Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische Friedrich-Wilhelms-Universität Bonn

Master Thesis

Learning Object-centric Latent Dynamics for Model-based Continuous Control

Author:
Jan Niklas Ewertz

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Prof. Dr. Maren Bennewitz

Date: September 17, 2024

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Bonn, September 17, 2024

Place, Date

Signature

Abstract

This thesis introduces a novel approach to model-based reinforcement learning that leverages object-centric representations to enhance sample efficiency, performance, and interpretability in visual control tasks. We extend an Object-Centric Video Prediction (OCVP) framework to function in an action-conditioned manner and integrate it with the Dreamer algorithm, developing the first object-centric model-based reinforcement learning method capable of solving continuous control tasks using only visual input. Our approach learns a structured world model that predicts environmental dynamics in terms of individual objects, allowing for more granular and interpretable scene representations. To effectively utilize these object-centric representations within the reinforcement learning framework, we develop specialized modules that enable reasoning over object-based state descriptions.

Extensive experiments on a suite of simulated robotic manipulation tasks demonstrate the effectiveness of our approach. Our object-centric model consistently outperforms both a non-object-centric baseline with significantly more parameters and the state-of-the-art DreamerV3 algorithm, particularly in tasks requiring relational reasoning between objects. The method exhibits improved sample efficiency and rapid learning, highlighting its potential for applications where data collection is costly or time-consuming. Furthermore, we show that our object-centric models are capable of adjusting to the changing state-distribution characteristic of many non-trivial reinforcement learning problems, overcoming a key limitation of prior object-centric reinforcement learning methods.

Our work demonstrates that learning object-centric representations from pixels is a viable paradigm for model-based reinforcement learning, opening up promising avenues for developing more efficient and capable reinforcement learning agents that can understand and interact with the world in ways that more closely resemble human cognition.

Contents

1. Introduction

Reinforcement learning (RL) has demonstrated remarkable success in solving complex control problems across diverse domains. From mastering strategic games like chess and Go (Silver, Hubert, et al. 2018) to navigating dynamic environments in video games (Hafner, T. Lillicrap, Norouzi, et al. 2020; Hafner, Pasukonis, et al. 2023; Mnih, Kavukcuoglu, Silver, Graves, et al. 2013; Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015) and executing intricate robotic manipulation tasks (Mosbach and Behnke 2024; Pavlichenko and Behnke 2023), RL has shown its potential to revolutionize how machines learn and adapt. However, despite these impressive achievements, current RL approaches often face significant limitations that hinder their widespread application in real-world scenarios.

One of the primary challenges in RL is the need for vast amounts of training data. Many successful RL applications have relied on millions of interactions with the environment to achieve high performance. This data-intensive nature poses a substantial barrier in domains where data collection is expensive, time-consuming, or potentially dangerous, such as in robotics or autonomous driving. Additionally, many current RL methods are often confined to narrow, specific domains, struggling to generalize their learned behaviors to new tasks or environments without extensive retraining.

These limitations raise a fundamental question in the field of artificial intelligence and robotics: how can we develop RL agents that learn efficiently and generalize effectively to new tasks without the need for extensive retraining? Addressing this question is crucial for advancing RL beyond constrained laboratory settings and into real-world applications where adaptability and sample efficiency are paramount.

Recent advancements have highlighted the potential of model-based approaches to address these challenges (Hafner, T. Lillicrap, J. Ba, et al. 2019; Hafner, T. Lillicrap, Fischer, et al. 2019b; Hafner, T. Lillicrap, Norouzi, et al. 2020; Hafner, Pasukonis, et al. 2023). By learning a world model that predicts the consequences of actions, model-based RL offers a structured approach to overcome limitations inherent in model-free methods. This allows for both reduction of costly interactions with the environment and explicit reasoning about action consequences.

However, while model-based RL offers significant benefits, it also introduces new

challenges, particularly when dealing with high-dimensional state spaces such as visual inputs. Traditional approaches to modeling environment dynamics often struggle with the complexity and variability inherent in visual scenes, especially those involving multiple objects and their relationships.

This is where object-centric representations (Ferraro et al. 2023; Haramati, Daniel, and Tamar 2024; Jiang et al. 2019; Kipf et al. 2021; Locatello et al. 2020; Yoon et al. 2023; Zadaianchuk, Seitzer, and Martius 2020) come into play, offering a promising solution to further enhance model-based RL. The key insight driving this research is that understanding a scene in terms of its constituent objects and their interactions can provide a powerful inductive bias for learning and reasoning. Instead of treating the environment as a monolithic entity, an object-centric approach decomposes the scene into distinct objects, each with its own properties and dynamics.

Despite these potential benefits, the combination of object-centric representations and model-based RL remains underexplored, likely due to the complexity of integrating two challenging learning problems: 1) Learning meaningful object representations from raw sensory input and using these to model complex action-conditioned environment dynamics, and 2) Using the object-centric dynamics model in a reinforcement learning process to derive optimal behavior.

To the best of our knowledge, no prior method has successfully learned an object-centric world model solely from pixels and utilized it effectively in model-based RL for continuous control tasks. This gap in the literature is testament to the intricate challenges inherent to building a tractable and efficient method for this problem and presents an exciting opportunity for innovation, which motivated this thesis.

To address this gap in the literature, this thesis introduces a novel method that extends the Object-Centric Video Prediction (OCVP) framework of (Villar-Corrales, Wahdan, and Behnke 2023) to operate in an action-conditioned manner. We then build a model-based reinforcement learning framework, akin to the Dreamer algorithm (Hafner, T. Lillicrap, J. Ba, et al. 2019; Hafner, T. Lillicrap, Norouzi, et al. 2020; Hafner, Pasukonis, et al. 2023), that employs this action-conditioned OCVP model as its dynamics model, resulting in the first object-centric model-based reinforcement learning algorithm capable of solving continuous control tasks solely from pixels.

Our approach leverages the insight that the structure derived from object-centric representations is mutually beneficial for both predicting future states and learning optimal action selection. This synergy enhances the agent's ability to reason about object properties and their interactions, leading to more efficient learning and better generalization.

To validate our method, we conducted extensive experiments on simulated

robotic manipulation tasks. These experiments were designed to showcase the algorithm's ability to reason about object properties and interactions, demonstrating its effectiveness in complex, visually-rich environments.

The contributions of this thesis are threefold:

1. We extend the OCVP framework developed by (Villar-Corrales, Wahdan, and Behnke 2023) to function in an action-conditioned manner, enabling its use in reinforcement learning contexts.

2. We adapt the Dreamer algorithm (Hafner, T. Lillicrap, J. Ba, et al. 2019; Hafner, T. Lillicrap, Norouzi, et al. 2020; Hafner, Pasukonis, et al. 2023) to utilize our action-conditioned OCVP model as its dynamics model, developing to the best of our knowledge the first object-centric model-based reinforcement learning algorithms for continuous control tasks from visual input alone.

3. We provide comprehensive testing and comparison of our new method on simulated robotic object manipulation tasks, demonstrating its improved interpretability in decision-making, superior ability to reason about object properties and interactions, and capability to adjust its object-centric model to a continuously evolving state-distribution.

The remainder of this thesis is organized as follows: Chapter 2 provides the fundamental concepts necessary for understanding our approach, including visual continuous control, reinforcement learning, and the transformer architecture. Chapter 3 reviews related work in model-based reinforcement learning and object-centric approaches, providing context for our contributions. Chapter 4 presents our methodology in detail, explaining the process of building an object-centric world model and its integration into a model-based RL framework. Chapter 5 describes our experimental setup, including the design of our robotic manipulation tasks and the baselines we compare against. We present and discuss our results, providing both quantitative comparisons and qualitative analyses. Chapter 6 concludes with a discussion of the implications of our work, its limitations, and potential directions for future research.

Through this work, we aim to contribute to the development of more efficient, generalizable, and interpretable reinforcement learning agents for complex visual control tasks, particularly in scenarios involving multiple objects and their interactions.

2. Fundamentals

This chapter provides an overview of the key concepts and techniques that form the foundation of our object-centric model-based reinforcement learning approach. We begin by discussing visual continuous control and the partially observable Markov decision process (POMDP) framework. We then review the fundamentals of reinforcement learning, including Q-learning and policy gradient methods. Finally, we introduce the transformer architecture, which plays a crucial role in our method for processing object-centric representations. Understanding these fundamental concepts is essential for grasping the motivation behind and implementation of our proposed approach.

2.1. Visual Continuous Control

Visual continuous control describes a group of problems where an agent chooses outputs in a continuous actions space based on visual inputs. This configuration of input and output spaces makes visual continuous control tasks highly relevant for robotics.

Visual data, such as images or videos, serve as a versatile representation of diverse environments. Limiting the input to vision is practical due to the richness of information cameras can provide and their widespread availability (Levine et al. 2016).

Continuous control refers to the process of making decisions in an environment where actions are not discrete but can take any value within a continuous range. Unlike discrete control, where actions are selected from a finite set, continuous control involves selecting actions from an infinite set of possible values. This type of control is particularly relevant in robotics, where precise and smooth movements are often required. For example, controlling the joint angles of a robotic arm or the speed and direction of a mobile robot involves continuous action spaces (T. P. Lillicrap et al. 2015).

In the context of robotics, visual continuous control is crucial because it allows robots to operate in dynamic and unstructured environments. For example, robotic manipulation tasks often require precise control based on visual feedback

to handle objects of varying shapes, sizes, and orientations. Similarly, autonomous navigation relies on visual inputs to perceive obstacles, recognize landmarks, and make path-planning decisions.

However, visual continuous control presents several challenges. One major challenge is processing high-dimensional visual data, which requires significant computational resources and sophisticated algorithms to extract relevant features. Additionally, these tasks often involve partial observability, where the agent cannot directly perceive the entire state of the environment and must infer it from limited visual inputs (Kaelbling, Littman, and Cassandra 1998).

2.1.1. Partially Observable Markov Decision Process (POMDP)

A Partially Observable Markov Decision Process (POMDP)(Kaelbling, Littman, and Cassandra 1998) is a mathematical framework used to model decision-making problems where the agent has incomplete information about the state of the environment, as is the case in most visual continuous control tasks. Unlike a Markov Decision Process (MDP), where the agent has full observability of the state, a POMDP accounts for scenarios where the agent only has access to partial observations, leading to uncertainty about the true state.

A POMDP is defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$, where: S represents the set of states in the environment; A denotes the set of actions available to the agent; $T : S \times A \times S \to [0, 1]$ is the state transition function, which defines the probability of transitioning from one state to another given a specific action; $R : S \times A \to \mathbb{R}$ is the reward function, which specifies the immediate reward received after taking an action in a given state; Ω is the set of possible observations the agent can receive; $O : S \times A \times \Omega \to [0, 1]$ is the observation function, which defines the probability of receiving a particular observation given a state and action; $\gamma \in [0, 1]$ is the discount factor, which represents the importance of future rewards compared to immediate rewards.

In a POMDP, the agent does not directly observe the state $s \in S$. Instead, it receives an observation $o \in \Omega$, which provides partial information about the state (see Figure 2.1).

Solving a POMDP involves finding a policy $\pi : \Omega \to A$ that maximizes the expected cumulative reward:

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{2.1}$$

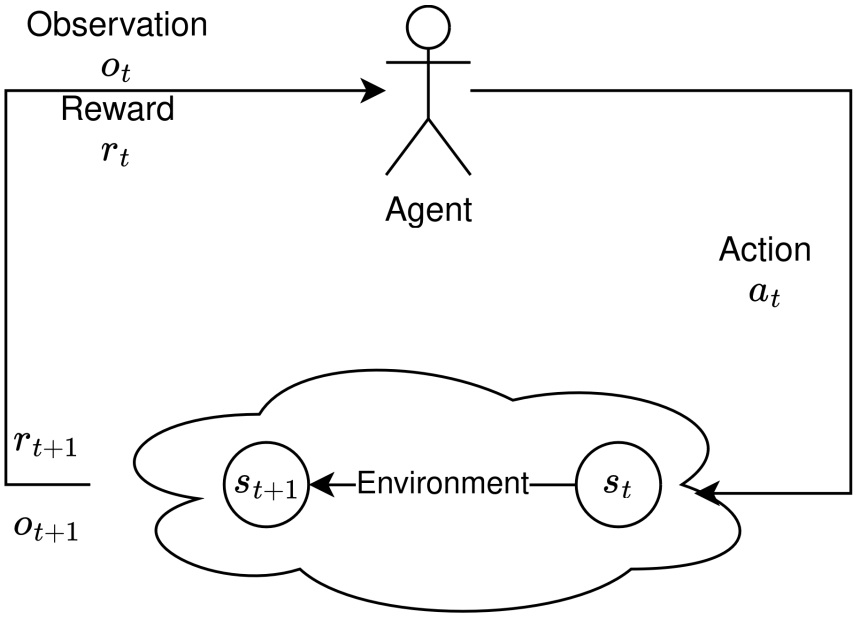


Figure 2.1: Components and information flow of a POMDP.

2.2. Reinforcement Learning (RL)

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions by interacting with its environment. Unlike supervised and unsupervised learning, RL involves learning from the consequences of actions, rather than from a static dataset.

In supervised learning, the algorithm is trained on a labeled dataset, where each input is paired with the correct output. The objective is to learn a mapping from inputs to outputs that generalizes well to unseen data. Common applications include classification and regression tasks. In unsupervised learning, the algorithm is provided with an unlabeled dataset and seeks to discover the underlying structure or distribution of the data. Techniques such as clustering and dimensionality reduction fall under this category (Mohri, Rostamizadeh, and Talwalkar 2018; Sutton and Barto 2018).

Reinforcement learning, in contrast, deals with sequential decision-making problems. The agent interacts with the environment in discrete time steps. At each time step, the agent observes the current state of the environment, selects an action based on its policy, and receives a reward as feedback. The environment then transitions to a new state, and the process repeats. The agent's goal is to learn a policy that maximizes the expected cumulative reward over time (Kaelbling, Littman, and Moore 1996).

A key distinction of RL is its emphasis on exploration and exploitation. Exploration involves trying new actions to discover their effects, while exploitation leverages known actions to maximize reward. Balancing these two aspects is critical for effective learning. Various strategies, such as ϵ-greedy and softmax action selection, are used to manage this trade-off (Sutton and Barto 2018).

In summary, the RL framework consists of two core components: the agent and the environment. The agent is the learner or decision maker, while the environment is the external system with which the agent interacts. The environment is characterized by its state space, which represents all possible configurations, and its action space, which defines the set of all possible moves the agent can make. At each time step, the agent observes the current state, selects an action, and receives a reward as feedback from the environment. The goal of RL is to learn a policy, which is a strategy for selecting actions, that maximizes the expected cumulative reward over time. To achieve this, RL algorithms often employ techniques such as estimating value functions, which predict the expected future rewards for states or state-action pairs.

2.2.1. Q-Learning

Q-learning is a fundamental algorithm in the field of model-free off-policy reinforcement learning. Developed by (Watkins 1989), it aims to find the optimal action-selection policy for any given finite Markov decision process (MDP) by learning the value of actions, represented as Q-values. These Q-values estimate the expected return (i.e., sum of discounted future rewards) of taking a given action in a particular state and following the optimal policy thereafter.

The core idea behind Q-learning is to iteratively update the Q-values based on the Bellman equation (Bellman 1957). The Bellman equation, which is used in Q-learning, provides a recursive decomposition that expresses the value of a state-action pair as the sum of the immediate reward and the discounted value of the following state. Mathematically, this is represented as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.2}$$

Here, s and a denote the current state and action, respectively, r is the immediate reward received after taking action a in state s, s' is the subsequent state, α is the learning rate, and γ is the discount factor which weights the importance of future rewards.

A key contribution of Q-learning is its convergence proof, published in (Watkins and Dayan 1992). Under the conditions that all state-action pairs continue to be

updated and the learning rate decays appropriately, Q-Learning is proven to converge to the optimal Q-values with probability 1. Despite this theoretical appeal, it also comes with a significant limitation: as a tabular method, it is only practical for problems with discrete action and state spaces. When applied to environments with large or continuous state spaces, the table of Q-values becomes excessively large, rendering the method computationally infeasible.

This limitation of Q-learning has led to various enhancements and innovations, one of the most significant being the development of Deep Q-Networks (DQNs) (Mnih, Kavukcuoglu, Silver, Graves, et al. 2013; Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015).

Deep Q-Network (DQN)

The Deep Q-Network (DQN) represents a significant advancement in the field of reinforcement learning, as it was the first deep learning model to successfully learn control policies directly from high-dimensional sensory input. Introduced by (Mnih, Kavukcuoglu, Silver, Graves, et al. 2013; Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015), DQN uses a convolutional neural network (CNN) to approximate the action-value function (Q-function) from raw pixel data, thereby addressing the scalability issues inherent in traditional Q-learning.

DQN was tested on a suite of 49 Atari 2600 games (M. G. Bellemare et al. 2013), demonstrating remarkable performance by outperforming the best existing reinforcement learning methods on 43 games and performing at a level comparable to that of a professional human games tester across the entire set. The core innovation of DQN is its ability to process high-dimensional visual inputs using CNNs, leveraging techniques previously successful in computer vision tasks.

Two key techniques also strongly contributed to the success of DQN: experience replay and the use of a target network. Experience replay involves storing the agent's experiences (state, action, reward, next state) in a replay buffer and sampling mini-batches of experiences to update the Q-network. This approach breaks the correlation between consecutive experiences and smooths out learning by reusing past experiences. By sampling from a diverse set of past experiences, the agent can learn more effectively, as it reduces the risk of the model becoming biased by recent transitions. This technique also increases data efficiency by allowing multiple updates from the same experience, which is particularly important when collecting data is expensive or time-consuming.

The target network, essentially a copy of the Q-network, is used to generate the target Q-values for the update step. This target network is updated less frequently than the Q-network, providing a stable reference point for learning. This stability

helps mitigate the risk of oscillations and divergence during training, which can occur when the Q-values are directly computed from the same network that is being updated. By using a fixed target for several updates, the learning process becomes more robust and less prone to the feedback loops that can destabilize learning.

DQN formulates its policy implicitly by choosing the action with the maximum Q-value in each state. While this maximization is trivial for discrete action spaces, searching for the optimal action in continuous spaces can quickly become infeasible. Methods that directly express a parametrized policy that maps from states to action can handle this domain more naturally.

2.2.2. Policy Gradient Methods

Policy gradient methods (Sutton and Barto 2018; Sutton, McAllester, et al. 1999) represent a class of reinforcement learning algorithms that directly optimize the policy, rather than estimating value functions to implicitly search for an optimal policy. Unlike value-based methods like Q-learning and its extensions, policy gradient methods maintain and improve a parameterized policy that maps states to actions.

The fundamental idea behind policy gradient methods is to adjust the parameters of the policy in a direction that maximizes expected cumulative rewards. The policy is typically represented by a neural network, parameterized by θ, that outputs the probability distribution over actions for a given state. The objective is to find the parameters θ that maximize the expected return $J(\theta)$, defined as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{2.3}$$

where τ denotes a trajectory (sequence of states, actions, and rewards), π_θ is the policy parameterized by θ, r_t is the reward at time step t, and γ is the discount factor.

The policy gradient theorem provides a way to compute the gradient of the expected return with respect to the policy parameters. This gradient can then be used to update the policy parameters via gradient ascent. The update rule for the policy parameters is given by:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2.4}$$

where α is the learning rate.

A key advantage of policy gradient methods is their applicability to continuous action spaces. They do not require discretizing the action space, which can lead to the curse of dimensionality. Instead, they can handle high-dimensional and continuous action spaces more efficiently. Additionally, policy gradient methods can naturally incorporate stochastic policies, which can be beneficial for exploration in complex environments.

However, a significant challenge with policy gradient methods is the high variance in gradient estimates, which can make learning unstable and slow. Actor-Critic methods (Konda and Tsitsiklis 1999) address this issue by combining policy-based and value-based approaches. In these methods, the "actor" learns and improves the policy for selecting actions, while the "critic" estimates the value function to evaluate the actor's decisions. This hybrid approach helps stabilize learning and improves efficiency by reducing the variance of the gradient estimates.

Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an actor-critic algorithm designed to operate over continuous action spaces. Developed by (T. P. Lillicrap et al. 2015), DDPG extends the Deterministic Policy Gradient (DPG) algorithm (Silver, Lever, et al. 2014) by leveraging deep learning techniques to enable more efficient learning in high-dimensional environments.

DDPG combines elements from both value-based and policy-based reinforcement learning methods. The core idea is to use a deterministic policy, represented by an actor network, which maps states directly to specific actions. This actor network is trained to maximize the expected return by following the gradient of the Q-function, which is estimated by a critic network. The critic network, parameterized by θ^Q, learns to approximate the action-value function $Q(s, a)$ by minimizing the loss function:

$$L(\theta^Q) = \mathbb{E}_{(s,a,r,s')}\Big[\big(r + \gamma Q(s', \mu(s'|\theta^\mu)|\theta^Q) - Q(s, a|\theta^Q)\big)^2\Big] \tag{2.5}$$

Here, $\mu(s|\theta^\mu)$ represents the actor network, and the term $r + \gamma Q(s', \mu(s'|\theta^\mu)|\theta^Q)$ serves as the target for the Q-value.

To stabilize the learning process, DDPG employs two key techniques, which were already discussed in the section on DQN: experience replay and target networks. Like introduced in DQN, target networks are copies of the actor and critic networks that are updated slowly to provide consistent targets for the Q-value updates. But unlike DQN, the weights of the target networks $\theta^{Q'}$ and $\theta^{\mu'}$ are updated using a

soft update mechanism:

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{2.6}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{2.7}$$

where $\tau \ll 1$ is a small constant. This method ensures that the target networks evolve smoothly, reducing the risk of divergence and instability during training.

DDPG has demonstrated strong performance on a variety of simulated physics tasks, including cartpole swing-up, robotic manipulation, and legged locomotion.

Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) is an off-policy actor-critic algorithm designed to address the challenges of sample efficiency and stability in deep reinforcement learning, particularly for continuous action spaces. Developed by (Haarnoja et al. 2018), SAC combines the benefits of entropy maximization (Ziebart 2010) with off-policy learning.

The core innovation of SAC lies in its use of the maximum entropy framework, which augments the standard reinforcement learning objective with an entropy term. This approach encourages the agent to explore more widely by favoring policies that maximize both expected return and entropy. The resulting objective can be expressed as:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{2.8}$$

where $\mathcal{H}(\pi(\cdot|s_t))$ represents the entropy of the policy π at state s_t, and α is a temperature parameter that controls the trade-off between reward maximization and entropy maximization. This entropy term incentivizes the agent to act as randomly as possible while still achieving high rewards, leading to improved exploration and robustness against model errors.

