k_1 \cup k_2} = \eta_{k_1} + \eta_{k_2} - \eta_{k_2}^{\text{prior}}$, ensuring that prior mass is not double-counted.

Candidate pairs are proposed by a fast heuristic that (i) samples up to $n_{\text{pair}} = 2000$ used clusters, (ii) computes their mutual expected log-likelihood under the other's parameters, and (iii) retains the highest-scoring pairs. Each proposal is accepted *iff* the merged ELBO (line 7) is not smaller than the current ELBO (line 2). The procedure is spelled out in Algorithm 2.

Algorithm 2 Bayesian model reduction for the recurrent mixture model

Input	**Output**	**Hyper-parameters**
$\mathcal{M}$: Posterior rMM	Reduced model $\mathcal{M}'$	n_{pair}, n_{samples} pruning interval ΔT_{BMR}

$\mathcal{D} = \{(\mathbf{c}_i, \mathbf{d}_i)\}_{i=0}^{n_{\text{samples}}} \sim \mathcal{M}$ ▷ Draw n_{samples} pairs of continuous and discrete data samples from $\mathcal{M}$
$\mathcal{L}^{(0)} \leftarrow \text{ELBO}(\mathcal{M}, \mathcal{D})$ ▷ Compute current ELBO
$\mathcal{P} = \{(k_1, k_2)_i\}_{i=0}^{n_{\text{pairs}}}$ ▷ Draw up to n_{pair} candidate pairs by heuristic overlap
for $s = 1$ **to** $|\mathcal{P}|$ **do**
  $(k_1, k_2) \leftarrow \mathcal{P}_s$
  $\mathcal{M}^{\text{try}} \leftarrow \text{MERGE}(\mathcal{M}, k_1, k_2)$
  $\mathcal{L}^{\text{try}} \leftarrow \text{ELBO}(\mathcal{M}^{\text{try}}, \mathcal{D})$
  if $\mathcal{L}^{\text{try}} \geq \mathcal{L}^{(s-1)}$ **then**
    $\mathcal{M} \leftarrow \mathcal{M}^{\text{try}}$ **and** $\mathcal{L}^{(s)} \leftarrow \mathcal{L}^{\text{try}}$ ▷ Set current model to the candidate
  else
    $\mathcal{L}^{(s)} \leftarrow \mathcal{L}^{(s-1)}$
return $\mathcal{M}$

A.11 Planning with active inference

In active inference, policies $\pi = a_{0:H}$ are selected that minimize expected Free Energy [33]:

$$p(\pi) = \sigma(-G(\pi)), \text{ with}$$
$$G(\pi) = \sum_{\tau=0}^{H} -\big(\mathbb{E}_{q(\mathcal{O}_\tau|\pi)}[\underbrace{\log p(r_\tau|\mathcal{O}_\tau, \pi)}_{\text{Utility}} - \underbrace{\mathrm{D}_{KL}(q(\boldsymbol{\alpha}_{\text{rmm}}|\mathcal{O}_\tau, \pi) \parallel q(\boldsymbol{\alpha}_{\text{rmm}}))}_{\text{Information gain (IG)}}]\big), \tag{57}$$

with H the planning horizon and σ the softmax function. However, as the number of possible policies grows exponentially with a larger planning horizon, enumerating all policies at every timestep becomes infeasible. Therefore, we draw inspiration from model predictive control (MPC) solutions such as Cross Entropy Method (CEM) and model predictive path integral (MPPI), which can be cast as approximating an action posterior by moment matching [38].

In particular, we sample P policies of horizon H, and evaluate their expected Free Energy G. Instead of sampling actions for each future timestep τ uniformly, we maintain a horizon-wise categorical proposal $p(a_\tau)$. After every planning step, we keep the top-k samples with minimum G, and importance weight to get a new probability for each action $p(a_\tau)$ at future timestep τ:

$$p(a_\tau) = \sum_k \frac{\exp(-G(a_\tau^{(k)}))}{\sum_j \exp(-G(a_\tau^{(j)}))} \tag{58}$$

Instead of doing multiple cross-entropy iterations per planning step, we maintain a moving average of $p(a_\tau)$. At every instant, the first action of the current best policy is actually executed by the agent. Our planning loop is hence composed of the following three stages (see Alg. 3).

Sampling policies. For each iteration we draw

$$\underbrace{P - R}_{\text{CEM}} + \underbrace{R}_{\text{random}} \quad \text{policies}$$

where the first set is sampled i.i.d. from the proposal $p(a_\tau)$ and the remaining R are "exploratory" sequences generated by a (smoothed) random walk. In addition, the previous best plan is always injected in slot 0 and the A constant action sequences occupy slots $1{:}A$ to guarantee coverage.

