Relational Object-Centric Actor-Critic

Leonid Ugadiarov UGADIAROV@AIRI.NET
AIRI & MIPT, Moscow, Russia

Vitaly Vorobyov VOROBEV.VITALY.V@PHYSTECH.EDU
FRC CSC RAS & MIPT, Moscow, Russia

Aleksandr Panov PANOV@AIRI.NET
AIRI & MIPT & FRC CSC RAS, Moscow, Russia

Editors: Biwei Huang and Mathias Drton

Abstract

The advances in unsupervised object-centric representation learning have significantly improved its application to downstream tasks. Recent works highlight that disentangled object representations can aid policy learning in image-based, object-centric reinforcement learning tasks. This paper proposes a novel object-centric reinforcement learning algorithm that integrates actor-critic and model-based approaches by incorporating an object-centric world model within the critic. The world model captures the environment's data-generating process by predicting the next state and reward given the current state-action pair, where actions are interventions in the environment. In model-based reinforcement learning, world model learning can be interpreted as a causal induction problem, where the agent must learn the causal relationships underlying the environment's dynamics. We evaluate our method in a simulated 3D robotic environment and a 2D environment with compositional structure. As baselines, we compare against object-centric, model-free actor-critic algorithms and a state-of-the-art monolithic model-based algorithm. While the baselines show comparable performance on more manageable tasks, our approach outperforms them in more challenging scenarios with a more significant number of objects or more complex dynamics.

Keywords: object-centric representations, graph neural networks, actor-critic, model-based reinforcement learning

1. Introduction

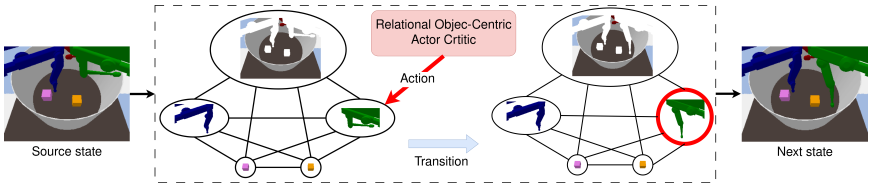

Figure 1: A high-level overview of the proposed method. ROCA learns the policy by extracting object-centric representations from the source image and treating them as a complete graph.

One of the primary problems in visual-based reinforcement learning (RL) is determining how to represent the environment's state efficiently. The most common approach is to encode the entire input image, which is then used as input for the policy network Mnih et al. (2015); Zhang et al.

(2021). However, previous studies (Santoro et al., 2017) have shown that such representations may fail to capture meaningful relationships and interactions between objects in the state. Object-centric representations (OCR) can be introduced to overcome this issue. Such representations are expected to result in more compact models with enhanced generalization capabilities (Keramati et al., 2018). State-of-the-art unsupervised OCR models (Kirilenko et al., 2024; Singh et al., 2022; Kirilenko et al., 2023; Engelcke et al., 2022) have a fundamental appeal for RL as they do not require additional data labeling for training. Recent studies (Stanić et al., 2022; Yoon et al., 2023) have shown that object-centric state factorization can improve model-free algorithms' generalization ability and sample efficiency.

Model-based methods (Sutton and Barto, 2018) represent a promising approach for enhancing data efficiency in RL. In model-based reinforcement learning (MBRL), the agent constructs models of transition and reward functions based on its interactions with the environment. The world model, in this context, describes the underlying data-generating process of the environment by capturing the causal relationships between actions, states, and rewards. By considering actions as interventions in the environment, the learning of the world model can be framed as a causal induction problem (Schölkopf et al., 2021). The agent performs multi-step planning to select the optimal action using the model's predictions without interacting with the environment. Model-based algorithms can be more efficient than model-free algorithms, provided the world model's accuracy is sufficiently high. State-of-the-art MBRL methods, which incorporate learning through imagination (Hafner et al., 2023) and look-ahead search with a value equivalent dynamics model (Ye et al., 2021), master a diverse range of environments.

To further enhance sample efficiency, a promising direction is to combine both approaches by developing a world model that leverages object representations and explicitly learns to model relationships between objects (Zholus et al., 2022). An example of this approach is the contrastively trained transition model CSWM (Kipf et al., 2020). It uses a graph neural network to approximate the dynamics of the environment and simultaneously learns to factorize the state and predict changes in the state of individual objects. CSWM has shown superior prediction quality compared to traditional monolithic models.

However, OCR models demonstrate high quality in relatively simple environments with strongly distinguishable objects (Wu et al., 2023). Additionally, in object-structured environments, actions are often applied to a single object or a small number of objects, simplifying the prediction of individual object dynamics. In more complex environments, the world model must accurately bind actions to objects to predict transitions effectively. Despite recent progress (Biza et al., 2022), no fully-featured dynamics models considering the sparsity of action-object relationships have been proposed. These challenges make it difficult to employ object-centric world models in RL. For instance, the CSWM model has not been utilized for policy learning in offline or online settings.

Our research is focused on value-based MBRL as object-based decomposition of value function could contribute to the training of object-centric world model consistent with policy. From the perspective of extracting causal relationships while learning a world model, an object-centric representation leads to the decomposition of these causal relationships that are responsible for the world model's dynamic part. The model-based approach identifies the effects of agent actions on individual objects and predicts changes in their features or those of small groups. This decomposition simplifies the identified causal relationships, accelerating the learning process for the world model and the agent's policy.

We introduce the Relational Object-Centric Actor-Critic (ROCA), an off-policy object-centric model-based algorithm inspired by the Soft Actor-Critic (SAC) (Haarnoja et al., 2018, 2019; Christodoulou, 2019) that operates with both discrete and continuous action spaces. The ROCA algorithm uses the pre-trained SLATE model (Singh et al., 2022), which extracts representations of the individual objects from the input image. Like CSWM (Kipf et al., 2020), we utilize a structured transition model based on graph neural networks. Our reward, state-value, and actor models are graph neural networks designed to align with the object-centric structure of the task. Inspired by TreeQN (Farquhar et al., 2018), we use a world model in the critic module to predict action values. The ROCA algorithm is the first to successfully apply a GNN-based object-centric world model for policy learning in the MBRL setting.

We conducted experiments in 2D environments with simple-shaped objects and visually more complex simulated 3D robotic environments to evaluate the algorithm's quality. The proposed algorithm demonstrates high sample efficiency and outperforms the object-oriented variant of the model-free PPO algorithm (Schulman et al., 2017), which uses the same SLATE model as a feature extractor and is built upon the transformer architecture. Furthermore, our method performs better than the state-of-the-art MBRL algorithm DreamerV3 (Hafner et al., 2023).

Our contributions can be summarized as follows:

- We propose a novel architecture that combines a value-based model-based approach with the actor-critic SAC algorithm by incorporating a world model into the critic module.
- We extended the SAC algorithm by introducing a new objective function to train the model-based critic.
- We propose a GNN-based actor to pool object-centric representations.
- We modified the GNN-based CSWM transition model by adjusting its edge model: we pass a pair of slots along with an action into the edge model.

2. Related Work

Object-Centric Representation Learning Recent advancements in machine learning research have been dedicated to developing unsupervised OCR algorithms (Ramesh et al., 2021; Locatello et al., 2020b; Engelcke et al., 2022). These methods aim to learn structured visual representations from images without relying on labeled data, modeling each image as a composition of objects. This line of research is motivated by its potential benefits for various downstream tasks, including enhanced generalization and the ability to reason over visual objects. The SPACE (Lin et al., 2020) method integrates both the spatial attention model and the spatial mixture model. The spatial attention model identifies objects within the scene, whereas the spatial mixture model focuses on separating the remaining background. This combination allows SPACE to effectively differentiate foreground objects and intricate backgrounds. One notable approach in this field is Slot-Attention (Locatello et al., 2020b), which represents objects using multiple latent variables and refines them through an attention mechanism. Building upon this, SLATE (Ramesh et al., 2021) further improves the performance by employing a Transformer-based decoder instead of a pixel-mixture decoder. Another recently proposed approach that doesn't rely on Slot-Attention is the Deep Latent Particles (DLP) method (Daniel and Tamar, 2024). DLP encodes an input image into a disentangled latent space,

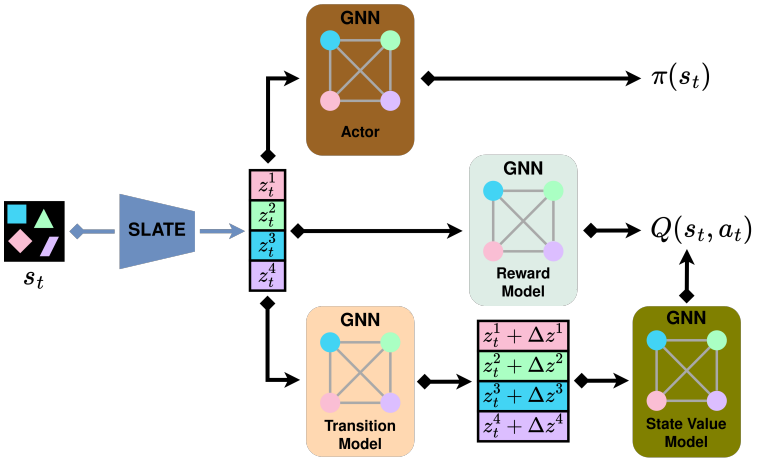

Figure 2: ROCA overview. Framework consists of a pre-trained frozen SLATE model, which extracts object-centric representations from an image-based observation, and GNN-based modules: a transition model, a reward model, a state-value model, and an actor model. The transition and reward models form a world model. The world model and the state-value model together constitute the critic module, which predicts Q-values.

structured as a set of particles with interpretable attributes. These include the particle's position, scale, a "depth" attribute in pixel space to determine which particle is in front in case of overlap, a transparency attribute, and latent features that capture the visual appearance of the region surrounding each particle.