SAC has demonstrated state-of-the-art performance on various continuous control benchmark tasks, such as those in the OpenAI gym benchmark suite (Brockman et al. 2016) and the rllab implementation of the complex Humanoid task (Duan et al. 2016), which involves a high-dimensional action space. Compared to other off-policy algorithms like DDPG, SAC is more stable and less sensitive to hyperparameter settings, making it easier to apply to a wide range of tasks.

2.3. Transformer

The Transformer architecture, introduced by (Vaswani et al. 2017), represents a major advancement in the field of deep learning, particularly for natural language processing (NLP). Unlike recurrent neural networks (RNNs) (Rumelhart, Hinton, and Williams 1986) and long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber 1997), which process data sequentially, Transformers are designed to handle input data in parallel, significantly improving computational efficiency and enabling the processing of much larger datasets.

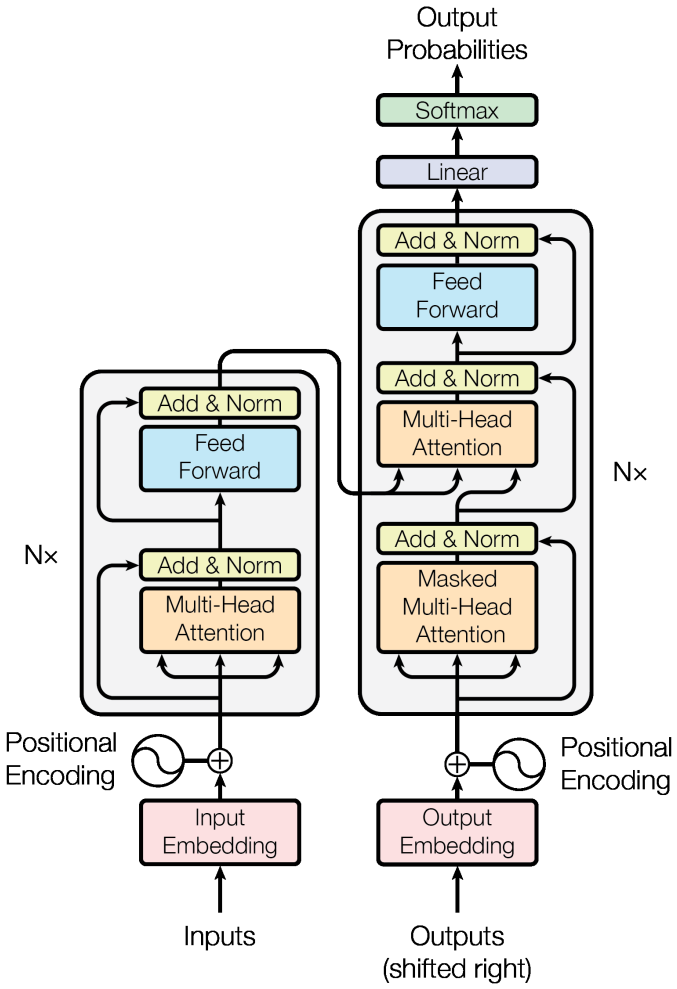


Figure 2.2: Transformer architecture (Vaswani et al. 2017).

The classic Transformer architecture (see Figure 2.2) is composed of an encoder-decoder structure. Both the encoder and decoder are built using stacks of identical layers. The encoder's primary function is to process the input sequence, i.e., generate a continuous representation for each symbol in the input sequence, while

the decoder uses these representations to produce the output sequence of symbols.

Each encoder layer consists of two main components: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The decoder layers are similar to the encoder layers but include an additional multi-head attention mechanism that performs attention over the encoder's output. This allows the decoder to access the encoded input sequence while generating the output sequence.

2.3.1. Key Components of Transformer Architecture

The Transformer architecture is built upon several key components. These components include multi-head attention, positional encoding, feed-forward networks, and the use of layer normalization and residual connections.

Attention

The attention mechanism (see Figure 2.3) is at the heart of the Transformer architecture. It allows the model to weigh the importance of different tokens in a sequence relative to each other, regardless of their positions. This is achieved by computing attention scores for each token with respect to all other tokens in the sequence. The attention mechanism is defined by the equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.9}$$

where Q, K, and V represent the query, key, and value matrices, respectively, and d_k is the dimension of the key vectors. This mechanism enables the model to capture long-range dependencies and contextual relationships more effectively than traditional RNNs or LSTMs within the input sequence.

Multi-Head Attention

Multi-head attention (see Figure 2.4) is an extension of the attention mechanism that enhances the model's ability to focus on different aspects of the relationships in the input sequence simultaneously. Instead of computing a single set of attention scores, the multi-head attention mechanism divides the input into multiple heads, each of which computes its own set of attention scores. These scores are then concatenated and linearly transformed to produce the final output. The formula for multi-head attention is given by:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{2.10}$$

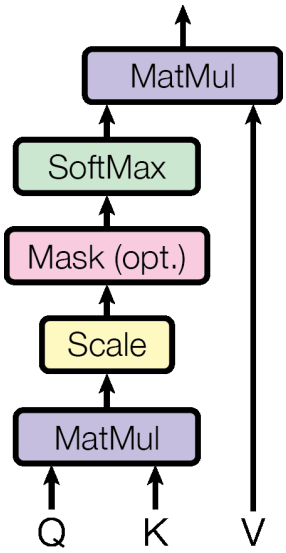


Figure 2.3: Scaled Dot-Product Attention (Vaswani et al. 2017).

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, and W_i^Q, W_i^K, W_i^V, and W_i^O are learned projection matrices.

Positional Encoding

Since Transformers do not inherently consider the order of tokens, positional encoding is used to inject sequence information into the model. Positional encodings are added to the input embeddings to provide information about the positions of tokens in the sequence. These encodings are derived from sine and cosine functions of different frequencies, allowing the model to distinguish between different positions. The formula for positional encoding is:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.11}$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.12}$$

where pos is the position and i is the dimension. This encoding ensures that each position has a unique representation that can be learned by the model.

Feed-Forward Networks

Each layer of the Transformer contains a position-wise fully connected feed-forward network. These networks consist of two linear transformations with a ReLU activation in between. The feed-forward network operates independently on each

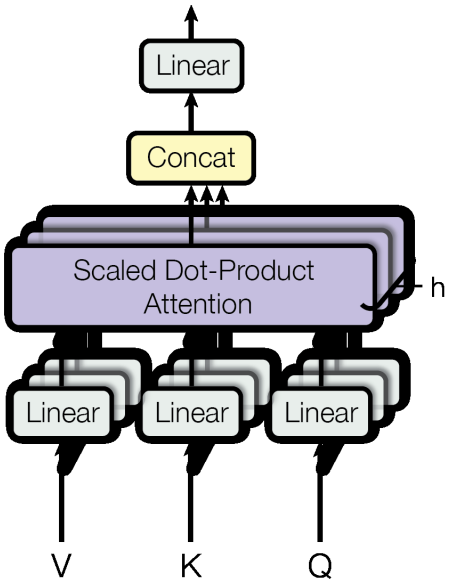


Figure 2.4: Multi-Head Attention (Vaswani et al. 2017).

position and provides non-linearity. The formula for the feed-forward network is:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.13}$$

where W_1, W_2, b_1, and b_2 are learned parameters. This component contributes to the model's ability to process and transform the input representations effectively.

Layer Normalization and Residual Connections

Layer normalization (J. L. Ba, Kiros, and Hinton 2016) and residual connections (He et al. 2016) are employed within each encoder and decoder layer to stabilize and accelerate training. Layer normalization normalizes the input across the features, helping to maintain a stable distribution of activations. Residual connections, on the other hand, add the input of each sub-layer to its output, enabling the gradient to flow directly through the network. This helps to mitigate the vanishing gradient problem and allows for deeper architectures. The formula for a residual connection is:

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x)) \tag{2.14}$$

where Sublayer(x) represents the output of the multi-head attention or feed-forward network.

2.3.2. Vision Transformer (ViT)

The Vision Transformer (ViT) represents a significant advancement in applying Transformer models to computer vision tasks. Introduced by (Dosovitskiy et al. 2020), ViT leverages the self-attention mechanisms of Transformers, originally developed for natural language processing, to process images. This approach marks a departure from the traditional convolutional neural networks (CNNs) that have dominated the field of computer vision.

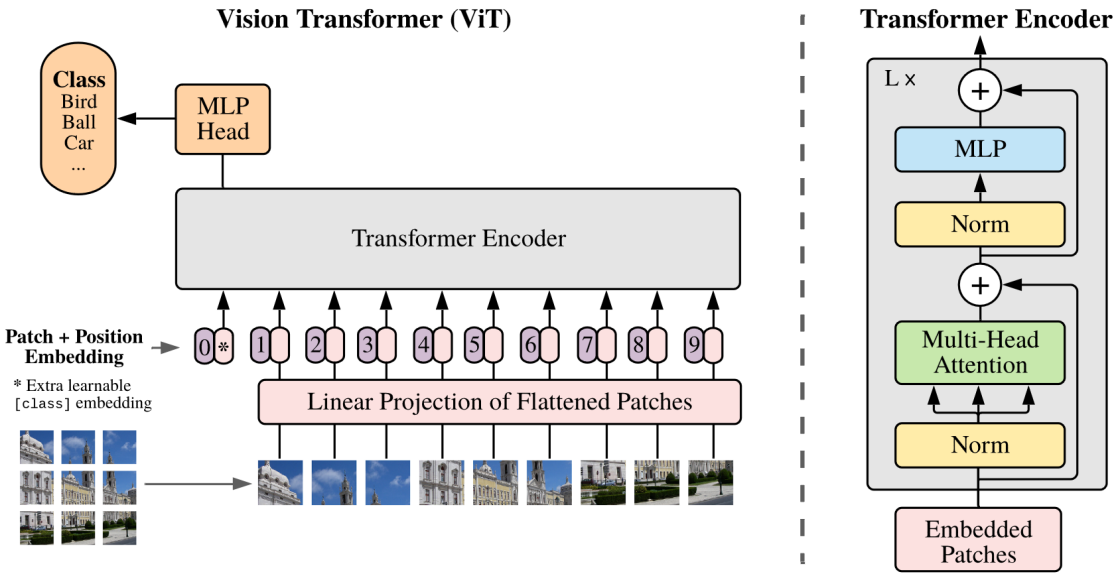


Figure 2.5: Vision Transformer (ViT) architecture (Dosovitskiy et al. 2020).

The Vision Transformer (see Figure 2.5) adapts the standard Transformer architecture to handle image data. Instead of processing a sequence of words, ViT divides an image into a sequence of fixed-size, non-overlapping patches. Each patch is treated as a token, similar to how words are treated in NLP applications. Typically, an image is divided into N patches, each of size 16×16 pixels, resulting in a sequence of N patches, where N is determined by the image size and patch size.

Each patch is then linearly embedded into a fixed-dimension vector, forming the input to the Transformer. To retain positional information, since the order of patches is crucial for understanding the spatial structure of images, positional embeddings are added to the patch embeddings.

The embedded patches, along with their positional encodings, are fed into a standard Transformer encoder. The encoder consists of multiple layers of multi-head self-attention and feed-forward neural networks, identical to those used in NLP Transformers.

The output of the Transformer encoder is a sequence of vectors, one for each patch, that collectively represent the image. A special classification token ([class]) is typically appended to the sequence of patch embeddings before being fed into the Transformer. The final representation corresponding to this [class] token is used for image classification tasks, as it aggregates information from all patches.

Vision Transformers have demonstrated competitive performance on several benchmark datasets, often surpassing traditional CNNs, especially when trained on large-scale datasets. One of the key advantages of ViTs is their ability to model long-range dependencies and global context more effectively than CNNs, which rely on local receptive fields and hierarchical feature extraction.

In addition to classification tasks, Vision Transformers have been adapted for various other computer vision applications, such as object detection, segmentation, and image generation.

3. Related Work

This chapter provides an overview of key research areas related to our work on object-centric model-based reinforcement learning. We begin by discussing model-based reinforcement learning approaches, which form the foundation of our method. We then explore object-centric learning techniques and their applications in reinforcement learning, highlighting both model-free and model-based approaches in this emerging field.

3.1. Model-based Reinforcement Learning

Model-based reinforcement learning is an approach in the realm of reinforcement learning, where the agent explicitly learns a model of the environment. The fundamental idea is to use this learned model to simulate future states and rewards, enabling the agent to plan and make decisions without directly interacting with the real environment at every step.

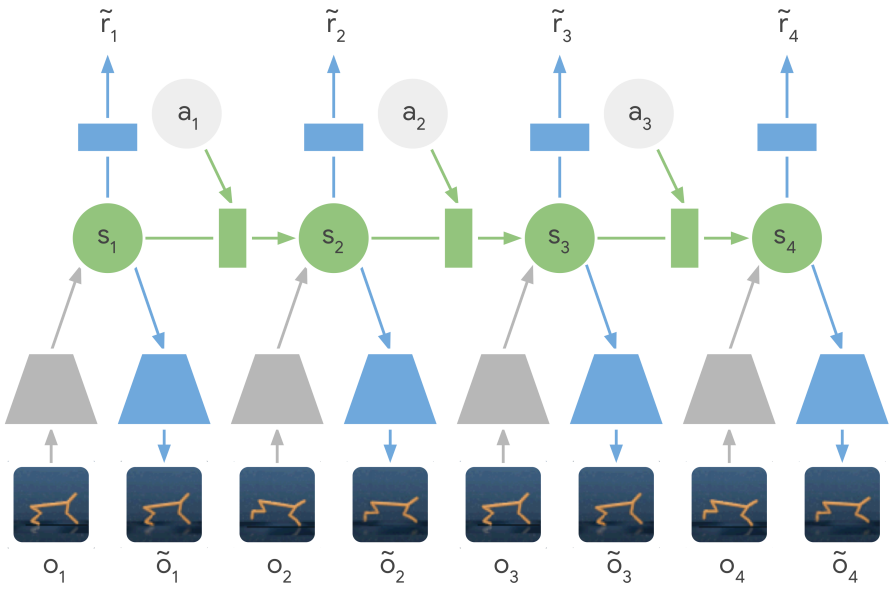


Figure 3.1: Structure of a typical dynamics model used in model-based RL (Hafner, T. Lillicrap, Fischer, et al. 2019a).

At the core of model-based RL is the learning of a dynamics model (see Figure 3.1), which should capture different aspects of the environment depending on the task and method. This model typically includes a transition model, which models the transition function $p(s_{t+1}|s_t, a_t)$. By doing this, it encapsulates the probability distribution over the next state s_{t+1} given the current state s_t and action a_t. Alongside, a reward model, which models the reward function $p(r_t|s_t)$, predicts the reward r_t of getting to state s_t. In addition, it may be necessary to include an observation model that models the observation function $p(o_t|s_t)$ if the task to be solved is to be modeled as a POMDP, i.e., we only perceive the environment through observations o_t. This observation model often has an encoder-decoder structure, where the encoder translates observations o_t into latent states s_t and the decoder implements the opposite direction. The latter is necessary, for example, to be able to formulate a reconstruction loss for the transition model.

One of the primary advantages of model-based RL is improved sample efficiency. By using the learned model to simulate interactions, the agent can gather a vast amount of experience without needing to interact with the real environment. This is particularly beneficial in scenarios where real-world interactions are expensive, time-consuming, or risky. As a result, model-based RL can achieve better performance with fewer real-world samples compared to model-free approaches. This efficiency is crucial in domains such as robotics and autonomous driving, where physical trials are costly and potentially hazardous.

Transferability is another significant benefit of model-based RL. One major dimension of this transferability is between different tasks or goals within the same environment. Once an agent has learned the underlying dynamics of an environment, it can often adapt to optimize for different reward functions with minimal retraining. This ability to reuse learned dynamics models across tasks reduces the need for extensive data collection when pursuing new objectives. For instance, a robot that has learned the physical dynamics of manipulating objects could quickly adapt to new tasks like stacking, sorting, or assembling without having to relearn the fundamental physics. Beyond task transfer, dynamics models can sometimes be adapted to similar but distinct environments. For example, a model trained in simulation might be fine-tuned for use in a real-world setting, enhancing the practical utility of the learned policies.

Explainability is also enhanced in model-based RL compared to model-free approaches. Since model-based RL explicitly models the environment's dynamics, it provides a clearer understanding of how different actions influence future states and rewards. This transparency aids in diagnosing and troubleshooting the agent's behavior, making it easier to identify the causes of failures or suboptimal performance. Additionally, the interpretability of the learned model can be valuable in

applications where understanding the decision-making process is crucial for safety and compliance.

3.1.1. Deep Planning Network (PlaNet)

Deep Planning Network (PlaNet) is a model-based reinforcement learning algorithm introduced by (Hafner, T. Lillicrap, Fischer, et al. 2019b). PlaNet is designed to learn environment dynamics from images and to perform planning in a compact latent space, significantly reducing the computational cost compared to planning directly in the high-dimensional observation space.

At the core of PlaNet is the Recurrent State Space Model (RSSM), which combines both deterministic and stochastic components to model the environment's dynamics. By modeling both deterministic and stochastic transitions, RSSM can handle the uncertainty and partial observability inherent in many tasks. PlaNet uses the learned RSSM for planning to select optimal actions through the Cross Entropy Method (CEM) (Rubinstein 1997). CEM is a population-based optimization algorithm that iteratively improves a distribution over action sequences to maximize expected rewards. The process involves sampling a set of candidate action sequences from a Gaussian distribution, simulating trajectories using the RSSM to evaluate the expected rewards of each action sequence, and updating the distribution based on the top-performing action sequences. This iterative process allows PlaNet to efficiently search for optimal actions in the latent space

PlaNet was tested on a variety of continuous control tasks from the DeepMind Control Suite (Tassa et al. 2018). Compared to model-free methods like A3C (Mnih, Badia, et al. 2016) and D4PG (Barth-Maron et al. 2018), PlaNet achieves comparable performance with substantially fewer interactions with the environment (e.g., 200 times less). However, PlaNet also faces several challenges. Despite planning in the latent space, the computational cost of evaluating numerous action sequences online can be high. Also, planning is done over a limited horizon, which can sometimes lead to suboptimal action selection.

3.1.2. Dreamer

Dreamer (Hafner, T. Lillicrap, J. Ba, et al. 2019) represents a significant advancement in model-based reinforcement learning, distinguishing itself from previous approaches like PlaNet by incorporating the learned dynamics model within an actor-critic framework. This integration enables Dreamer to use an additional value network to account for rewards beyond the immediate planning horizon and an actor network to compute actions more efficiently. These components allow

Dreamer to learn long-sighted behaviors through backpropagation of gradients through the dynamics model predictions.

The actor network in Dreamer is trained to predict good actions by backpropagating the gradients of rewards through sequences of predicted states, a process that is not feasible for model-free methods. This method enables Dreamer to reflect how minor adjustments to its action selection influence future rewards, thus refining the actor network to maximize rewards. To account for rewards beyond the prediction horizon, the value network estimates the discounted future rewards for each latent state. These rewards and values are then backpropagated to the actor network, optimizing it to select better actions.

Dreamer differs from PlaNet in several key aspects. Whereas PlaNet searches for the optimal action by predicting the potential reward of various action sequences, Dreamer avoids this computationally expensive process by separating planning and acting. Once the actor network is trained on the predicted sequences, it can directly compute actions for interaction with the environment without additional searching. Furthermore, Dreamer considers rewards beyond the immediate planning horizon by using the value function

In evaluations on 20 visual control tasks of the DeepMind Control Suite (Tassa et al. 2018), Dreamer outperforms the model-free baseline D4PG (Barth-Maron et al. 2018), achieving an average score of 823 compared to 786, while requiring 20 times fewer interactions with the environment. Additionally, Dreamer surpasses the final performance of PlaNet across nearly all tasks.

DreamerV2 (Hafner, T. Lillicrap, Norouzi, et al. 2020) introduced several significant improvements over its predecessor. It replaced Gaussian latents with categorical latent states using straight-through gradients, enabling better capture of multimodal dynamics. The algorithm employed KL balancing, scaling prior cross-entropy and posterior entropy separately in the KL loss to encourage learning an accurate temporal prior. For discrete action spaces like Atari, DreamerV2 switched to REINFORCE gradients for policy optimization. The model size was also increased from 13 million to 22 million parameters.

Building on these advancements, DreamerV3 (Hafner, Pasukonis, et al. 2023) incorporated additional robustness techniques. These included observation symlog transformation, free bits for KL loss, and a 1% uniform mixture for all categorical distributions. To handle varying reward scales across different tasks, DreamerV3 implemented percentile return normalization. The algorithm also introduced the symexp twohot loss for reward and critic models, stabilizing the learning process. Network architecture improvements included the use of Block GRU, RMSNorm normalization, and SiLU activation functions. The optimizer was enhanced with adaptive gradient clipping and LaProp. Additionally, DreamerV3 expanded the

replay buffer capacity and added the ability to store and update latent states.

These cumulative improvements enabled DreamerV3 to achieve state-of-the-art performance across a wide range of reinforcement learning benchmarks, demonstrating robust learning and generalization capabilities across diverse tasks and environments.

3.2. Object-centric Reinforcement Learning

Object-centric reinforcement learning aims to leverage structured representations of environments in terms of distinct objects and their interactions. This approach offers several potential benefits, including improved sample efficiency, better generalization, and more interpretable learned behaviors. In this section, we review key developments in object-centric representation learning and its application to reinforcement learning tasks.

3.2.1. Object-centric Representation Learning

Object-centric representation learning is a paradigm in machine learning that focuses on developing models capable of understanding and manipulating scenes based on the discrete objects they contain (see Figure 3.2). This approach contrasts with traditional methods that process scenes as undifferentiated wholes. The key benefit of object-centric representation learning is its ability to provide more interpretable and generalizable representations by explicitly modeling the objects within a scene and their interactions. This capability allows models to better handle complex tasks such as scene understanding, object manipulation, and interaction prediction, making them particularly useful for applications in robotics and computer vision.

Object-centric methods are advantageous because they can systematically generalize to new compositions and variable numbers of objects, which is challenging for holistic scene representation approaches. By disentangling the scene into objects, these methods enable the learning of more robust and modular representations, facilitating transfer learning and improving performance on tasks requiring high-level reasoning and manipulation of individual entities.

