Universiteit Leiden

Master Computer Science

On the efficiency and robustness of structured world models combined with pre-learned object-centric representations

Name:	Jonathan Collu
Student ID:	s3169952
Date:	July 11, 2023
Specialisation:	Artificial Intelligence
1st supervisor:	Thomas Moerland
2nd supervisor:	Aske Plaat

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The ability to perceive and reason about individual components in a visual scene is a fundamental aspect of human cognition, enabling a robust understanding of the world and its dynamics. Replicating these abilities in artificial systems would mean reaching a significant milestone toward building intelligent agents. While the last few years have been characterized by an increasing intensity of publications in the object-centric representation learning area, the current solutions still present non-negligible constraints. Most of the proposed architectures tend indeed to represent per-object information in ways that do not consent to isolate specific features such as position, angle, scale, shape, and color. Other methods instead succeed in shaping a disentangled and interpretable latent space, though they are particularly expensive in terms of data and gradient steps, even on simple synthetic datasets. These drawbacks probably contributed to limiting the adoption of such techniques in fields such as reinforcement learning and planning, that in contrast, would strongly benefit from encoding the visual information in a structured and interpretable way. Nonetheless, various approaches either using pre-learned object-based representations (as [33] in model-fre reinforcement learning and [30] in planning) or learning structured encoding (as C-SWM [13]) arose. The latter presents a class of world models enabling learning entities' representations and interactions, though these models cannot disambiguate multiple instances of the same object and have not been tested in environments requiring learning more features than just the position. This thesis proposes Slot Attention-based Structured World Models (SA-SWM), a class of world models replacing the C-SWM object extractor with a pretrained Slot Attention encoder, and provides an augmented version of Spriteworld [29] to challenge object-based world models with more complex dynamics. The results showed that SA-SWMs drastically improve the performances and generalization abilities of C-SWMs, which showed to be particularly prone to overfitting in the proposed environment. The absence of disentanglement in Slot Attention encodings leaves space for a wide margin of improvement, as it compromises their utility for relational reasoning. The latter undesired fact prevented us from collecting meaningful observations on the involvement of Slot Attention and SA-SWMs in a reinforcement learning setting.

Contents

1 Introduction

Being able to distinguish individual components of a visual scene and reason in terms of their features and relations allows humans to build a solid comprehension of the environment they are part of. Hence, such abilities are considered fundamental aspects of human cognition [25], and therefore a necessary step to take in order to build intelligent artificial systems.
In line with these preliminaries, the number of computer vision approaches focusing on unsupervised object discovery significantly grew in the last few years. Works such as [17, 14, 4, 16, 3, 24, 9], took important steps toward this goal, and methods such as [4, 16] have already been involved respectively in planning and reinforcement learning tasks.
The usage of such structured representations has been hypothesized to bring numerous advantages to reinforcement learning; prominent among the others are improved sample efficiency and generalization to novel and out-of-distribution scenes. Works such as [33, 8, 10, 34], provided empirical shreds of evidence of these assumptions.
As the environmental dynamics are usually task-independent, learning a structured model of the environment would bring a boost to generalization. Hence such a model would allow an artificial agent to develop knowledge regarding the (non-task-specific) consequences of its action on the rest of the environment. Such knowledge can be exploited to learn new tasks with no need to re-learn how the environment works every time the objective changes. Although affected by several limitations, the most notable approaches to learning an object-based structured world model are, to the best of our knowledge, COBRA [30] and C-SWM [13]. For instance, the visual model adopted in COBRA is [4], which produces accurate and disentangled per-object representations, being at the same time far less training and data efficient than more recent SOTA approaches based on Slot Attention [16] (which, however, offer a latent space much less interpretable). The C-SWM visual module instead does not manage to disambiguate multiple instances of the same object and struggles to encode complex object features using just the contrastive objective. Moving the focus on the transition models, COBRA presents a simple MLP-based model applied object-wisely that cannot model object interactions. This limitation could be ignored in the experiments in [30], as the Spriteworld environment [29] used to test the method does not include any form of physics. The C-SWM transition model instead is implemented as a graph neural network (GNN) [22] and can therefore model object interactions. Nevertheless, the environment involved in the experimentation presents very simple physics (e. g collisions do not cause the movement of static objects) that can be modeled with a single round of message passing. To overcome these limitations we propose SA-SWMs, a class of structured world models inherited by C-SWMs that removes the contrastive objective and replaces the vision model with a pretrained Slot Attention encoder. We further enhance the transition model through an iterative message-passing mechanism that allows learning complex object interactions and dynamics. In this regard, we extended Spriteworld with rudimental physics, preventing object occlusion and allowing an embodied agent to carry multiple objects at once and in a chain. The results obtained show that, in addition to overcoming the instance disambiguation problem, SA-SWMs drastically improve C-SWMs generalization capabilities as the pre-learned object embeddings are much more informative

than those learned contrastively. Nonetheless, the object representations offered by Slot Attention do not separate the information related to specific features (e. g position, color, shape, scale, angle, etc...) into specific vector components, but instead mix this information. In this way, slight transitions in space can determine significant and sometimes unpredictable shifts in the latent space limiting generalization and accuracy in predicting future environment states. This kind of representation does not allow us to grasp many of the features characterizing an object and, as the reward functions of the tasks investigated rely on these features, it was not possible to learn an object-based policy to solve such tasks.

The document is structured as follows: Section 2 recalls concepts useful to understand the rest of the document and introduces the notation adopted; Section 3 offers a wide overview of previous works related with object-centric representation learning and its involvement in reinforcement learning; Section 4 describes the environment and the tasks proposed; Section 5 introduces the methodology; Section 6 furnish quantitative and qualitative measures of SA-SWMs and C-SWMs performances; and Section 7 draws extensive conclusion regarding the observations.

2 Background and preliminaries

This section provides the reader with concepts and notation useful to understand the rest of the document. Specifically, it provides an overview of the reinforcement learning objectives and methods with a specific focus on the reason behind the need for planning over a learned model of the environment. After these preliminary, we provide a description of the methods on which we built the ones proposed in this work.

Reinforcement Learning Reinforcement Learning is a machine learning paradigm whose aim is to build autonomous agents able to take decisions finalized to the reaching of a goal. Different from supervised learning, this approach does not rely on labeled data and instead needs interactions with an environment whose feedback measures the usefulness of actions based on the context. Formally, reinforcement learning addresses Markov Decision Process (MDP) optimization problems. MDPs are defined [20] by the tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, p(s_0), \gamma\}$ where: $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to p(\mathcal{S})$ is the transition function of the environment, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the reward function, and γ is a discount factor. The initial state of the environment is sampled from the distribution $p(s_0)$, then at each timestep $t \in \mathbb{N}$ the agent executes an action a_t and the environment returns the next state $s_{t+1} \sim \mathcal{T}(\cdot \mid s_t, a_t)$ and a reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$. The function with which the agent selects its action is known as the policy function and is defined as $\pi : \mathbf{S} \to p(\mathcal{A})$.

$$\tau_0^n = \{s_0, a_0, r_0, s_1, ..., s_n, a_n, r_n, s_{n+1}\} \tag{1}$$

Denotes an n-steps long trajectory (trace) starting from the initial state and

$$\mathcal{R}(\tau_t^N) = \sum_{i=0}^{N} \gamma^i \cdot r_{t+i} \tag{2}$$

defines the discounted cumulative reward of an N-steps long trace starting from an arbitrary timestep t.

The value function $V^\pi : \mathcal{S} \to \mathbb{R}$ expressed as:

$$V^\pi(s) = \mathbb{E}_{\pi,\mathcal{T}} \left[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} \middle| s_t = s \right] \tag{3}$$

ndicates the expected cumulative reward following the policy π from state s, while the action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ expressed as:

$$Q^\pi(s, a) = \mathbb{E}_{\pi,\mathcal{T}} \left[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} \middle| s_t = s, a_t = a \right] \tag{4}$$

indicates the expected cumulative reward following the policy π from state s' after taking action a in the state s.

Given these preliminaries, we can now move on to how we can build and optimize a policy. In this regard, we can distinguish between two approaches:

- **Value-Based**: in this approach, the policy is not explicitly learned. Instead, we optimize a Q-function, and the policy is derived as $\pi(s) = argmax_a(Q(s,a))$. The Q-function can be stored in a table (Tabular RL) or approximated by a neural network (Deep RL). This approach is only suitable for discrete action spaces. A very popular method belonging to this class is DQN [18].
- **Policy-Based**: in this approach the policy is explicitly defined and optimized. This approach is suitable for both discrete and continuous action spaces. In this class of methods we find (among the others) REINFORCE [32], Actor-Critic [15], and PPO [23]. The latter is the method adopted in this work and is introduced in the next paragraph.

Proximal Policy Optimization Proximal Policy Optimization (PPO) is a class of policy gradient methods introduced in [23] to overcome the limitations of classical policy gradient methods. Specifically, the objective of classical policy-based algorithms is to maximize a function

$$J(\theta) = \frac{1}{M}\sum_{i=1}^{M} \mathbb{E}_{\tau_i \sim \pi_\theta(\tau_i)} \left[\hat{A}_t \cdot \log \pi_\theta(a_t|s_t) \right] \tag{5}$$

where M is the number of trajectories τ_i sampled from the policy π parametrized by θ, and $\hat{A}_t$ is an arbitrary advantage function. As highlighted in [23], optimizing such a function empirically leads to very large policy updates, and the large magnitude of policy changes limits sample efficiency.

To overcome this limitation and improve sample efficiency over the classical policy gradient methods, PPO introduces a simple and elegant surrogate objective defined as follows:

let the ratio

$$\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{6}$$

be denoted as $r_t(\theta)$. Since the goal is to prevent action probabilities from increasing or decreasing "too much" after a gradient update, the goal is to keep $r_t(\theta)$ close to one. To do this, the authors of [23] introduce a clip term in the objective function to ensure $r_t(\theta)_{clipped}$ differs at most ϵ from 1. Then taking the minimum between $r_t(\theta)\hat{A}_t$ and $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)$ the objective

discourages taking large steps. The clipped surrogate objective is then defined as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{old}}(\tau)} \left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{7}$$

where ϵ is a hyperparameter.

Model-based reinforcement learning As clear at this point of the reading, reinforcement learning is a useful tool designed to optimize a value or a policy function (or both, as in Actor-Critic). Optimizing such functions allows agents to "understand" how their actions impact the reward they receive, but not to learn their consequences on the environment. In most of the MDPs, the transition function is unknown to the agent, and being unaware of it leads to two undesired consequences:

- the agent cannot predict the next state and can only interact with the environment to collect the experience necessary to learn how to act in a way that maximizes the reward it gets;
- this way the agent can only move forward in time and cannot backtrack its steps as they alter the environment irreversibly.