Object-Centric Representations and Model-Free RL Stanić et al. (2022) uses Slot-Attention as an object-centric feature extractor and examines the performance and generalization capabilities of RL agents. In another study (Sharma et al., 2023), a multistage training approach is proposed, involving fine-tuning a YOLO model (Jocher et al., 2022) on a dataset labeled by an unsupervised object-centric model. The frozen YOLO model is then employed as an object-centric features extractor in the Dueling DQN algorithm. Object representations are pooled using a graph attention neural network before being fed to the Q-network. Yoon et al. (2023) proposes using a transformer encoder as a pooling layer in the PPO (Schulman et al., 2017) to aggregate the object representations extracted from an input image by object-centric models of different types. Yi et al. (2022) uses a pre-trained SPACE model to extract object-centric representations and unsupervisedly groups them into a set of categories. For each category, an independent neural network module is used in the PPO, improving the sample efficiency of the approach compared to both monolithic and object-centric baselines. SMORL (Zadaianchuk et al., 2020) and SRICS (Zadaianchuk et al., 2022b) are object-centric, model-free methods specifically designed for goal-conditioned manipulation tasks. SRICS utilizes GNNs as transition models but is limited to non-visual observations. SMORL is suitable for visually-based tasks, as it uses a patch-based image representation model, SCALOR (Jiang et al., 2020). However, its key limitation is the assumption that multi-object tasks can be addressed sequentially and independently, ignoring potential interactions between objects. Haramati et al. (2024) combines DLP and TD3 (Fujimoto et al., 2018) with a transformer backbone for goal-conditioned manipulation

tasks. Additionally, the authors design an intrinsic reward based on the disentangled features of objects provided by DLP. The proposed method outperforms state-of-the-art goal-based approaches.

Object-Centric Representations and MBRL As related work in object-oriented MBRL, we consider Watters et al. (2019). It uses MONet (Burgess et al., 2019) as an object-centric features extractor and learns an object-oriented transition model. However, unlike our approach, this model does not consider the interaction between objects and is only utilized during the exploration phase of the RL algorithm. Recently proposed MBRL algorithm FOCUS (Ferraro et al., 2023) is based on a modified version of RSSM (Hafner et al., 2022) and incorporates an online-learning object-centric encoder. FOCUS accelerates training by using an exploration bonus and requires segmentation masks for its decoder. These masks are either provided by the simulator in a supervised regime or generated by the Segment Anything Model (Kirillov et al., 2023) after fine-tuning with ground-truth segmentation masks in a weakly supervised regime.

3. Background

3.1. Markov Decision Process

We consider a simplified version of the object-oriented MDP (Diuk et al., 2008):

$$\mathcal{U} = (\mathcal{S}, \mathcal{A}, T, R, \gamma, \mathcal{O}, \Omega), \tag{1}$$

where $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_K$ — a state space, $\mathcal{S}_i$ — an individual state space of the object i, $\mathcal{A}$ — an action space, $T = (T_1, \ldots, T_K)$ — a transition function, $T_i = T_i(T_{i1}(s_i, s_1, a), \ldots, T_{iK}(s_i, s_K, a))$ — an individual transition function of the object i, $R = \sum_{i=1}^{K} R_i$ — a reward function, $R_i = R_i(R_{i1}(s_i, s_1, a), \ldots, R_{iK}(s_i, s_K, a))$ — an individual reward function of the object i, $\gamma \in [0; 1]$ — a discount factor, $\mathcal{O}$ — an observation space, $\Omega : \mathcal{S} \to \mathcal{O}$ — an observation function. The goal of reinforcement learning is to find the optimal policy:
$\pi^* = \text{argmax}_\pi \mathbb{E}_{s_{t+1} \sim T(\cdot|s_t, a_t), a_{t+1} \sim \pi(\cdot|s_{t+1})} \left[\sum_{i=0}^{\tau} \gamma^t R(s_t, a_t)\right]$ for all s_0 where τ is the number of time steps.

In model-based approach the agent uses the experience of interactions with the environment to build a world model that approximates the transition function $\hat{T} \approx T$ and the reward function $\hat{R} \approx R$ and use its predictions as an additional signal for policy learning.

3.2. Soft Actor-Critic

Soft Actor-Critic (SAC) (Haarnoja et al., 2018, 2019) is a state-of-the-art off-policy reinforcement learning algorithm for continuous action settings. The goal of the algorithm is to find a policy that maximizes the maximum entropy objective:

$$\pi^* = \text{argmax}_\pi \sum_{i=0}^{\tau} \mathbb{E}_{(s_t, a_t) \sim d_\pi} \left[\gamma^t (R(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))\right]$$

where α is the temperature parameter, $\mathcal{H}(\pi(\cdot|s_t)) = -\log \pi(\cdot|s_t)$ is the entropy of the policy π at state s_t, d_π is the distribution of trajectories induced by policy π. The soft action-value function $Q_\theta(s_t, a_t)$ parameterized using a neural network with parameters θ is trained by minimizing the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim D} \left[\left(Q_\theta(s_t, a_t) - R(s_t, a_t) - \gamma \mathbb{E}_{s_{t+1} \sim T(s_t, a_t)} V_{\bar{\theta}}(s_{t+1})\right)^2\right] \tag{2}$$

where D is a replay buffer of past experience and $V_{\bar{\theta}}(s_{t+1})$ is estimated using a target network for Q and a Monte Carlo estimate of the soft state-value function after sampling experiences from the D.

The policy π is parameterized using a neural network with parameters ϕ. The parameters are learned by minimizing the expected KL-divergence between the policy and the exponential of the Q-function:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D}\big[\mathbb{E}_{a_t \sim \pi_\phi(\cdot|s_t)}\big[\alpha \log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t, a_t)\big]\big] \tag{3}$$

The objective for the temperature parameter is given by:

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t)}\big[-\alpha(\log \pi(a_t|s_t) + \bar{H})\big] \tag{4}$$

where $\bar{H}$ is a hyperparameter representing the target entropy. In practice, two separately trained soft Q-networks are maintained, and then the minimum of their two outputs are used to be the soft Q-network output.

While the original version of SAC solves problems with continuous action space, the version for discrete action spaces was suggested by Christodoulou (2019). In the case of discrete action space, $\pi_\phi(a_t|s_t)$ outputs a probability for all actions instead of a density. Such parametrization of the policy slightly changes the objectives 2, 3 and 4.

4. Relational Object-Centric Actor-Critic

Figure 2 outlines the high-level overview of the proposed actor-critic framework (ROCA). As an encoder we use SLATE (Singh et al., 2022), a recent object-centric model. SLATE incorporates a dVAE (van den Oord et al., 2018) for internal feature extraction, a GPT-like transformer (Ramesh et al., 2021) for decoding, and a slot-attention module (Locatello et al., 2020c) to group features associated with the same object. We refer to the appendix B.2 for a more detailed description of SLATE. In ROCA the pre-trained frozen SLATE model takes an image-based observation s_t as input and produces a set of object vectors, referred to as slots, $\boldsymbol{z}_t = (z_t^1, \ldots, z_t^K)$ (K - the maximum number of objects to be extracted). An actor model encapsulates the current agent's policy and returns an action for the input state $\boldsymbol{z}_t$. Critic predicts the value $Q(\boldsymbol{z}_t, a)$ of the provided action a sampled from the actor given the current state representations $\boldsymbol{z}_t$. It is estimated using the learned transition model, reward model, and state-value model. The input state representation $\boldsymbol{z}_t = (z_t^1, \ldots, z_t^K)$ is treated as a complete graph while being processed by GNN-based components of the ROCA.