Slot Attention

Slot Attention (Locatello et al. 2020) is one of the first methods with which the learning of object centric representations could be practically implemented without the need for specifically annotated data. It is designed to interface with perceptual

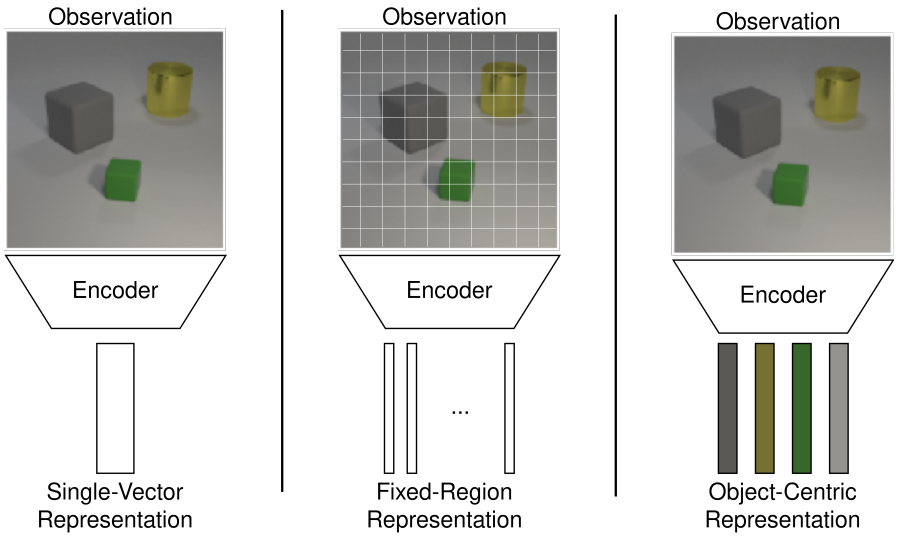


Figure 3.2: Different representation learning paradigms.

representations, such as those generated by convolutional neural networks (CNNs), and produces a set of task-dependent abstract representations called "slots". Each slot is intended to bind to a particular object or part of the input scene through an iterative and competitive attention process.

The Slot Attention mechanism (see Algorithm 1) begins with the initialization of slots as independent samples from a Gaussian distribution with shared, learnable parameters. These slots serve as abstract representations that will be refined to capture different objects or components of the input scene. Both the input feature vectors and the slots are normalized using LayerNorm to ensure stability and improve convergence during training.

A softmax function is applied to the dot product of transformed input features and slots, creating a competitive process where slots compete to explain parts of the input. This competition is crucial for the slots to specialize and bind to specific parts of the scene. The input features are then assigned to slots based on the attention weights, and the values are aggregated using a weighted mean. This aggregation step ensures that each slot receives a summary of the features it is responsible for.

The slots are updated through a recurrent update function, typically a Gated Recurrent Unit (GRU), which integrates the aggregated features and refines the slot representations. An optional residual MLP with ReLU activation can be used to further enhance the slot updates. This iterative process of attention calculation, feature aggregation, and slot updating continues until the slots converge to stable representations of the objects or components within the scene.

Algorithm 1 Slot Attention module (Locatello et al. 2020). The input is a set of N vectors of dimension D_{inputs} which is mapped to a set of K slots of dimension D_{slots}. The slots are initialized by sampling their initial values as independent samples from a Gaussian distribution with shared, learnable parameters $\mu \in \mathbb{R}^{D_{\text{slots}}}$ and $\sigma \in \mathbb{R}^{D_{\text{slots}}}$. In experiments, the number of iterations is for example set to $T = 3$.

Input: inputs $\in \mathbb{R}^{N \times D_{\text{inputs}}}$, slots $\sim \mathcal{N}(\mu, \text{diag}(\sigma)) \in \mathbb{R}^{K \times D_{\text{slots}}}$
Layer params: k, q, v: linear projections for attention; GRU; MLP; LayerNorm (x3)
inputs = LayerNorm(inputs)
for $t = 1 \rightarrow T$ **do**
    $\text{slots}_{\text{prev}} = \text{slots}$
    slots = LayerNorm(slots)
    $\text{attn} = \text{Softmax}\left(\frac{1}{\sqrt{D}} k(\text{inputs}) \cdot q(\text{slots})^T, \text{axis} = \text{'slots'}\right)$
    $\text{updates} = \text{WeightedMean}(\text{weights} = \text{attn} + \epsilon, \text{values} = v(\text{inputs}))$
    $\text{slots} = \text{GRU}(\text{state} = \text{slots}_{\text{prev}}, \text{inputs} = \text{updates})$
    $\text{slots} = \text{slots} + \text{MLP}(\text{LayerNorm}(\text{slots}))$
end for
return slots

Slot Attention excels in scenarios where the goal is to discover and represent objects without explicit supervision. For instance, it can be used in an autoencoder framework to encode images into a set of slot representations and then decode them back to reconstruct the original image. This process enables the model to learn object-centric representations that capture the underlying structure of the scene.

3.2.2. Model-free Object-centric Reinforcement Learning

Model-free object-centric reinforcement learning methods integrate object-centric representations into traditional model-free RL algorithms. These approaches aim to exploit the structure provided by object-centric representations without explicitly modeling environment dynamics. We discuss several notable works in this area, examining their strengths and limitations in handling complex visual environments and object interactions.

SMORL

SMORL (Self-supervised Multi-object Reinforcement Learning) as introduced by (Zadaianchuk, Seitzer, and Martius 2020) combines SCALOR (Jiang et al. 2019), an object-centric encoder, with a goal-conditioned reinforcement learning approach. To effectively utilize the set of representations generated by SCALOR, SMORL

introduces a specialized goal-conditioned attention mechanism.

The SMORL pipeline begins with pre-training SCALOR on data collected from a random policy. Subsequently, a goal-conditioned policy is trained to achieve randomly sampled goal configurations. During evaluation, the system receives a goal observation, which is encoded by SCALOR to extract the desired goal configuration of objects. The policy then attempts to sequentially position objects into their goal configurations by cycling through the encoded goals.

SMORL was evaluated on two robotic manipulation tasks: Visual Push and Visual Rearrange. In Visual Push, the objective is to move objects from fixed starting positions to random target positions. Visual Rearrange presents a more challenging scenario, requiring the rearrangement of multiple objects from random starting positions to random target configurations. Visual reinforcement learning experiments were conducted with both single and dual object scenarios.

Performance-wise, SMORL achieved comparable results to the best baselines in the Visual Push task and outperformed them in the more complex Visual Rearrange task. However, it's worth noting that SMORL's performance fell short of that achieved by SAC (Soft Actor-Critic) (Haarnoja et al. 2018) which was provided with ground truth object configurations. The authors attribute this performance gap to SMORL's limitations in handling occlusions and imperfections in SCALOR-generated representations.

Interestingly, SMORL demonstrated some generalization capabilities across different numbers of objects. For instance, a model trained on the two-object Visual Rearrange task performed comparably on the single-object variant to a model specifically trained for that scenario.

While SMORL offers an innovative approach to multi-object reinforcement learning, it does have certain limitations. The method assumes that objects can be sequentially arranged into their goal configurations and requires observations of the goal configuration. Additionally, it struggles with occlusion handling. Furthermore, the evaluation was limited to relatively simple robotic manipulation tasks, leaving room for investigation of more complex scenarios in future work.

Pre-training Object-centric Representations for Reinforcement Learning

(Yoon et al. 2023) conducted a systematic investigation into the effectiveness of pre-training object-centric representations (OCRs) for reinforcement learning tasks. The study introduced a benchmark comprising various object-centric tasks, including object interaction and relational reasoning, designed with visually simple 2D scenes and a more complex 3D robotic reaching task.

The authors found that OCR pre-training showed performance improvements

in relational reasoning tasks compared to other representation types, but did not consistently outperform other methods in tasks that don't require reasoning about object relationships. OCR pre-training demonstrated improved sample efficiency in tasks where understanding object relationships is crucial. In out-of-distribution generalization tests, OCR pre-training showed more robust performance than ground truth state representations for unseen object colors.

The study emphasized the significance of model architecture, revealing that transformer-based pooling layers were crucial in effectively utilizing object-centric representations, particularly in tasks requiring relational reasoning. The effectiveness of OCR pre-training was maintained even in the visually more complex 3D robotic reaching task, where perfect object segmentation was not achievable.

However, the work has several limitations. The simplicity of the environments used, particularly the 2D tasks with unicolored objects on black backgrounds, raises questions about the generalizability of the findings to more complex scenarios. As the benefits of OCR pre-training appear to be task-specific and not universally applicable to all object-centric scenarios, the limited complexity of the tasks makes it difficult to assess how the observed sample efficiency would translate to more challenging environments.

Furthermore, the authors note in the conclusion, the study is lacking environments with partial observability, where objects can be occluded. This is a common challenge in more complex, realistic tasks, and its absence limits the applicability of the findings to many more realistic scenarios. While the 3D robotic reaching task provides some evidence of applicability to more realistic scenarios, this was only demonstrated on a single task and still did not really address the issue of occlusion. More diverse and complex environments, including those with partial observability, would be needed to fully validate the claims and understand the potential of OCR pre-training in practical reinforcement learning applications.

Entity Interaction Transformer (EIT)

The Entity Interaction Transformer (EIT), introduced by (Haramati, Daniel, and Tamar 2024), represents a novel approach to goal-conditioned reinforcement learning from pixel inputs. At its core, EIT utilizes Deep Latent Particles (DLP) (Daniel and Tamar 2022) to generate object-centric representations from visual inputs. These representations are then processed by a transformer architecture, which enables the model to reason about relationships between entities in the scene.

EIT was tested on a suite of simulated tabletop robotic object manipulation environments where the goal was to push cubes into target configurations. The

EIT method was compared with a baseline method that uses unstructured representations and with SMORL. EIT performs better than or equivalently to the best performing baseline in each task. Additionally, the performance of EIT on tasks requiring complex object interactions is significantly better than the performance of SMORL, demonstrating its ability to reason about relationships between multiple entities. The authors also report strong generalization capabilities, with EIT showing the ability to generalize zero-shot to tasks involving up to 6 objects when trained on only 3 objects.

However, the method is not without limitations. EIT assumes full observability of the environment, which may limit its applicability in more realistic scenarios with partial observability. The evaluation of EIT has been primarily limited to a single type of environment, raising questions about its performance in more diverse settings. Furthermore, as noted by the authors, EIT requires a pretrained DLP model. While pretraining on data collected using a random policy worked well for the domains tested, more complex tasks may necessitate more sophisticated pretraining approaches or an online method that integrates DLP training with policy learning.

3.2.3. Model-based Object-centric Reinforcement Learning

Model-based object-centric reinforcement learning is a very new and largely unexplored field that combines the benefits of model-based RL with structured object representations. This emerging area aims to learn actionable world models that capture object dynamics and interactions, potentially enabling more efficient planning and decision-making. Despite the limited research in this domain, we have identified an interesting and quite recent approach that merits discussion. In the following, we present this method, examining its novel contributions to this nascent field and considering its implications for future research in model-based object-centric RL.

FOCUS

FOCUS, introduced by (Ferraro et al. 2023), presents an approach to model-based reinforcement learning that incorporates an object-centric world model. The method builds upon the DreamerV2 algorithm (Hafner, T. Lillicrap, Norouzi, et al. 2020), replacing its standard world model with an object-centric variant.

The world model in FOCUS shares structural similarities with the Recurrent State Space Model (RSSM) used in PlaNet (Hafner, T. Lillicrap, Fischer, et al. 2019b) and Dreamer (Hafner, T. Lillicrap, J. Ba, et al. 2019). A key modification

is in the decoder component, which extracts a latent representation for each object from the latent RSSM state, then decodes each object into a mask and an image reconstruction, including RGB and depth information where applicable.

Training of this object-centric world model involves an additional reconstruction loss, computed on the masked reconstructions and the masks themselves. This approach requires object-specific masks for supervision. In real-world experiments, where such masks are not inherently available, the authors utilized SAM-masks (Segment Anything Model) (Kirillov et al. 2023) to provide the necessary object-specific supervision.

FOCUS introduces an exploration strategy based on a maximum entropy formulation of the latent object representations. This approach aims to encourage exploration of the environment in an object-centric manner. The method was evaluated on both simulated and real-world robotic manipulation tasks. The authors report that FOCUS performed comparably to or better than the baselines tested, including DreamerV2, across the experimental tasks. The results also indicated strong exploration performance.

However, FOCUS has certain limitations and assumptions that are important to consider. It requires object-specific masks for training the world model, which may not always be readily available or easy to generate in all environments. The method also relies on proprioceptive and depth information in addition to visual data, which may limit its applicability in scenarios where such information is not accessible. While FOCUS uses object-centric representations in its world model and exploration strategy, it still relies on the singular latent state of the RSSM for reward prediction, value estimation, and action selection.

In comparison, our proposed method aims to address some of these limitations. Our approach learns object-centric representations in a fully unsupervised manner from visual input alone, without requiring segmentation masks for supervision. We do not assume access to proprioceptive or depth information, instead learning all necessary representations solely from visual input. Our method utilizes object-centric representations more extensively, incorporating them into reward and value prediction, as well as the action selection process. We model object interactions and their state transitions explicitly in the transition model, with each object represented by its own latent representation.

4. Method

In this chapter, we introduce our novel method that performs model-based reinforcement learning using an object-centric world model. Our approach centers on two key innovations: First, we develop a structured action-conditioned world model that predicts environmental dynamics in terms of individual entities or objects. This allows for a more granular and interpretable representation of the scene. Second, we design reinforcement learning models capable of effectively reasoning over these object-centric representations to solve control problems. To provide a clear overview of our method, Algorithm 2 presents the core steps of our algorithm.

4.1. Dynamics Model Learning

A significant aspect of our work is the introduction of an object-centric dynamics model to model-based RL. This approach offers distinct advantages: it enhances the prediction of environmental dynamics by enabling the model to more explicitly learn interactions between objects, and it facilitates the learning of behaviors that rely on understanding the relationships between objects.

4.1.1. Slot Attention for Video (SAVi)

To achieve an object-centric representation of the environment, we adopt a preliminary step of pre-training, following an approach similar to those used by (Haramati, Daniel, and Tamar 2024; Villar-Corrales, Wahdan, and Behnke 2023; Yoon et al. 2023; Zadaianchuk, Seitzer, and Martius 2020). This pre-training step is designed to establish a robust object-centric representation before proceeding to the main training phase. We utilize the Slot Attention for Video (SAVi) method, as introduced by (Kipf et al. 2021), which builds upon the Slot Attention mechanism proposed by (Locatello et al. 2020). SAVi effectively extends the Slot Attention framework to handle video inputs by employing a predictor-corrector framework that iteratively refines object representations across frames.

In the predictor step, the model generates an initial estimate $\hat{\mathcal{Z}}_{t+1}$ for the slots at the next time step $t+1$, based on the slots outputted by the corrector $\mathcal{Z}_t$ from the current time step t. This prediction is obtained by applying a transformer to the

Algorithm 2 Overview of Training Procedure

Require: Number of object-centric autoencoder pretraining episodes N_p, number of seed episodes N_s, number of behavior learning episodes N_b, update steps per behavior learning episode G, object-centric autoencoder fine-tuning ratio f

▷ Generate pretraining dataset with random actions:
Initialize environment *env* and pretraining dataset D_p
for episode $e_p = 1$ to N_p **do**
    Observe initial observation $\boldsymbol{o}_0 = env.reset()$ and store it in D_p
    for timestep $t = 0$ to max_timestep - 1 **do**
        Sample random action $\boldsymbol{a}_t$ from action space $A(env)$
        Execute action $\boldsymbol{a}_t$ in *env* and observe $\boldsymbol{o}_{t+1}$
        Store $\boldsymbol{o}_{t+1}$ in D_p
    end for
end for
▷ Pretraining of the object-centric autoencoder:
Randomly initialize the object-centric autoencoder Z_θ (e.g., SAVi)
Pretrain Z_θ on D_p using a reconstruction loss
▷ Main training:
Randomly initialize models: Transition Model P_θ, Reward Model R_θ, Action Model A_θ, Value Model V_θ
Initialize experience replay buffer D with N_s random seed episodes
for episode $e = 1$ to N_b **do**
    for update step $g = 1$ to G **do**
        Sample mini-batch of data sequences from D
        Update models P_θ, R_θ, A_θ, V_θ using sampled mini-batch
        if $(f \cdot g \cdot e) \bmod 1 = 0$ **then**
            Update Z_θ on sampled mini-batch
        end if
    end for
    Observe initial observation $\boldsymbol{o}_0 = env.reset()$ and store it in D
    for timestep $t = 0$ to max_timestep - 1 **do**
        Sample action $\boldsymbol{a}_t \sim A_\theta(\boldsymbol{o}_t)$
        Execute action $\boldsymbol{a}_t$ in *env* and observe $\boldsymbol{o}_{t+1}$, reward r_t
        Store $(\boldsymbol{o}_{t+1}, \boldsymbol{a}_t, r_t)$ in D
    end for
end for

current slots to predict the temporal dynamics and account for object interactions through self-attention.

In the corrector step, the predicted slots $\hat{\mathcal{Z}}_{t+1}$ are refined using information from the current frame. The frame $\boldsymbol{o}_t$ is first processed by a convolutional neural network (CNN) to extract spatial features $\boldsymbol{h}_t$. The refinement is done using Slot Attention, where the predicted slots $\hat{\mathcal{Z}}_{t+1}$ interact with the encoded features $\boldsymbol{h}_t$ to update the slot representations.

The slots $\mathcal{Z}_t = \{\boldsymbol{z}_t^1, \ldots, \boldsymbol{z}_t^N\}$ can be decoded independently into RGB ($\hat{\boldsymbol{o}}_t^n$) and mask predictions ($\hat{\boldsymbol{m}}_t^n$). To obtain the combined reconstructed frame $\hat{\boldsymbol{o}}_t$, the masks are normalized across the slot dimension via softmax, multiplied with the RGB predictions, and then added together:

$$\hat{\boldsymbol{o}}_t = \sum_{n=1}^{N} \boldsymbol{m}_t^n \odot \hat{\boldsymbol{o}}_t^n, \qquad \boldsymbol{m}_t^n = \underset{N}{\text{softmax}}(\hat{\boldsymbol{m}}_t^n), \qquad \hat{\boldsymbol{m}}_t^n, \hat{\boldsymbol{o}}_t^n = \text{dec}(\boldsymbol{z}_t^n) \tag{4.1}$$

In our implementation, we closely follow the SAVi framework as described by (Kipf et al. 2021), including the use of learnable slot initializations. This approach, where slots are initialized as a fixed set of learnable vectors, provides a consistent starting point for the attention mechanism across different video sequences, contributing to stable and accurate object representations in our experiments.

We pre-train SAVi on a dataset generated using randomly selected actions in the environment. The model is trained using a reconstruction loss, calculated as the squared difference between the reconstructed frame $\hat{\boldsymbol{o}}_t$ and the ground truth frame $\boldsymbol{o}_t$:

$$\mathcal{L}_{\text{SAVi}} = \frac{1}{T} \sum_{t=1}^{T} \|\hat{\boldsymbol{o}}_t - \boldsymbol{o}_t\|_2^2 \tag{4.2}$$

Unlike most other approaches that use a fixed pre-trained object-centric encoder, we make the design choice to further fine-tune SAVi during the main reinforcement learning training loop. We update SAVi's parameters at a lower frequency than the other components, using a fraction f of the gradient steps. This fine-tuning allows SAVi to potentially adapt to environments that differ significantly between random and learned behavior. For example, in manipulation tasks, a trained policy may frequently lift objects above the table, while random actions rarely produce such configurations. By fine-tuning, we enable SAVi to maintain accurate object representations across the full range of states encountered during training and execution of the learned policy.

4.1.2. Action Conditioned Object Centric Video Prediction

For our state transition model, we adapt the approach presented by (Villar-Corrales, Wahdan, and Behnke 2023). Specifically, we employ a modified version of their OCVP-Seq module, which demonstrated superior performance in their experiments.

The method proposed by (Villar-Corrales, Wahdan, and Behnke 2023) builds upon the Slot Attention for Video (SAVi) framework (Kipf et al. 2021) to achieve object-centric video prediction. SAVi is used as a scene parsing module to decompose input video frames into a set of object representations, referred to as slots. These slots capture the properties and states of individual objects in the scene. The key idea of their approach is to learn a predictor module that can model the temporal dynamics and interactions of these object slots, enabling the prediction of future object states and, consequently, future video frames. This object-centric approach allows for more structured and interpretable predictions compared to traditional frame-level video prediction methods.

The OCVP-Seq module, as introduced by (Villar-Corrales, Wahdan, and Behnke 2023), is designed to decouple the processing of temporal dynamics and object interactions. It employs two specialized multi-head self-attention mechanisms: temporal attention and relational attention. Temporal attention updates each object slot by aggregating information from that same object up to the current time step, while relational attention models multi-object interactions by jointly processing all slots from the same time step. These attention mechanisms are applied sequentially, allowing for iterative refinement of the slots with both temporal and relational information.

Our key modification to OCVP-Seq lies in incorporating action information into the prediction process. While the original model only considered observations $\{\boldsymbol{o}_0, \ldots, \boldsymbol{o}_C\}$ in the form of their latent slot representations $\{\mathcal{Z}_0, \ldots, \mathcal{Z}_C\} = \{\{\boldsymbol{z}_0^1, \ldots, \boldsymbol{z}_0^N\}, \ldots, \{\boldsymbol{z}_C^1, \ldots, \boldsymbol{z}_C^N\}\}$, our adapted model also takes into account the selected actions $\{\boldsymbol{a}_0, \ldots, \boldsymbol{a}_C\}$ when predicting future frames (e.g., $\hat{\boldsymbol{o}}_{C+1}$).

To accommodate this change with minimal architectural modifications, we map the actions (emb_a) into the token space of the predictor and append them to the respective slots (also mapped into the token space by emb_z) of the corresponding time step (see Figure 4.1). This results in the following transformation of the predictor function:

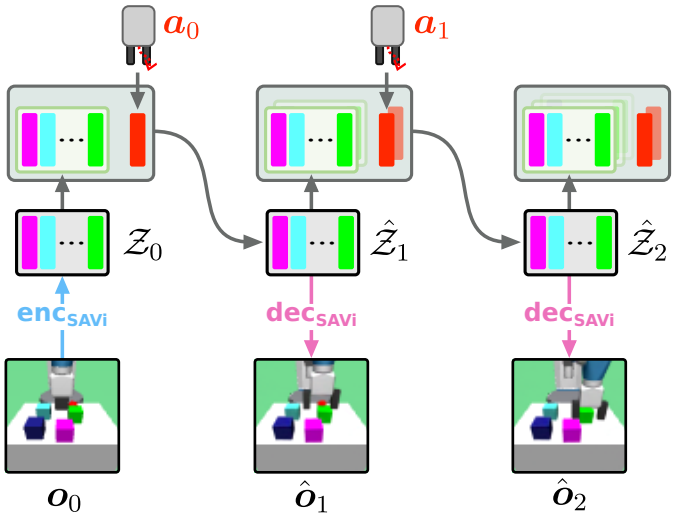


Figure 4.1: The action-conditioned object-centric transition model.

$$\begin{aligned}\hat{\boldsymbol{o}}_{C+1} = \text{dec}_{\text{SAVi}}\big(P_\theta(\{\text{emb}_\text{z}(\boldsymbol{z}_0^1),\ldots,\text{emb}_\text{z}(\boldsymbol{z}_0^N)\},\text{emb}_\text{a}(\boldsymbol{a}_0),\ldots,\\ \{\text{emb}_\text{z}(\boldsymbol{z}_C^1),\ldots,\text{emb}_\text{z}(\boldsymbol{z}_C^N)\},\text{emb}_\text{a}(\boldsymbol{a}_C))\big) \qquad (4.3)\\ = \text{dec}_{\text{SAVi}}\big(P_\theta(\{\text{tok\_}\boldsymbol{z}_0^1,\ldots,\text{tok\_}\boldsymbol{z}_0^N,\text{tok\_}\boldsymbol{a}_0\},\ldots,\\ \{\text{tok\_}\boldsymbol{z}_C^1,\ldots,\text{tok\_}\boldsymbol{z}_C^N,\text{tok\_}\boldsymbol{a}_C\})\big) \qquad (4.4)\end{aligned}$$

In this new formulation, the embedded actions essentially function as an additional slot token for each time step, and are treated as such in both relational and temporal attention mechanisms. The blue colored terms in the equation represent the newly added action embeddings ($\text{emb}_\text{a}(\boldsymbol{a}_t)$) and their corresponding tokens ($\text{tok\_}\boldsymbol{a}_t$). These action tokens are appended to the slot tokens for each time step, allowing the predictor to incorporate action information directly into its processing. It's worth noting that while the transformer encoder processes these additional action tokens, their output is not used in predicting the future observation and is discarded.