Evaluating a policy. Each policy is rolled forward S times through the world-model to obtain per-step predictions of reward $\hat{r}_\tau$ and information gain $\widehat{\text{IG}}_\tau$, averaged over samples. We calculate an

Algorithm 3 Planning algorithm

Require: current posterior state q, proposal $p(a_\tau)$, best plan $\tilde{\mathbf{a}}$

// Sample P candidate policies

$\mathcal{A}_{\text{cem}} \leftarrow \text{Cat}(p(a_\tau))^{\otimes(P-R)}$

$\mathcal{A}_{\text{rand}} \leftarrow \text{RANDOM}(R)$

$\mathcal{A} \leftarrow [\tilde{\mathbf{a}},\ \text{const}_{0:A-1},\ \mathcal{A}_{\text{cem}},\ \mathcal{A}_{\text{rand}}]$

// Evaluate

for all $\mathbf{a}^{(p)} \in \mathcal{A}$ **do**

  $(\hat{r}_{0:H-1}, \widehat{\text{IG}}_{0:H-1}) \leftarrow \text{Rollout}(q, \mathbf{a}^{(p)})$

  $G^{(p)} \leftarrow \sum_\tau \gamma_{\text{discount}}^\tau (\hat{r}_\tau + \lambda_{\text{IG}}\, \widehat{\text{IG}}_\tau)$

// Refit proposal

$\mathcal{K} \leftarrow$ top-K indices of $-G^{(p)}$

for $\tau = 0$ **to** $H-1$ **do**

  $\hat{p}(a_\tau) \leftarrow \text{softmax}\big(\texttt{temperature}^{-1}\, \text{hist}\{a_\tau^{(p)}\}_{p\in\mathcal{K}}\big)$

  $p(a_\tau) \leftarrow \alpha_{\text{smooth}}\, \hat{p}(a_\tau) + (1-\alpha_{\text{smooth}})\, p(a_\tau)$

return first action of best plan, updated $p(a_\tau)$, best plan $\tilde{\mathbf{a}}$

expected Free Energy G, where in addition, we weigh the information gain term with a scalar λ_{IG} to trade off exploration and exploitation, as well as apply temporal discounting:

$$G = \frac{1}{S}\sum_{s=0}^{S-1}\sum_{\tau=0}^{H-1} \gamma_{\text{discount}}^\tau \left(\hat{r}_\tau^s + \lambda_{\text{IG}}\, \widehat{\text{IG}}_\tau^s\right), \qquad \gamma_{\text{discount}} \in [0,1),\ \lambda_{\text{IG}} = \texttt{info\_gain}\ \text{ weight.}$$

Proposal update. Let $\mathcal{K}$ be the indices of the top-$K = \lfloor \texttt{topk\_ratio} \cdot P \rfloor$ policies by (negative) expected Free Energy. For every horizon step τ we form the empirical action histogram of $\{\mathbf{a}_{k,\tau}\}_{k\in\mathcal{K}}$, convert it to probabilities with a tempered softmax (`temperature`) and perform an exponential-moving-average update

$$p(a_\tau)^{\text{new}} = \alpha_{\text{smooth}}\, p(a_\tau) + (1-\alpha_{\text{smooth}})\, p(a_\tau)^{\text{old}}, \qquad \alpha_{\text{smooth}} = \texttt{alpha}.$$

B Hyperparameters

We list the hyperparameters used for main AXIOM results shown in Figure 3 in Table 3. For the `fixed_distance` ablations, we fixed $r_{\text{min}} = 1.25$ for the games Explode, Bounce, Impact, Hunt, Gold, Fruits, and fixed $r_{\text{min}} = 1.25$ for the games Jump, Drive, Cross, and Aviate.

C Computational resources, costs and scaling

AXIOM and baseline models were trained and evaluated on A100 40G GPUs. All models use a single A100 GPU per environment. AXIOM and BBF train a single environment to 10k steps in about 30min, whereas DreamerV3 trains in 2.5h. The corresponding per-step breakdown of average inference and planning times can be seen in Table 2.

C.1 Planning and inference time

For AXIOM, each time step during training can be broken down into planning and model inference. To quantify the planning time scaling we perform ablations over the number of planning policies (P). Since model inference correlates with the number of mixture model components, we evaluate the scaling using the environments Explode (few objects) and Cross (many objects). Figure 5 shows that the planning time scales linearly with the number of policies rolled out (left panel), and how model inference time scales with the number of mixture model components (right panel).

D Gameworld 10k Environments

In this section we provide an informal description of the proposed arcade-style environments, inspired by the Arcade learning environment [25]. To this end, our environments have an observation space

Table 3: Hyperparameters of the AXIOM agent trained to play all 10 games of Gameworld as reported in the main text (see Figure 3).