4.1. Transition Model

We approximate the transition function using a graph neural network Kipf et al. (2020) with an edge model $\texttt{edge}_T$ and a node model $\texttt{node}_T$ which takes a factored state $\boldsymbol{z}_t = (z_t^1, \ldots, z_t^K)$ and action $\boldsymbol{a}_t$ as input and predicts changes in factored states $\Delta \boldsymbol{z}$. The action is provided to the node model $\texttt{node}_T$ and the edge model $\texttt{edge}_T$ as shown in the appendix in Figure 8. The factored representation of the next state is obtained via $\hat{\boldsymbol{z}}_{t+1} = \boldsymbol{z}_t + \Delta \boldsymbol{z}$. Since we treat the set of slots as a complete graph, the complexity of the update rule (5) is quadratic in the number of slots. The same applies to all GNN models in the ROCA.

$$\Delta z^i = \texttt{node}_T(z_t^i, a_t^i, \sum_{i \neq j} \texttt{edge}_T(z_t^i, z_t^j, a_t^i)) \tag{5}$$

4.2. Reward Model

The reward model uses almost the same architecture as the transition model. Still, we average object embeddings returned by the node models and feed the result into the MLP to produce the scalar reward. The reward model is trained using the mean squared error loss function with environmental rewards r_t as target (6).

$$\begin{cases} \texttt{embed}^i_R = \texttt{node}_R(z^i_t, a^i_t, \sum_{i \neq j} \texttt{edge}_R(z^i_t, z^j_t, a^i_t)) \\ \hat{R}(\boldsymbol{z}_t, a_t) = MLP(\sum_{i=1}^{K} \texttt{embed}^i_R / K) \end{cases} \tag{6}$$

4.3. State-Value Model

The state-value function is approximated using a graph neural network $\hat{V}$, which does not depend on actions in either the edge model $\texttt{edge}_V$ or the node model $\texttt{node}_V$. As in the reward model, we average object embeddings returned by the node models and feed the result into the MLP to produce the scalar value.

$$\begin{cases} \texttt{embed}^i_V = \texttt{node}_V(z^i_t, \sum_{i \neq j} \texttt{edge}_V(z^i_t, z^j_t)) \\ \hat{V}(\boldsymbol{z}_t) \quad = MLP(\sum_{i=1}^{K} \texttt{embed}^i_V / K) \end{cases} \tag{7}$$

4.4. Actor Model

The actor model uses the same GNN architecture as the state-value model but employs different MLP heads for continuous and discrete action spaces. In the case of the continuous action space, it returns the mean and the covariance of the Gaussian distribution. For the discrete action space, it outputs the probabilities for all actions.

$$\begin{cases} \texttt{embed}^i_{actor} = \texttt{node}_{actor}(z^i_t, \sum_{i \neq j} \texttt{edge}_{actor}(z^i_t, z^j_t)) \\ \mu(\boldsymbol{z}_t) \quad = MLP_\mu(\sum_{i=1}^{K} \texttt{embed}^i_{actor} / K) \\ \sigma^2(\boldsymbol{z}_t) \quad = MLP_{\sigma^2}(\sum_{i=1}^{K} \texttt{embed}^i_{actor} / K) \\ \pi(\boldsymbol{z}_t) \quad = MLP_\pi(\sum_{i=1}^{K} \texttt{embed}^i_{actor} / K) \end{cases} \tag{8}$$

4.5. Critic Model

In the critic, we use a world model to predict action-values. Specifically, we employ a Q-function decomposition based on the Bellman equation. It was initially introduced in the Q-learning TreeQN algorithm (Farquhar et al., 2018):

$$\hat{Q}(\boldsymbol{z}_t, a_t) = \hat{R}(\boldsymbol{z}_t, a_t) + \gamma \hat{V}(\boldsymbol{z}_t + \Delta \boldsymbol{z}) \tag{9}$$

where $\hat{R}$ — the reward model (6), $\hat{V}$ — the state-value model (7), $\boldsymbol{z}_t + \Delta \boldsymbol{z}$ — the next state prediction, generated by the transition model (5). Since the critic's output values are computed using the world model, we refer to our approach as a value-based model-based method.

4.6. Training

The SLATE model is pre-trained on the data set of trajectories collected with a uniform random policy (300K observations for Shapes2D tasks and 1M observations for the Object Reaching task). Following the original paper (Singh et al., 2022), we apply decay on the dVAE temperature τ from

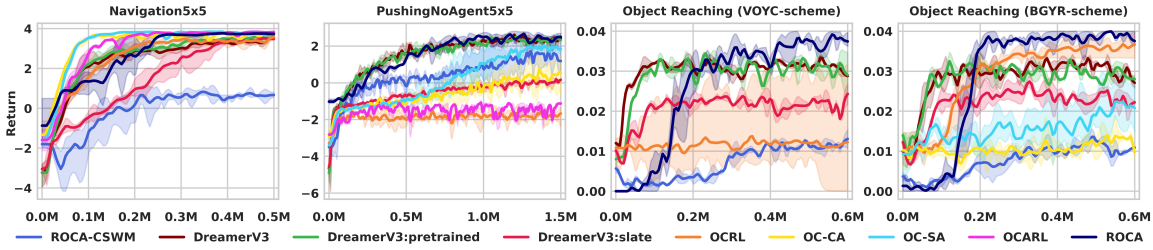

Figure 3: Return averaged over 30 episodes and three seeds for ROCA, ROCA-CSWM, DreamerV3, DreamerV3:pretrained, DreamerV3:slate, OCRL, OC-CA, OC-SA, and OCARL. ROCA learns faster or achieves higher metrics than the baselines. Shaded areas indicate standard deviation.

1.0 to 0.1 and a learning rate warm-up for the parameters of the slot-attention encoder and the transformer at the start of the training. After pre-training, we keep the parameters of the SLATE model frozen.

To train all the other components of ROCA we use SAC objectives (2, 3, 4). For both continuous and discrete environments, a conventional double Q-network architecture is used in the critic module. Additionally, we use the data sampled from the replay buffer to train the world model components. The transition model is trained using the mean squared error loss function to minimize the prediction error of the object representations for the next state, given the action. The reward model is trained using the mean squared error loss function with environmental rewards r_t as targets.

$$J_{WM} = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}\sim D}\left[\beta_T \|\boldsymbol{z}_t + \Delta\boldsymbol{z} - \boldsymbol{z}_{t+1}\|^2 + \beta_R\left(\hat{R}(\boldsymbol{z}_t, a_t) - r_t\right)^2\right] \tag{10}$$

In total, we use four optimizers. The temperature parameter, the actor, and the value model use individual optimizers. The transition and reward models share the world model optimizer.

Due to the stochastic nature of the SLATE model, object-centric representation can shuffle at each step. To enforce the order of object representation during the world model objective (10) optimization, we pre-initialize the slots of the SLATE model for the next state $\boldsymbol{z}_{t+1}$ with the current values $\boldsymbol{z}_t$.

5. Environments

The efficiency of the proposed ROCA algorithm was evaluated in the 3D robotic simulation environment CausalWorld (Ahmed et al., 2020) on the Object Reaching task as it was done in (Yoon et al., 2023), and in the compositional 2D environment Shapes2D (Kipf et al., 2020) on the Navigation and PushingNoAgent tasks. Current state-of-the-art slot-based object-centric models struggle to extract meaningful object-centric representations in visually complex environments (Locatello et al., 2020a; Engelcke et al., 2021). As a result, testing object-centric RL algorithms in visually rich environments, like Habitat (Szot et al., 2022), becomes challenging due to the low quality of representations. However, the visual complexity of the selected environments enables object-centric models to extract high-quality object representations. This allows us to focus on the problem of object-centric MBRL, which is the primary objective of this paper.

Object Reaching Task In this task, a fixed target object (violet cube) and a set of distractor objects (orange, yellow, and cyan cubes) are randomly placed in the scene. The agent controls a tri-finger robot and must reach the target object with one of its fingers (the other two are permanently fixed) to obtain a positive reward and solve the task. The episode ends without reward if the finger first touches one of the distractor objects. The action space in this environment consists of the three continuous joint positions of the moveable finger. During our experiments, we discovered that one of the baseline algorithms is sensitive to the choice of color scheme for the cubes. Therefore, we also conducted experiments in the task with the original color scheme (Yoon et al., 2023): the color of the target cube is blue, and the colors of the distracting cubes are red, yellow, and green.