To enable multi-step prediction, we can provide the predictor with additional future actions $\{\boldsymbol{a}_{C+1},\ldots,\boldsymbol{a}_{C+T}\}$. Concurrently, predictions from previous time steps (in the form of their slot predictions) are fed back into the predictor, making it autoregressive as introduced by (Villar-Corrales, Wahdan, and Behnke 2023).

For training, we adopt the loss function proposed by (Villar-Corrales, Wahdan, and Behnke 2023). This loss function comprises two components: one that ensures the predicted future frames closely match the actual future frames ($\mathcal{L}_{\text{dyn}_I}$), and another that enforces similarity between the slot predictions (from which the pre-

dicted future frames are rendered) and the slot encodings of SAVi for the future frames ($\mathcal{L}_{\text{dyn}_O}$). The total loss for the predictor is defined as:

$$\mathcal{L}_{\text{dyn}} = \mathcal{L}_{\text{dyn}_I} + \mathcal{L}_{\text{dyn}_O} \tag{4.5}$$

where $\mathcal{L}_{\text{dyn}_I}$ is the image prediction loss:

$$\mathcal{L}_{\text{dyn}_I} = \frac{1}{T}\sum_{t=1}^{T} \|\hat{\boldsymbol{o}}_{t+C} - \boldsymbol{o}_{t+C}\|_2^2 \tag{4.6}$$

and $\mathcal{L}_{\text{dyn}_O}$ is the object slot prediction loss:

$$\mathcal{L}_{\text{dyn}_O} = \frac{1}{T \cdot N}\sum_{t=1}^{T}\sum_{n=1}^{N} \|\hat{\boldsymbol{z}}_{t+C}^{n} - \boldsymbol{z}_{t+C}^{n}\|_2^2 \tag{4.7}$$

Here, T is the number of predicted frames, C is the number of context frames, N is the number of object slots, $\hat{\boldsymbol{o}}_{t+C}$ and $\boldsymbol{o}_{t+C}$ are the predicted and ground truth frames respectively, and $\hat{\boldsymbol{z}}_{t+C}^{n}$ and $\boldsymbol{z}_{t+C}^{n}$ are predicted and reconstructed SAVi object slots.

Unlike SAVi, which is pre-trained, our predictor model (serving as the transitions model in our method) is trained during the main training loop on sequences randomly sampled from the replay buffer.

4.2. Reinforcement Learning

In this section, we detail how the learned slot representations and the prediction model are integrated into our reinforcement learning framework. Our approach builds upon the Dreamer algorithm (Hafner, T. Lillicrap, J. Ba, et al. 2019; Hafner, T. Lillicrap, Norouzi, et al. 2020; Hafner, Pasukonis, et al. 2023), employing an actor-critic architecture for model-based reinforcement learning.

To implement this framework, we introduce three key components: a reward, value, and action model. Each of these models takes the slot history as input and performs a regression task, with the action model potentially producing a multidimensional output.

A central challenge in this approach is effectively processing the slot sets from individual time steps to generate the required output. Given that this task is common to all three models, we have developed a unified architecture, termed the slot processing module. Importantly, each of the reward, value, and action models incorporates its own instance of this slot processing module, allowing for

specialized processing while maintaining a consistent approach to handling slot representations.

In the following subsections, we first introduce the slot processing module, which forms the architectural backbone of our reinforcement learning models. We then describe in detail the reward model, value model, and action model, explaining how each utilizes its own instance of the slot processing module to perform its specific function within the actor-critic framework.

4.2.1. Slot Processing Module

The slot processing module is a core component designed to process the sets of slot representations generated by the SAVi encoder. This module aggregates information across multiple time steps to produce outputs, which are subsequently used by model heads to predict rewards, values, or actions in the reinforcement learning framework.

The primary motivation behind the slot processing module is to address challenges like occlusions and complex temporal dependencies that arise when learning from visual inputs in dynamic environments. By relying on all past time steps' slot representations, the module can maintain a more comprehensive understanding of the environment.

Inspired by the Vision Transformer (ViT) (Dosovitskiy et al. 2020), which effectively handles sets of inputs treated as tokens to produce a singular output, and the slot pooling method introduced in (Yoon et al. 2023) the slot processing module utilizes a transformer encoder architecture. The goal is to relate and compare object properties represented in the slots across different time steps efficiently. A visualization of the slot processing module can be found in Figure 4.2.

The module takes as input a sequence of slot representations $\mathcal{Z} \in \mathbb{R}^{T \times N \times D}$, where T is the number of time steps, N is the number of slots per time step, and D is the slot dimension. For processing, these slots are treated as tokens and passed through a transformer encoder. Additionally, a learnable output token **OUT** is introduced for each time step t, which is responsible for producing the final output for that time step. Unlike the singular class token in the Vision Transformer, we use multiple output tokens corresponding to each time step, ensuring that the module can predict outputs for each time step when needed (e.g., reward prediction).

The transformer encoder receives the slot tokens and the output token as inputs. The attention mechanism within the transformer allows each token, including the output token, to attend to all other tokens from their time step and previous time steps which is ensured by masking the attention weights, enabling the module to capture relationships between objects represented by the slots. Additionally, we

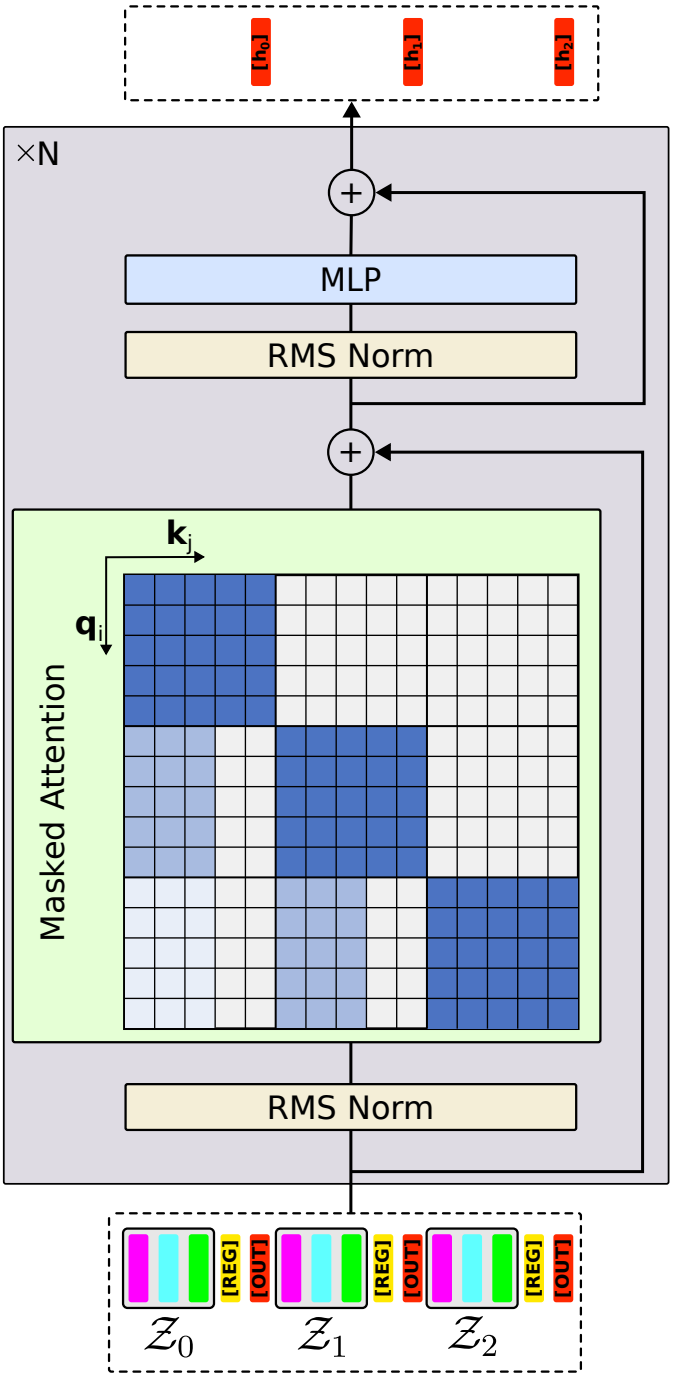


Figure 4.2: The slot processing module. Including a visualization of the attention weight biases and masking in the self attention blocks.

made sure through masking that tokens belonging to time step t cannot directly interact with output tokens (and register tokens, which will be explained next) of time steps $m < t$.

Inspired by the findings of (Darcet et al. 2023), the module also incorporates L learnable register tokens $\mathbf{REG}^l$ for each time step t, which are basically treated like additional output tokens per time step. These tokens serve to offload intermediate computations from the output tokens and help the module focus on relevant slots while ignoring background or irrelevant slots. The output of these register tokens is not directly used, but assists in refining the final output generated by the output tokens.

To encode the position information, we employ Attention with Linear Biases (ALiBi) (Press, Smith, and Lewis 2021) instead of classical absolute position encoding. ALiBi introduces linear biases directly into the attention scores, effectively encoding the recency of tokens, which is crucial when dealing with varying sequence lengths. This approach also helps the module generalize to longer sequences than those seen during training.

The linear bias in ALiBi for a token at position i attending to a token at position j is given by:

$$\text{Bias}_{i,j} = -\alpha|i - j| \tag{4.8}$$

where α is a manually set parameter that controls the slope of the bias. The slope parameter differs for each attention head to ensure that each attention head potentially is able to focus on different time spans. This bias is shared among all tokens (slots, output, and register tokens) within a single time step.

The attention mechanism used in our transformer encoder layers is mathematically described as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \text{Mask} + \text{Bias}\right)V \tag{4.9}$$

where Q, K, and V represent the query, key, and value matrices derived from the input tokens, d_k is their dimensionality and the Mask is added to ensure that tokens attend only to other tokens as described above.

Figure 4.2 illustrates the combined effects of ALiBi and our masking approach on the attention weight calculations. The intensity of cell shading in the masked attention block corresponds to the magnitude of the bias introduced by ALiBi: darker cells indicate a higher bias, resulting in stronger attention between the corresponding tokens. Conversely, white cells represent masked areas where attention between tokens is prohibited, resulting in attention weights of zero. The block-like

structure observed in the figure arises from the uniform ALiBi bias assignment to slots and additional tokens (output and register tokens) within the same time step. Some of these blocks appear incomplete, with sections removed in the area of the additional tokens. This truncation demonstrates that these tokens are prevented from attending to their counterparts in previous time steps.

The final output $\boldsymbol{h}_t$ for each time step t is generated from the corresponding output token after passing through the transformer encoder. This output is then used by subsequent model heads to predict rewards, values, or actions.

This architecture allows our module to process variable-length sequences of slots, capturing both local and global dependencies across time steps. The additional register tokens provide the module with auxiliary memory, offering flexibility in how it processes and stores information across the temporal dimension.

By design, this slot processing module can handle potential occlusions or temporary loss of information in individual frames — a common challenge in visual reinforcement learning. The ability to aggregate information across multiple time steps allows the module to maintain a more robust understanding of the environment, even when individual observations may be incomplete or noisy.

4.2.2. Reward Model

The reward model is an integral component of the world model, working alongside the SAVi encoder and the transition model to predict rewards from a given sequence of slot representations. This model plays a crucial role in the actor-critic framework, as it enables the value and action models to learn from rewards generated by the world model rather than actual environment rewards.

Architecture

The reward model's architecture is built upon a slot processing module instance. This module processes the sequence of slot representations, $\{\mathcal{Z}_0, \mathcal{Z}_1, \ldots, \mathcal{Z}_T\}$, derived from the observations through the SAVi encoder. The processed representations, $\{\boldsymbol{h}_0^r, \boldsymbol{h}_1^r, \ldots, \boldsymbol{h}_T^r\}$, are then passed through a Multi-Layer Perceptron (MLP) head to predict the rewards.

Reward Prediction

Instead of directly predicting a scalar reward value, the MLP head outputs logits for a softmax distribution over K exponentially spaced bins $b_i \in \mathcal{B}$. The bins are

defined as:

$$b_i = \text{symexp}\left(\frac{2i}{K-1} - 1\right) \cdot M, \quad i \in 0, 1, ..., K-1 \tag{4.10}$$

where M is a scalar that determines the maximum absolute bin value, and symexp (Webber 2012) is defined as:

$$\text{symexp}(x) = \text{sign}(x)(\exp(|x|) - 1) \tag{4.11}$$

The predicted reward $\hat{r}_t$ is then computed as the expectation over these bins:

$$\hat{r}_t = \text{softmax}(f_\theta(\boldsymbol{h}_t^r))^T \mathcal{B} \tag{4.12}$$

where $f_\theta(\boldsymbol{h}_t^r)$ represents the MLP head's output logits for the reward prediction corresponding to time step t.

This approach allows the model to efficiently represent a wide range of reward values while maintaining precision for both small and large rewards.

Two-Hot Encoding

For training, the true reward r_t is first transformed using the symmetric logarithm (symlog) function, and this transformed value is then encoded using a two-hot encoding strategy (Marc G. Bellemare, Dabney, and Munos 2017; Schrittwieser et al. 2020). The symlog function is defined as:

$$\text{symlog}(x) = \text{sign}(x) \log(|x| + 1) \tag{4.13}$$

This transformation helps to compress the range of reward values, allowing for better handling of both small and large magnitudes.

The symlog-transformed reward $\text{symlog}(r_t)$ is then distributed across the two nearest bins in the exponentially spaced set. If $\text{symlog}(r_t)$ falls between bins b_k and b_{k+1}, it is encoded as:

$$\text{twohot}(\text{symlog}(r_t))_i = \begin{cases} \frac{b_{k+1} - \text{symlog}(r_t)}{b_{k+1} - b_k} & \text{if } i = k \\ \frac{\text{symlog}(r_t) - b_k}{b_{k+1} - b_k} & \text{if } i = k+1 \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

Unlike one-hot encoding, this encoding also allows any possible intermediate value between two bin centers to be represented precisely.

Loss Function

The model is trained by minimizing the negative log-likelihood of the two-hot encoded reward distribution under the predicted distribution. The loss function for the reward model, $\mathcal{L}_{\text{reward}}$, is defined as:

$$\mathcal{L}_{\text{reward}} = -\frac{1}{T+1}\sum_{t=0}^{T} \text{twohot}(r_t) \log \text{softmax}(f_\theta(\boldsymbol{h}_t^r)) \tag{4.15}$$

This loss function decouples the scale of gradients from the scale of the predicted values, allowing for more stable training across diverse reward scales. The strategy for training the reward model (including symexp/symlog and two-hot encoding) has been adopted from (Hafner, Pasukonis, et al. 2023).

Like the transition model, the reward model is trained during the main training loop on sequences that are randomly drawn from the replay buffer.

4.2.3. Actor-Critic Framework

Our approach builds upon the foundational training framework of the Dreamer algorithm (Hafner, T. Lillicrap, J. Ba, et al. 2019; Hafner, T. Lillicrap, Norouzi, et al. 2020; Hafner, Pasukonis, et al. 2023) to learn optimal behavior from our object-centric world model, comprising the SAVi encoder, object-centric transition model, and reward model.

The core of this method lies in the use of "latent imagination" for behavior learning (see Figure 4.3). The process, detailed in Algorithm 3, involves training both the actor (action model) and the critic (value model) exclusively through interactions with the world model in the latent space of object slots.

Each component of the world model is fully differentiable. This allows us to employ gradient-based optimization techniques, enabling the action model to learn how small adjustments in action selection influence expected future rewards. Consequently, the model can be trained to maximize these predicted rewards through backpropagation.

Algorithm 3 outlines the latent imagination process, demonstrating how we sample from the replay buffer, generate imagined trajectories, and update our models. This process leverages the compact and structured latent space of object-centric representations, to efficiently predict future states and rewards based on actions selected by the actor. The critic is trained to give the actor feedback on his behavior so that he learns to select actions that reflect far-sighted behavior.

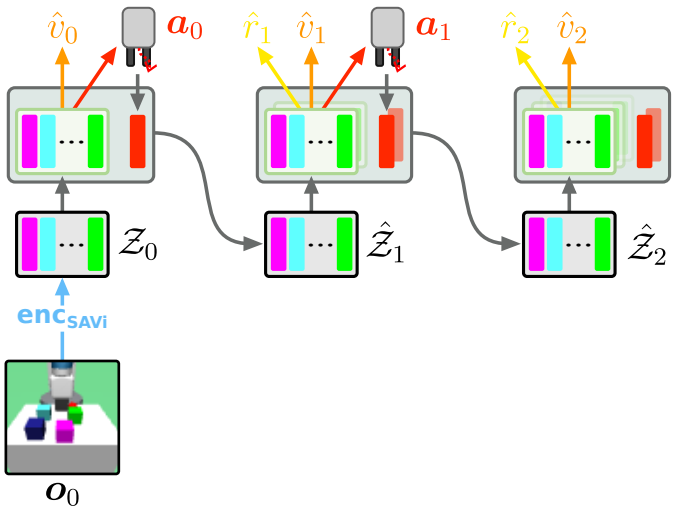


Figure 4.3: Learn behavior through latent imagination.

Algorithm 3 Object-Centric Actor-Critic Learning via Latent Imagination

Require: World model $W_\theta = \{Z_\theta, P_\theta, R_\theta\}$ (SAVi encoder, object-centric transition model, reward model), actor A_ϕ, critic V_ψ, replay buffer $\mathcal{D}$, imagination horizon H, learning rates α_ϕ, α_ψ

for each training iteration **do**
  Sample observation from replay buffer: $\boldsymbol{o}_0 \sim \mathcal{D}$
  Encode observation: $\mathcal{Z}_0 = Z_\theta(\boldsymbol{o}_0)$
  Initialize imagined trajectory: $\hat{\mathcal{Z}}_0 = \mathcal{Z}_0$
  for $h = 0$ to $H - 1$ **do**
    Sample action: $\boldsymbol{a}_h \sim A_\phi(\hat{\mathcal{Z}}_{0:h})$
    Predict next state: $\hat{\mathcal{Z}}_{h+1} = P_\theta(\hat{\mathcal{Z}}_{0:h}, \boldsymbol{a}_{0:h})$
  end for
  Predict rewards: $\hat{r}_{1:H} = R_\theta(\hat{\mathcal{Z}}_{0:H})_{1:H}$
  Predict values: $\hat{v}_{1:H} = V_\psi(\hat{\mathcal{Z}}_{0:H})_{1:H}$
  Compute value estimates $V^\lambda_{0:H-1}$ based on $\hat{r}_{1:H}$ and $\hat{v}_{1:H}$
  Update actor: $\phi \leftarrow \phi - \alpha_\phi \nabla_\phi \mathcal{L}_{\text{actor}}(V^\lambda_{0:H-1})$
  Update critic: $\psi \leftarrow \psi - \alpha_\psi \nabla_\psi \mathcal{L}_{\text{critic}}(V^\lambda_{0:H-1})$
end for

Value Model

The value model serves as the critic component in the actor-critic framework, which is needed for estimating expected long-term rewards. Its architecture mirrors that of the reward model, consisting of a slot processing module followed by a Multi-Layer Perceptron (MLP) head.

Unlike the reward model, which predicts immediate rewards, the value model estimates the sum of discounted future rewards. As there is no ground truth data for this, value estimates $V^{\lambda}_{0:H-1}$ are formed from predicted rewards $\hat{r}_{1:H}$ and values $\hat{v}_{1:H}$, which serve as target values for the value model training:

$$V^{\lambda}_{h} = \hat{r}_{h+1} + \gamma \left((1-\lambda)\hat{v}_{h+1} + \lambda V^{\lambda}_{h+1} \right) \qquad V^{\lambda}_{H} = \hat{v}_{H} \tag{4.16}$$

Here, γ is the discount factor, H is the imagination horizon and λ is a mixing parameter to trade off bias and variance in the training of the value model (Sutton and Barto 2018).

The value model is trained to predict these value estimates using a categorical distribution over exponentially spaced bins, similar to the reward model. The loss function for the value model is:

$$\mathcal{L}_{\text{value}} = -\frac{1}{H} \sum_{h=0}^{H-1} \text{twohot}(V^{\lambda}_{h})^{T} \log \text{softmax}(f_{\psi}(\boldsymbol{h}^{v}_{h})) \tag{4.17}$$

where $f_{\psi}(\boldsymbol{h}^{v}_{h})$ represents the output logits of the value model's MLP head for the processed state representation $\boldsymbol{h}^{v}_{h}$.

To enhance training stability, since the value model is trained on targets that depend on its own predictions, we maintain a target network, whose parameters are an exponential moving average (EMA) of the parameters of the value model. The value predictions of this target network serve in addition to the computed value estimates as training targets for the value model. This technique was introduced by (Hafner, Pasukonis, et al. 2023) but is similar to ideas by (Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015). The target network parameters are updated as

$$\psi_{\text{target}} \leftarrow \tau \psi_{\text{value}} + (1-\tau)\psi_{\text{target}}, \tag{4.18}$$

where τ is a small constant. We have decided to set τ to a value of 0.02 in our experiments.

Action Model

The action model, serving as the actor component in the framework, shares a similar architectural foundation with the reward and value models. It incorporates a slot processing module followed by a Multi-Layer Perceptron (MLP) head. However, unlike its counterparts that predict scalar values, the action model's MLP head is designed to output parameters for an action distribution.