Inference Hyperparameters	
Parameter	Value
τ_{SMM} (expansion threshold of the sMM)	5.7
τ_{iMM} (expansion threshold of the sMM)	-1×10^{2}
τ_{rMM} (expansion threshold of the rMM)	-1×10^{1}
τ_{tMM} (expansion threshold of the tMM)	-1×10^{-5}
$\mathcal{E}$ (maximum number of expansion steps)	10
ΔT_{BMR} (timesteps to BMR)	500
ζ (exponent of the damped $\mathbf{z}_{\text{presence}}$ likelihood)	0.01
$r_{\min}$	0.075
Planning Hyperparameters	
Parameter	Value
H (planning depth)	32
P (number of rollouts)	512
S (number of samples per rollout)	3
λ_{IG} (information gain weight)	0.1
γ_{discount} (discount factor)	0.99
top-k ratio	0.1
random sample ratio	0.5
temperature	10.0
α_{smooth}	1.0
Generative Model Parameters	
Structure	
K (max sMM components)	32
V (max iMM components)	32
L (max tMM components)	500
M (max rMM components)	5000
λ (from θ_{moving})	0.99
β (from θ_{moving})	0.01
γ_u	∞
ξ	1.0
ν_u	0.05
Dirichlet/Gamma priors	
$\gamma_{0,\text{R,G,B}}$	0.1
$\alpha_{0,\text{smm}}$	1
$\alpha_{0,\text{imm}}$	1×10^{-4}
$\alpha_{0,\text{tmm}}$	0.1
$\alpha_{0,\text{rmm}}$	0.1
Normal-Inverse–Wishart $(1:V)$	
$\mathbf{m}_{0,1:V,\text{type}}$	$\mathbf{0}$
$\kappa_{0,1:V,\text{type}}$	1×10^{-4}
$\mathbf{U}_{0,1:V,\text{type}}$	$\frac{1}{4} I_5$
$n_{0,1:V,\text{type}}$	11
Normal-Inverse–Wishart $(1:M)$	
$\mathbf{m}_{0,1:M,\text{rmm}}$	$\mathbf{0}$
$\kappa_{0,1:M,\text{rmm}}$	1×10^{-4}
$\mathbf{U}_{0,1:M,\text{rmm}}$	$625\, I_7$
$n_{0,1:M,\text{rmm}}$	15
Component–wise Dirichlet priors $(1:M)$	
$\mathbf{a}_{0,1:M,\text{type}}$	1×10^{-4}
$\mathbf{a}_{0,1:M,\text{interacting}}$	1×10^{-4}
$\mathbf{a}_{0,1:M,\text{o}}$	1×10^{-4}
$\mathbf{a}_{0,1:M,\text{tmm}}$	1×10^{-4}
$\mathbf{a}_{0,1:M,\text{action}}$	1×10^{-4}
$\mathbf{a}_{0,1:M,\text{reward}}$	1.0

of shape $210 \times 160 \times 3$, that corresponds to a RGB pixel arrays game screen. Agents interact via a set of 2 to 5 discrete actions for movement or game-specific interactions. As is standard practice, positive rewards (+1) are awarded for achieving objectives, while negative rewards (-1) are given for failures. Here is a brief description of the games:

Aviate. This environment puts the player in control of a bird, challenging them to navigate through a series of vertical pipes. The bird falls under gravity and can be made to jump by performing a "flap" action. The player's objective is to guide the bird through the narrow horizontal gaps between the pipes without colliding with any part of the pipe structure or the top/bottom edges of the screen. Any collision with a pipe, or going out of screen at the top or bottom results in a negative reward and ends the game.

Bounce. This environment simulates a simplified version of the classic game Pong, where the player controls a paddle to hit a ball against an AI-controlled opponent. The player has three discrete actions: move their paddle up, move it down, or keep it stationary, influencing the ball's vertical trajectory upon contact. The objective is to score points by hitting the ball past the opponent's paddle (reward +1), while preventing the opponent from doing the same (reward -1). The game is episodic, resetting once a point is scored by either side.

Cross. Inspired by the classic Atari game Freeway, this environment tasks the player, represented as a yellow square, with crossing a multi-lane road without being hit by cars. The player has three discrete actions: move up, move down, or stay in place, controlling vertical movement across eight distinct lanes. Cars of varying colors and speeds continuously traverse these lanes horizontally, wrapping around the screen. The objective is to reach the top of the screen for a positive reward; however, colliding with any car resets the player to the bottom of the screen and incurs a negative reward.

Driver. This environment simulates a lane-based driving game where the player controls a car from a top-down perspective, navigating a multi-lane road. The player can choose from three discrete actions: stay in the current position, move left, or move right, allowing for lane changes. The goal is to drive as far as possible, avoiding collisions with opponent cars that appear and move down the lanes at varying speeds. Colliding with another car results in a negative reward and ends the game.

Explode. In this game inspired by the arcade classic Kaboom!, the player controls a horizontal bucket at the bottom of the screen, tasked with catching bombs dropped by a moving bomber. The player can choose from three discrete actions: remain stationary, move left, or move right, allowing for precise horizontal positioning to intercept falling projectiles. A bomber continuously traverses the top of the screen, periodically releasing bombs that accelerate as they fall towards the bottom. Successfully catching a bomb in the bucket yields a positive reward, whereas allowing a bomb to fall off-screen results in a negative reward.

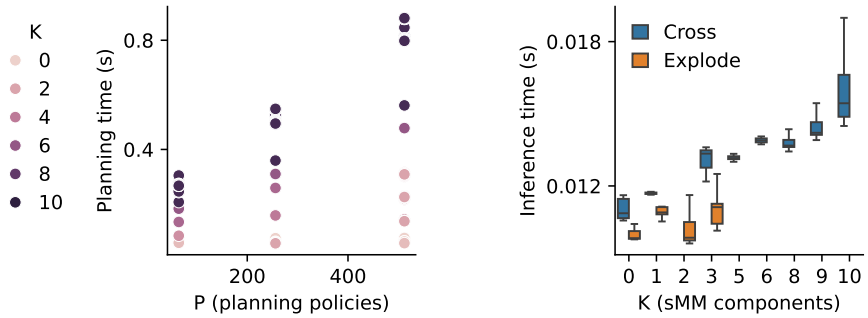


Figure 5: **Computational costs.** Scaling of planning time as a function of the number of policies (left), and model inference time as a function of the number of sMM components (right). All times measured on a single A100 GPU.