Navigation Task Shapes2D environment is a four-connected grid world where objects are represented as figures of simple shapes. One object — the cross is selected as a stationary target. The other objects are movable. The agent controls all movable objects. In one step, the agent can move an object to any free adjacent cell. The agent aims to collide the controlled objects with the target object. Upon collision, the object disappears, and the agent receives a reward of $+1$. When an object collides with another movable object or field boundaries, the agent receives a reward of -0.1, and the positions of objects are not changed. For each step in the environment, the agent receives a reward of -0.01. The episode ends if only the target object remains on the field. In the experiments, we use a 5x5-sized environment with five objects and a 10x10-sized environment with eight objects. The action space in the Shapes2D environment is discrete and consists of 16 actions for the Navigation 5x5 task (four movable objects) and 28 actions for the Navigation 10x10 task (seven movable objects).

PushingNoAgent Task The agent controls all movable objects as in the Navigation task, but collisions between two movable objects are permitted: both objects move in the direction of motion. The agent is tasked to push another movable object into the target while controlling the current object. The pushed object disappears, and the agent receives a reward of $+1$ for such an action. When the currently controlled object collides with the target object or field boundaries, the agent receives a reward of -0.1. When the agent pushes a movable object into the field boundaries, the agent receives a reward of -0.1. For each step in the environment, the agent receives a reward of -0.01. The episode ends if only the target object and one movable object remain on the field. In the experiments, we use a 5x5-sized environment with five objects.

6. Experiments

We utilize a single SLATE model for Navigation5x5 and PushingNoAgent5x5 tasks as they share the same observation space. However, we train a distinct SLATE model for Navigation10x10 and each version of the Object Reaching task. The appendix provides detailed information regarding the hyperparameters of the SLATE model in B.2 and the examples of attention maps produced by SLATE in Figure 7.

In continuous Object Reaching tasks, we conventionally use the dimension of the action space as the target entropy hyperparameter for ROCA. For 2D tasks with a discrete action space, we scale the entropy of a uniform random policy with the tuned coefficient. For more information on the hyperparameters of the ROCA model, please refer to the appendix C.

We compare ROCA with an object-centric model-free algorithm based on PPO, using the same pre-trained frozen SLATE model as a feature extractor. To combine the latent object representations

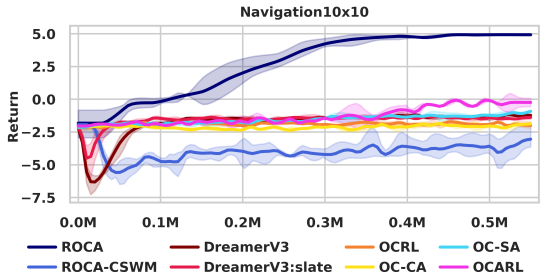

Figure 4: Return averaged over 30 episodes and three seeds for ROCA, ROCA-CSWM, DreamerV3, DreamerV3:slate, OCRL, OC-SA, OC-CA, and OCARL models in the Navigation 10x10 task. ROCA exhibits better performance than baselines but still does not solve the task. Shaded areas indicate standard deviation.

into a single vector suitable for the value and policy networks of the PPO, we used a Transformer encoder (Vaswani et al., 2023) as a pooling layer. We referred to the transformer-based PPO implementation provided by (Yoon et al., 2023) as the OCRL baseline. For the Object Reaching Task, we employed the same hyperparameter values as the authors. For Shapes2D tasks, we fine-tuned the hyperparameters of the OCRL baseline. The other object-centric model-free baselines are the self-attention OC-SA and cross-attention OC-CA versions from Stanić et al. (2022), as well as OCARL (Yi et al., 2022). The tested hyperparameters are listed in the appendix.

Since there are no established state-of-the-art object-centric MBRL algorithms, we have chosen the DreamerV3 (Hafner et al., 2023) algorithm as a MBRL baseline. In order to ensure a fair comparison between the ROCA and the DreamerV3, we conducted experiments where we trained the DreamerV3 with a pretrained encoder obtained from the DreamerV3 model that solves the task (DreamerV3:pretrained). For all the tasks, we conducted experiments using two different modes: one with the encoder frozen and another with the encoder unfrozen. However, we did not observe any improvement in the convergence rate compared to the DreamerV3 model that does not use the pretrained encoder. Additionally, we discovered that the pretrained world model significantly accelerates the convergence of DreamerV3, but this mode makes the comparison unfair to the ROCA. We also utilize the pretrained frozen SLATE model as an encoder in DreamerV3. In DreamerV3:slate, we obtain a single-vector representation of the current observation by concatenating the slot representations produced by SLATE and then feed it into DreamerV3. For the DreamerV3-based algorithms we use the default hyperparameter values from the official repository, except for the `train_ratio` parameter which we increased from 32 to 512, as we observed that more frequent training of models improves sample efficiency. Additionally, we implement ROCA-CSWM baseline, where we utilize CNN-based object encoder and train it online along with the world model using contrastive loss Kipf et al. (2020):

$$
\begin{aligned}
J_{WM}^{cswm} = \mathbb{E}_{s_t,a_t,r_t,s_{t+1}\sim D}\big[&\beta_T \|\boldsymbol{z}_t + \Delta\boldsymbol{z} - \boldsymbol{z}_{t+1}\|^2 \\
&+ \beta_R\big(\hat{R}(\boldsymbol{z}_t, a_t) - r_t\big)^2 + \beta_C \max\big(0, \gamma - \|\hat{\boldsymbol{z}}_t - \boldsymbol{z}_{t+1}\|^2\big)\big],
\end{aligned}
\tag{11}
$$

where γ is the margin and $\hat{\boldsymbol{z}}_t$ is the representation of the observation randomly sample from the replay buffer. Other loss functions in ROCA and ROCA-CSWM are the same.

The hyperparameters for ROCA-CSWM can be found in the appendix D.

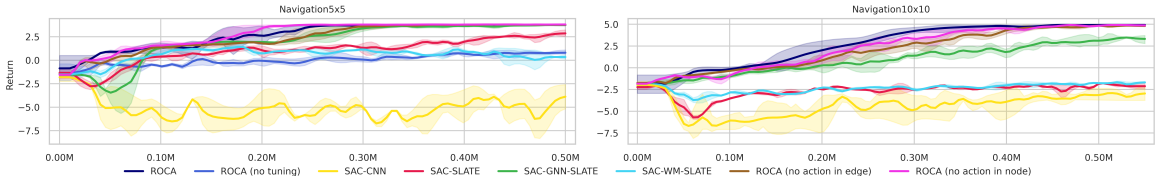

Figure 5: Ablation study. SAC-CNN — a version of SAC with a standard CNN encoder. SAC-SLATE — a version of SAC with a pretrained SLATE encoder which averages object emebeddings to obtain the embedding of the current state. SAC-WM-SLATE — a modification of SAC-SLATE which uses a monolithic world-model in its critic. SAC-GNN-SLATE — an object-centric version of SAC with a pretrained SLATE encoder which uses GNNs as actor and critic. ROCA (no-tuning) — a version of ROCA without target entropy tuning. ROCA (no action in edge) — a version of ROCA, in which `edge` functions do not take an action as input. ROCA (no action in node) — a version of ROCA, in which `node` functions do not take an action as input. ROCA and ROCA (no action in node) demonstrate similar performance. ROCA outperforms the other considered baselines. Return averaged over 30 episodes and three seeds. Shaded areas indicate standard deviation.

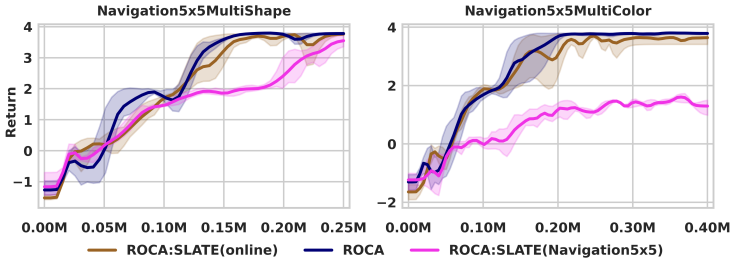

Figure 6: Return averaged over 30 episodes and three seeds for ROCA, ROCA:SLATE(online), ROCA:SLATE(Navigation5x5) in the modified versions of the Navigation5x5 task. These tasks introduce distribution shifts for the frozen SLATE used in ROCA:SLATE(Navigation5x5). ROCA and ROCA:SLATE(online) perform comparably, whereas ROCA:SLATE(Navigation5x5) fails to solve the Navigation5x5MultiColor task. Shaded areas indicate standard deviation.