Having access to a model of the environment, on the other hand, would allow the agent to make guesses about the next state and most importantly, the agent can interrogate the model instead of taking irreversible steps. In this way, it is possible to retrace the steps taken to plan and select the action that leads to the best outcome in the long run.

As planning over an entire state space is unfeasible, this class of approaches can only store a solution that applies just to a local subset of states. In contrast, reinforcement learning methods (assuming an adequate exploration) can approximate the entire state space and store a global solution.

The authors of [19] define Model-based reinforcement learning methods as the combination of planning and learning or more formally as a class of MDP algorithms that make use of a model and store a global solution.

For a taxonomy of planning methods and their integration with learning approaches, we refer the reader to [19], as in the rest of this section, we will only describe the one adopted in this work.

Monte Carlo Tree Search Monte Carlo Tree Search (MCTS) is a class of adaptive search algorithms used to approximate the utility of an action as the average reward of multiple simulated traces starting by taking that action from the current state.

MCTS algorithms can be simply explained as an iterative process involving the four steps described below and shown in Figure 1:

1. **Selection**: in this phase, the algorithm selects the next node to expand by visiting the current tree. The selection criterion, also known as tree policy is usually the Upper Confidence Bounds (UCB) strategy defined as:

$$UCB = \bar{X}_j + C_p\sqrt{\frac{ln(n)}{n_j}} \tag{8}$$

where n is the total number of visits made to the parent node, and n_j is the number of visits made to the child node j. C_p is a positive constant value controlling the degree of exploration. The left term of this formula represents exploitation and prioritizes choices that are known to lead to a higher reward, whereas the right term, is responsible for the exploration prioritizing the less visited nodes. The next node to expand is the one that maximizes the equation 8.

2. **Expansion**: the tree is expanded by adding one or several child nodes of the selected node.
3. **Rollout**: in this phase, we simulate an arbitrarily long trajectory (it can be an entire episode or a fixed number of steps) from the expanded node.
4. **Backpropagation**: the outcome of the simulation is backpropagated from the expanded to the root node.

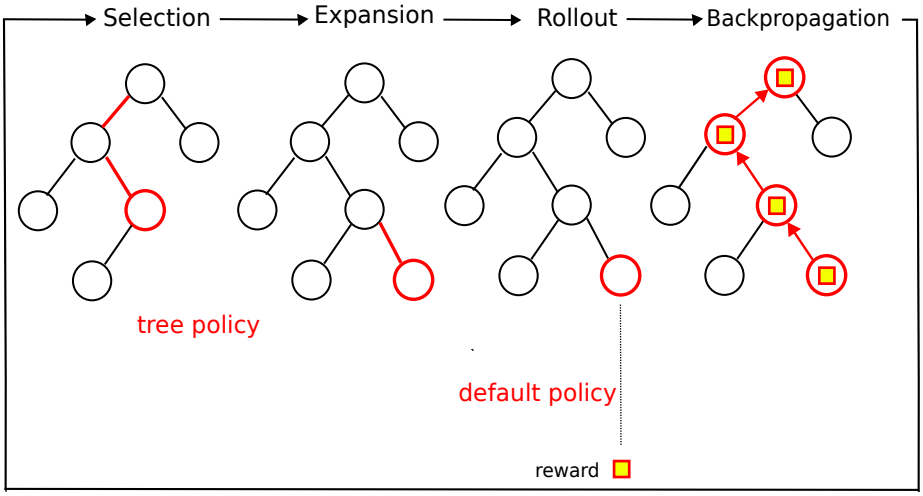

Figure 1: Visual progression of the MCTS iterative process. In the first phase, the algorithm selects a node present in the current tree based on Equation 8 (tree policy). At this point, the tree is expanded by adding a child node to the selected one and an n-step long simulation started from that node is performed. Such simulation is performed by exploiting the default policy and the reward estimated is then backpropagated up to the root.

C-SWM As in most cases the transition function of an environment is unknown, learning a model that approximates this function is a key challenge and a crucial step to take in order to build a model-based algorithm. Contrastive Learning of Structured World Models (C-SWM) [13] introduces a method to learn

a relational model of the environment based on the interactions of individual objects. The architecture presented is structured in three different sub-modules: object extractor E_{ext}, object encoder E_{enc}, and relational transition model. The object extractor is a function $E_{ext}(x) = m_1, ..., m_k$ mapping an RGB image x into a set of $\mathcal{K}$ feature maps meant to represent a mask binding to one of the $\mathcal{K}$ objects. It is implemented as a fully-convolutional network whose last layer produces at least $\mathcal{K}$ feature maps (at least one per object) and is activated by sigmoid. The object encoder is a function $E_{enc}(m_i) = z_i$ embedding each of the $\mathcal{K}$ masks to a latent vector $z_i \in \mathbb{R}^{\mathcal{D}}$, implemented as an MLP. The relational transition model is a function $T(z_t^k, a_t^k) = \Delta z_t^k$ computing the update for each of the abstract representation $z_{t,k}$ given an action $a_{t,k}$ associated. It is implemented as a Graph Neural Network (GNN) [22] using a single round of the following message passing updates:

$$e_t^{(i,j)} = f_{edge}([z_t^i, z_t^j]) \tag{9}$$

where $e_t^{(i,j)}$ is the edge (the relationship) between the nodes z_t^i and z_t^j, and $f_{edge}([z_t^i, z_t^j])$ is a simple MLP taking as input the concatenation of the vectors z_t^i and z_t^j.

$$\Delta z_t^j = f_{node}([z_t^j, a_t^j, \sum_{i \neq j} e_t^{(i,j)}]) \tag{10}$$

where Δz_t^j is the update of the object z_t^j given the application of action a_t^j and the sum of its incoming edges and f_{node} is an MLP.

The entire model is trained to minimize a loss function

$$\mathcal{L} = H + max(0, \gamma - \tilde{H}) \tag{11}$$

$$H = \frac{1}{K} \sum_{k=1}^{K} d(z_t^k + \Delta z_t^k, z_{t+1}^k), \quad \tilde{H} = \frac{1}{K} \sum_{k=1}^{K} d(\tilde{z}_t^k, z_{t+1}^k) \tag{12}$$

with H being the average Euclidean distance between the predicted next states (per object) $z_t^k + \Delta z_t^k$ and the encoding of the real next states z_{t+1}^k. This part of the objective is used to optimize the performance of the transition model. $\tilde{H}$ instead, is the average Euclidean distance between the encoding of a state $\tilde{s}_t$ randomly sampled from the experience replay. This part of the objective is meant to prevent the encoder from representing different states in a similar way.

Slot Attention The Slot Attention module proposed in [16] takes as input a set $\mathcal{X} \in \mathbb{R}^{\mathcal{N} \times \mathcal{D}_{input}}$ of $\mathcal{N}$ feature vectors augmented with positional embeddings, and produces a set $\tilde{\mathcal{S}} \in \mathbb{R}^{\mathcal{K} \times \mathcal{D}}$ of $\mathcal{K}$ slots. The feature vectors are projected in a $\mathcal{D}$-dimensional space through learned linear transformations k and v (respectively keys and values), while the slots (queries) are independently sampled from a Normal distribution defined by learnable parameters $\mu, \sigma \in \mathbb{R}^D$. The slots are then refined through an iterative mechanism involving multiple steps per iteration.

The first step consists in computing the attention matrix

$$\mathcal{A} = Softmax(\frac{k(x) \cdot q(s)^T}{\sqrt{\mathcal{D}_{slot}}}) \tag{13}$$

where x are the input feature vectors and s the sampled slots. The dot product between the input features and the slots relates each slot with parts of the image, then the resultant coefficients are normalized by softmax over the queries to ensure competition between the slots for explaining part of the image. At this point, the updates are obtained by combining the input values with the attention matrix normalized over the slots:

$$\mathcal{U} = W^T \cdot v(x), \quad W_{i,j} = \frac{\mathcal{A}_{i,j}}{\sum_{l=1}^{\mathcal{N}} \mathcal{A}_{l,j}} \tag{14}$$

The final updates of each iteration are obtained by a passage of Gated-recurrent-Unit GRU followed by an MLP with a residual connection.

In order to perform object discovery, the slots are decoded into images and masks that are then combined and summed up to obtain a single image. The training objective is therefore to minimize the mean squared error between the input image and the reconstructed one.

3 Related Work

Most existing deep learning-based approaches addressing sequential decision-making problems rely on unstructured and global visual inputs. The idea that disentangled representations of individual entities might enhance the way artificial agents learn to solve tasks, however, aroused the interest of several authors across time. A fascinating perspective on the role objects play in human cognition has been presented in [27]. This work focuses on the requirements object representations must fulfill to facilitate individual entities' discovery and the understanding of their dynamics, highlighting the limitations of current neural network architectures in satisfying these points. Encouraging results regarding the benefits structured representations may bring to reinforcement learning have been produced by [35, 6, 12]. Specifically, in [6], every possible state of an environment is represented in terms of objects and their interaction through First Order Logic (FOL). These perfect representations of states brought significant improvements, especially in sample efficiency, although they are suitable for only fully-deterministic finite MDPs and low-dimensional environments. [12] instead proposes Schema networks modeling causal-effect interactions between entities through Boolean logic operations. Such an approach allowed for zero-shot transfer learning on multiple altered versions of the environments the authors experimented with, providing empirical confirmation of the importance of modeling causal connections between individual entities. As the method proposed in [6], this does not scale well on more complex and high-dimensional environments. While these results suggest the usefulness of modeling features and dynamics of single objects, none of the described works provided a method to learn this ability. On the other hand, [35] proposes a reinforcement learning agent augmented with a graph network-based relational module. This module allows the agent to grasp meaningful insights about the relationships between individual entities, enhancing its generalization capabilities across novel scenes.

Object-centric representation learning As it appears logical, reasoning about an environment in terms of objects and their relationships requires the ability to disentangle a scene into its single components and represent them individually. For this reason, it is relevant to provide the reader with an overview of different computer vision approaches dealing with object-centric representation learning.

Recently, this field of research has been enriched with numerous proposals to approach the problem. Unsupervised methods such as the ones presented in [17, 14, 4, 16] encode input images into a set of latent vectors, each binding to a single entity in the represented scenes. Among the mentioned methods, the architecture presented in [16] obtained better or aligned results with respect to the others showing to be, at the same time, less expensive in terms of data and training time. For these characteristics, [16] appears to be a good candidate for the purpose of this work. Works such as [3, 24, 9] propose different extensions for [16] obtaining state-of-the-art performances.