Fruits. This game casts the player as a character who must collect falling fruits while dodging dangerous rocks. The player can perform one of three discrete actions: move left, move right, or stay in place, controlling horizontal movement at the bottom of the screen. Fruits of various colors fall from the top, granting a positive reward upon being caught in the player's invisible basket. Conversely, falling rocks, represented as dark grey rectangles, will end the game and incur a negative reward if collected.

Gold. In this game, the player controls a character, represented by a yellow square, from a top-down perspective, moving across a grassy field to collect gold coins and avoid dogs. The player can choose from five discrete actions: stay put, move up, move right, move down, or move left, enabling agile navigation across the screen. Gold coins are static collectibles that grant positive rewards upon contact, while dogs move dynamically across the screen, serving as obstacles that end the game and incur a negative reward if collided with.

Hunt. This game features a character navigating a multi-lane environment, akin to a grid, from a top-down perspective. The player has four discrete actions available: move left, move right, move up, or move down, allowing full two-dimensional movement within the game area. The screen continuously presents a flow of items and obstacles moving horizontally across these lanes. The player's goal is to collect beneficial items to earn positive rewards while deftly maneuvering to avoid contact with detrimental obstacles, which incur negative rewards, encouraging strategic pathfinding.

Impact. This environment simulates the classic arcade game Breakout, where the player controls a horizontal paddle at the bottom of the screen to bounce a ball and destroy a wall of bricks. The player has three discrete actions: move the paddle left, move it right, or keep it stationary. The objective is to eliminate all the bricks by hitting them with the ball, earning a positive reward for each destroyed brick. If the ball goes past the paddle, the player incurs a negative reward and the game resets. The game ends when all bricks are destroyed.

Jump. In this side-scrolling endless runner game, the player controls a character who continuously runs forward, encountering various obstacles. The player has two discrete actions: perform no action or initiate a jump allowing the character to avoid different types of obstacles. Colliding with an obstacle results in a negative reward and immediately resets the game.

E Additional results and ablations

E.1 Baseline performance on 100K

Extending the wall-clock budget to 100 K interaction steps sharpens the contrast between model-based and model-free agents. On Hunt, **DreamerV3** fails to make measurable progress over the entire horizon, remaining near its random-play baseline, whereas **BBF** continues to improve and ultimately attains a mean episodic return on par with the score our object-centric agent already reaches after only 10 K steps. In Gold both baselines do learn within 100 K steps, but their asymptotic performance still plateaus below the level our agent achieves in the much shorter 10 K-step regime (see Figure 6).