Results The graphs in Figure 3 depict how the episode return of ROCA and the baselines depend on the number of steps for Navigation 5x5, PushingNoAgent5x5, and two versions of the Object Reaching task. For the Navigation 5x5 task, ROCA performs better than the OCRL baseline. Although DreamerV3 shows slightly more stable and efficient learning than ROCA, ROCA eventually achieves a higher return. In the PushingNoAgent 5x5 task, ROCA and DreamerV3 demonstrate similar performance. For the Object Reaching task with the original color scheme (BGYR), the OCRL baseline initially demonstrates much better performance, but ROCA surpasses all baselines after 200K steps. We believe that the poor performance of the OCRL baseline in the Object Reaching task with VOYC color schema is due to its sensitivity to the quality of the SLATE model. One potential solution to overcome this issue could be increasing the number of training epochs for

the SLATE. Figure 4 demonstrates the results in the more challenging Navigation 10x10 task. All baselines fail to achieve a positive return. ROCA performs better than the baselines but can not solve the task entirely, as it only moves five out of seven objects to the target. In all tasks, DreamerV3 outperforms DreamerV3:slate, which indicates that DreamerV3 cannot efficiently utilize object-centric representations. The results for ROCA-CSWM highlight the difficulties of using contrastive loss for training the world model and the object encoder in an online setting.

Ablations ROCA is built upon SAC, and thus, the ablation study aims to assess the impact of the different modifications we introduced to the original SAC with a monolithic CNN encoder. Figure 5 illustrates the results of additional experiments estimating the effects of the pre-trained SLATE encoder, the object-centric actor and critic, the object-centric world model and the target entropy tuning. We evaluate the quality of several monolithic and object-centric versions of SAC and compare them with ROCA. SAC-CNN is standard monolithic version of SAC that utilizes the convolutional encoder from the original DQN implementation (Mnih et al., 2015). In SAC-SLATE, the CNN encoder is replaced with a pre-trained frozen SLATE encoder, while the other model components remain the same. To obtain the monolithic state representation z_t^* from the object-centric one $\boldsymbol{z}_t$, produced by the SLATE, we take the average over the object axis: $z_t^* = \sum_{i=0}^{K} z_t^i / K$. Note, that z_t^* is independent of the slot order in $\boldsymbol{z}_t$ and can be fed into the standard actor and critic MLPs. SAC-WM-SLATE builds upon SAC-SLATE and can be considered as a monolithic version of the ROCA. Its actor, state-value, reward, and transition models are implemented using MLPs. SAC-GNN-SLATE is an object-centric version of SAC and can be viewed as ROCA without the world model in the critic module. It uses a pretrained frozen SLATE encoder and GNN-based actor and critic modules. Additionally, we compare the ROCA with a variant where the target entropy is set to the default value, equal to the scaled entropy of the uniform random policy with coefficient 0.98 (Christodoulou, 2019). Also we compare ROCA with a variant ROCA (without actions in GNN edges) in which the `edge` functions of the transition (5) model and the reward model (6) do not take the action a_t as input. This architecture strictly follows the CSWM model (Kipf et al., 2020).

The ablation studies have shown that in the monolithic mode, the SLATE model significantly improves performance only in the relatively simple Navigation5x5 task. However, extending the critic with the world model does not improve the convergence rate. The object-centric SAC-GNN-SLATE outperforms all monolithic models. Finally, the ROCA, which uses an object-centric world model in the critic module, outperforms the SAC-GNN-SLATE. Note that we obtained the presented results after fine-tuning the hyperparameters for all of the models. Also we observe that providing actions to the `edge` function in GNNs slightly improve performance of the our algorithm.

Performance with Frozen SLATE Under Distribution Shifts We conducted two experiments to investigate the performance of our approach under distribution shifts. Specifically, we used SLATE trained on the Navigation5x5 task but train ROCA on modified versions of Navigation5x5. In the Navigation5x5MultiShape task, the shapes of three objects were randomly sampled at the beginning of each episode (with a probability of 0.5 for each object): the red circle could be replaced with a red diamond, the blue triangle with a blue pentagon, and the green square with a green scalene triangle. In the Navigation5x5MultiColor task, the colors of these objects were randomly sampled at the beginning of each episode (also with a probability of 0.5 for each object): the red circle could be replaced with a yellow circle, the blue triangle with a brown triangle, and the green square with a pink square. Both Navigation5x5MultiShape and Navigation5x5MultiColor introduce distribution shifts compared to the original Navigation5x5 task. We evaluated three versions of

ROCA in these experiments. ROCA:SLATE(Navigation5x5) — ROCA with a frozen SLATE trained on data collected from the original Navigation5x5 task. ROCA:SLATE(online) — ROCA with SLATE initialized with weights from the Navigation5x5 task but trained further online along with the policy on data from the replay buffer. ROCA — The standard version of ROCA with a frozen SLATE trained on data collected from the current task.

The results, presented in Figure 6, show that ROCA converges the fastest, as expected, since it does not experience any distribution shift. ROCA:SLATE(online) learns slightly slower but successfully converges in both tasks. In contrast, ROCA:SLATE(Navigation5x5) learns the slowest and only solves the Navigation5x5MultiShape task, failing to converge in Navigation5x5MultiColor within 400K steps. From these results, we conclude that ROCA can adapt to distribution shifts between offline and online data by training SLATE alongside the policy on data from the replay buffer.

The results of additional experiments with DreamerV3 using a pre-trained frozen encoder, sparsification of the GNNs in ROCA, evaluation of ROCA with the DINOSAur model instead of SLATE, and evaluation of ROCA on tasks with low-dimensional vector states are presented in the appendix.

7. Conclusion and Future Work

We presented ROCA, an object-centric off-policy value-based model-based reinforcement learning approach that uses a pre-trained SLATE model as an object-centric feature extractor. Our experiments in 3D and 2D tasks demonstrate that ROCA learns effective policies and outperforms object-centric model-free and model-based baselines. The world model is built upon a GNN architecture, showing that graph neural networks can be successfully applied in MBRL settings for policy learning. While we use the SLATE model as an object-centric feature extractor, in principle, we can replace SLATE with other slot-based object-centric models. However, ROCA does have limitations. Firstly, its world model is deterministic and may struggle to predict the dynamics of highly stochastic environments. Additionally, as our model is based on the SAC algorithm, it is sensitive to the target entropy hyperparameter, especially in environments with discrete action spaces (Xu et al., 2021; Zhou et al., 2023).

In our future work, we consider the primary task to be evaluating ROCA in more visually challenging environments. To accomplish this, we plan to replace SLATE with the recently proposed DLP (Daniel and Tamar, 2024) model, which has shown promising results in goal-conditioned tasks when paired with the model-free algorithm (Haramati et al., 2024).

Acknowledgments

This work was supported by Russian Science Foundation, grant No. 20-71-10116, https://rscf.ru/en/project/20-71-10116. The research was carried out using the infrastructure of the Shared Research Facilities "High Performance Computing and Big Data" (CKP "Informatics") of FRC CSC RAS (Moscow).

References

Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Yoshua Bengio, Bernhard Schölkopf, Manuel Wüthrich, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning, 2020.

Ondrej Biza, Robert Platt, Jan-Willem van de Meent, Lawson L.S. Wong, and Thomas Kipf. Binding actions to objects in world models. In *ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*, 2022. URL https://openreview.net/forum?id=HImz8BuUclc.

Christopher P. Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation, 2019.

Petros Christodoulou. Soft actor-critic for discrete action settings, 2019.

Tal Daniel and Aviv Tamar. Ddlp: Unsupervised object-centric video prediction with deep dynamic latent particles, 2024. URL https://arxiv.org/abs/2306.05957.

Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 240–247, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390187. URL https://doi.org/10.1145/1390156.1390187.

Martin Engelcke, Oiwi Parker Jones, and Ingmar Posner. Genesis-v2: Inferring unordered object representations without iterative refinement. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8085–8094. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/43ec517d68b6edd3015b3edc9a11367b-Paper.pdf.

Martin Engelcke, Oiwi Parker Jones, and Ingmar Posner. Genesis-v2: Inferring unordered object representations without iterative refinement, 2022.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning, 2018.

Stefano Ferraro, Pietro Mazzaglia, Tim Verbelen, and Bart Dhoedt. Focus: Object-centric world models for robotics manipulation, 2023. URL https://arxiv.org/abs/2307.02427.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL https://arxiv.org/abs/1802.09477.

Niklas Funk, Georgia Chalvatzaki, Boris Belousov, and Jan Peters. Learn2assemble with structured representations and search for robotic architectural construction. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1401–1411. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/funk22a.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models, 2022. URL https://arxiv.org/abs/2010.02193.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2023.