Object-centric representations in model-free Reinforcement learning Over the last few years, unsupervised object-centric representations found application in model-free reinforcement learning settings. An interesting investigation

on the effect these representations can have in learning multi-task policies is conducted in [34]. This work proposes an architecture composed by the SCALOR[10] encoder and a goal-conditioned attention policy. The encoder produces multiple latent variables (binding to a single object in the input image) that are fed together with the embedded goal to the policy network (which predicts actions based on the state representations and the goal vector). The results showed that encoding individual objects instead of the global scene improves the learning of multi-task policies. More recent approaches such as [33, 8] investigated the impact of unsupervised object representations in model-free reinforcement learning using PPO[23] to optimize the policy. Specifically, [8] compared the performances obtained(in terms of mean reward) by an agent when the visual input is encoded by a CNN, Slot Attention[16], and a GNN (based on the architecture presented in [2]). All of these models have not been pre-trained on visual tasks. The results suggest that GNNs produce better encoding than CNNs and Slot Attention in the environment tested, although more exhaustive experiments are needed to grasp more meaningful insights. The investigation the authors of [33] carried out instead focuses on the effect of pre-trained object-centric representations in reinforcement learning. In particular, this study tries to answer several questions about the utility of representing scenes as a composition of multiple objects.

Object-centric representations in model-based Reinforcement learning Learning to predict future states based on the dynamics of single entities would allow artificial agents to create accurate models of environments. An empirical example is provided by [28], which proposes an architecture composed of a vision module that infers a set of entity variables from the input and a dynamic module that predicts their transitions based on a sequence of actions. This architecture reaches state-of-the-art performances on tasks such as video prediction.

Works such as [26, 31] yield an example of how individual object representations are fundamental to learning environment physics. More specifically, [26] aims to learn common-sense physical reasoning through relational neural expectation maximization. [31] instead models the dynamics of physical systems to predict future trajectories and is able to infer the mass of invisible objects by their effect on the rest of the system.

Approaches closer to the scope of this work are COBRA[30] and C-SWM[13]. Both methods learn to discover objects in an unsupervised way and have a transition model that predicts the transition of each detected object given the actions associated. While in [30], MONet[4] (optimized through a pixel-based loss) performs the object discovery, [13] adopts a visual model consisting in a simple CNN whose optimization process does not involve pixel-based loss functions. Regarding the transition model, in [30] is implemented as an MLP, whereas in [13] as a graph neural network, meaning that the latter is able to model interactions between objects while the former is not.

4 Environments and tasks

As the main objective of this work is to investigate whether object-centric representations help understanding environment dynamics and learning behaviors, it is crucial to experiment with an environment whose scenes can be represented by Slot Attention. An ideal environment for this scope should present simple scenes composed of objects that possibly do not occlude each other. Even though Slot Attention can learn impressively accurate representations of scenes containing complex and overlapping objects (as shown in [16] with datasets such as Multi-dSprites and Clever [11]), simple scenes are still preferred as they are likely to be easier to represent and require less training time (and therefore computational effort).

The need for such type of environment has characterized other related research such as those conducted by the authors of [30], who have developed Spriteworld [29], which has been used in the more recent study proposed in[33].

Spriteworld is a simple 2-dimensional and squared world with a black-colored background. Its scenes can present a variable number of objects named sprites that can appear under a wide range of shapes, colors, sizes, and inclinations. The environment does not include physics and presents three different classes of tasks:

- **Goal Finding** where the agent has to carry a target sprite to a fixed goal position;
- **Clustering** where the agent has to group the sprites by their color;
- **Sorting** where the agent has to group the sprites by their color in a specific location of the map.

All of the mentioned tasks have a select-and-move action space, meaning that the agent is external to the world and, at each timestep, selects and moves a single sprite. Since the environment does not include physics a sprite moving toward another causes an occlusion. This is an undesired property as occlusions can challenge the visual model, whereas having interactions would make the application of C-SWM more interesting since the results obtained in [30] show that a simple MLP is enough to model the Spriteworld dynamics.

Interactive Spriteworld As explained in the above paragraph, Spriteworld is a valid environment to experiment with object-centric reinforcement learning [1], and to the best of our knowledge, is the only environment built for this purpose. Nevertheless, the absence of physical interactions between sprites does not allow us to fully investigate the potential of the GNN-based transition model proposed by [13]. For this reason, we extended the environment with rudimental physic rules that allow sprites interactions and avoid object occlusions. The more notable properties of our extension (which we refer to as Interactive Spriteworld) are:

[1] with this term we refer to all of the reinforcement learning approaches using object-centric encoders

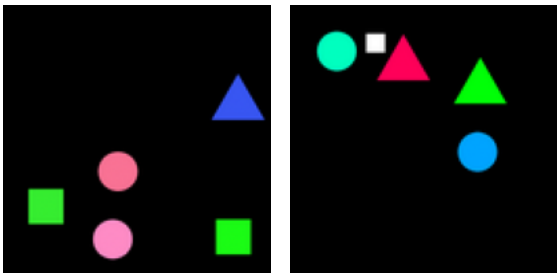

Figure 2: The left side of the figure illustrates a state sampled from the original environment, while on the right we can observe one sampled from our extended version. The sprite characterized by a smaller size and white-colored identifies the agent.

- the position of the sprites is clipped to keep them entirely in the frame (in the original version part of an object could be out of frame as its position is clipped to keep just its center of mass inside);
- when a moving sprite collides with one or more other objects, they move accordingly (a moving sprite transfer its motion to the objects it hits);
- the motion of an object propagates transitively to other objects (meaning that when an object moves a sprite towards another, the latter is moved consequently);
- the sprites never overlap;
- different from the original Spriteworld, an embodied agent can move more than one sprite per timestep.

Since these extensions required the implementation of an accurate collision detection mechanism and the modeling of physical interaction for each couple of shapes, we decided to limit the sprites to either circles, squares, or equilateral triangles; the latter can appear only in orientations where one of their edges is perfectly parallel with the x-axis.

It is finally fair to mention that the original Spriteworld includes a mode where an embodied agent can physically move the other objects, though no mechanism to avoid occlusions is present, and a single function is responsible for checking the occurrence of a collision for any possible couple of polygons (this affects the accuracy of the collision detection).

4.1 Tasks

All the tasks involved in the experiments consist of an embodied agent doted with four possible actions (move left, right, up, or down) having to reach a certain objective. For each task, the shape of the agent is sampled from a uniform distribution (containing all three possible shapes), its size is a fraction of that of the other objects, it is always white-colored, and the maximum episode length is set to 100. Figure 3 shows a frame of the environment for each of the four tasks involved.

Goal finding The goal-finding task requires the agent to reach a target sprite characterized by a red color while avoiding all the other sprites. The task is considered solved when the distance between the agent and the target is less than or equal to a threshold δ. Defining $d(agt, tgt)$ as the distance agent-target, at each step, the reward is computed as $r_t = \delta - d(agt, tgt)$ and if an obstacle is touched, a -1 is summed to the reward.

Goal finding interactive Similar to the task described above, this task requires the agent to reach a red-colored target sprite and avoid the obstacles, but in this case, there is an additional challenge: the agent has to move the target sprite to touch another goal-sprite colored yellow. Again the task is considered solved when the distance between the red target-sprite and the yellow target-sprite is less than or equal to a threshold δ.
The goal-finding task requires the agent to reach a target sprite characterized by a red color while avoiding all the other sprites. The task is considered solved when the distance between the agent and the target is less than or equal to a threshold δ. Defining $d(agt, tgt1)$ as the distance agent-target1 and $d(tgt1, tgt2)$ the distance between the red and the yellow sprite, at each step, the reward is computed as $r_t = (\delta - d(agt, tgt)) + (\delta - \text{d(tgt1, tgt2)})$ and if an obstacle is touched, a -1 is summed to the reward. The first term is meant to incentivize the agent to move toward the first target to reduce the need for intensive exploration.

Clustering In this task, the agent's objective is to group the sprites based on a property between shape or color. In order to evaluate the goodness of the clustering assignments, we make use of the Davies-Bouldin clustering metric [5]. Denoting the Davies-Bouldin Index as DBI, the reward at each timestep is calculated as $\frac{1}{DBI} - \delta$, where δ is a termination threshold. The task is considered solved when the inverse Davies-Bouldin Index is higher than equal to a δ.

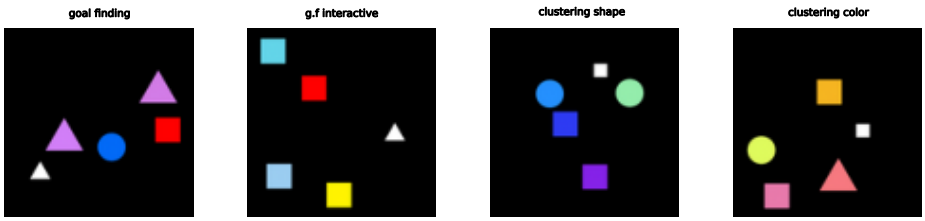

Figure 3: Visualization of states sampled from the environment under different tasks. In the task where the agent is required to cluster the other sprites by shape (third image from left to right), it is expected to join the right cluster based on its shape

5 Methodology

The authors of [13] introduce Contrastively-trained Structured World Models (C-SWMs), a class of structured world models whose goal is to learn to represent scenes as a composition of individual objects and predict action-conditioned transitions accounting for their relations. As the idea has the potential to bring several benefits over classical world models, such as better generalization properties and a more interpretable latent space, it is affected by several limitations mentioned by the authors. Prominent among the others is the inability of the object extractor to disambiguate multiple instances of the same object. This limitation comes from the fact that a feed-forward CNN (the architecture chosen for the object extractor) needs either distinct visual features or labels to distinguish one entity from another. Such a limitation can affect the performance of this class of models in environments such as the one described in the previous section, hence as noticeable in Figure 3, it can present scenes containing identical or very similar objects.

Contribution The main contribution of this work is to enhance the class of models proposed in [13] by replacing the CNN-based object extractor and the MLP-based object encoder with the encoder of a pretrained object-centric model such as Slot Attention [16]. Such an extension should increase the quality of the representations fed to the transition model making it more inclined to generalize and easier to interpret. The original method has further been extended with an iterative GNN module to predict the transitions of the objects given the updates obtained at the previous iteration (necessary since interactive Spriteworld allows sprites to move in chains).