Specifically, the MLP head predicts a mean $\mu_{\boldsymbol{a}_t}$ and a standard deviation $\sigma_{\boldsymbol{a}_t}$ to parameterize a normal distribution $\mathcal{N}(\mu_{\boldsymbol{a}_t}, \sigma_{\boldsymbol{a}_t}|\mathcal{Z}_{0:t})$ over possible actions. This distributional approach to action selection offers several advantages, particularly in terms of exploration. During training, actions can be sampled from this distribution, implementing an adaptive exploration strategy that modulates the degree of exploration based on the current state. This is especially beneficial in environments where certain states require more cautious exploration to avoid potentially catastrophic outcomes.

To encourage exploration while maintaining good performance, we employ entropy regularization (Haarnoja et al. 2018; Williams and Peng 1991) in the actor's loss function. This approach adds the entropy of the predicted action distribution to the loss, incentivizing the model to maintain a degree of randomness in its policy without sacrificing expected returns. The resulting loss function for the actor is formulated as:

$$\mathcal{L}_{\text{actor}} = -\sum_{h=0}^{H-1} \frac{V_h^\lambda}{\max(1, \text{scale}_V)} + \eta\mathcal{H}(\mathcal{N}(\mu_{\boldsymbol{a}_h}, \sigma_{\boldsymbol{a}_t}|\mathcal{Z}_{0:h})) \tag{4.19}$$

where V_h^λ represents the value estimate at step h, $\mathcal{H}$ is the entropy of the action distribution, and η is a hyperparameter controlling the strength of the entropy regularization.

To adapt to varying scales of value estimates across different environments, we use a normalization factor scale_V which was introduced by (Hafner, Pasukonis, et al. 2023). This factor is computed using an exponential moving average (EMA) of the difference between the 95th and 5th percentiles of the value estimates:

$$\text{scale}_V = \text{EMA}\left(\text{Per}(V_h^\lambda, 95) - \text{Per}(V_h^\lambda, 5), 0.99\right) \tag{4.20}$$

This adaptive scaling ensures that the actor's learning remains stable across a wide range of environments with potentially different reward scales and dynamics.

4.2.4. Implementation Details

In this section, we detail the explicit model configuration of the model components we discussed in the previous sections.

World Model Components

The SAVi encoder consists of a convolutional neural network (CNN) with four layers, each having 32 channels and a kernel size of 5. The encoder processes 64×64 RGB images without downsampling. The slot attention mechanism uses a variable number of slots (typically between 2 and 10 depending on the experiment), each with a dimension of 128. The decoder mirrors the encoder structure but with 64 channels per layer and includes upsampling to reconstruct the original image size. We use 3 slot attention iterations for the first frame and 1 for subsequent frames in the slot attention process of SAVi. SAVi employs Layer Normalization (J. L. Ba, Kiros, and Hinton 2016) and ReLU activation functions.

SAVi also includes a predictor module, implemented as a transformer block with 4 attention heads and an MLP size of 256. This predictor is used to generate initial slot estimates for subsequent frames based on the slots from the previous frame.

The SAVi model is pretrained using the Adam (Kingma 2014) optimizer on approximately one million frames for 400,000 gradient steps. During pretraining, we use a batch size of 64 and an initial learning rate of 1e-4. We employ learning rate warmup for the first 2,500 gradient steps, followed by cosine annealing (Loshchilov and Hutter 2016) for the remaining steps. Gradient clipping with a maximum norm of 0.05 is applied during training.

The object-centric transitions model, based on the OCVP-Seq architecture, is implemented as a transformer encoder (with temporal and relational attention). It uses 4 layers, each with 8 attention heads for both types of attention. The token dimension is set to 256, and the hidden dimension in the feed-forward layers is 512. The model incorporates residual connections. Like SAVi, it uses Layer Normalization and ReLU activations. The transitions model employs absolute positional encoding.

Reinforcement Learning Components

The reward model, value model, and action model all utilize a slot processing module as their backbone. This module is implemented as a transformer encoder with 4 layers, 8 attention heads, a token dimension of 256, and a hidden dimension in the feed-forward layers of 512. Each model includes 4 learnable register tokens to enhance information processing. These models use RMSNorm (Zhang and Sen-

nrich 2019) for normalization and SiLU (Hendrycks and Gimpel 2016) activation functions. For positional encoding, these models use Attention with Linear Biases (ALiBi).

For the reward and value models, we use a categorical distribution with 255 bins, spanning the range from -20 to 20. The action model outputs the mean and standard deviation for a Normal distribution, which is then transformed to bound the actions within the environment's action range.

Training Procedure

We also use the Adam optimizer for the main training phase. Different learning rates are used for various components: 1e-4 for both the transition and reward models, and 3e-5 for the SAVi encoder, the action and value models. To stabilize training, we employ gradient clipping with different maximum norms: 0.05 for the SAVi model, 3.0 for the transition model, and 10.0 for the reward, value, and action models.

For all components in the main training phase, we also use learning rate warmup for the first 2,500 gradient steps.

Additionally, we implement the exponential moving average (EMA) for the target value network with a decay rate of 0.98. We use an imagination horizon of 15 steps for behavior learning. The λ-parameter is set to 0.95.

5. Experiments

To evaluate the effectiveness of our object-centric model-based reinforcement learning approach, we conducted a series of experiments on simulated robotic manipulation tasks. This chapter describes our experimental setup, including the environments we designed, the baselines we compared against, and both quantitative and qualitative analyses of our results. We begin by detailing the suite of robotic tasks, then present our evaluation methodology and findings, demonstrating the advantages of our approach in terms of sample efficiency, performance, and interpretability.

5.1. Environments

To test our model, we developed a suite of robotic manipulation tasks based on the framework introduced by (R. Li et al. 2020). These environments are simulated using MuJoCo (Todorov, Erez, and Tassa 2012). The basic structure across all environments consists of a robot arm mounted on a base, positioned near a table where the manipulation tasks take place.

We designed three categories of tasks: Reach, Push, and Pick-and-Place. Each category has two variants: Specific and Distinct. The Specific variant requires the robot to interact with a target object of a particular color, while the Distinct variant challenges the robot to identify and manipulate the object that is visually distinct from the others. This latter variant is particularly interesting as it requires the agent to compare object properties, testing its ability to reason about relationships between entities in the scene

In all environments, the robot is controlled by a 4-dimensional action vector $\mathbf{a} = [a_x, a_y, a_z, a_{\text{grip}}] \in [-1, 1]^4$, where the first three components represent the desired movement direction of the end effector, and the fourth component controls the gripper. In Reach and Push tasks, a_{grip} is ignored. In these tasks, the gripper is fixed in the closed configuration, as it is not required to solve the tasks.

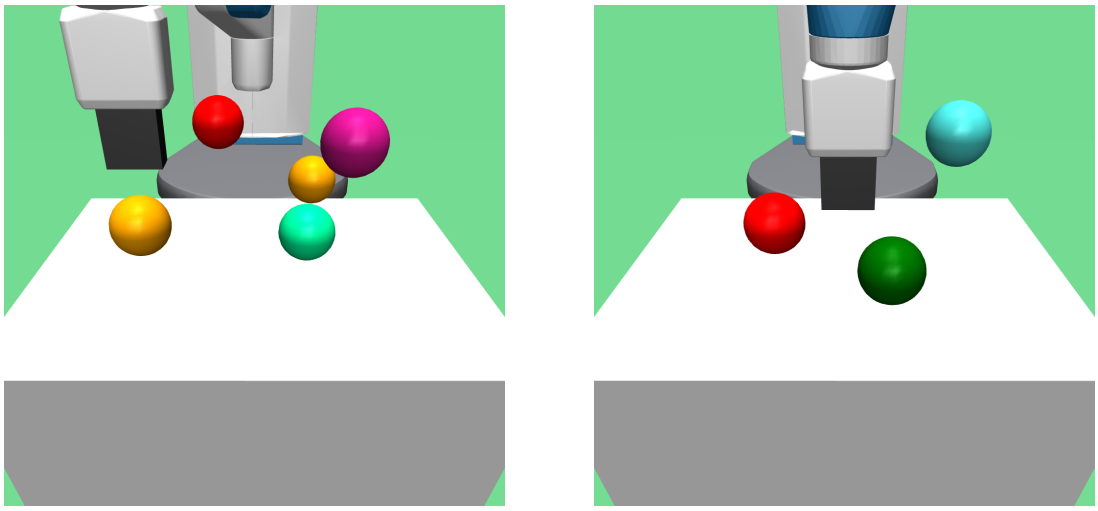
For all tasks, we define the following constants:

- $t_1 = 20$ and $t_2 = 10$: Temperature parameters that determine the steepness of the reward function.

- $d_{\text{success}} = 0.05$: The distance threshold (in meters) for considering a task successful.

5.1.1. Reach Environments

In Reach tasks, the robot arm must touch a spherical target while being distracted by other targets. The positions of the targets are generated randomly. Figures 5.1 and 5.2 illustrate these environments.



(a) Four distractor targets

(b) Two distractor targets

Figure 5.1: Reach Specific environment

Reach Specific

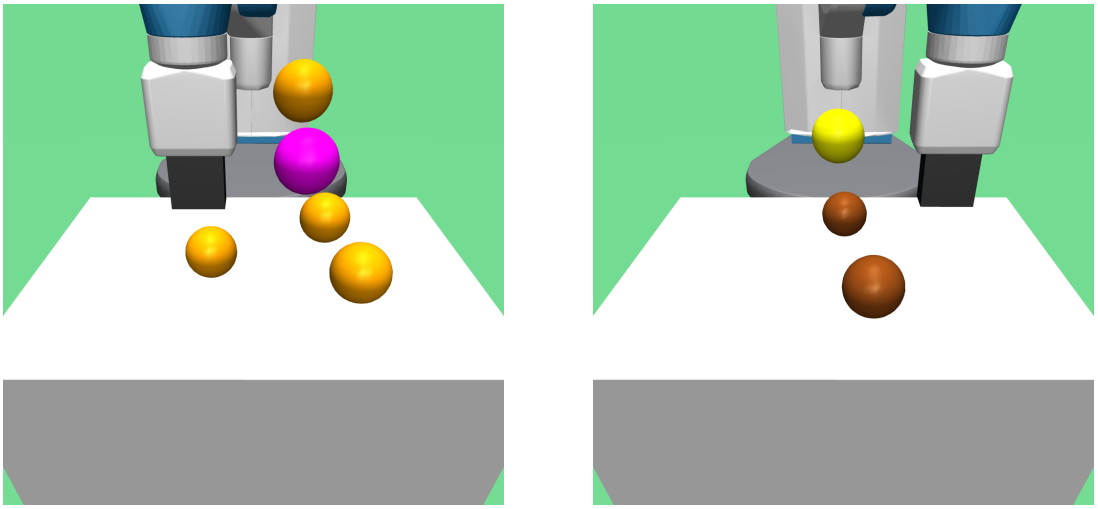
The goal is to reach the red sphere among differently colored distractor spheres. The number (between zero and four) and color of the distractor targets is determined randomly. The reward is calculated as:

$$r = \exp(-t_1 \cdot \|\mathbf{p}_{\text{e}} - \mathbf{p}_{\text{target}}\|_2) \tag{5.1}$$

where $\mathbf{p}_{\text{e}}$ is the position of the end effector and $\mathbf{p}_{\text{target}}$ is the position of the target sphere.

Reach Distinct

The robot must touch the sphere that is a different color from the others. The reward function is the same as in Reach Specific, but the target is the uniquely



(a) Four distractor targets (b) Two distractor targets

Figure 5.2: Reach Distinct environment

colored sphere. The number of distractors here is at least two to make the task solvable. Success at the end of the episode in both variants is defined as:

$$\text{success} = \begin{cases} 1 & \text{if } \|\mathbf{p}_\text{e} - \mathbf{p}_\text{target}\|_2 < d_\text{success} \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

5.1.2. Push Environments

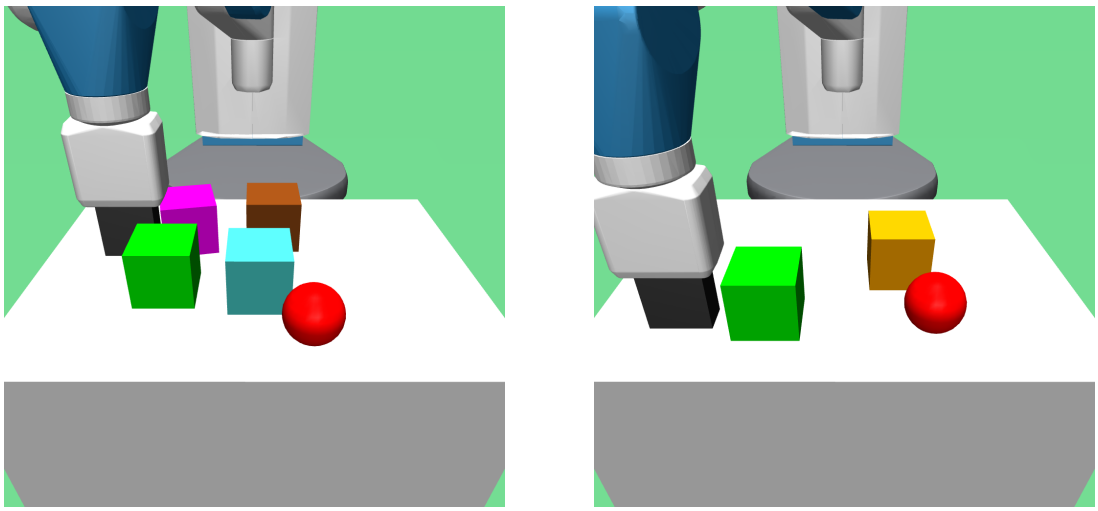
Push tasks require the robot to slide a cube to a target location (red sphere). The positions of the cubes and the target are generated randomly on the table. Figures 5.3 and 5.4 show these environments.

Push Specific

The agent must push the green cube to the red target area among differently colored distractor cubes. The number (between zero and three) and color of the distractors cubes is determined randomly. The reward is:

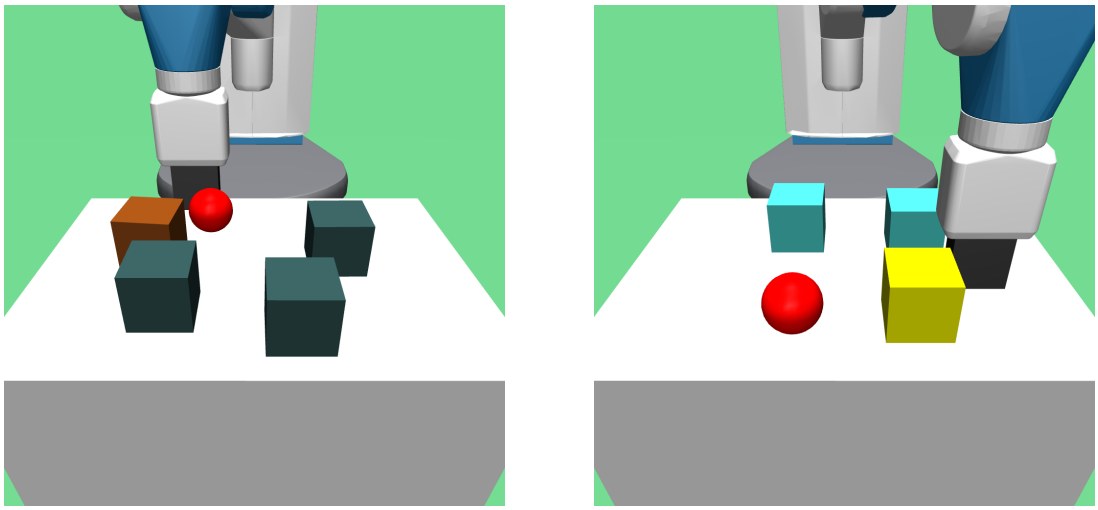
$$r = 0.9 \cdot \exp(-t_1 \cdot \|\mathbf{p}_\text{cube} - \mathbf{p}_\text{target}\|_2) + 0.1 \cdot \exp(-t_2 \cdot \|\mathbf{p}_\text{e} - \mathbf{p}_\text{cube}\|_2) \tag{5.3}$$

where $\mathbf{p}_\text{cube}$ is the position of the green cube.



(a) Three distractor cubes (b) One distractor block

Figure 5.3: Push Specific environment



(a) Three distractor cubes (b) Two distractor block

Figure 5.4: Push Distinct environment

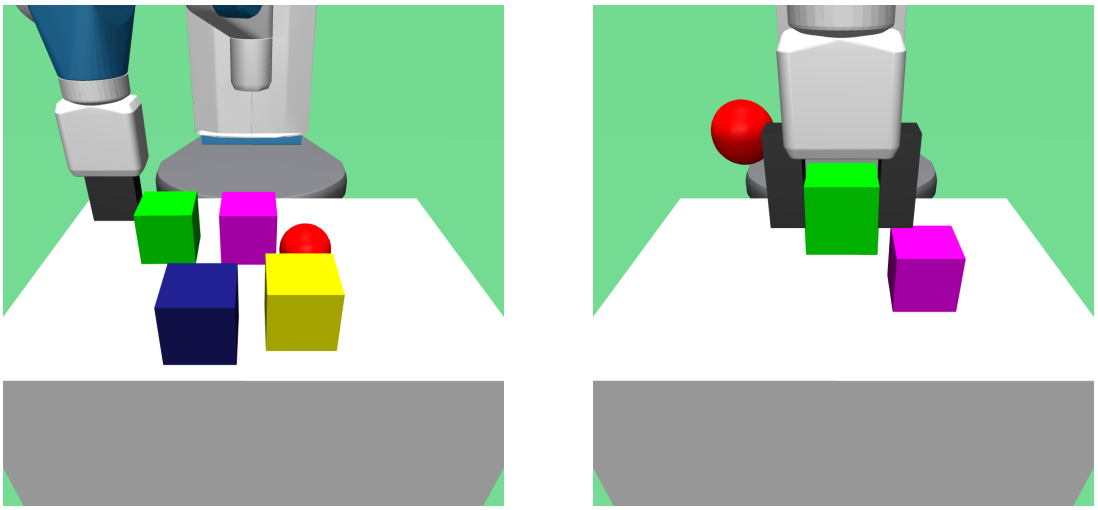
Push Distinct

The goal is to push the uniquely colored cube to the target area. The number of distractor cubes here is again at least two to make the task solvable. The reward function is the same as in Push Specific, but $\mathbf{p}_{\text{cube}}$ refers to the position of the distinctly colored cube. Success (at the end of the episode) for both variants is defined as:

$$\text{success} = \begin{cases} 1 & \text{if } \|\mathbf{p}_{\text{cube}} - \mathbf{p}_{\text{target}}\|_2 < d_{\text{success}} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

5.1.3. Pick and Place Environments

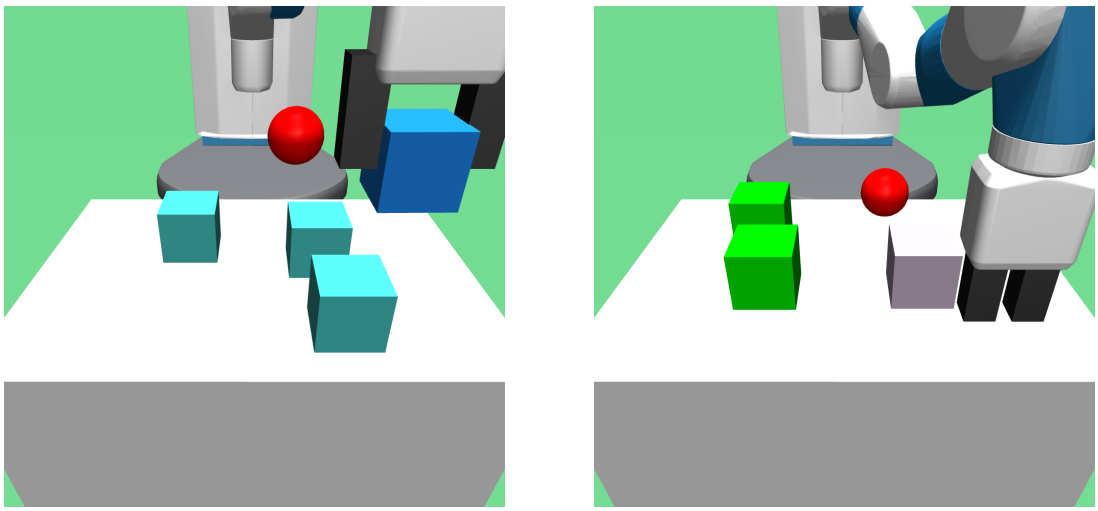
Pick-and-Place tasks require placing a cube at a target location. The difference to the Push tasks is that the target (red ball) is in the air in 50 percent of cases (decided randomly during episode generation). It is therefore possible in these tasks for the gripper jaws to be opened in order to grab and lift cubes. The number and color of the cubes is determined as in the Push environments. Figures 5.5 and 5.6 depict these environments.



(a) Three distractor cubes

(b) One distractor block

Figure 5.5: Pick-and-Place Specific environment



(a) Three distractor cubes (b) Two distractor block

Figure 5.6: Pick-and-Place Distinct environment

Pick-and-Place Specific

The robot must place the green cube on the red target area among differently colored distractor cubes. The reward function is identical to the Push tasks.

Pick-and-Place Distinct

The agent must identify and place the uniquely colored cube on the target area. The reward function is the same as in Pick-and-Place Specific, with $\mathbf{p}_{\text{cube}}$ referring to the position of the distinctly colored cube. The success criterion for both variants is the same as in the Push tasks.

In Reach environments, episodes terminate after 50 steps and in Push and Pick-and-Place environments after 100 steps. Colors for objects are sampled from a predefined set of 16 colors when needed. In our environments, targets are massless spheres whereas the objects to be manipulated are cubes which have a mass associated with them.

5.2. Evaluation

This section presents a comprehensive evaluation of our proposed method against two baselines: DreamerV3, representing the state-of-the-art in model-based reinforcement learning, and a variant of our model without object-centric representations. We first describe the baseline methods we compare our model against.

We then present quantitative results, including learning curves, final performance comparisons, and an analysis of sample efficiency. Finally, we provide qualitative insights into our model's behavior through visualizations of learned representations and attention mechanisms.

5.2.1. Baselines

To evaluate the effectiveness of our object-centric model-based reinforcement learning approach, we compare it against two baseline methods across the environments described in Section 5.1.

The first baseline is DreamerV3, the latest iteration of the Dreamer algorithm family developed by (Hafner, Pasukonis, et al. 2023). DreamerV3 represents the state-of-the-art in model-based reinforcement learning and can even be regarded as one of the most powerful reinforcement learning algorithms available. To ensure a fair comparison, we use the DreamerV3 variant with approximately 12 million parameters, matching the total parameter count of our proposed method. For this baseline, we utilize the official implementation provided by the authors of DreamerV3, adapting it to incorporate our custom environments.