Dan Haramati, Tal Daniel, and Aviv Tamar. Entity-centric reinforcement learning for object manipulation from pixels, 2024. URL https://arxiv.org/abs/2404.01220.

Jindong Jiang, Sepehr Janghorbani, Gerard de Melo, and Sungjin Ahn. Scalor: Generative world models with scalable object representations, 2020. URL https://arxiv.org/abs/1910.02384.

Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, Imyhxy, , Lorna, (Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, Tkianai, YxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - yolov5 sota realtime instance segmentation, 2022. URL https://zenodo.org/record/3908559.

Ramtin Keramati, Jay Whang, Patrick Cho, and Emma Brunskill. Fast exploration with simplified models and approximately optimistic planning in model based reinforcement learning, 2018.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gax6VtDB.

Daniil Kirilenko, Alexandr Korchemnyi, Konstantin Smirnov, Alexey K Kovalev, and Aleksandr I Panov. Quantized disentangled representations for object-centric visual tasks. In *Pattern Recognition and Machine Intelligence. PReMI 2023. Lecture Notes in Computer Science*, volume 14301, pages 514–522. Springer Cham, 2023. doi: 10.1007/978-3-031-45170-6_53. URL https://link.springer.com/chapter/10.1007/978-3-031-45170-6_53.

Daniil Kirilenko, Vitaliy Vorobyov, Alexey Kovalev, and Aleksandr Panov. Object-centric learning with slot mixture module. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=aBUidW4Nkd.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023. URL https://arxiv.org/abs/2304.02643.

Zhixuan Lin, Yi-Fu Wu, Skand Vishwanath Peri, Weihao Sun, Gautam Singh, Fei Deng, Jindong Jiang, and Sungjin Ahn. Space: Unsupervised object-oriented scene representation via spatial attention and decomposition, 2020. URL https://arxiv.org/abs/2001.02407.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11525–11538. Curran Associates, Inc., 2020a. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/8511df98c02ab60aea1b2356c013bc0f-Paper.pdf.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention, 2020b.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention, 2020c.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021. doi: 10.1109/JPROC.2021.3058954.

Maximilian Seitzer, Max Horn, Andrii Zadaianchuk, Dominik Zietlow, Tianjun Xiao, Carl-Johann Simon-Gabriel, Tong He, Zheng Zhang, Bernhard Schölkopf, Thomas Brox, and Francesco Locatello. Bridging the gap to real-world object-centric learning, 2023.

Vishal Sharma, Aniket Gupta, Prayushi Faldu, Rushil Gupta, Mausam ., and Parag Singla. Object-centric learning of neural policies for zero-shot transfer over domains with varying quantities of interest. In *PRL Workshop Series – Bridging the Gap Between AI Planning and Reinforcement Learning*, 2023. URL https://openreview.net/forum?id=mQtyk75pYZ.

Gautam Singh, Fei Deng, and Sungjin Ahn. Illiterate dall-e learns to compose. In *ICLR*, 2022.

Aleksandar Stanić, Yujin Tang, David Ha, and Jürgen Schmidhuber. Learning to generalize with object-centric agents in the open world survival game crafter, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat, 2022.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai, 2024. URL https://arxiv.org/abs/2410.00425.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P. Burgess, and Alexander Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration, 2019.

Ziyi Wu, Nikita Dvornik, Klaus Greff, Thomas Kipf, and Animesh Garg. Slotformer: Unsupervised visual dynamics simulation with object-centric models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=TFbwV6I0VLg.

Yaosheng Xu, Dailin Hu, Litian Liang, Stephen McAleer, Pieter Abbeel, and Roy Fox. Target entropy annealing for discrete soft actor-critic, 2021.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data, 2021.

Qi Yi, Rui Zhang, Shaohui Peng, Jiaming Guo, Xing Hu, Zidong Du, Xishan Zhang, Qi Guo, and Yunji Chen. Object-category aware reinforcement learning, 2022. URL https://arxiv.org/abs/2210.07802.

Jaesik Yoon, Yi-Fu Wu, Heechul Bae, and Sungjin Ahn. An investigation into pre-training object-centric representations for reinforcement learning, 2023.

Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Self-supervised visual reinforcement learning with object-centric representations, 2020. URL https://arxiv.org/abs/2011.14381.

Andrii Zadaianchuk, Georg Martius, and Fanny Yang. Self-supervised reinforcement learning with independently controllable subgoals, 2022a.

Andrii Zadaianchuk, Georg Martius, and Fanny Yang. Self-supervised reinforcement learning with independently controllable subgoals, 2022b. URL https://arxiv.org/abs/2109.04150.

Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-2FCwDKRREu.

Artem Zholus, Yaroslav Ivchenkov, and Aleksandr Panov. Factorized World Models for Learning Causal Relationships. In *ICLR Workshop on the Elements of Reasoning: Objects, Structure and Causality*, 2022. URL https://openreview.net/forum?id=BCGfDBOIcec.

Haibin Zhou, Zichuan Lin, Junyou Li, Qiang Fu, Wei Yang, and Deheng Ye. Revisiting discrete soft actor-critic, 2023.

Appendix A. Environments

A.1. Shapes2D

- **Navigation 5x5**: Number of objects: 5; Observation size: 50x50; Number of actions: 16; Episode length: 100.
- **PushingNoAgent 5x5**: Number of objects: 5; Observation size: 50x50; Number of actions: 16; Episode length: 100.
- **Navigation 10x10**: Number of objects: 8; Observation size: 100x100; Number of actions: 28; Episode length: 100.

A.2. CausalWorld

- **Object Reaching**: Observation size: 64x64; Action dimensionality: 3; Episode length: 100.

Appendix B. Object-Centric Representation Models

B.1. Datasets

Shapes2D Training datasets of 300K observations are collected using a uniform random policy for every task, with the exception of Navigation 5x5 and PushingNoAgent 5x5. Since these tasks share the same observation space, we use only the Navigation 5x5 dataset.

CausalWorld For the Object Reaching task with the VOYC color scheme, we collect following the procedure described in the official OCRL repository. For the BGYR color scheme, we do not collect data as we use the pretrained SLATE model provided by the authors.

B.2. SLATE

Our code is based on the OCRL repository: https://github.com/jsikyoon/OCRL. Hyperparameters for SLATE are listed in Tables 1 and 2. For the Object Reaching task with the BGYR color scheme, we use model weights from the original OCRL repository.

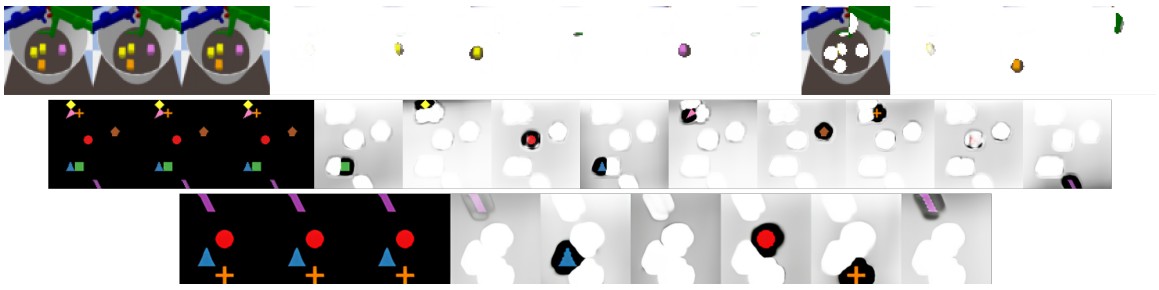

Figure 7: Visualization of attention maps produced by SLATE in Object Reaching and Shapes2D tasks

		Shapes2D	CausalWorld
Learning	Training dataset size	300000	1000000
	Temp. Cooldown	1.0 to 0.1	
	Temp. Cooldown Steps	30000	
	LR for DVAE	0.0003	
	LR for CNN Encoder	0.0001	
	LR for Transformer Decoder	0.0003	
	LR Warm Up Steps	30000	
	LR Half Time	250000	
	Dropout	0.1	
	Clip	0.05	
	Batch Size	32	
	Epochs	100	
DVAE	Vocabulary Size	32	4096
CNN Encoder	Hidden Size	64	
Slot Attention	Iterations	5	3
	Slot Heads	1	
	Slot Dim.	64	192
	MLP Hidden Dim.	256	192
Transformer Decoder	Layers	4	
	Heads	4	
	Hidden Dim.	128	192

Table 1: Common Hyperparameters for SLATE

Task	Resize	Num. Slots	Pos Channels
Navigation 5x5	64	6	5
PushingNoAgent 5x5	64	6	5
Navigation 10x10	96	9	8
Object Reaching	64	10	4

Table 2: Task-Specific Hyperparameters for SLATE

Appendix C. ROCA

`edge` and `node` models in the GNN-based Transition model, Reward model, and Actor model are MLPs consisting of three hidden layers with 512 units each, along with LayerNorm and ReLU activations. The MLPs following the concatenation-readout layer in the Actor, Critic, Reward Predictor, and Continue Predictor models consist of a single layer with 512 units.