5.1 Slot Attention and pretraining

Before building a world model based on Slot Attention's representations, it was necessary to ensure that Slot Attention could represent the Spriteworld scenes consistently. By carrying out several experiments, Slot Attention emerged to have the following undesired behaviors:

1. The random sampling of the slots leads to different outcomes with the same inputs, no guarantee of "good" encodings at the first draw, and representations without a fixed order.
2. When having more slots than objects to represent, the model tends to separate the information of one entity into two vectors to avoid empty masks.
3. The per-object masks produced by the decoder tend to share information about the background.

In order to correct the first two Slot Attention practices, it was primarily important to fix the number of sprites that can appear in Spriteworld at once. In this way, the number of slots is always equal to the number of discoverable objects, and more importantly, we could replace the random slots' initialization with learned parameters. This solution solves both behaviors 1 and 2. Since the loss of the transition model compares the predicted slot vectors with the ones resulting

from the encoding of the next state in a pair-wise way, maintaining the order of the representations constant avoids computationally expensive sortings based on similarity measures. As the third behavior is given by the background being constant across the scenes, Slot Attention has been pretrained by augmenting the images with random noise. The noise is applied with a two-thirds probability and is constant for all the pixels. In this way, the sprites remain distinguishable from the background as it changes color across different samples.

The results are obtained by training Slot attention for roughly 250,000 gradient steps with the configuration used in [16] for the Multi-DSprites dataset and can be observed in Figure 4. As it is noticeable the reconstruction and the division in slots are pretty much perfect, while the order in which the slots are represented is preserved from one state to the next. Unfortunately, some scenes can still be misrepresented as in Figure 5 and it was not possible to detect the causes of this phenomenon. We tried many different configurations and longer training, but none of them brought any improvement.

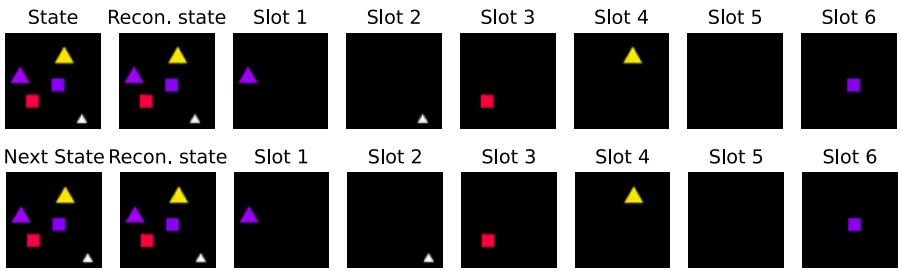

Figure 4: Visualization of the Slot Attention behavior on consecutive visual scenes extracted from our version of Spriteworld. The images labeled as "Slot #" are obtained by multiplying the slots with the respective masks and the reconstruction is their sum.

Figure 5: Visualization of a Slot Attention misrepresentation of a visual scene extracted from our version of Spriteworld. The images labeled as "Slot #" are obtained by multiplying the slots with the respective masks and the reconstruction is their sum.

5.2 Slot Attention-based Structured World Model (SA-SWM)

Once the object-centric visual model is able to adequately process scenes' information to produce per-object representations as the ones shown in Figure 4, the C-SWM object-extractor and object-encoder can be replaced in-block

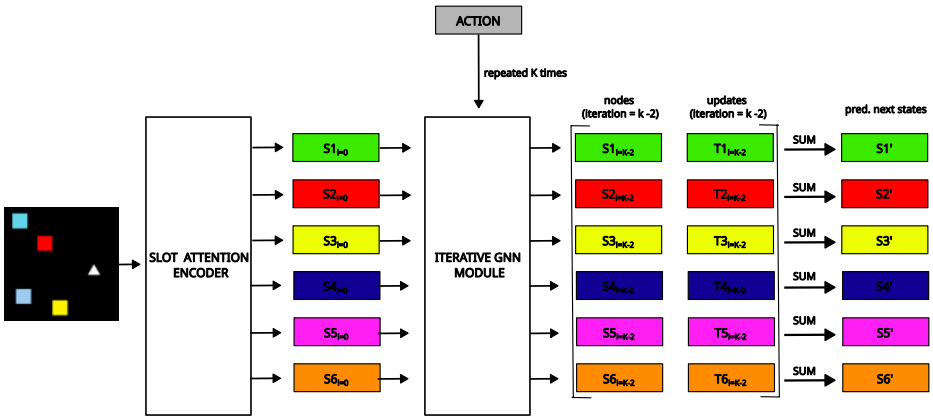

Figure 6: Given an input image, the Slot Attention encoder produces 6 vectors encoding information related to each of the 6 objects present in the scene (5 + the background). These vectors are then fed together with an action to the transition model (defined in Algorithm 1) to predict the next state. The latter is obtained by summing the nodes derived from the updates of the second-to-last iteration to the ones predicted at the end.

by the pretrained Slot Attention encoder. In this way, we have a single encoding function $E(x) = \{z_1, ..., z_k\}$ projecting the input image x into a set of K vectors $z_i \in \mathbb{R}^D$ explaining entities individually. As the model is trained in a self-supervised manner using targets produced by the same encoder, freezing its parameters ensures targets remain constant throughout the entire training process. Therefore, having a pretrained visual model brings a double advantage. In the first instance, the optimization focuses just on the transition model resulting in faster gradient updates (with equal transition models) and a simplified version of the objective in Equation 11. Hence, as there is no need to optimize the encoder, the contrastive term can be removed from the objective obtaining:

$$\mathcal{L} = \frac{1}{K} \sum_{k=1}^{K} d(z_t^k + \Delta z_t^k, z_{t+1}^k) \tag{15}$$

The second benefit is the fact that for every couple of consecutive visual scenes (x_t, x_{t+1}) the corresponding representations $(\{z_t^1, ..., z_t^k\}, \{z_{t+1}^1, ..., z_{t+1}^k\})$ are fixed, resulting in a more stable and efficient training.

We further adapted the original transition model to be suitable for a more complex environment such as Interactive Spriteworld. As previously mentioned in the dedicated section, when a sprite A collides with a sprite B, B is moved accordingly, and if its transition results in a collision with a sprite C this is moved in turn with the chain ending when the last object does not collide with any other. For this reason, a single round of message passing is not sufficient to model the dynamics of the environment, therefore we need to iterate this process multiple times. In order to allow the GNN to take into account the previous node

updates, the edge function in Equation 9 is turned into $f_{edge}([z^i_{t+1}, z^j_{t+1}, \Delta z^i_t])$. The intuition is that the update Δz^i_t contains information about the direction of the transition of z^i_t into z^i_{t+1}, thus in case of collision between z^i_{t+1} and z^j_{t+1}, the edge function can predict a relation vector $e^{(i,j)}$ that can be interpreted as the force applied by z^i_{t+1} on z^j_{t+1} given Δz^i_t.

The node function predicts node updates based on the associated action and the sum of all its incoming edges as in the original version.

As can be observed in Algorithm 1, the Iterative GNN Module takes as input a set of K vectors $z^i_t \in \mathbb{R}^D$ produced by Slot Attention and an action a_t (one-hot vector). Before starting the loop, the algorithm builds a $K \times K$ adjacency matrix with the diagonal set to zero (to avoid relations between an object and itself) representing a fully-connected graph, and initializes the per-object updates Δz^i_0 to zero. At this point, the edge function is fed with a set of $K \times (K-1)$ vectors obtained as the concatenation of z^i_0, z^j_0, and Δz^i_0 with the auxiliary of the adjacency matrix. Then the node updates Δz^j_1 are obtained as in Equation 10 and the nodes z^i_1 are obtained by summing all the nodes z^i_0 with the corresponding updates Δz^i_1. This first phase before the loop is meant to predict the position of the agent after it takes the action a_t, while the iterative part is meant to predict the transition of every other object based on the forces they apply on each other. For this reason, the actions to be fed to the node function are set to zero.
At every iteration n the edges are obtained as $e^{(i,j)}_t = f_{edge}([z^i_n, z^j_n, \Delta z^i_{n-1}])$, the node transitions as $\Delta z^j_n = f_{node}([z^j_n, 0, \sum_{i \neq j} e^{(i,j)}_t])$, and the nodes as $z^i_{n+1} = z^i_n + \Delta z^i_n$. Finally, since in the worst case, the agent motion causes the movement of all the other objects, the number of iterations is set to $K-1$ where K is the number of discoverable objects. The overall SA-SWM architecture is shown in Figure 6

Algorithm 1 Iterative GNN Module

Require: $\mathcal{S} = \{z^1, ..., z^K\} \in \mathbb{R}^{K \times D}$ Slots vectors, a_t action
  $z^i_0 \leftarrow \{0\}^D$
  $i, j \leftarrow get\_inds\_from\_adj\_matrix(K)$
  $e^{(i,j)}_0 \leftarrow f_{edge}([z^i_0, z^j_0, \Delta z^i_0])$
  $\Delta z^j_1 \leftarrow f_{node}([z^j_0, a_t, \sum_{i \neq j} e^{(i,j)}_0])$
  $z^i_1 \leftarrow z^i_0 + \Delta z^i_1$
  for $n = 1 \rightarrow K$ **do**
    $e^{(i,j)}_n \leftarrow f_{edge}([z^i_n, z^j_n, \Delta z^i_n])$
    $\Delta z^j_{n+1} \leftarrow f_{node}([z^j_n, 0, \sum_{i \neq j} e^{(i,j)}_n])$
    $z^i_{n+1} \leftarrow z^i_{n+1} + \Delta z^i_{n+1}$
  end for

5.3 Reward Predictor, Policy, and Value Networks

In order to plan over an object-centric world model such as the one described above, it is indispensable that all the other useful components involved in planning exploit the structured information in a clever way. In a model-based reinforcement learning setting that uses PPO to learn a policy for an arbitrary

task, we need to learn a value function $V^\pi(s)$ (Equation 3) to estimate the advantages $\hat{A}_t = \mathcal{R}(\tau_t^\infty) - V^\pi(s_t)$ ($\mathcal{R}(\tau_t^N)$ is defined in Equation 2), a policy function $\pi(a_t|s_t)$ returning a probability distribution over the action space given the state s_t, and (eventually) a reward function $R(s_t, a_t, s_{t+1}) = r \in \mathbb{R}$. Since in all the tasks involved in our experiments the reward is determined uniquely by the state, the reward function has been modeled for simplicity as $r = R(s_t)$. As the objective is to exploit the model (and the reward predictor) to learn a policy, all the states s_t collected from real interactions with the environment (when predicting s_{t+1} using the model we obtain a factorized representation) need to be encoded as $s_t = \{z_t^1, ..., z_t^k\}$ by the SA-SWM object-encoder.