The second baseline is a variant of our proposed method, designed to isolate the impact of object-centric representations. In this variant, we replace the Slot Attention for Video (SAVi) (Kipf et al. 2021) module with a convolutional neural network (CNN) autoencoder, while keeping the rest of the architecture largely unchanged. This modification results in a model that generates a single latent vector representation per observation, as opposed to multiple object-centric slot representations.

The CNN autoencoder used in this baseline consists of an encoder and a decoder. The encoder comprises four convolutional layers with 64, 128, 256, and 512 channels respectively, each followed by batch normalization and a ReLU. Importantly, each convolutional layer uses a stride of 2, resulting in downsampling at each step. The output of the final convolutional layer is flattened and passed through a linear layer to produce a 512-dimensional latent vector (making it five times as large as the slots representations used in our method). The decoder mirrors this structure, starting with a linear layer to reshape the latent vector, followed by four transposed convolutional layers that progressively upsample the feature maps. The final layer uses a sigmoid activation to produce the reconstructed image.

To compensate for the lack of multiple latent vectors and to ensure a fair comparison, we increased the capacity of this baseline model. The total parameter count for this baseline is approximately 60 million, which is five times larger than our proposed method and the DreamerV3 baseline (both of which have about 12

million parameters).

This increased capacity is achieved by expanding the size of the transformer networks in the actor, critic, and transition models. Specifically, the hidden dimensions in these models were increased from 512 to 1024. The token dimension in these models was also increased from 256 to 512.

By comparing our proposed method against these two baselines, we aim to demonstrate not only its overall performance in relation to the current state-of-the-art, but also to quantify the specific benefits derived from its object-centric approach to model-based reinforcement learning.

5.2.2. Quantitative Evaluation

To evaluate the performance of our method and compare it to the baselines, we conducted extensive experiments across our suite of robot manipulation tasks. Each method was trained twice on each task with different random seeds to account for variability. The training hyperparameters used are detailed in Table 5.1.

Figure 5.7 presents the episode returns obtained during training for each method. These episodes were subsequently added to the replay buffer. It's important to note that our method underwent pre-training for 40,000 episodes in Reach tasks and 20,000 episodes in Push and Pick-and-Place tasks. To account for this, we shifted the curves of our method by the corresponding number of episodes.

To visualize the episode returns effectively, we applied an exponential moving average with a factor of 0.999 for smoothing. We then subsampled a total of 150,000 points for each training run and calculated the mean value across the two seeds for each point. The standard deviation is represented by the shaded area in a slightly lighter color.

Figures 5.8 and 5.9 illustrate the average return and success rate, respectively, on 100 seeded test episodes. For these evaluations, we tested all models every 10,000 steps on the same set of 100 test episodes. Unlike the training curves, these figures show raw data without smoothing or downsampling. Again, we present the mean and standard deviation across the two seeds.

The final performance of the fully trained models is summarized in Figures 5.10 and 5.11, which show the average returns and success rates achieved on 1,000 test episodes.

Notably, our object-centric model consistently outperforms the baseline that does not use an object-centric environment representation. This demonstrates the power of decomposing the environment into multiple objects. Even though the baseline model has five times more parameters than our method, it fails to match its performance, often by a significant margin.

Table 5.1: Hyperparameters Robot Environments

Category	**Parameter**	**Value**
SAVi Pretraining	Pregenerated Episodes (N_p)	40k (Reach) 20k (Push/Pick-and-Place)
	Number of Slots (N)	8 (Reach/Push) 10 (Pick-and-Place)
	Learning Rate	1e-4
	Gradient Steps	400k
	LR Warmup Steps	2500
	Batch Size	64
	Chunk Size	10
Behaviour Learning	Imagination Horizon (H)	15
	Discount Factor (γ)	0.96 (Reach) 0.98 (Push/Pick-and-Place)
	λ	0.95
	Actor entropy factor (η)	3e-4
	Critic EMA Decay ($1-\tau$)	0.98
Training	SAVi Learning Rate	3e-5
	Predictor Learning Rate	1e-4
	Reward Learning Rate	1e-4
	Actor Learning Rate (α_ϕ)	3e-5
	Critic Learning Rate (α_ψ)	3e-5
	Gradient Steps per Episodes (G)	1 (Reach) 2 (Push/Pick-and-Place)
	Gradient Steps per Ep. (SAVi) (f)	0.1 (Reach) 0.2 (Push/Pick-and-Place)
	LR Warmup Steps	2500
	Batch Size	64 (RL + SAVi) 32 (Predictor)
	Chunk Size	10 (SAVi) 16 (RL + Predictor)
	Predictor Seed Frames (C)	1
	Number of Episodes (N_b)	150k (Reach Specific) 200k (Reach Distinct) 270k (Push/Pick-and-Place)
	Number of Seed Episodes (N_s)	5
	Action Repeat	2
	Replay Capacity (Episodes)	100k (Reach) 50k (Push/Pick-and-Place)

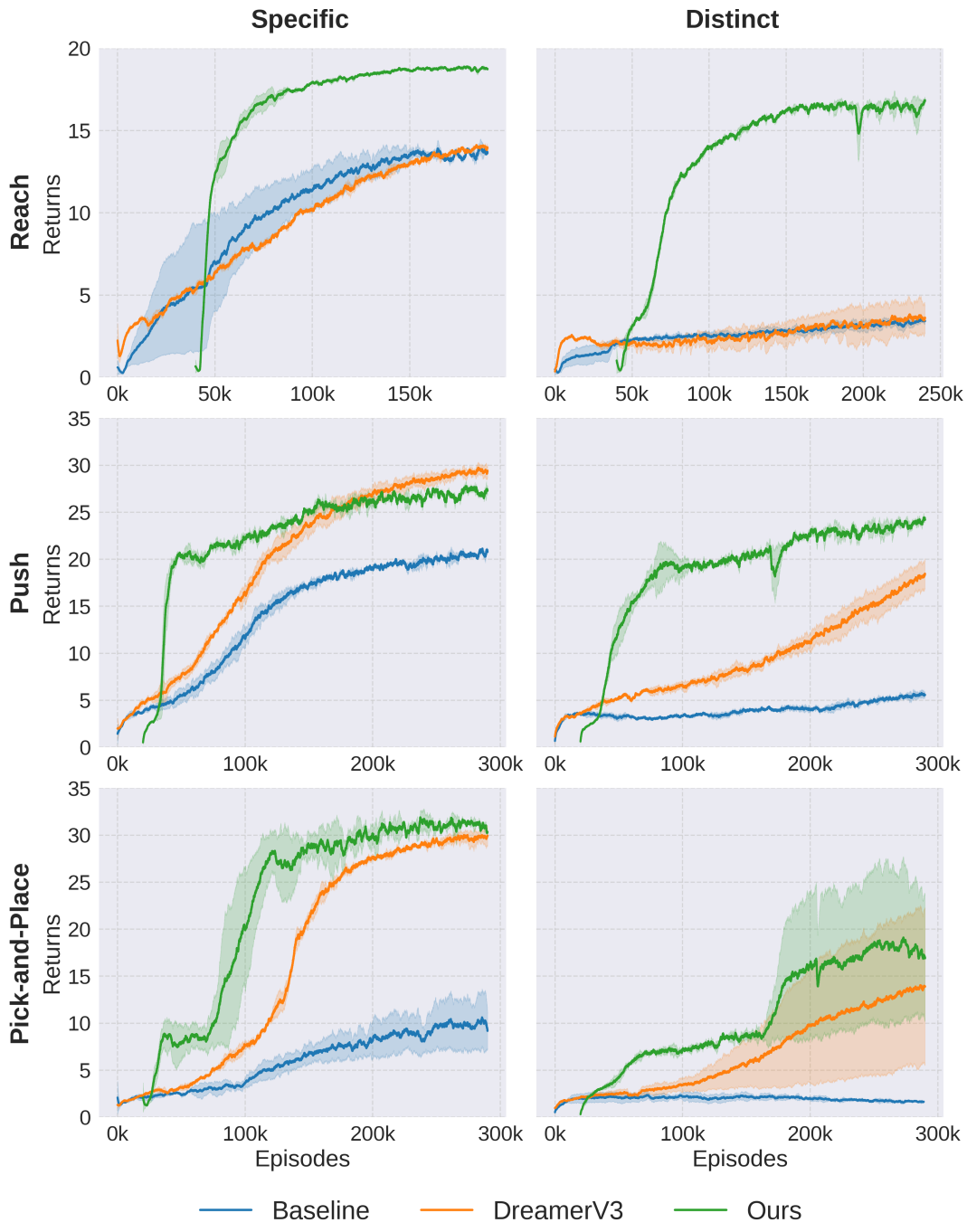


Figure 5.7: Returns received on episodes during training.

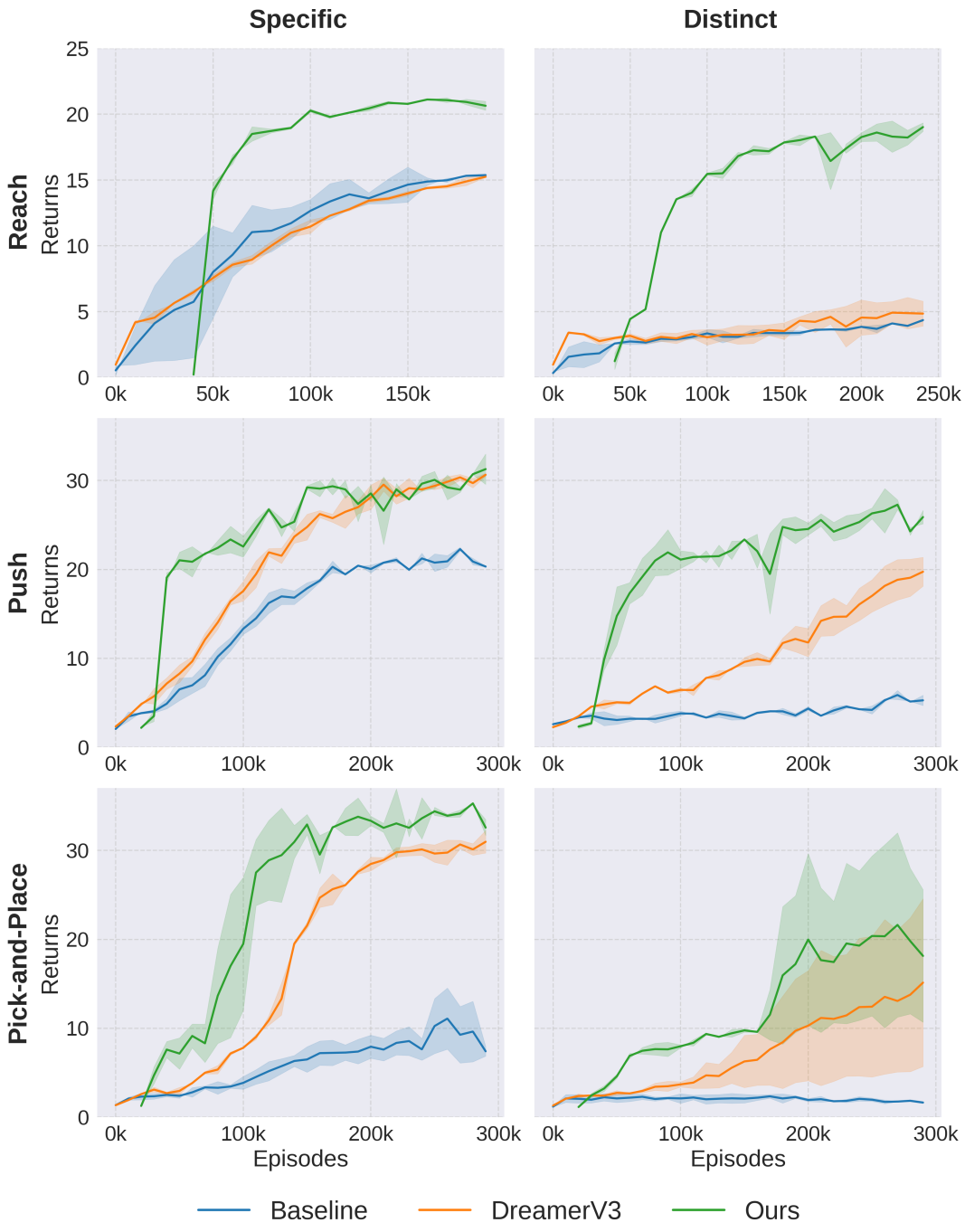


Figure 5.8: Average returns received on test episodes during training.

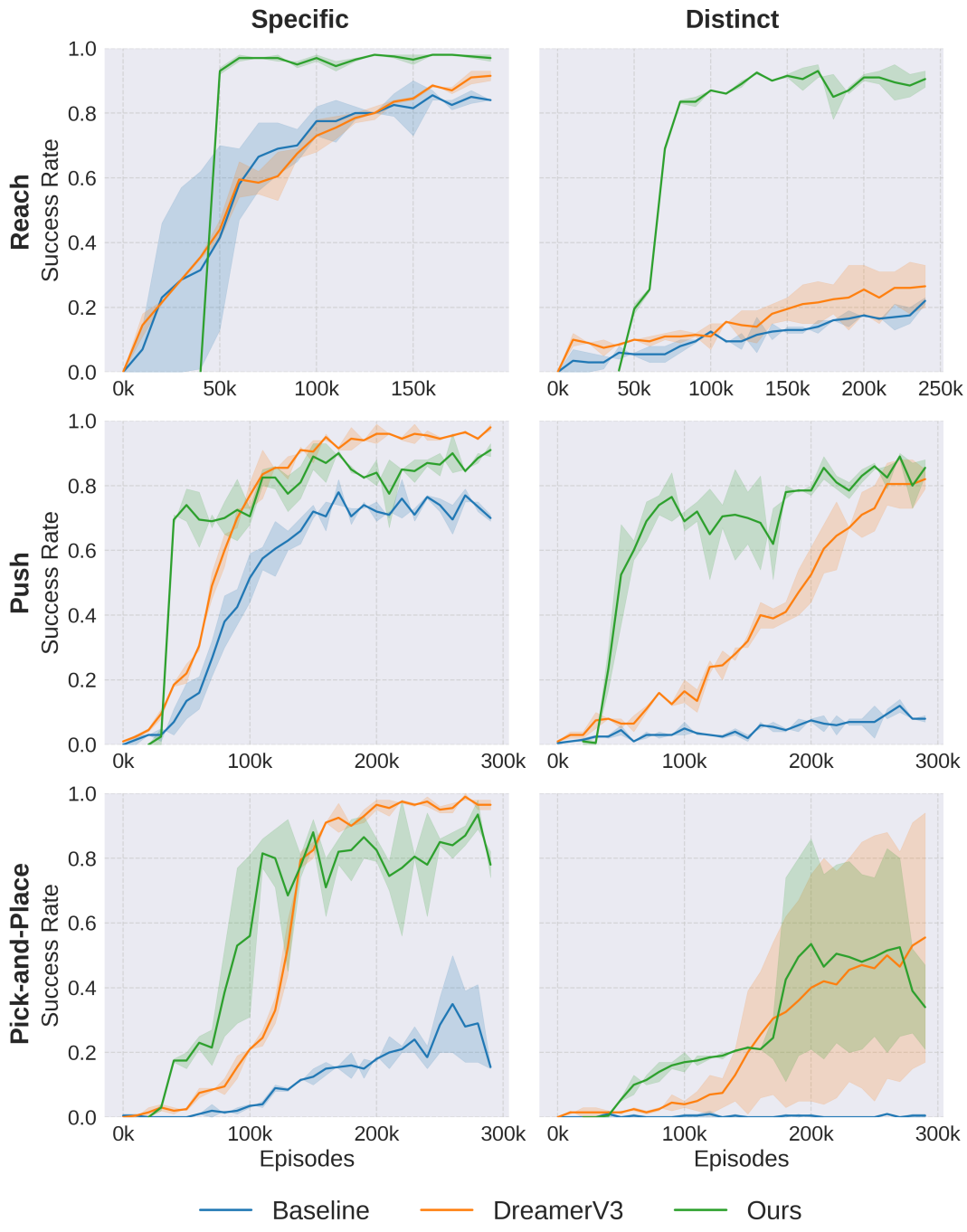


Figure 5.9: Success rates achieved on test episodes during training.

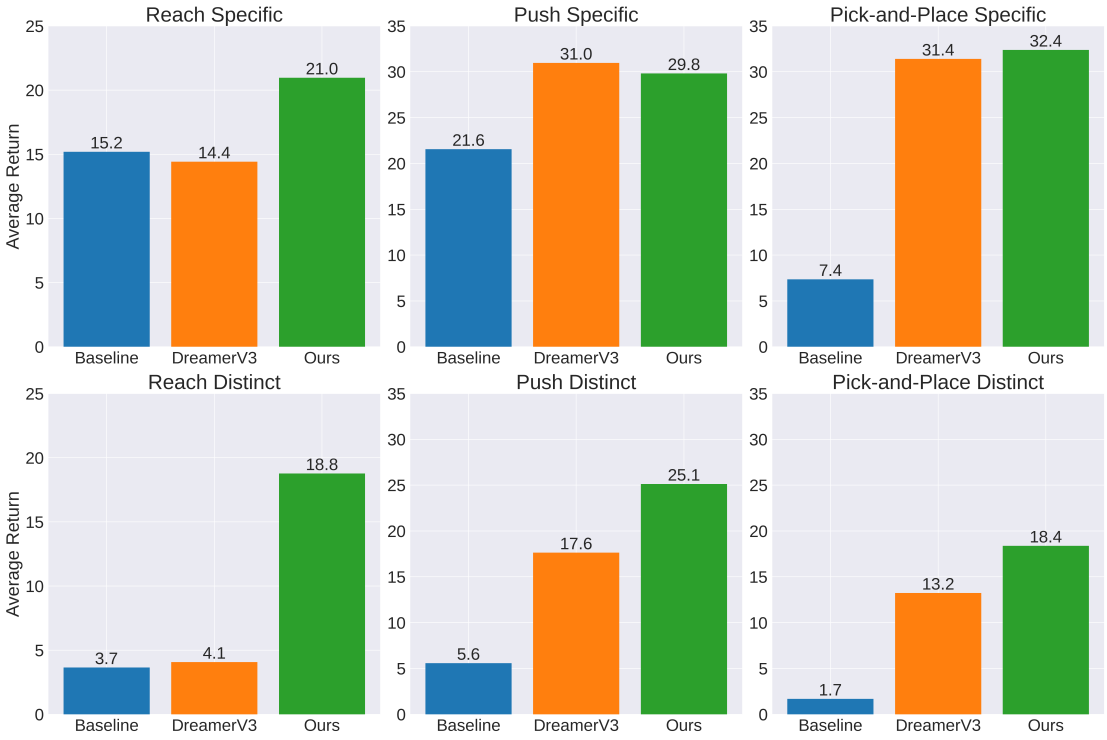


Figure 5.10: Average returns received by fully trained models.

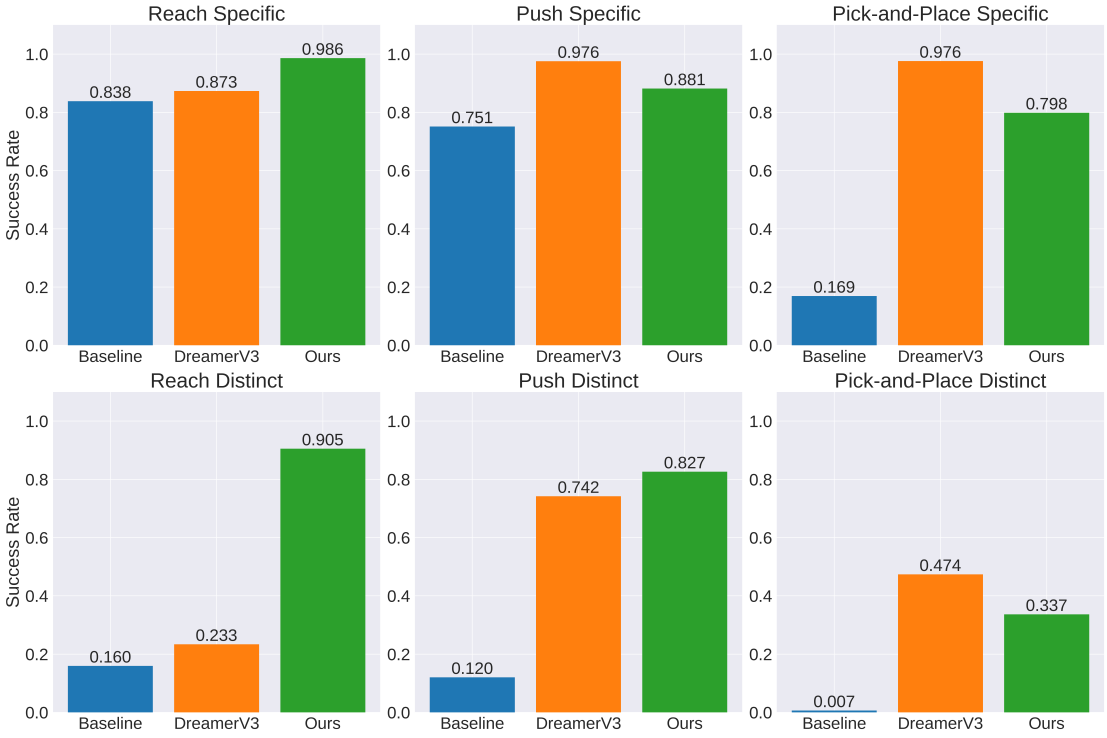


Figure 5.11: Average success rates achieved by fully trained models.

When compared to the state-of-the-art method DreamerV3, our model shows competitive or superior performance, as evident in Figure 5.10. This advantage is particularly pronounced in the Distinct tasks, suggesting that our object-centric approach is especially beneficial for tasks requiring relational reasoning between objects.

While our success rate (Figures 5.9 and 5.11) is occasionally lower than DreamerV3, it's important to note that this metric only captures performance at the final step and uses a binary criterion. So this criterion might give the reader a better idea about the performance of the models, because it is easier to interpret than an arbitrary number, like it is the case with the episode return. But in contrast, the models are trained to maximize expected return, which provides a more nuanced measure of performance throughout the episode.

Another key advantage of our method is its improved sample efficiency, particularly evident in Figures 5.8 and 5.7. Our method often learns rapidly, achieving good average returns much earlier in training compared to the baselines. This efficiency could be crucial in applications where data collection is costly or time-consuming.