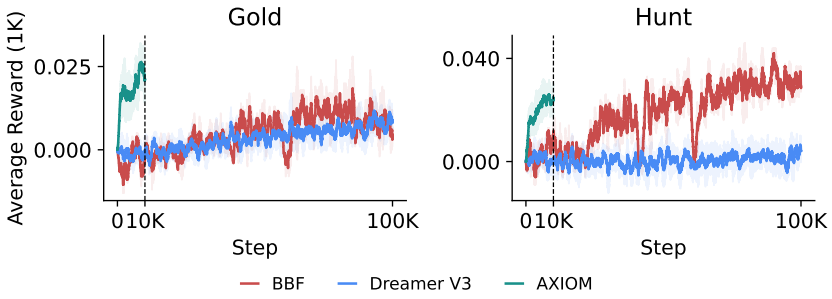


Figure 6: **100K performance on Gold & Hunt.**

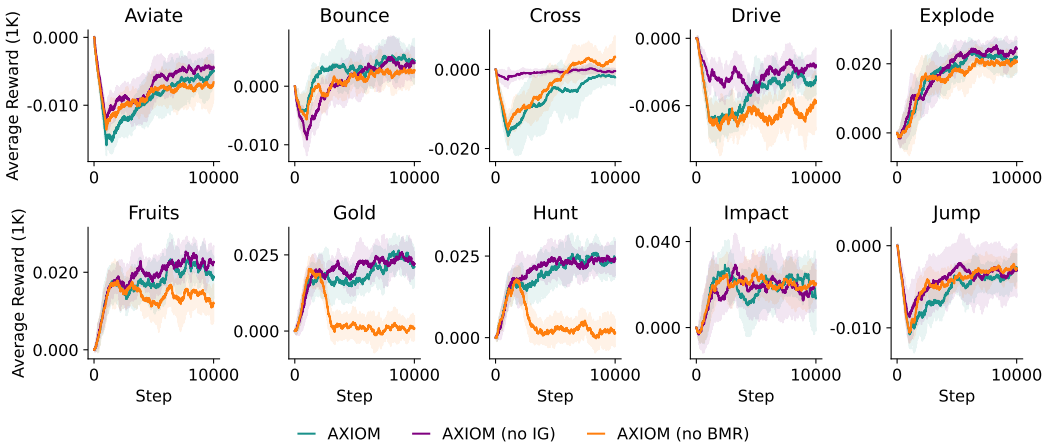


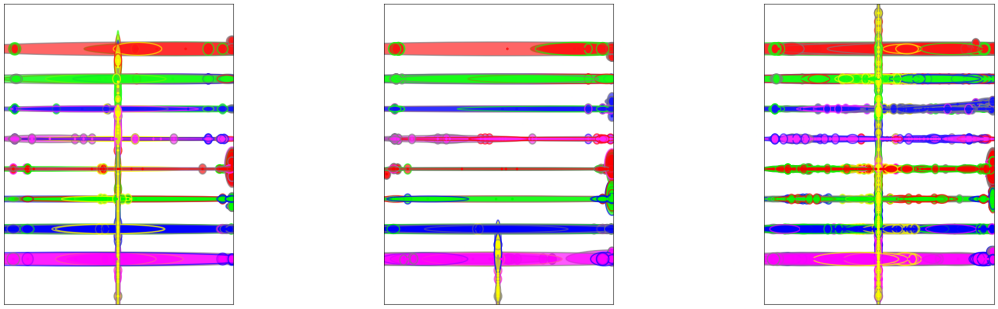
Figure 7: **Performance of AXIOM ablations**. Average reward over the final 1,000 frames across 10 `Gameworld 10K` environments for three AXIOM variants: the full AXIOM model, a version without Bayesian Model Reduction (AXIOM (no BMR)), and a version excluding information gain during planning (AXIOM (no IG)).

E.2 Ablations

No information gain. When disabling the information gain, we obtain the purple curves in Figure 7. In general, at first glance there appears to be little impact of the information gain on most games. However, this is to be expected, as in Figure 4c we showed that e.g. for Explode, the information gain is only driving performance for the first few hundred steps, after which expected utility takes over. In terms of cumulative rewards, information gain is actually hurting performance on most games where interactions between player and object result in a negative reward. This is because these interaction events will be predicted as information-rich in the beginning, encouraging the agent to experience these multiple times. This is especially apparent in the Cross game, where the no-IG-ablated agent immediately decides not to attempt crossing the road at all after the first few collisions. Figure 8 visualizes the created rMM clusters, which illustrates how no information gain kills exploration in Cross. We hence believe that information gain will play a more important role in hard exploration tasks, which is an interesting direction for future research.

No Bayesian Model Reduction. The orange curves in Figure 7 show the impact of disabling the Bayesian Model Reduction (BMR). BMR clearly has a crucial impact on both Gold and Hunt, which are the games where the player can move freely around the 2D area. In this case, BMR is able to generalize the dynamics and object interactions spatially by merging clusters together. The exception to this is once again Cross, where disabling BMR actually yields the best performing agent. This is again explained by the interplay with information gain. As BMR will merge similar clusters together, moving up without colliding will be assigned to a single, often visited cluster. This will render this cluster less informative from an information gain perspective, and the agent will be more attracted to collide with the different cars first. However, when disabling BMR, reaching each spatial location will get its own cluster, and the agent will be attracted to visit less frequently observed locations, like the top of the screen. This can also be seen qualitatively if we plot the resulting rMM clusters in Figure 8c. This begs the question on when to best schedule BMR during the course of learning. Clearly, BMR is crucial to generalize observed events to novel situations, but when done too early in learning, it can be detrimental for learning. Further investigating this interplay remains a topic for future work.

Planning rollouts and samples. As we sample rollouts at each timestep during the planning phase, there is a clear tradeoff between the number of policies and rollout samples to collect in terms of computation time spent (see Figure 5) and the quality of the found plan. We performed a grid search, varying the number of rollouts $[64, 128, 256, 512]$ and number of samples per rollout $[1, 3, 5]$, evaluating 3 seeds each. The results, shown in Figure 9 shows there are no significant performance differences, but more rollouts and drawing more than one sample seem to perform slightly better on



(a) AXIOM (b) AXIOM (no IG) (c) AXIOM (no BMR)

Figure 8: **Visualizations of the rMM clusters on Cross for information gain and BMR ablations.** Each Gaussian cluster depicts a particular dynamics for a particular object type, colored by the object color, and the edge color of a nearby "interacting" object. (a) AXIOM has various small clusters for the player object (yellow) interacting with the colored cars in the various lanes, and elongated clusters that model the player dynamics of moving up or down. (b) Without information gain, the player collides with the bottom most cars, and then stops exploring because of the negative expected utility. (c) Without BMR, all player positions get small clusters assigned, which in this case helps the player to cross, as visiting these locations is now rendered information gaining.

average. Therefore for our main evaluations we used 512 policies and 3 samples per policy, but the results in Figure 5 Figure 9 suggest that when compute time is limited, scaling the number of policies down to 128 or 64 is a viable way to increase efficiency without sacrificing performance.