Because the reward predictor has to infer the quality of a state given a set of object vectors, we need to compute their relations in a similar manner as the transition model. To provide a practical example, in clustering tasks, a good reward predictor should compare objects' properties and return high scores when objects sharing the same property (e.g. color or shape) have positions close in space and vice versa, whereas in goal-finding tasks it has to relate the position of the agent with those of the target sprite and obstacles. In order to allow the reward predictor to develop the desired behavior, we implemented it as a Relational Network (RN) [21] (as in [30]). An RN is usually composed of a function $f_{pair} : \mathbb{R}^{2D_{in}} \rightarrow \mathbb{R}^{D_{rel}}$ producing pair-wise relations given a set of vectors (it can be seen as the edge function of a GNN) and a function $f_{global} : \mathbb{R}^{D_{rel}} \rightarrow \mathbb{R}^{D_{out}}$ that, given the aggregation of these relations, provides a single output. In our implementation both functions are MLPs and the predicted reward is computed as $r_t = f_{global}(\sum_{i<j} f_{pair}([z_t^i, z_t^j]))$. The value network has the same exact architecture as the reward predictor and is of course trained with a different objective.

The policy network architecture differs from the one described above as the RN outputs a vector that is further processed by a feed-forward network producing the final distribution over the action space. It is important to mention that the policy and value networks do not share parameters.

5.4 Model-based RL Algorithm

This section describes the model-based algorithm used to learn a policy that solves the tasks described in the dedicated section. In each of the following paragraphs, we provide a textual description of the main components of the algorithm illustrated in Algorithm 2 and explicit details ignored in Algorithm 2 for readability purposes.

MCTS The implementation of MCTS is mostly standard with some small customizations. Firstly, the rollout has a fixed number of steps and is executed using the policy $\pi_\theta(a|s)$, the action is sampled from the distribution at training time and is the greediest at inference. When in training mode, the algorithm returns the best action (according to the tree policy), and in addition, the log probability associated as it will be used by PPO to compute the ratio in Equation 6 in the form $exp(\log \pi_\theta(a_t|s_t) - \log \pi_{\theta_{old}}(a_t|s_t))$. Furthermore, when the planning budget is set to zero, the algorithm simply returns $a \sim \pi_\theta(a|s)$ at training time and $a = argmax_a \pi_\theta(a|s)$ at inference. Even when the budget is higher than zero, during training there is a preliminary phase without planning

to allow the policy to freely explore the environment, and the reward predictor to learn an approximation of the reward function "adequate" for planning.

PPO The chosen PPO implementation for our experiments strictly follows the structure of the one proposed in the GitHub repository [1]. Different from the cited one, we have separate networks, optimizers, and objectives for actor and critic.

Input The algorithm takes as input a pretrained structured world model, the maximum number of timesteps, the budget to allocate for planning with MCTS, the allowed number of extra steps collectible (after every real episode) by interrogating the model, and the number of steps to take before any PPO update. There are of course other arguments such as learning rates and PPO hyperparameters that are not included for conciseness.

Observation buffer For the sake of clarity, it is relevant to mention that there are two different observation buffers: one storing states, actions, log probabilities, state values, rewards, and done(s) used for PPO updates and cleared right after; and one storing states and rewards only used to train the reward predictor with a way higher capacity that is implemented as a FIFO queue. The latter choice was made to optimize the reward predictor on larger windows of samples to make the training more stable and less biased.

Real vs. Imagined Episodes At every real episode (first nested loop in Algorithm 2) the state returned by the environment at the previous timestep is turned into multiple latent vectors through the encoder of the world model. These vectors are fed to the value network to estimate the state value and passed to MCTS which predicts a locally-optimal action by exploiting the model, the policy, and the reward predictor. The agent then takes the chosen action and the environment provides the next state with the associated reward; the count of timesteps is incremented and after U steps a PPO update is performed. An imagined episode (second nested loop in Algorithm 2) proceeds mostly as the same, with the exception that only the first state is sampled from the environment while the states obtained after taking an action are estimated by the model as the rewards are inferred by the reward predictor. These additional steps are not taken into consideration in the global count as the timesteps budget is meant for real interactions with the environment.

Algorithm 2 Model-based RL Algorithm

Require: M: model, env: environment, T: max number of timesteps, B: planning budget, I: imagination steps, U: steps before update

$O \leftarrow \emptyset$ ▷ initialize observation buffer
$\pi_\theta \leftarrow PolicyNet(M.slots\_dim, env.action\_space.dim)$
$V_\phi \leftarrow RelationalNet(M.slots\_dim, 1)$
$R_\psi \leftarrow RelationalNet(M.slots\_dim, 1)$
$t \leftarrow 0$

while $t < T$ **do**
  $s \leftarrow env.reset()$
  $done \leftarrow false$

  while not done **do**
    $z \leftarrow M.encoder(s)$
    $v \leftarrow V_\phi(z)$
    $a \leftarrow MCTS(M, R_\psi, z, B)$ ▷ if $B = 0$ $a \sim \pi_\theta(a|z)$
    $s, r, done \leftarrow env.step(a)$
    $t \leftarrow t + 1$
    $O \leftarrow O \cup \{(z, v, r)\}$

    if U mod $t = 0$ **then**
      $PPO(\pi_\theta, V_\phi, O)$
    end if
  end while

  $\mathcal{L}(\psi) \leftarrow \frac{1}{|O|} \sum_{i=1}^{|O|} (R_\psi(z_i) - r_i)^2$
  $\psi \leftarrow \psi - \alpha \nabla \mathcal{L}(\psi)$ ▷ update reward predictor

  $z \leftarrow M.encoder(env.reset())$
  for $i = 0 \rightarrow I$ **do** ▷ update the policy with imagination steps

    $v \leftarrow V_\phi(z)$
    $a \leftarrow MCTS(M, R_\psi, z, B)$ ▷ if $B = 0$ $a \sim \pi_\theta(a|z)$
    $r = R_\psi(z)$
    $O \leftarrow O \cup \{(z, v, r)\}$
    $z \leftarrow M.transition\_model(z, a)$

    if U mod $(t + i) = 0$ **then**
      $PPO(\pi_\theta, V_\phi, O)$
    end if

    if $r >= 0$ **then**
      break
    end if

  end for
end while

6 Experiments

The objective of the following experiments is to observe:

- whether better object representations allow to overcome the limitations of C-SWM and improve its generalization capabilities in environments with complex dynamics;
- the behavior of the best world model from the previous experiment when applied in a model-based reinforcement learning algorithm.

6.1 Metrics

World Models The metrics chosen to evaluate the performances of the transition model are those adopted in [13], namely: Hits at rank k (H@k) and Mean Reciprocal Rank (MMR), as they allow performing the evaluation directly in latent space

Hits at rank k Let a predicted object vector be ranked based on its distance with the corresponding target vector. The H@k score is 1 when the rank of the inferred vector does not exceed k, 0 otherwise. As in [13], we report the H@k averaged over the entire dataset.

Mean Reciprocal Rank MRR is defined as the average of the inverse rank of all the n samples present in the evaluation dataset, in formula

$$MRR = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{rank_n}$$

where $rank_n$ is the rank of the n-th sample.

We refer the reader to [7] for more details on these and other ranking-based evaluation metrics.

RL agents' performances The performances of the RL agents are evaluated through the cumulative reward obtained averaged over multiple repetitions and their success rate. The success rate is defined as the number of times the agent completes a fixed task divided by the number of trials.

6.2 World models

As the following experiments aim to validate the hypothesis that pre-learned object-centric representations should overcome C-SWMs limitations and improve their generalization, we consider the baseline a C-SWM adopting the iterative GNN module described in the methods section. We further augment the number of feature maps from one to four per object to model the positional information (as in the original paper) as other relevant information such as the shape and size of the object. The results are obtained using the configurations in Table 2 and are averaged over four independent repetitions. The training dataset

consists of $6 \cdot 10^4$ samples, one-third of which were collected with a random policy acting in the environment, and the remaining part by physically playing with the environment to ensure collecting enough complex interactions (that would be unlikely to collect by acting randomly). The test set is divided into three disjointed subsets sorted in ascending order of difficulty, each containing 10^3 unseen samples structured in 10 episodes of 100 timesteps. The first set contains 10 "collision-free" trajectories where the agent is the only sprite moving, the second set includes several sequences of steps (per episode) where the agent carries one sprite at once, and the last contains complex trajectories where the agent can carry multiple sprites at once or in a chain. This division is meant to highlight the strengths and limits of both classes of models and help individuate and interpret their behaviors under different settings.

Quantitative results As in [13], our findings include the ranking scores (represented as a percentage) obtained traversing the latent space from the source to the (encoded) target state using the learned model. The reported results encompass the mean and standard error of scores obtained from four independent runs.

Table 1: As in [13], we present the ranking outcomes for multi-step prediction in the latent space, highlighting the best mean scores in bold.

		1 Step		5 Steps		10 Steps	
	Model	H@1	MRR	H@1	MRR	H@1	MRR
test 1	**SA-SWM**	**98.2**$_{\pm 1.3}$	**98.8**$_{\pm 0.8}$	**93.5**$_{\pm 1.1}$	**95.3**$_{\pm 0.8}$	**88.0**$_{\pm 2.1}$	**92.4**$_{\pm 1.1}$
	C-SWM	72.0$_{\pm 18.5}$	82.2$_{\pm 11.9}$	36.0$_{\pm 23.4}$	53.5$_{\pm 19.5}$	23.7$_{\pm 18.6}$	42.7$_{\pm 17.7}$
test 2	**SA-SWM**	**97.5**$_{\pm 1.1}$	**97.9**$_{\pm 0.9}$	**86.8**$_{\pm 0.8}$	**90.2**$_{\pm 1.0}$	**73.8**$_{\pm 1.6}$	**81.2**$_{\pm 0.5}$
	C-SWM	89.7$_{\pm 7.4}$	94.2$_{\pm 4.1}$	50.0$_{\pm 14.9}$	69.6$_{\pm 10.0}$	20.3$_{\pm 13.2}$	48.0$_{\pm 10.0}$
test 3	**SA-SWM**	**97.2**$_{\pm 0.4}$	**98.0**$_{\pm 0.3}$	**89.0**$_{\pm 0.6}$	**93.5**$_{\pm 0.7}$	**68.3**$_{\pm 3.5}$	**79.6**$_{\pm 2.0}$
	C-SWM	86.3$_{\pm 9.0}$	92.3$_{\pm 5.2}$	59.3$_{\pm 10.5}$	75.5$_{\pm 7.5}$	19.3$_{\pm 12.6}$	48.2$_{\pm 10.5}$
train	**SA-SWM**	**100**$_{\pm 0.0}$	**100**$_{\pm 0.0}$	99.0$_{\pm 0.2}$	99.1$_{\pm 0.2}$	95.7$_{\pm 0.7}$	96.4$_{\pm 0.5}$
	C-SWM	**100**$_{\pm 0.0}$	**100**$_{\pm 0.0}$	**100**$_{\pm 0.0}$	**100**$_{\pm 0.0}$	**100**$_{\pm 0.0}$	**100**$_{\pm 0.0}$