Long-Horizon Reasoning

To further test the models' ability to retain information over longer time horizons, we conducted an additional experiment on the Push Distinct task. After the initial observation, all but one distractor cube were removed from the simulation, as if the cubes have fallen off the table. We evaluated this "Disappearing Objects" variant using the same 1,000 seeded episodes as in Figures 5.10 and 5.11. Figure 5.12 compares the results with the original task performance.

Our method showed resilience to this modification, with performance decreases of only 8.76% in average return and 9.79% in success rate. In contrast, DreamerV3 experienced more significant drops of 17.05% and 27.90% respectively. The baseline method slightly improved, likely due to the simplified decision space with fewer objects. These results suggest that our object-centric approach better maintains relevant information about temporarily invisible objects, a useful capability for long-horizon reasoning in dynamic environments.

These results collectively demonstrate the effectiveness of our object-centric approach in model-based reinforcement learning, particularly for tasks requiring understanding and manipulation of multiple objects in a scene and potentially needing to reason about their properties.

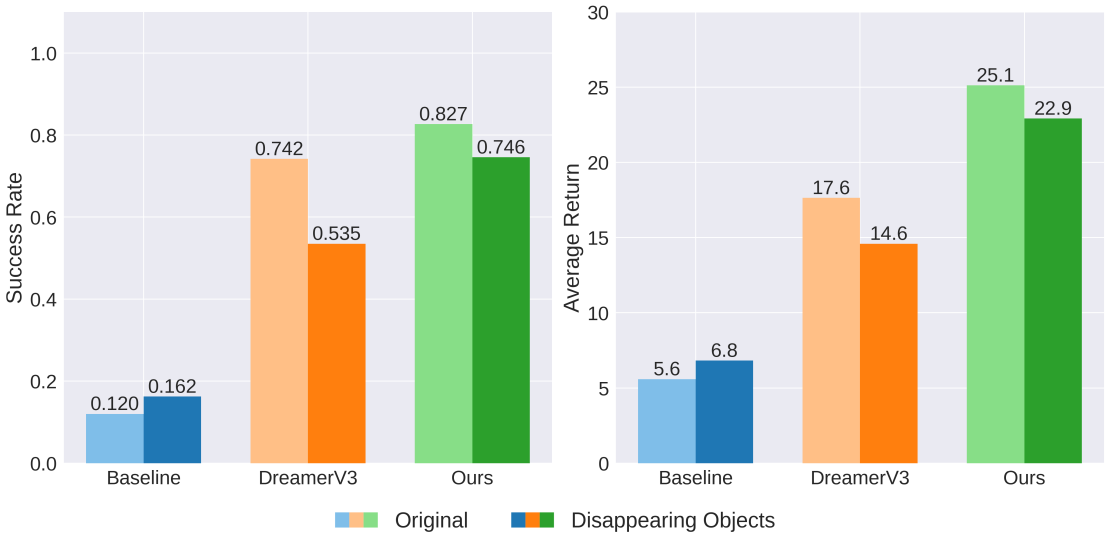


Figure 5.12: Model performance comparison on Push Distinct when all but one distractor cube are removed after the initial observation.

5.2.3. Qualitative Evaluation

To gain deeper insights into our model's performance and behavior, we conducted a qualitative analysis focusing on open-loop predictions, the visualization of attention weights, and evaluation of the SAVi fine-tuning.

Open-Loop Predictions

Figures 5.13 and 5.14 illustrate open-loop prediction sequences for the Pick-and-Place Distinct and Push Distinct environments, respectively. These visualizations demonstrate how our trained transitions model predicts future slots and, consequently, future frames based on a single context frame and an action sequence.

Figure 5.13 showcases a rollout in the Pick-and-Place Distinct environment, depicting a successful grasp of the target cube and its subsequent repositioning to the target location. The SAVi model effectively decomposes the initial scene into individual objects, a decomposition that is maintained by the transition model throughout the prediction sequence. Both the robot's movement and the resulting motion of the grasped block are predicted with high accuracy, demonstrating the model's ability to capture complex object interactions. Furthermore, the model successfully predicts and handles occlusions, as evidenced by the continued accurate prediction of the occluded object in the slot at the bottom of the frame.

Figure 5.14 further illustrates the model's capabilities in the Push Distinct environment. This sequence showcases the model's capacity to predict complex

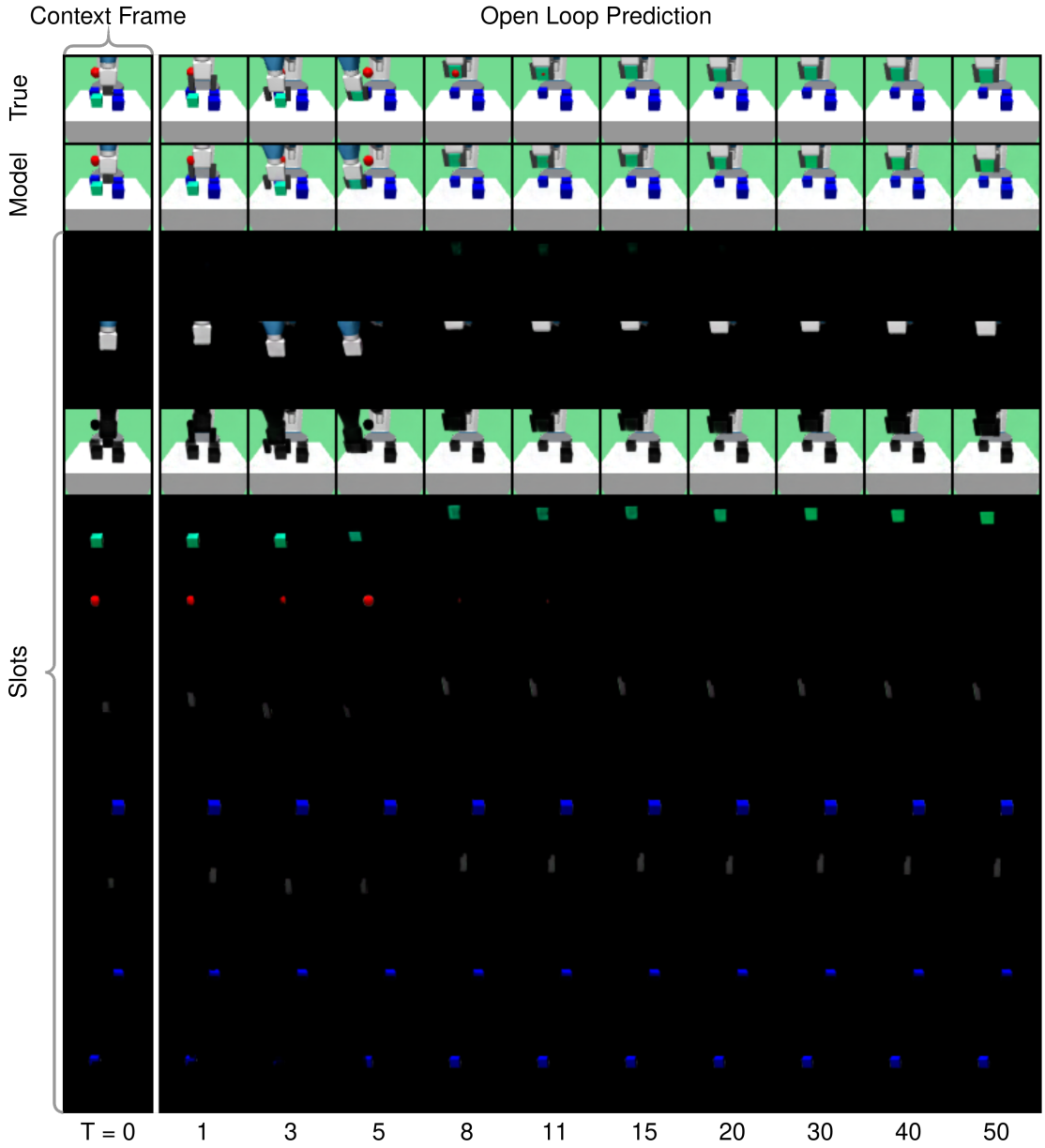


Figure 5.13: Pick-and-Place Distinct (with 3 distractor cubes) open loop prediction roll-out.

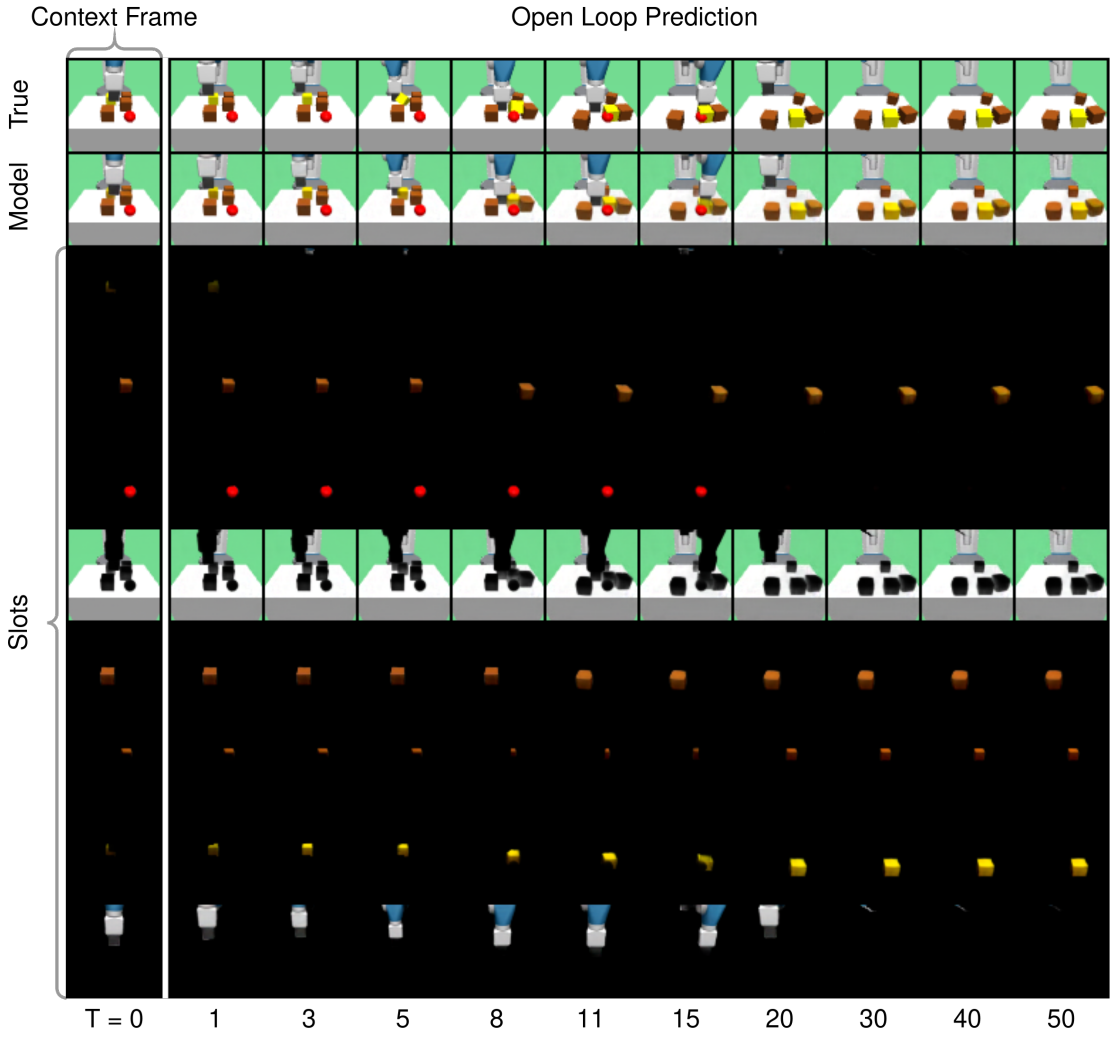


Figure 5.14: Push Distinct (with 3 distractor cubes) open loop prediction rollout.

interactions between multiple cubes across numerous frames. As the robot pushes one block, it may collide with others, setting off a chain of interactions. Our model successfully captures these cascading effects, predicting not just the movement of the directly manipulated object, but also the resulting movements of other objects in the scene.

Attention Visualization

To understand what our trained action model focuses on when selecting actions, we analyzed the attention weights of the output token in the slot processing module instance of the action model. Figures 5.15 and 5.16 provide visualizations of these attention weights for short and long contexts, respectively.

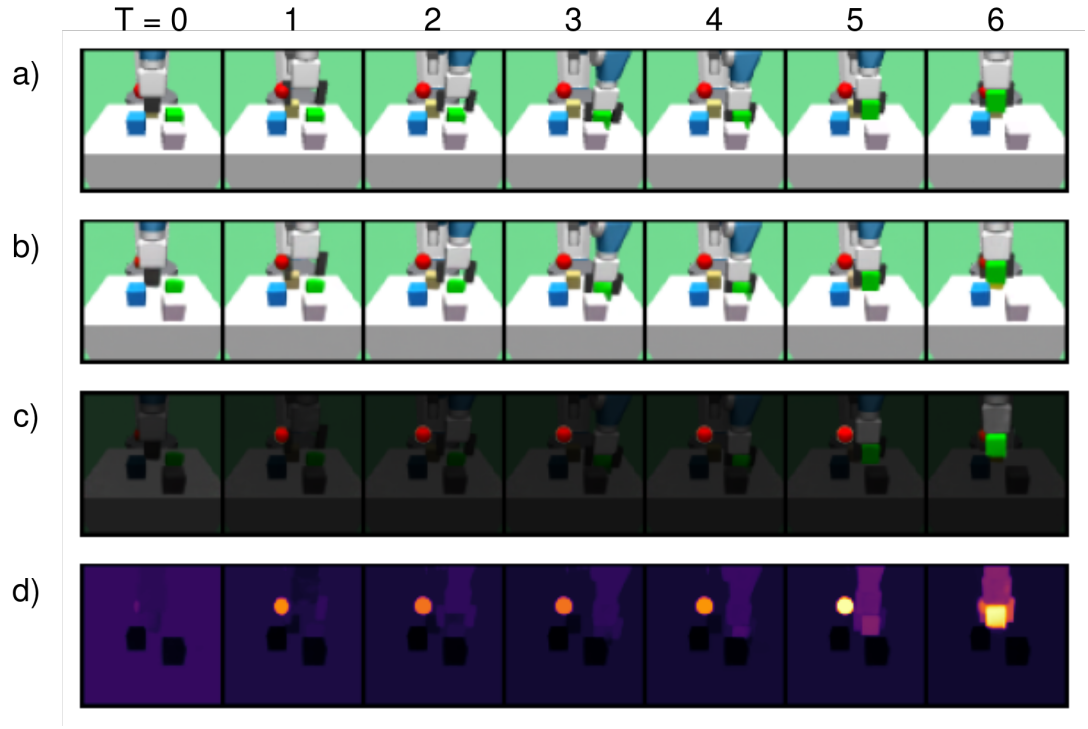


Figure 5.15: Action model attention visualization on Pick-and-Place Specific (3 distractor cubes) with short context. a) shows the original observation sequence. b) shows the reconstruction of the observations by the SAVi model. c) shows the reconstruction where each slot reconstruction is multiplied with its attention weight. c) visualizes the color mapped attention weights.

Figure 5.15 shows the attention weights of the output token in the action model for the Pick-and-Place Specific task at time step 6. It should be noted that the action model can consider the entire observation sequence in the form of the slot set

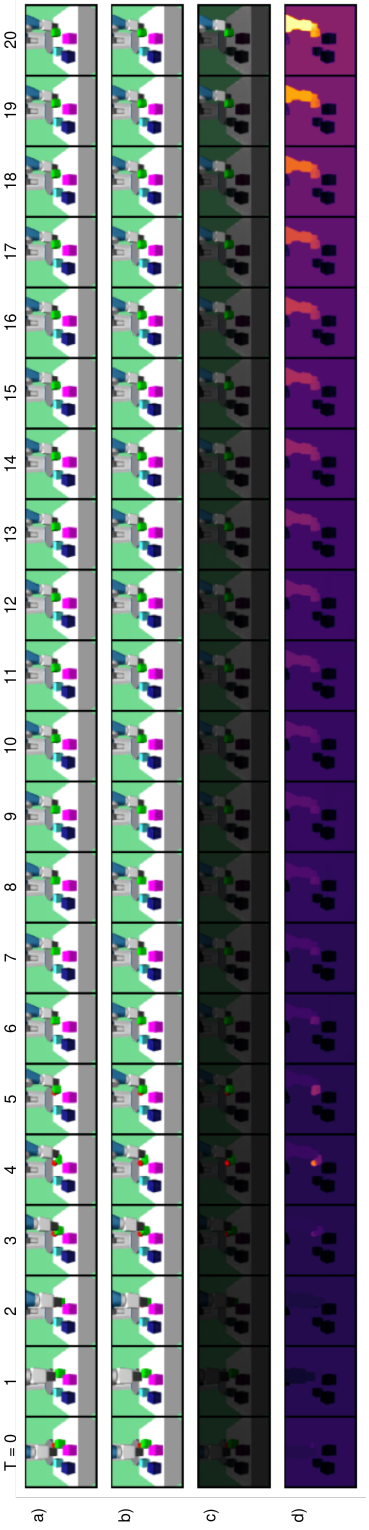


Figure 5.16: Action model attention visualization on Push Specific (3 distractor cubes) with long context. For descriptions on what each row shows see Figure 5.15

sequence in order to make a decision. The visualized attention weights therefore all refer to the output of the action model for time step 6.

The model primarily focuses on the target cube that has already been grasped in the current frame. Despite the target being occluded in the current time step, the model shows robustness by assigning a strong attention weight to the target's slot in the previous time step. Notably, distractor cubes receive very low attention weights, indicating that the model has learned to ignore irrelevant objects for the task at hand.

Figure 5.16 presents attention weights for a longer context in the Push Specific environment, revealing additional insights. In this case, the shown context is even longer than the maximum context of 15 on which the action model is trained during latent imagination. The effect of Attention with Linear Biases (ALiBi) (Press, Smith, and Lewis 2021) is evident, with attention weights decreasing for more temporally distant frames. We intended this recency bias because, in principle, the current environment configuration (in the form of the current observation) should be most important when it comes to selecting the next action. ALiBi also helps the model to select actions on sequences that are longer than sequences that were observed during training.

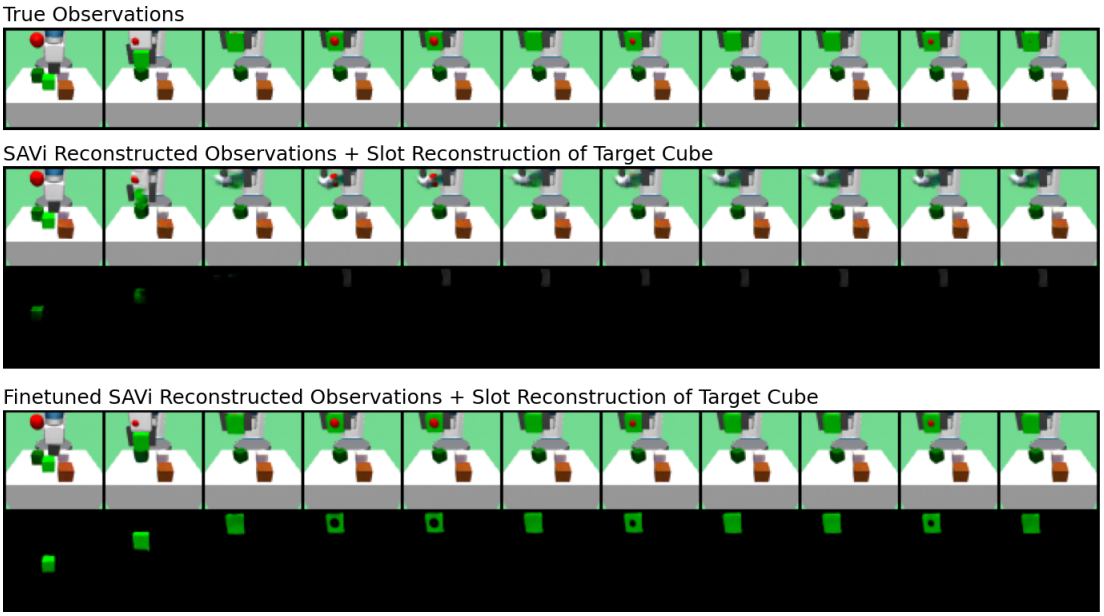
Interestingly, the model overcomes the ALiBi-induced negative attention weight offset to extract information about the target's position when it becomes occluded from time step 4 onwards. This behavior allows the model to infer that the target cube is already correctly positioned despite the occlusion, demonstrating the model's ability to make informed decisions based on past observations.

SAVi fine-tuning

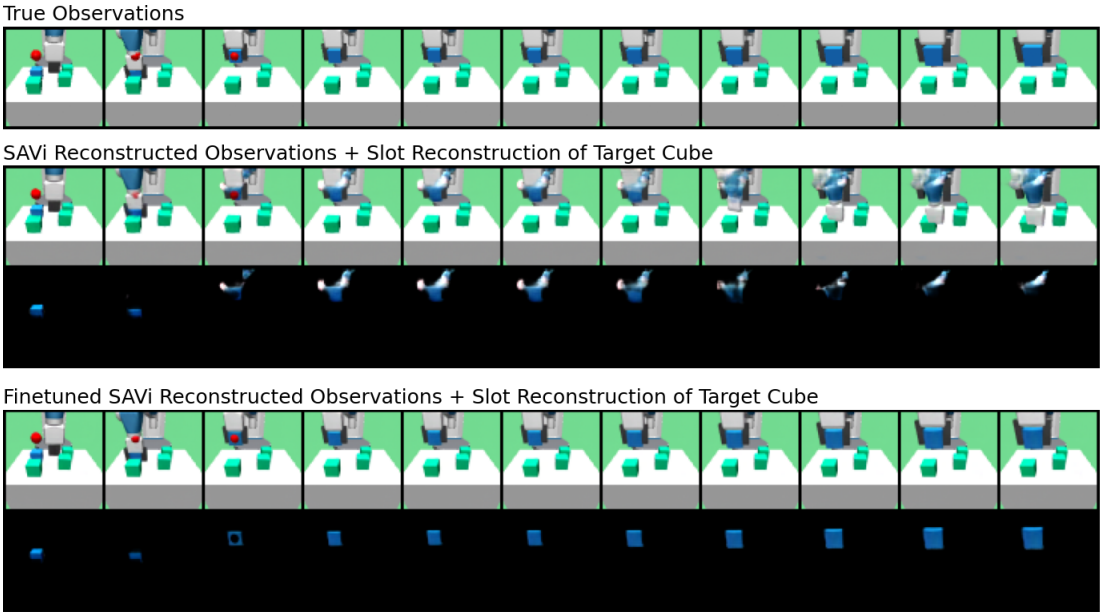
Our experiments demonstrate the importance of fine-tuning the SAVi model during the main reinforcement learning training, particularly in environments where object configurations differ significantly between random and learned behaviors. This is especially evident in the Pick-and-Place tasks, where learned policies frequently lift objects above the table — a situation that is highly unlikely to occur during random exploration.