E.3 Perturbations

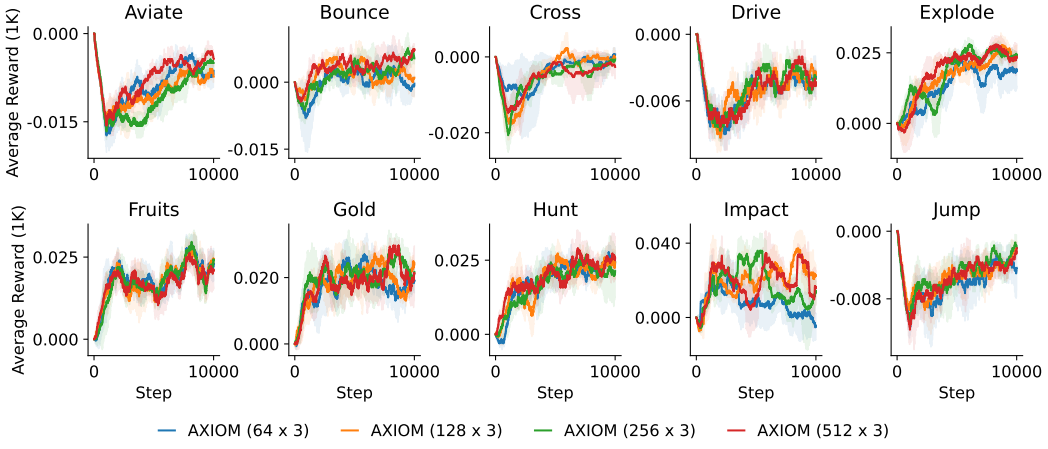
Perturbations. One advantage of the `Gameworld 10k` benchmark is its ability to apply homogeneous perturbations across environments, allowing us to quantify how robust different models are to changes in visual features. In our current experiments, we introduce two types of perturbations: a color perturbation, which alters the colors of all sprites and the background (see Figure 10b), and a shape perturbation, which transforms primitives from squares into circles and triangles (see Figure 10c).

To assess model robustness, we apply each perturbation halfway through training (at 5,000 steps) and plot the average reward for Axiom, Dreamer, and BBF across each game in Figure 11. Under the shape perturbation, Axiom demonstrates resilience across games. We attribute this to the identity model (**iMM**), which successfully maps the new shapes onto existing identities despite their altered appearance. Under the color perturbation, however, Axiom's performance often drops - suggesting the identity model initially treats the perturbed sprites as new objects - but then rapidly recovers as it reassigns those new identities to the previously learned dynamics.

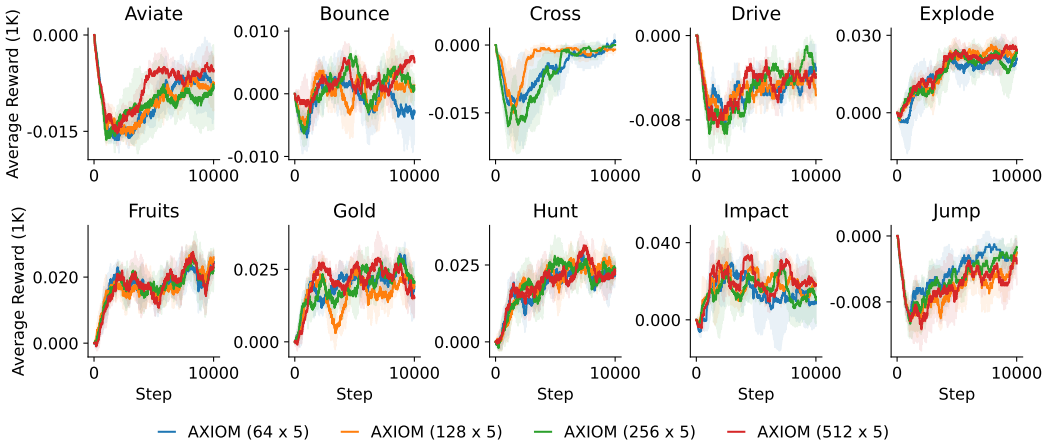
Our results also show that BBF and Dreamer are robust to shape changes. For the color perturbation, Dreamer - like Axiom - sometimes experiences a temporary performance decline (for example, in Explode) but then recovers. BBF, by contrast, appears unaffected by either perturbation. We hypothesize that this resilience stems from applying the perturbation early in training - before BBF has converged - so that altering visual features has minimal impact on its learning dynamics.

Remapped slot identity perturbations In this perturbation, shown by the purple line in Figure 11, we performed a special type of perturbation to showcase the 'white-box', interpretable nature of AXIOM's world model. For this experiment, we performed a standard 'color perturbation' as described above, but after doing so, we encode knowledge about the unreliability of object color into AXIOM's world model. Specifically, because the latent object features learned by AXIOM are directly interpretable as the colors of the objects in the frame, we can remove the influence of the latent dimensions corresponding to color from the inference step that extracts object identity (namely, the inference step of the iMM), and instead only use shape information to perform object

(a)

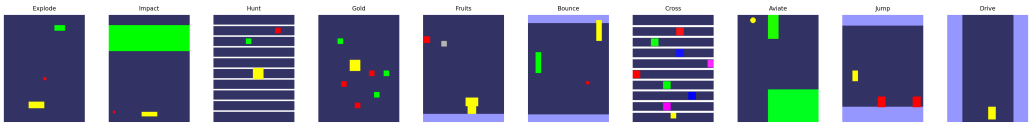


(b)

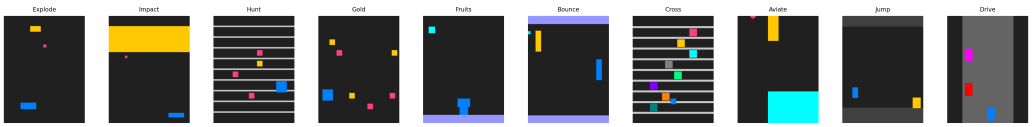


(c)