The quantitative measures reported in Table 1 illustrate a significant distance in terms of performance and generalization between the proposed class of models and the baseline. We can hence observe that the baseline performance drastically deteriorates as the number of steps to take in the latent space increases, and in all three splits, the H@1 registered after a 10-step prediction is around 20%, while SA-SWM in the worst case reached a score close to 70%. Even though SA-SWM (similarly to C-SWM) becomes less accurate as the number of future steps grows up to ten, it showed to be more robust than the baseline as the H@1 obtained with a ten-step prediction varies from around 88% (for the easiest split) to 68% (for the hardest).
The fact that SA-SWM consistently achieves scores close to 100% in 1-step prediction suggests that the model successfully learns to predict how actions affect the agent position; the small margin of error can be given either from Slot Attention errors (as in fig 5) or small imprecisions in predicting the agent transition. Although both options are reasonable, the latter appears more likely

as the fact that the first set H@1 scores decrease (almost constantly) by roughly 5% (between an n-steps and an (n+5)-steps prediction) suggests a linearly accumulating error. Moreover, as the training set contains samples that Slot Attention misrepresents, we expect this phenomenon to be almost irrelevant for the metric involved as the source and the target states are (wrongly) encoded with a pattern that the transition model can (probably) learn. We further notice that while the standard deviation from the mean is reasonable for SA-SWM, with 3.1 being the highest value, C-SWM showed a very high variance in the results concerning the three test splits. From this observation, we can conclude that C-SWMs are much more sensitive to weight initialization and other random factors in the training process than SA-SWMs are. Furthermore, we show the results obtained by both classes of models on the training set to highlight that the baseline always converges on the training dataset and achieves an average score of 100% independently from the number of steps. This behavior strongly supports the initial hypothesis that the lack of information about the object properties (different from position) makes C-SWMs prone to overfitting. Even though the baseline has four feature maps per object (differently from the C-SWM used in [13] experiments) to model different object characteristics, the contrastive objective is probably too "vague" to learn complex features as it only pushes the model to represent divergent states differently. There (might) exist numerous approaches to attain this objective, which can explain the substantial variance observed in the baseline test outcomes. On the other hand, SA-SWM exhibited much better generalization capabilities and significantly outperformed the baseline on every test set and independently from the number of steps involved in the prediction. The proposed class of models reached a score close to 90% in most cases, making exceptions for 10-step predictions in the dataset involving object interactions. These results can be explained (again) by the error in one-step predictions accumulating along the steps and through Slot Attention's embeddings (mostly). Hence, the per-object representations Slot Attention produces contain mixed information, and there is no specific component in the latent vector to explain a single property (such as position, scale, shape, and color), meaning that the transition model has no information about individual features. This entangled latent space represents a hard limit to SA-SWMs generalization and complicates how it predicts object transitions as it cannot ignore useless (in this specific case) properties such as color.

Qualitative results This paragraph shows the effect of the transitions in pixel space. The visual images are obtained by predicting the target state after n actions with the best model out of the four repetitions and decoding those predictions using the Slot Attention decoder.

As can easily be observed in Figure 7, the proposed world model predicts the agent motion coherently with the concrete transitions in the environment. Accordingly, with the quantitative results in the previous paragraph, Figure 7 shows less accurate predictions as the number of steps rises. We can observe in the last column that the portion of the agent that differs from the actual target state gets more visible in proportion to the number of steps. This visualization also supports the hypothesis that minimal errors in single-step predictions accumulate and become more evident as the model takes multiple steps in the latent space. Nevertheless, the predictions are impressively accurate and allow the model to

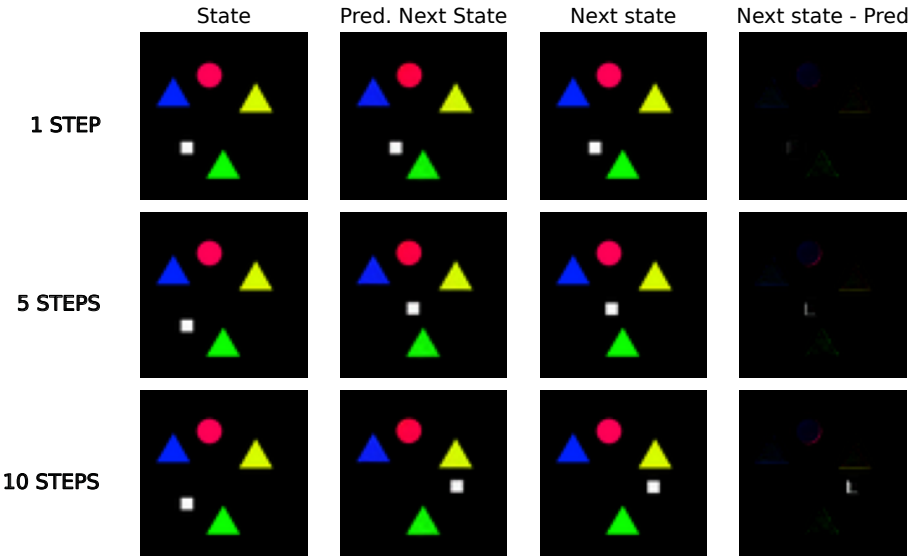

Figure 7: Visualization of the behavior of the transition model when the agent is the only sprite moving in the environment. The first column shows the source states, the second exhibits the prediction after the number of steps indicated in the row, and the third shows the real target state after the same number of steps. The last column illustrates how the predictions differ from the targets (a full black image means no error).

be involved in planning (at least in tasks that do not demand object collisions, such as the Goal-finding task).

In Figure 8, we provide a visualization of different behaviors exhibited by the transition model when dealing with situations where the agent carries one sprite. By focusing on the 1-step row, it is possible to observe that the model learned to predict a transition involving two objects quite accurately. It is also noticeable that the error in the prediction is comparable with the one produced after ten steps on the first test set. A possible explanation for this phenomenon is that while the agent always appears with the same color, other sprites have colors sampled from a uniform continuous distribution, which makes it likely to draw pigments unseen at training time. As the encoding fed to the transition model contains entangled information about objects' features, learning a local movement pattern for a fixed-color entity is much easier than a general and color-invariant one. Having disentangled representations would instead allow encoding objects' movements consistently and regardless of the color component.

Moving the attention to the 5-step row, we realize that the transition model predictions remain consistent even when the agent carries another object for five steps in the latent space. The slight error visible in the visualization of the difference between prediction and target is comparable with the previous and finds its explanation in error accumulation and entangled latent space.

The 10-step row offers a view of a particular behavior that may ulteriorly

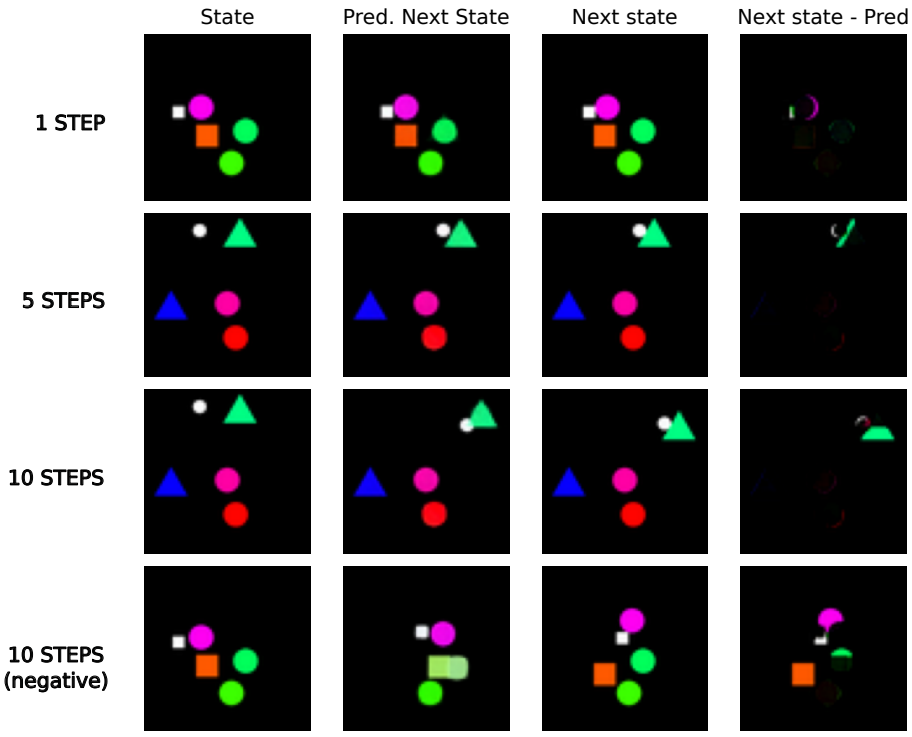

Figure 8: Visualization of the behavior of the transition model when the agent can move one sprite at a time. The first column shows the source states, the second exhibits the prediction after the number of steps indicated in the row, and the third shows the real target state after the same number of steps. The last column illustrates how the predictions differ from the targets (a full black image means no error).

sustain that the limits in generalization shown by the model are due to nonoptimal embeddings forming an intricate latent space. Hence, we can observe the model predicting all the steps toward the right quite accurately, as prediction and target mostly disagree for the lower part of the triangle. Moreover, the inferred agent position matches the target with an error aligned with those that emerged in previous tests. Given these preliminaries, it would make sense that the model mispredicted the shape of the carried object with either a circle or a square, making the agent movement toward the bottom inconsequential for the other sprite involved in the collision.

The last row of Figure 8 illustrates a situation where Slot Attention fails to represent the scene objects. It is evident that in such a situation, the model predictions are meaningless, and involving it in planning tasks (in episodes affected by these representations) would be counter-productive.