The target entropy parameter is set to -3 for the Object Reaching task. For tasks with a discrete action space, we scale the entropy of a uniform random policy with a coefficient of 0.6. This results

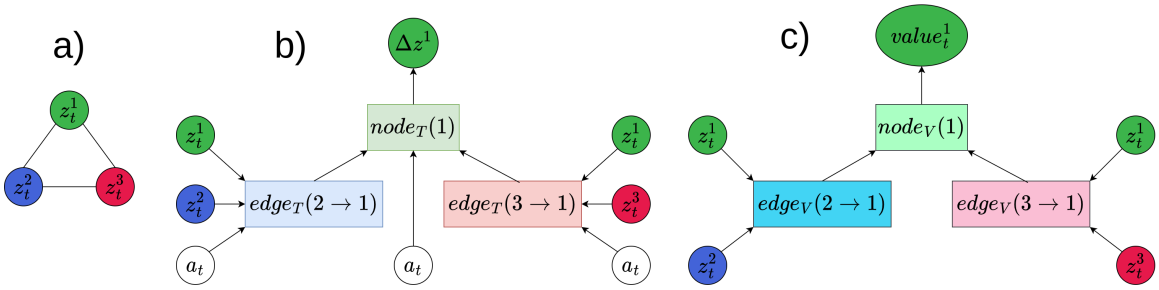

Figure 8: Overview of GNN-based transition model and state-value model. **a)** Representation of the state as a complete graph. **b)** Transition model: message-passing update scheme for the embedding of object 1. **c)** State-value model: message-passing update scheme for the state-value prediction for the object 1.

Gamma	0.99
Buffer size	1000000
Batch size	128
τ_{polyak}	0.005
β_T	1
β_R	1
Buffer prefill size	5000
Number of parallel environments	16
Learning rate	0.0003
Train frequency	Every step in environment
Target network update frequency	Every training step

Table 3: Hyperparameters for ROCA

in values of 1.66 for the Navigation 5x5 and PushingNoAgent 5x5 tasks, and 2 for the Navigation 10x10 task. The other hyperparameters are listed in Table 3.
Tested hyperparameters:

- Target entropy (Object Reaching): [-1, -2, -3, -4, -5]
- Target entropy (Shapes2D) — scaling coefficients for the entropy of a uniform random distribution: [0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
- Batch size: [64, 128, 256]
- Target network update frequency (every N training steps): [1, 2, 3]

Appendix D. ROCA-CSWM

We use a CSWM-like encoder (https://github.com/tkipf/c-swm), which consists of four convolutional layers with BatchNorm layers and ReLU activations between them:

- kernel_size: 3, stride: 1, padding: 1, out_channels: 32
- kernel_size: 3, stride: 1, padding: 1, out_channels: 32
- kernel_size: 5, stride: 2, padding: 1, out_channels: 32
- kernel_size: 5, stride: 2, padding: 1, out_channels: `num_slots`

The value of the `num_slots` parameter depends on the task: Navigation 5x5 and PushingNoAgent 5x5 — 5, Navigation 10x10 — 8, Object Reaching — 6.

The output of the final convolutional layer is flattened and fed into an MLP consisting of three layers with ReLU activations between them. This MLP is shared between slots. Before the final layer, we use layer normalization. We use 128 hidden units in the MLP. The output dimension of the final layer in the MLP is the same as the Slot Dim. parameter in Table 1. The other hyperparameters of ROCA-CSWM are listed in Table 4.

Gamma	0.99
Buffer size	1000000
Batch size	256
τ_{polyak}	0.005
β_T	1
β_R	1
β_C	1
γ_{margin}	0.025
Buffer prefill size	5000
Number of parallel environments	16
Learning rate	0.0003
Train frequency	Every step in environment
Target network update frequency	Every training step

Table 4: Hyperparameters for ROCA-CSWM

Tested hyperparameters:

- Target entropy (Object Reaching): [-1, -2, -3, -4, -5]
- Target entropy (Shapes2D): Scaling coefficients for the entropy of a uniform random distribution: [0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
- Batch size: [64, 128, 256, 512]
- Target network update frequency (every N training steps): [1, 2, 3]
- γ_{margin}: [5, 1, 0.5, 0.025, 0.01]

Appendix E. DreamerV3, DreamerV3:pretrained

For DreamerV3, we use the `medium` preset for configuration parameters with the `train_ratio` parameter increased to 512. We also ran experiments with the `large` and `xlarge` presets but did not notice any improvements. We resize observations to the same sizes as OCR models (Table 2).
Our code is based on the original DreamerV3 repository: https://github.com/danijar/dreamerv3

Appendix F. DreamerV3:slate

Similarly, for DreamerV3:slate, we use the `medium` preset because the `large` and `xlarge` presets did not produce performance gains. We also set the `train_ratio` parameter to 512. The input of DreamerV3:slate is the concatenation of slots, which are vectors of size `num_slots` × `slot_dim` (Tables 1 and 2).
Our code is based on the original DreamerV3 repository: https://github.com/danijar/dreamerv3

Appendix G. OCRL

Our code is based on the OCRL repository: https://github.com/jsikyoon/OCRL
PPO-related tested hyperparameters:

- Entropy coefficient: [0, 0.001, 0.01, 0.025, 0.05, 0.075, 0.1]
- Clip range: [0.1, 0.2, 0.4]
- Epochs: [10, 20, 30]
- Batch size: [32, 64, 128]
- GAE lambda: [0.90, 0.95]

The best hyperparameters we found are:

- Entropy coefficient: 0.01
- Clip range: 0.2
- Epochs: 10
- Batch size: 32
- GAE lambda: 0.95

Appendix H. Ablation Study

We perform a grid search to find the best hyperparameters for the baselines. We use the same parameter ranges as those used for ROCA.

Appendix I. Additional Experiments with DreamerV3

To ensure a fair comparison with DreamerV3, we conducted experiments using a pretrained encoder obtained from the DreamerV3 model that solves the task. For all tasks, we tested two different modes: one with the encoder frozen and another with the encoder unfrozen. However, we did not observe any improvement in the convergence rate. The results are shown in Figure 9.

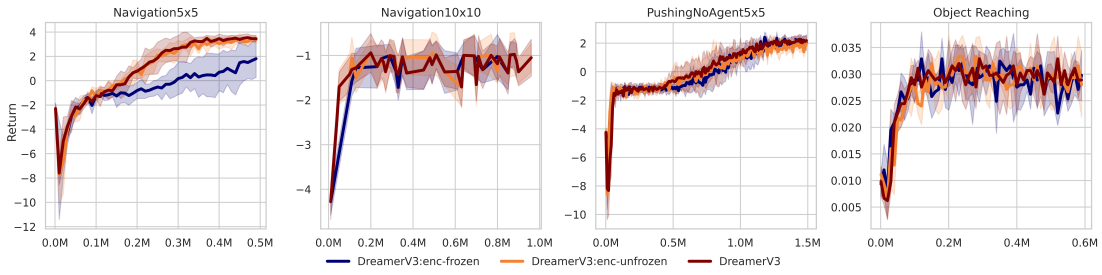

Figure 9: The plots illustrate the impact of encoder pretraining on the DreamerV3 algorithm. *DreamerV3* is the default version that trains its encoder from scratch. *DreamerV3:enc-frozen* is a version with a pretrained frozen encoder. *DreamerV3:enc-unfrozen* is a version with a pretrained unfrozen encoder. Returns are averaged over 30 episodes and three seeds.

Appendix J. Evaluation of Out-of-Distribution Generalization to Unseen Colors

We evaluated the generalization of ROCA to unseen colors of distractor objects in the Object Reaching task. When tested with the same colors it was trained on, the model achieved a success rate of 0.975 ± 0.005. However, when new colors were used for the distractor objects, the success rate dropped to 0.850 ± 0.005. The results were averaged over three instances of the ROCA model, with each instance evaluated over 100 episodes.