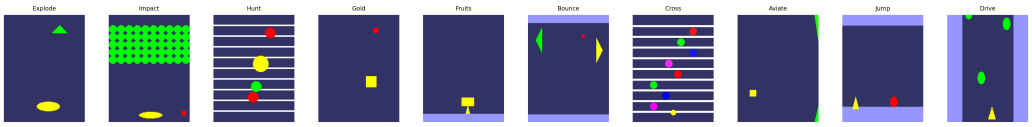
Figure 9: **Ablation on the amount of sampled policies.** The label indicates the number of policies $\times$ number of samples for that policy (a) 1 Sample (b) 3 Samples (c) 5 Samples



(a) No perturbation.



(b) Color perturbation (sprites and background recolored).



(c) Shape perturbation (primitives replaced with circles and triangles).

Figure 10: **Perturbations.** Sample frames from each of the ten environments under (a) no perturbation, (b) a color perturbation, and (c) a shape perturbation.

type inference. In practice, what this means is that slots that changed colors don't rapidly get assigned new identities, meaning the same identity-conditioned dynamics (clusters of the rMM) can be used to predict and explain the behavior of the same objects, despite their color having changed. This explains the absence of an effect of perturbation for some games when using this 'color remapping' trick at the time of perturbation, especially the ones where object identity can easily be inferred from shape, such as Explode. Figure 12 shows the iMM identity slots, with and without the 'remapping trick'. Impact on performance for all games is shown in 11d). For games where certain objects have the same shape (e.g., rewards and obstacles in Hunt, or fruits and rocks in Fruits), this remapping trick has no effect because shape information alone is not enough to infer object type and thus condition the dynamics on object types. In such cases, one might use more features to infer the object identity, such as position or dynamics, but extending our model to incorporate these to further improve robustness is left as future work.

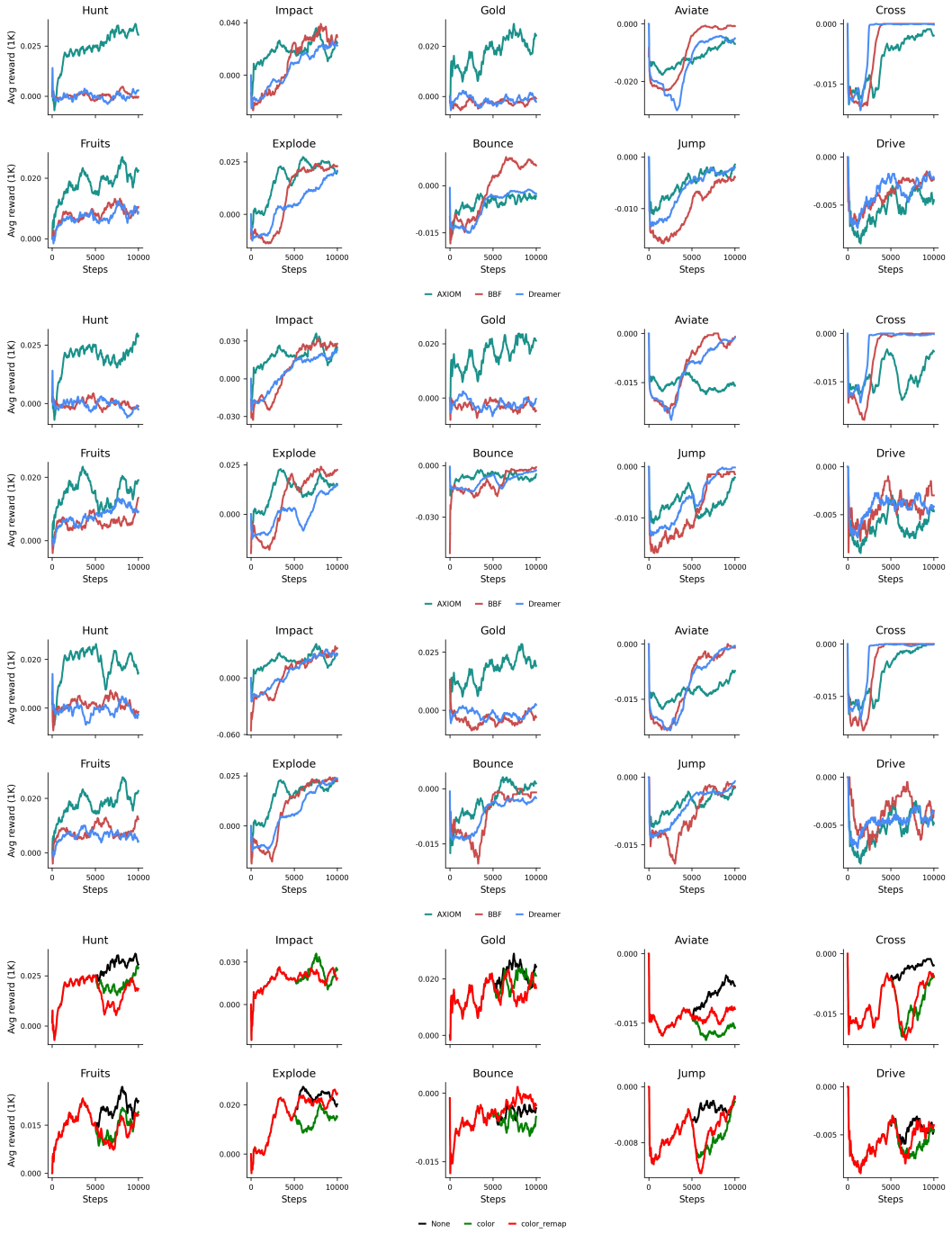
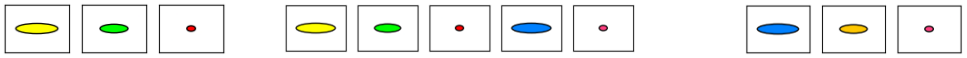