The visual outcomes deriving from the experiments on the third test split were significantly affected by the problematics measured in the previous analysis. In this regard, it was not achievable to investigate whether the proposed class

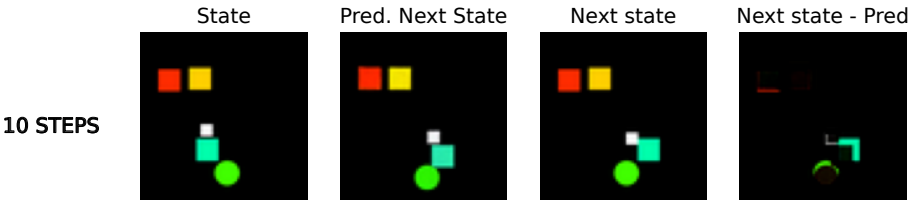

Figure 9: Visualization of the behavior of the transition model when the agent can move more than one sprite at a time. The first column shows the source states, the second exhibits the prediction after the number of steps indicated in the row, and the third shows the real target state after the same number of steps. The last column illustrates how the predictions differ from the targets (a full black image means no error).

of world models can manage complex situations involving the interaction of multiple objects. Fortunately, it was possible to find the example shown in Figure 9 that exhibits curious conduct. Even though the figure illustrates a wrong prediction because the teal square was supposed to be more on the right when the agent took its step toward the bottom, assuming the teal square position was correct, the predicted transition would be close to reality for all the three objects interacting. This finding arouse ulterior curiosity regarding the performance of the world model in the presence of optimal disentangled representations.

Model-based RL Agents Unfortunately, the encodings produced by Slot Attention (and consequently by SA-SWM) did not allow finding any configuration enabling to solve any of the tasks proposed. After various experiments, we found that neither the critic network nor the reward predictor could learn to approximate the value and the reward functions, respectively. The fact that both networks could not minimize a squared error, not even fixing the sprites appearing in an episode (to at least overfit this situation), led to conclude that the embeddings do not allow us to find any relational pattern. As the objective is to enable object-oriented relational reasoning, the architectures used (Relational Networks) necessitate exploiting relationships intra-objects to estimate the quality of a state or determine which action would lead to the best outcome. For example, even in a simple task where the agent has to reach (the only) red sprite, a reward predictor should relate the agent's position with that of the red object to guess the reward of a state. But if there is no way to infer whether a sprite is red, the network cannot associate an observed reward based on the distance agent-red object. This concept extends to all the other tasks as all their reward functions rely on distances or similarities between sprites sharing one or more properties.

To validate these hypotheses, we extrapolated some data points from the test split, encoded them in latent space, and labeled them (based on their features) to perform Principal Component Analysis (PCA). We reduced the embeddings into 2-dimensional vectors to investigate whether clusters of points sharing certain features exist and which information we cannot grasp. An ideal object-centric model should encode the information dividing vectors into subparts, each

explaining always the same individual object property. Performing PCA on those individual parts would bring perfect clusters based on the label. In this case, we can only observe if there are discriminants that allow the world model to reach the results shown in this section but not to learn either a reward, value, or policy function.

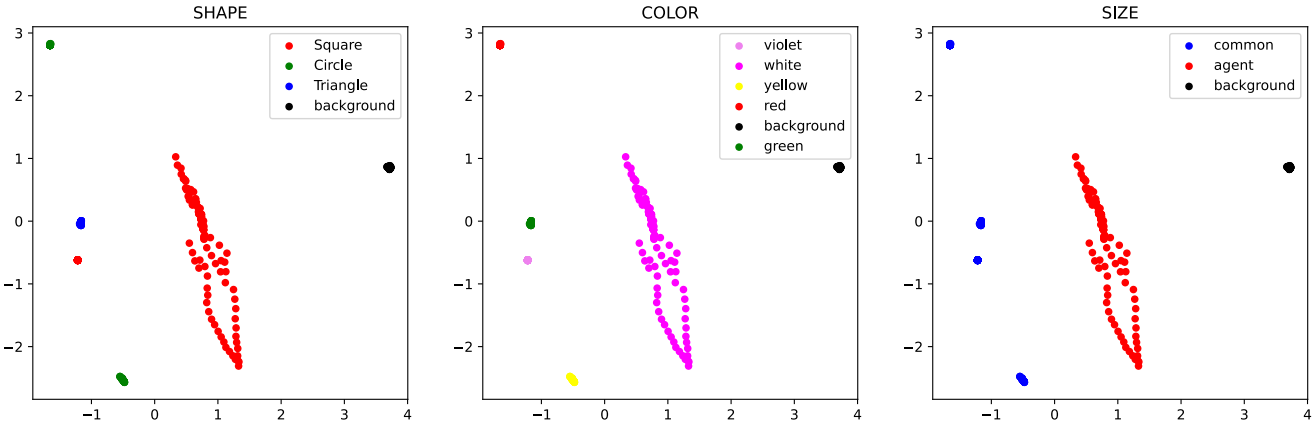

Figure 10: A 2-dimensional visualization of the latent space obtained encoding samples from a single episode in the first test split (the agent is the only sprite moving).

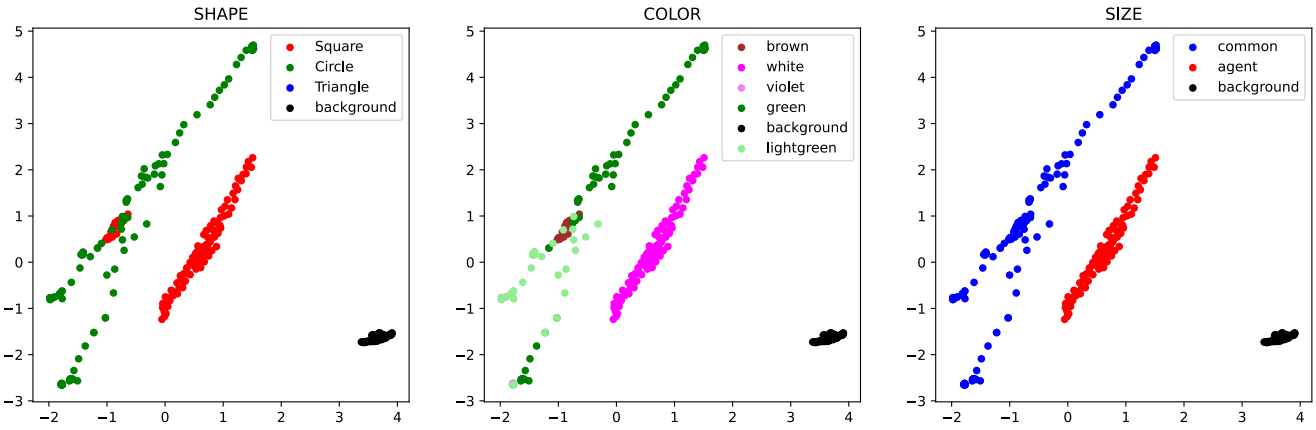

Figure 11: A 2-dimensional visualization of the latent space obtained encoding samples from a single episode in the second test split (all sprites can move one at a time).

As visible in Figures 10, 11, and 12, there is no way to find a correlation between shapes or colors belonging to ordinary sprites. On the other hand, observing the plots concerning the size, we can see three distinct clusters: standard sprites size, agent size, and background, and in the color plots, we can individuate a separate group for black, one for white, and one for all the other colors. We can therefore agree that, given these representations, we can

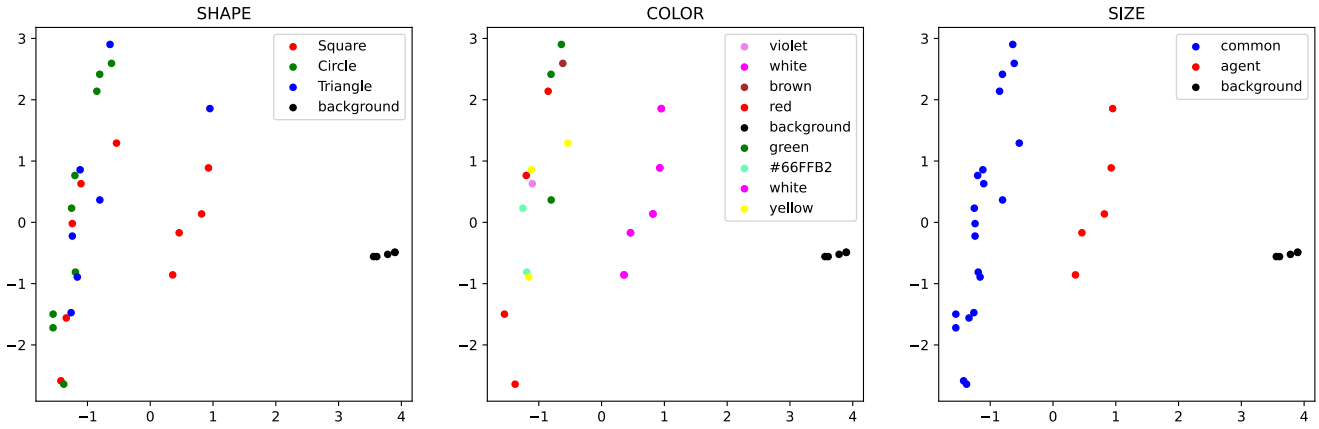

Figure 12: A 2-dimensional visualization of the latent space obtained encoding samples from five different episodes (different objects) in the second test split (all sprites can move one at a time).

only discriminate the agent or the background from the rest of the sprites. This information is (as expected) insufficient to make relational reasoning, but the reader may wonder why it was enough for learning a robust world model.

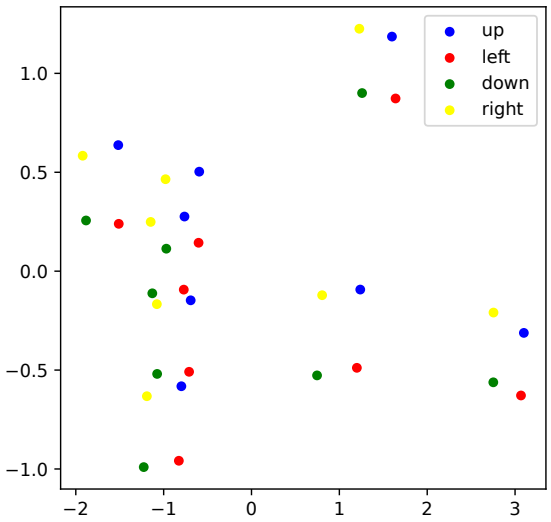

Figure 13: Observation of the effect of actions on the agent in 2-dimensional space. The four actions seem to create a rectangle independently from the original position occupied by the agent

The transition model only needs to detect if two objects are colliding, which can be imprecise if the shape is mispredicted, but we can generally assume that with a heterogenous and vast dataset, it can learn to determine if two objects are touching. Moreover, the agent can be easily discriminated from the context, making it easy for the model to understand which sprite is affected by the action,

and as evident in Figure 13, the way actions alter the agent position follows a strict pattern. The transition model can therefore model the agent transitions in space and detect sprites collisions easily; from these observations and the results obtained, we can assume that every moving object follows some pattern that the model successfully learned. Nevertheless, as previously discussed, having ideal representations would improve generalization and facilitate learning.