Appendix K. Sparse Interactions Modeling

In ROCA, we assume dense interactions among objects, modeled using complete graphs. However, interactions in real-world scenarios often occur sparsely rather than densely. While GNNs can emulate these naturally sparse interactions through their neural representations, some works explicitly model these interactions using a sparse graph Zadaianchuk et al. (2022a). To determine if modeling sparse interactions could improve our algorithm's efficiency, we implemented two modifications.

In **ROCA:sparse (GNN)**, we implemented an additional GNN-based $Interaction_{GNN}$ model. The `node` function of this model produces a weight $0 \leq w_t^i \leq 1$ for each object. This weight represents the degree of involvement of object i at time step t in interactions. These weights w_t^i are used as multipliers for `edge` functions in the transition, reward, state-value, and actor models. For example, the modified Transition model is:

$$\Delta z^i = \texttt{node}_T(z_t^i, a_t^i, \sum_{j \neq i} w_t^i w_t^j \texttt{edge}_T(z_t^i, z_t^j, a_t)). \tag{12}$$

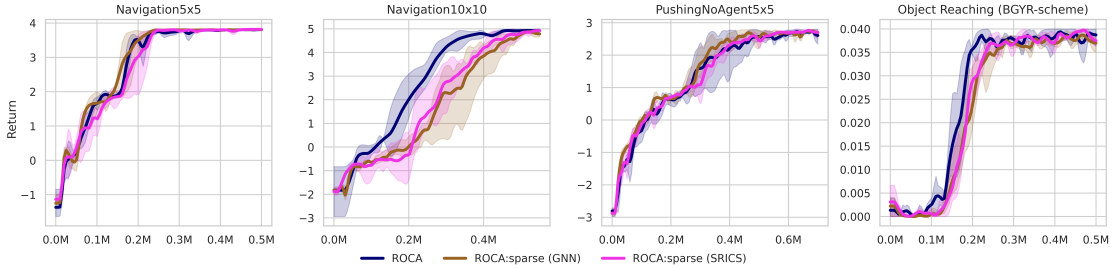

Figure 10: The plots illustrate the performance of variations of the ROCA model with explicit modeling of sparse interactions between objects. The results show that ROCA performs better than the modifications.

No additional loss function is used to train the $Interaction_{GNN}$ model.

In **ROCA:sparse (SRICS)**, we employ the approach proposed with the SRICS model Zadaianchuk et al. (2022a). We implement an $Interaction_{SRICS}$ model as an `edge` function. The output w_t^{ij} of $Interaction_{SRICS}$ is treated as a parameter of a Bernoulli distribution. A value of one indicates an interaction between objects i and j at time step t, while a value of zero indicates no interaction. Similar to $Interaction_{GNN}$, we use w_t^{ij} as a multiplier for `edge` functions in the transition, reward, state-value, and actor models. For example, the modified Transition model is:

$$\Delta z^i = \texttt{node}_T(z_t^i, a_t^i, \sum_{j \neq i} w_t^{ij} \texttt{edge}_T(z_t^i, z_t^j, a_t)). \tag{13}$$

To enforce sparsity, we use the Kullback-Leibler divergence $D_{KL}(w_t^{ij}, p_{prior})$ between the Bernoulli distribution induced by $Interaction_{SRICS}$ and the prior distribution p_{prior}:

$$p_{prior}(k; p) = \begin{cases} p, & \text{if } k = 1, \\ 1 - p, & \text{if } k = 0. \end{cases} \tag{14}$$

As in the original implementation Zadaianchuk et al. (2022a), we use $p = 0.05$, which means a low probability of interactions. We add the KL-divergence to the loss functions of the Critic, Actor, and World Model.

Figure 10 demonstrates the results of our experiments with the ROCA modifications. We did not observe any significant improvements from modeling sparse interactions.

Appendix L. Performance of ROCA with DINOSAur Slot-Extractor Model

We additionally trained ROCA, OCRL, and DreamerV3 baselines on the PushCube task in the ManiSkill (Tao et al., 2024) environment. In both OCRL and ROCA, we replaced SLATE with DINOSAur (Seitzer et al., 2023), as it is expected to perform better in more visually complex domains. In our experiments, we found that selecting four slots yielded the best average FG-ARI (foreground-adjusted Rand Index) value of 65% ± 10% on a validation set of 10K images (compared to three, five, and six slots). To calculate FG-ARI, we used ground-truth four-class segmentation: background, robotic arm, target goal region, and cube. The results are presented in Figure 11. We

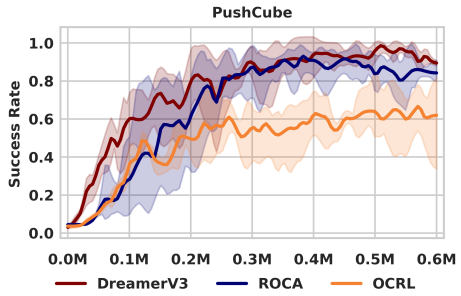

Figure 11: The plots illustrate the performance of ROCA and the baselines on the PushCube task in the Maniskill environment. Success rate is averaged over 30 episodes and three seeds.

found that ROCA outperforms OCRL but learns more slowly than DreamerV3. We attribute this to the lower quality of object-centric representations produced by DINOSAur in the PushCube task, compared to SLATE in the Shapes2D environment, where FG-ARI scores exceed 90%.

Appendix M. Performance of ROCA with Low-dimensional Vector States

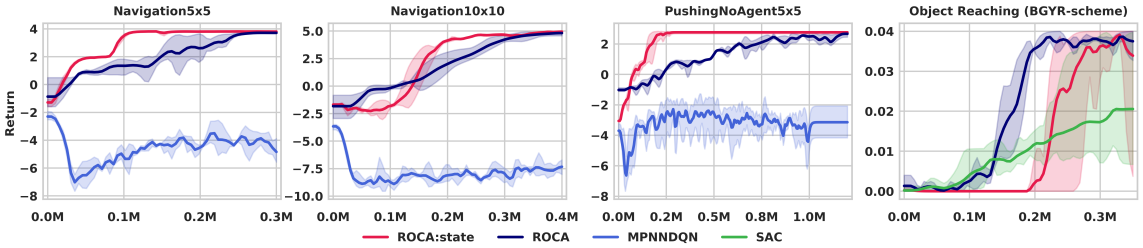

Figure 12: The plots illustrate the performance of ROCA and models that use low-dimensional vector states as input: MPNNDQN and ROCA:state. Since MPNNDQN cannot be used in tasks with continuous action spaces, we replace it with SAC for the Object Reaching task. The return is averaged over 30 episodes and three seeds.

We conducted experiments on variants of the Shapes2D and Object Reaching tasks, where ROCA is trained on low-dimensional states of objects. We compare the performance of the standard ROCA algorithm (as presented in this paper) with ROCA:state (where vector states of objects are used as input instead of slots) and MPNNDQN (Funk et al., 2022)—a modification of DQN (Mnih et al., 2015) based on a multi-head attention GNN.

MPNNDQN is built on top of DQN, and because of this, it cannot be applied to tasks with continuous action spaces, such as Object Reaching. The original architecture of MPNNDQN is tightly coupled to the environment, as described in the paper. It assumes that the environment provides information about which vertices are adjacent in the graph representation of the scene. However, our environments do not provide this information, so in our adaptation, we use complete graphs. Additionally, MPNNDQN was designed for action spaces with a hierarchical structure, where actions are defined relative to pairs of objects. As a result, the original MPNNDQN produces sets of action

values for every pair of objects. In contrast, our environments require action values for the entire scene, so we add a non-learned readout layer that pools these values over all pairs by summation. The results, presented in Figure 12, show that ROCA:state outperforms the standard ROCA on the Shapes2D tasks. In contrast, ROCA converges faster than ROCA:state on the Object Reaching task. For our version of MPNNDQN, we performed a limited hyperparameter search, but the algorithm still underperforms. We attribute this behavior to the specific use case for which MPNNDQN was originally designed. For completeness, we also include results from Soft Actor-Critic (SAC) on the Object Reaching task, demonstrating that ROCA:state outperforms SAC

Appendix N. Compute Resources

The experiments were carried out on a cluster with eight Nvidia Tesla V100 GPUs (32 GB each), 48 Intel(R) Xeon(R) Gold 2.60 GHz CPU cores, and 1536 GB RAM. We also used a cluster with three Nvidia Titan RTX GPUs (24 GB each), 80 Intel(R) Xeon(R) Gold 2.60 GHz CPU cores, and 1024 GB RAM. Additionally, we occasionally used a desktop computer with an Nvidia RTX 2070 GPU, 6 Intel(R) Core i5-9600K CPUs, and 32 GB RAM. The rough estimate of the total computational cost required to reproduce the experiments is approximately 13,000 CPU-hours and 5,000 GPU-hours.