Figure 11: **Impact of perturbations on average reward.** Smoothed 1k-step average rewards for Axiom, BBF, and Dreamer across ten games under (a) no perturbation, (b) color perturbation, (c) shape perturbation, and (d) Axiom's color perturbation with and without remapping.



(a) no perturbation (b) color perturbation (c) color perturbation with remap

Figure 12: **iMM identity slots on Explode.** (a) On Explode, the iMM constructs a slot for the player, bomber and bomb respectively. (b) When color is perturbed, novel slots are created for the blue (player) and the pink bomb, and the yellow enemy is mapped onto the old player slot. (c) However, with color remapping, the blue player, yellow bomber and pink bomb are correctly remapped to the old player and bomber slots based on their shape.

F Related works

Object-Centric World Models. The first breakthroughs in deep reinforcement learning, that leveraged deep Q networks to play Atari games [39], were not model-based, and required training on millions of images to reach human-level performance. To this end, recent works have leveraged model-based reinforcement learning, which learns world models and hence generalizes using fewer environment interactions [40, 41]. A notable example is the Dreamer set of models, which relies on a mix of recurrent continuous and discrete state spaces to model the dynamics of the environment [36, 42, 43]. This class of world models simulates aspects of human cognition, such as intuitive understanding of physics and object tracking [5, 7]. To this end, it is possible to add prior knowledge to this class of architectures, in a way that specific structures of the world are learned faster and better. For example, modeling interactions at the object level has shown promising performance in improving sample efficiency, generalization, and robustness across many tasks [9–12].

In recent years, the field of object segmentation has gained momentum thanks to the introduction of models like IODINE [44] and Slot Attention [45], which leverages the strengths and efficiency of self-attention to enforce competition between slot latents in explaining pixels in image data. The form of self-attention used in slot attention is closely-related to the E- and M-steps used to fit Gaussian mixture models [46, 47], which inspired our, where AXIOM segments object from images using inference and learning of the Slot Mixture Model. Examples of improvements over this seminal work include Latent Slot Diffusion, which improves upon the original work using diffusion models and SlotSSM [48] which uses object-factorization not only as an inductive bias for image segmentation but also for video prediction. Recent works that have also proposed object-centric, model-based approaches are FOCUS, that confirms how such approaches help towards generalization in the low data regime for robot manipulation [49], and OC-STORM and SSWM, that use object-centric information to predict environment dynamics and rewards [14, 50]. To conclude, SPARTAN proposes the use of a large transformer architecture that identifies sparse local causal models that accurately predict future object states [13]. Unlike OC-STORM, which uses pre-extracted object features using a pre-trained vision foundational model and segmentation masks, AXIOM learns to identify segment objects online without object-level supervision (albeit so far we have only tested AXIOM on simple objects like monochromatic polygons). AXIOM also grows and prunes its object-centric state-space online, but like OC-STORM plans using trajectories generated from its world model.

Bayesian Inference. Inference, learning, and planning in our model are derived from the active inference framework, that allows us to integrate Bayesian principles with reinforcement learning, balancing reward maximization with information gain by minimizing expected free energy [15, 16]. To learn the structure of the environment, we drawn inspiration from fast structure learning methods [24], that first add mixture components to the model [51] and then prunes them using Bayesian model reduction [21, 22, 24]. Our approach to temporal mixture modeling shares conceptual similarities with recent work on structure-learning Gaussian mixture models that adaptively determine the number of components for perception and transition modeling in reinforcement learning contexts [52]. An important distinction between AXIOM's model and the original fast structure learning approach [23], is that AXIOM uses more structured priors (in the form of the object-centric factorization of the sMM and the piecewise linear tMM), and uses continuous mixture model likelihoods, rather than purely discrete ones. The transition mixture model we use is a type of truncated infinite switching linear dynamical system (SLDS)[29, 53, 54]. In particular, we rely on a recent formulation called the recurrent SLDS [19], that introduces dependence of the switch state on the continuous state, to address two key limitations of the standard SLDS: state-independent transitions and context-blind dynamics. Our innovation is in how we handle the recurrent connection of the rSLDS: we do this

using a *generative*, as opposed to *discriminative*, model for the switching states. This allows for more flexible conditioning of the switch state on various information sources (both continuous and discrete), as well as a switch dependence that is quadratic in the continuous features; this overcomes the intrinsic linear separability assumptions made by using a classic softmax likelihood over the switch state, as used in the original rSLDS formulation [19, 55].