7 Discussion & Conclusion

In light of the result observed in the previous section, it is evident that the quality of the object-centric representations determines the robustness of structured world models. Dealing with vectors describing entities without disentangling the information related to their fundamental properties does not allow us to infer most of their features. Consequently, when feeding a transition model with embedding obtained encoding test images, there is no guarantee that its prediction will agree with reality. To provide an example supported by visual results, in Figure 8, we see our model that significantly mispredicts the target state in the 10-step row. As inferring the shape of the teal triangle from its embedding is hard (or maybe impossible), the model erroneously guessed either a circle or a square. Both embodiments would indeed not be affected by the agent moving to the bottom, remaining in the position predicted by the model.

Moreover, having this chaotically mixed information does not allow the model to ignore properties that are useless in predicting the transition of an object in space (such as color in this specific environment). In other words, the transition model has to update the object vectors in a way that preserves all the features (except positions when objects move). On the other hand, having specific components explaining individual features would allow a model to consider only the useful information and update just the sub-vector bound to the position. This would favor generalization to unseen objects and enhance the robustness of the world model in multi-step prediction. On the contrary, updating the entire vector could bring updates disrupting the object representation altering its texture and contours, making predictions less accurate as the number of steps to take in latent space increases.

SA-SWMs vs. C-SWMs Even though the representations are far from optimal, we observed that SA-SWMs worked impressively better than C-SWMs in the given environment. Exploiting pre-learned representations (that even if not disentangled, still succeed in explaining objects individually) allows a loss function entirely dedicated to optimizing the transition model and prevents the encoder from learning embeddings minimizing the loss without encoding the desired information.

Furthermore, as previously mentioned, the contrastive term in the loss function is (probably) not sufficient to model complex features, as it is reasonable to think that exists countless ways to move embeddings of dissimilar states away into latent space. As a consequence, the solutions optimizing the loss and modeling the desired features are strongly outnumbered by poor local minima.

Despite the high variance registered in C-SWM performances endorses this hypothesis, we believe a C-SWM can learn good representations with rigorous and extensive hyperparameter tuning. Regardless, such a tuning procedure is likely to be far more computationally expensive than pre-training an object-centric model as training-efficient as Slot Attention.

Lastly, Slot Attention is not affected by instance disambiguation issues, allowing it to overcome one of the limitations named by T.Kipf et. al in [13].

Reinforcement learning The analysis of the latent space in 2-dimensions favors the explanation that sustains features such as color and shape cannot be inferred from the Slot Attention encodings (or at least not easily). Consequently, architectures such as Relational Networks that rely on affinities between couples of vectors, struggle to approximate functions depending on those properties such as reward, value, and policy functions. In order to simplify the problem and verify that it did not rise instead from implementation errors or non-optimal parameter settings, we tried to approach the easiest task (goal finding) with a simple 1-step search as in [30]. In this way, we only train the reward predictor through mean squared error, and in case it does not even manage to overfit, we can conclude that the representations do not allow us to learn a simple linear function, based on object relations. In this task, the reward function is inversely proportional to the distance between the agent and the target sprite which is identified by the red color. It came out that after about 10^5 timesteps, the trained reward predictor accuracy was aligned with that achievable with random guesses. The experiments conducted in [30] showed instead that a number of environment steps between 10^2 and 10^3 is sufficient to reach a success rate of 90% on a task with a pretty similar reward function. Though these results should not surprise as the MONet [4] encoder (the visual model used in COBRA[30]) provide the reward predictor (a Relational Network) with latent vectors sharing the same representational semantic. These vectors are hence composed of subparts explaining object properties in a disentangled way and the order in which these features are encoded is constant. Watters et. al [30](appendix G) recognized that these representational properties could be a determining factor in the development of a robust agent. In light of these results, the hypothesis that disentangled representations are indispensable to enable relational reasoning and allow learning property-comparison tasks with a few interactions with the environment (less than 10^3), becomes even more solid. It should not be excluded that with more complex models and different architectures, it might be possible to exploit the actual representations to learn some of the proposed tasks (given an extensive tuning of the hyperparameters). Nevertheless, this is not the direction that this work is meant to follow. Contrarily, this work aims to provide ulterior evidence that interpretable object-based encodings enable current reinforcement learning approaches to be more sample efficient and prone to generalization. Complex models fed with non-optimal representations would be instead more data-hungry and inclined to overfitting.

The latent space analysis further explained why the representations were instead enough to learn a satisfying world model. In particular, it showed that it is particularly easy to discriminate the agent from the other sprites and map its moves based on the actions, justifying the valuable results it obtains (especially) when the agent does not interact much with the other objects.

Conclusion & Future Work In conclusion, this study has shed light on the importance of disentanglement in object-centric representation to allow agents to reason about objects and their specific properties. We further introduced a class of object-based world models that overcomes some of the limitations affecting C-SWMs while enhancing their performances and generalization properties on the environment used in our experiments. In this regard, we extended Spriteworld

[29] to include physics in order to require structured world models able to learn complex interactions between objects. We hope the introduction of a more challenging environment stimulates interest in the field to realize solutions to solving its tasks, or even to build different environments to challenge future object-based reinforcement learning agents. In future projects, it would be interesting to ulteriorly enhance C-SWMs to overcome their other limitations listed in [13]. In particular, it would be interesting to enhance the GNN-based transition model to approximate an object-wise probability distribution over the possible transitions. In this way, it would be possible to take into account the stochasticity that can characterize the transition function of more complex and non-deterministic environments. Another possible improvement suggested by T.Kipf et. al [13] is to extend the transition model with some sort of memory mechanism (such as for example RNNs or Neural Turning Machines) to unbound the transition model from the assumption that the current state and action contain all the information needed to predict the next one. As a first step to take in the immediate future, our intention is to replicate our investigation replacing Slot Attention with MONet to definitively prove our conclusion on disentanglement, and finally provide empirical results for the application of structured world models in a model-based reinforcement learning algorithm.

References

[1] N. Barhate. Minimal pytorch implementation of proximal policy optimization. https://github.com/nikhilbarhate99/PPO-PyTorch, 2021.

[2] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[3] M. Chang, T. L. Griffiths, and S. Levine. Object representations as fixed points: Training iterative refinement algorithms with implicit differentiation, 2022.

[4] N. W. R. K. I. H. M. B. A. L. Christopher P. Burgess, Loic Matthey. Monet: Unsupervised scene decomposition and representation. 2019.

[5] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

[6] C. Diuk, A. Cohen, and M. L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247, 2008.

[7] C. T. Hoyt, M. Berrendorf, M. Gaklin, V. Tresp, and B. M. Gyori. A unified framework for rank-based evaluation metrics for link prediction in knowledge graphs. *arXiv preprint arXiv:2203.07544*, 2022.

[8] V. Jankovics, M. G. Ortiz, and E. Alonso. Efficient entity-based reinforcement learning. *arXiv preprint arXiv:2206.02855*, 2022.

[9] B. Jia, Y. Liu, and S. Huang. Improving object-centric learning with query optimization, 2022.

[10] J. Jiang, S. Janghorbani, G. De Melo, and S. Ahn. Scalor: Generative world models with scalable object representations. *arXiv preprint arXiv:1910.02384*, 2019.

[11] R. Kabra, C. Burgess, L. Matthey, R. L. Kaufman, K. Greff, M. Reynolds, and A. Lerchner. Multi-object datasets. https://github.com/deepmind/multi-object-datasets/, 2019.

[12] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1809–1818. PMLR, 06–11 Aug 2017.

[13] T. Kipf, E. van der Pol, and M. Welling. Contrastive learning of structured world models, 2019.

[14] R. K. N. W. C. B. D. Z. L. M. M. B. A. L. Klaus Greff, Raphaël Lopez Kaufman. Multi-object representation learning with iterative variational inference. 2019.

[15] V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[16] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention, 2020.

[17] O. P. J. I. P. Martin Engelcke, Adam R. Kosiorek. Genesis: Generative scene inference and sampling with object-centric latent representations. 2019.

[18] 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.

[19] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[20] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[21] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30, 2017.

[22] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[24] G. Singh, F. Deng, and S. Ahn. Illiterate dall-e learns to compose, 2021.

[25] E. S. Spelke and K. D. Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

[26] S. Van Steenkiste, M. Chang, K. Greff, and J. Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

[27] S. van Steenkiste, K. Greff, and J. Schmidhuber. A perspective on objects and systematic generalization in model-based rl. *arXiv preprint arXiv:1906.01035*, 2019.

[28] R. Veerapaneni, J. D. Co-Reyes, M. Chang, M. Janner, C. Finn, J. Wu, J. Tenenbaum, and S. Levine. Entity abstraction in visual model-based reinforcement learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1439–1456. PMLR, 30 Oct–01 Nov 2020.

[29] N. Watters, L. Matthey, S. Borgeaud, R. Kabra, and A. Lerchner. Spriteworld: A flexible, configurable reinforcement learning environment. https://github.com/deepmind/spriteworld/, 2019.

[30] N. Watters, L. Matthey, M. Bosnjak, C. P. Burgess, and A. Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019.

[31] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti. Visual interaction networks: Learning a physics simulator from video. *Advances in neural information processing systems*, 30, 2017.

[32] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992.

[33] J. Yoon, Y.-F. Wu, H. Bae, and S. Ahn. An investigation into pre-training object-centric representations for reinforcement learning. *arXiv preprint arXiv:2302.04419*, 2023.

[34] A. Zadaianchuk, M. Seitzer, and G. Martius. Self-supervised visual reinforcement learning with object-centric representations. *arXiv preprint arXiv:2011.14381*, 2020.

[35] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, et al. Deep reinforcement learning with relational inductive biases. In *International conference on learning representations*, 2019.

A Hyperparameters

	Configurations	SA-SWM	C-SWM
Learning	lr	0.0005	0.0005
	batch size	512	512
	epochs	600	600
	num objects (w/bg)	6	6
	hinge margin	-	1
CNN Encoder	layer1 activation	-	leaky relu
	layer1 filter size	-	9×9 (zero padding)
	layer1 features maps	-	16
	layer2 activations	-	sigmoid
	layer2 filter sizes	-	1×1
	layer2 features maps	-	6×4
MLP Encoder	hidden dim	-	512
	embedding dim	-	64
Slot Attention Encoder	slots	6	-
	iterations	3	-
	embedding	64	-

Table 2: Hyperparameters for SA-SWM and C-SWM