Figures 5.17a and 5.17b illustrate the impact of SAVi fine-tuning on the Pick-and-Place Specific and Distinct tasks, respectively. The top row in each figure shows the true observations, the middle rows display the reconstructions and slot representation of the target cube from the pre-trained SAVi model, and the bottom rows shows the same for the fine-tuned SAVi model.

In the Pick-and-Place Specific task (Figure 5.17a), we observe that the pre-trained SAVi model struggles to accurately represent the target cube when it is



(a) Pick-and-Place Specific environment.



(b) Pick-and-Place Distinct environment.

Figure 5.17: Impact of SAVi finetuning.

lifted off the table. The slot reconstruction (shown below the full reconstructions) becomes distorted and loses its cubic shape. In contrast, the fine-tuned SAVi model maintains a clear and consistent representation of the cube throughout the lifting action.

The Pick-and-Place Distinct task (Figure 5.17b) further highlights this effect. Here, the pre-trained model not only distorts the shape of the lifted cube but also fails to maintain its color, blending it with the robot arm. The fine-tuned model, however, preserves both the shape and color of the distinct cube accurately, even when it is being manipulated in the air.

These results underscore the necessity of adapting the object-centric representation model to the full range of object configurations encountered during task execution. Without fine-tuning, the world model may fail to accurately represent important states, potentially leading to suboptimal policy learning. The fine-tuned SAVi model demonstrates improved robustness and generalization, enabling more effective learning in complex manipulation tasks where objects undergo significant changes in position and context.

6. Conclusion

In this thesis, we introduced a novel approach to model-based reinforcement learning that leverages object-centric representations to enhance sample efficiency, performance, and interpretability in visual control tasks. By extending the Object-Centric Video Prediction (OCVP) framework of (Villar-Corrales, Wahdan, and Behnke 2023) to function in an action-conditioned manner and integrating it with the Dreamer algorithm (Hafner, T. Lillicrap, J. Ba, et al. 2019; Hafner, T. Lillicrap, Norouzi, et al. 2020; Hafner, Pasukonis, et al. 2023), we developed, to the best of our knowledge, the first object-centric model-based reinforcement learning method capable of solving continuous control tasks using only visual input.

Our extensive experiments on a suite of simulated robotic manipulation tasks demonstrated the effectiveness of our approach. The object-centric model consistently outperformed both a non-object-centric baseline with significantly more parameters and the state-of-the-art DreamerV3 algorithm, particularly in tasks requiring relational reasoning between objects. The improved sample efficiency and rapid learning exhibited by our method highlight its potential for applications where data collection is costly or time-consuming. Further, we showed that object-centric models are capable of adjusting to the changing state-distribution characteristic of many non-trivial reinforcement learning problems, thereby overcoming one of the key limitations of prior methods. Overall, we have demonstrated that learning object-centric representations from pixels is a viable paradigm for model-based reinforcement learning, thereby opening up many interesting avenues for future research.

One key area for improvement is the incorporation of stochastic state representations, similar to those used in the Dreamer family of algorithms. Currently, our model uses deterministic representations, which may limit its ability to capture uncertainty in the environment. By introducing stochasticity into the state representations, we could potentially improve the model's robustness in partially observable or stochastic environments. This modification would allow the model to maintain multiple hypotheses about the current state, leading to more nuanced and flexible decision-making.

Another promising direction for future work is the development of object-centric exploration techniques. Drawing inspiration from the FOCUS approach (Ferraro

et al. 2023), we could design exploration strategies that specifically encourage the agent to interact with different objects or explore novel object configurations. This object-centric approach to exploration could be particularly beneficial in environments with a diverse range of objects, as it would naturally guide the agent to learn about the properties and interactions of various entities in its environment. Such a technique could significantly improve the efficiency of learning, especially in the early stages of training when the agent has limited knowledge about its environment.

It is worth noting that advances in the field of learning object-centric representations could directly benefit our method. As our approach uses the Slot Attention for Video (SAVi) framework (Kipf et al. 2021) as a key component, improvements in object-centric representation learning could be integrated as a drop-in replacement for SAVi. This modularity allows our method to benefit from future developments in object-centric learning without requiring significant architectural changes. As research in this area progresses, we can expect to see improvements in the quality and robustness of object representations, which could translate into better performance and generalization capabilities for our reinforcement learning approach.

In conclusion, this work has demonstrated the potential of combining object-centric representations with model-based reinforcement learning for visual control tasks. By providing a structured and interpretable way to reason about objects and their interactions, our approach opens up new possibilities for developing more efficient and capable reinforcement learning agents. As we continue to advance in this direction, incorporating stochastic state representations, object-centric exploration techniques, and leveraging improvements in object-centric representation learning, we move closer to creating AI systems that can understand and interact with the world in ways that more closely resemble human cognition.

A. Additional Experiments

A.1. Finger Spin (DM Control)

To evaluate the versatility of our object-centric model-based reinforcement learning approach, we conducted preliminary experiments on environments that are not inherently object-centric. Specifically, we chose the Finger Spin task from the DeepMind Control Suite (Tassa et al. 2018). This selection was motivated by the fact that the Dreamer family of algorithms (Hafner, T. Lillicrap, J. Ba, et al. 2019; Hafner, T. Lillicrap, Norouzi, et al. 2020; Hafner, Pasukonis, et al. 2023) has been extensively evaluated on this suite, potentially allowing for a fair comparison.

While time constraints prevented a comprehensive study, our initial results are promising and offer insights into the capabilities and limitations of our approach. Figure A.1 demonstrates that our action-conditioned object-centric video prediction performs well on this task, accurately capturing the dynamics of the finger and the spinning object. This suggests that our model can effectively learn and predict the behavior of systems even when they are not explicitly designed with multiple distinct objects.

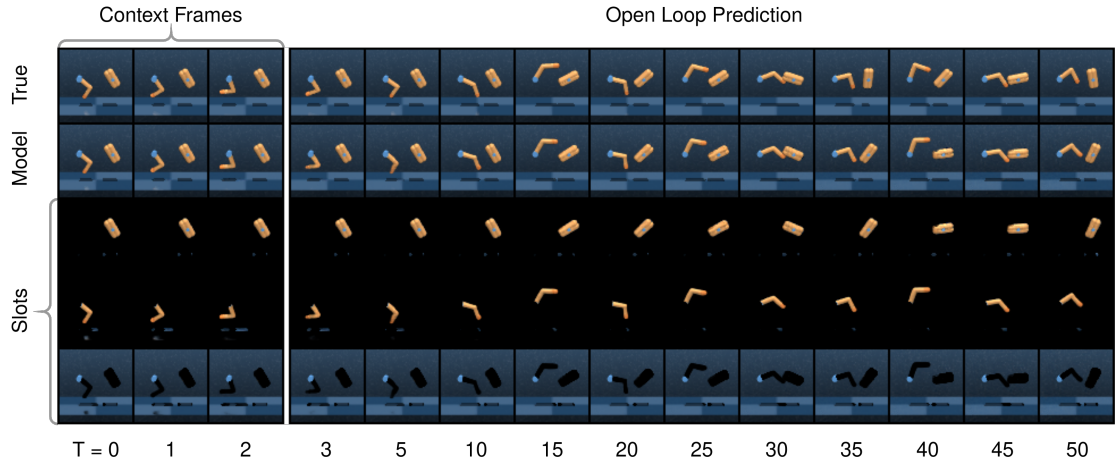


Figure A.1: Finger Spin open loop prediction roll-out.

Furthermore, as shown in Figure A.2, our model achieved notably good returns on the Finger Spin task. This performance indicates that our approach can suc-

cessfully learn effective policies for control tasks that do not necessarily require reasoning about multiple interacting objects.

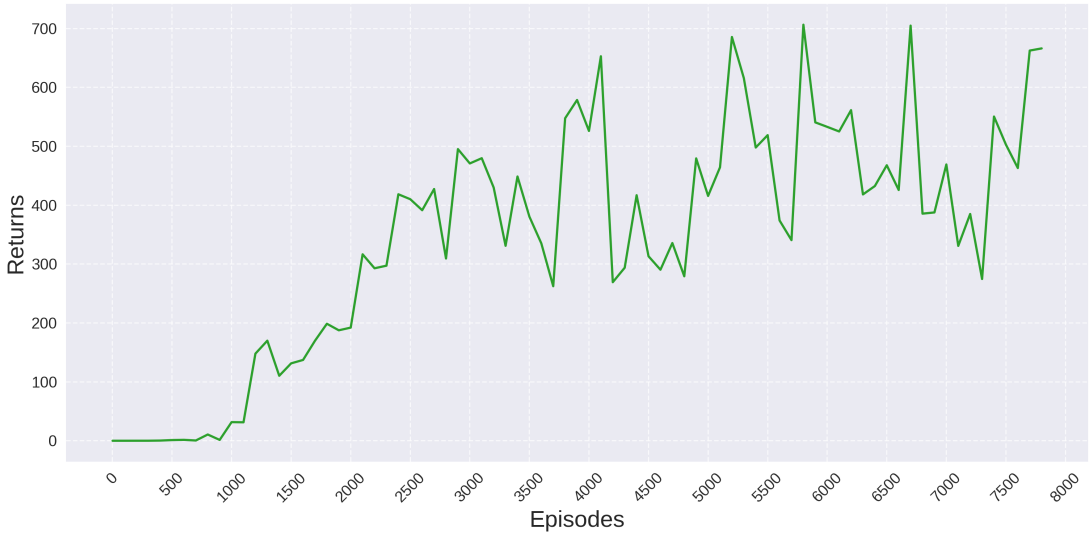


Figure A.2: Finger Spin returns received on test episodes during training.

However, it is important to note that our method appears to be less sample efficient than DreamerV3 (Hafner, Pasukonis, et al. 2023) in this context. This observation aligns with our expectation that the advantages of our object-centric approach may be less pronounced in tasks that do not involve complex multi-object interactions or reasoning.

A.2. Meta-World

To further evaluate the versatility of our object-centric model-based reinforcement learning approach, we conducted preliminary experiments on selected tasks from the Meta-World benchmark (Yu et al. 2020). This evaluation aimed to demonstrate the efficacy of our method in manipulating objects beyond the unicolored cubes commonly used in related object-centric RL research (Haramati, Daniel, and Tamar 2024; Yoon et al. 2023; Zadaianchuk, Seitzer, and Martius 2020). Meta-World provides a diverse array of robotic manipulation tasks, offering a suitable testbed for this kind of evaluation. Our investigation focused on two specific tasks: Press Button and Hammer.

A.2.1. Press Button

The Press Button task requires the agent to manipulate a robotic arm to press a button on a box. This task tests the agent's ability to perform precise movements and interact with small objects in a 3D environment.

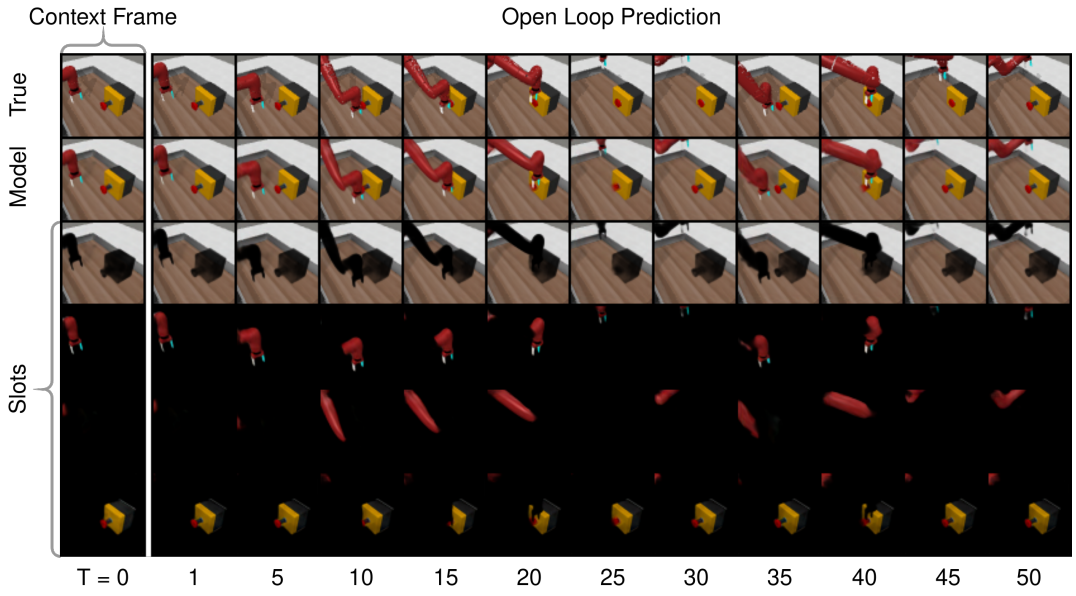


Figure A.3: Press Button open loop prediction roll-out.

Figure A.3 illustrates our model's capability to decompose the Press Button environment and predict its dynamics. The predictions accurately capture the robotic arm's movements, demonstrating some grasp of inverse kinematics, and its interaction with the button.

Our approach achieved significant success in this task, reaching a success rate of 91.2% over 1000 test episodes after 20,000 training episodes. An episode is considered successful if the button is pressed at least once during the episode. Figure A.4 illustrates the learning progress, showing an increase in returns as training progresses.

A.2.2. Hammer

The Hammer task presents a more complex challenge, requiring the agent to grasp a hammer and use it to drive a nail into a wall. This task tests the agent's ability to manipulate tools and perform multi-step actions.

Figure A.5 showcases our model's predictions for the Hammer task. As observed in the Press Button task, the model successfully decomposes the environment and

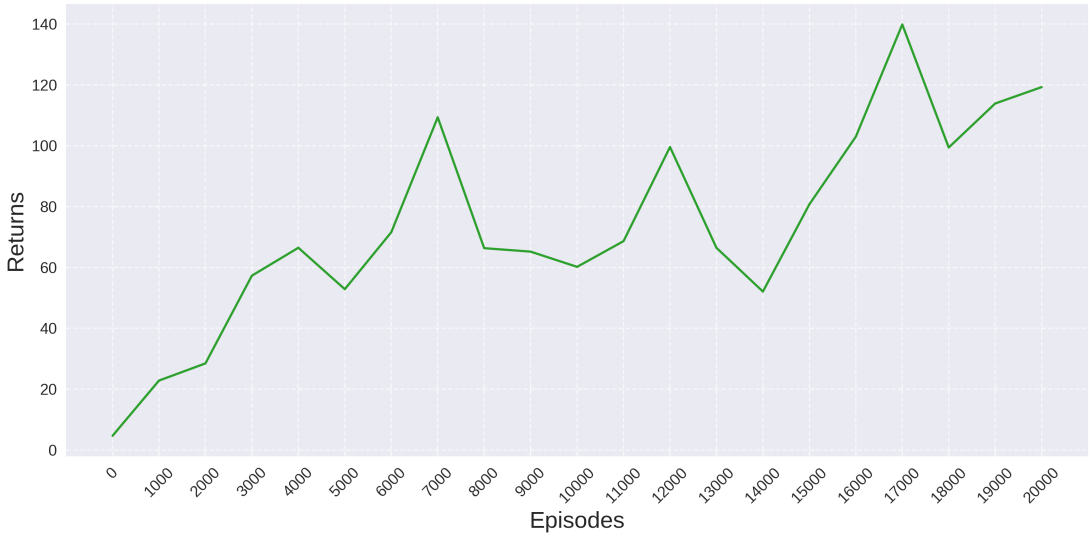


Figure A.4: Press Button returns received on test episodes during training.

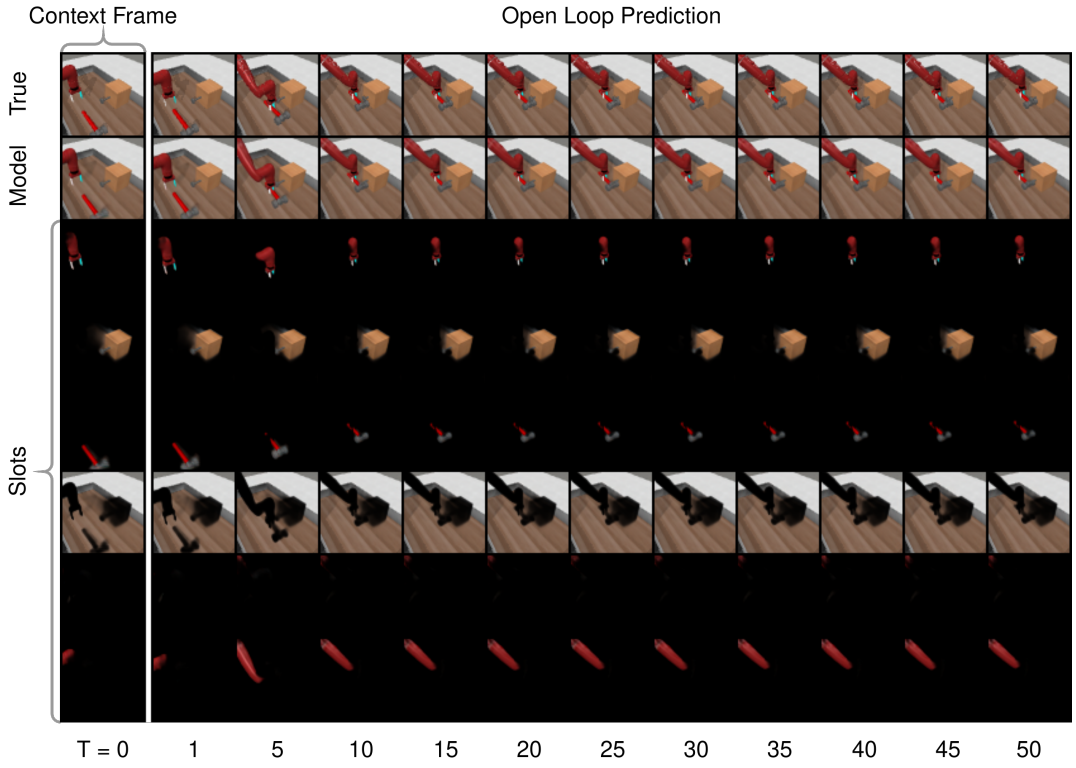


Figure A.5: Hammer open loop prediction roll-out.

demonstrates an understanding of the robot arm's inverse kinematics. Moreover, it accurately predicts the hammer's movement following interaction with the robot arm.

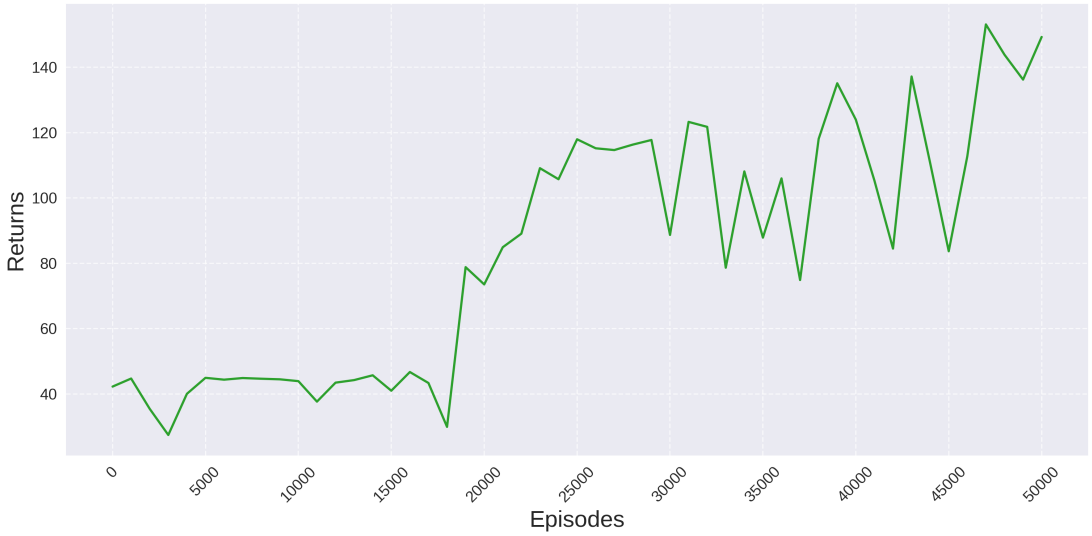


Figure A.6: Hammer returns received on test episodes during training.

Despite accurate predictions, our approach achieved a success rate of only 0.7% over 1000 test episodes after 50,000 training episodes on the Hammer task. Here, success is defined as driving the nail more than 9 cm into the wall. This limited success may be attributed to the agent learning to grip and move the hammer towards the nail (as evidenced in Figure A.5), but failing to consistently strike the nail. Extended training periods, which were beyond the scope of this thesis, might address this limitation. Nevertheless, Figure A.6 indicates that the agent continues to improve its performance over time, even if this improvement has not yet translated to a high success rate.

These experiments demonstrate that our object-centric approach can effectively model and predict dynamics in environments featuring diverse objects, extending beyond the manipulation of simple unicolored cube-like objects prevalent in related work (Haramati, Daniel, and Tamar 2024; Yoon et al. 2023; Zadaianchuk, Seitzer, and Martius 2020). This underscores that the use of unsupervised learned object-centric representations (in our case, utilizing SAVi (Kipf et al. 2021)) does not constrain our method to manipulating only visually simple objects such as unicolored cubes. Instead, it showcases the potential of our approach to generalize to more complex, realistic manipulation tasks.

List of Figures

List of Tables

List of Algorithms

Bibliography

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). "Layer normalization." In: *Arxiv preprint arxiv:1607.06450*.

Barth-Maron, Gabriel, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap (2018). "Distributed distributional deterministic policy gradients." In: *Arxiv preprint arxiv:1804.08617*.

Bellemare, M. G., Y. Naddaf, J. Veness, and M. Bowling (2013). "The arcade learning environment: an evaluation platform for general agents." In: *Journal of artificial intelligence research* 47, pp. 253–279.

Bellemare, Marc G., Will Dabney, and Rémi Munos (2017). "A distributional perspective on reinforcement learning." In: *International conference on machine learning*. PMLR, pp. 449–458.

Bellman, Richard (1957). *Dynamic programming*. Princeton: Princeton University Press.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "Openai gym." In: *Arxiv preprint arxiv:1606.01540*.

Daniel, Tal and Aviv Tamar (2022). "Unsupervised image representation learning with deep latent particles." In: *Arxiv preprint arxiv:2205.15821*.

Darcet, Timothée, Maxime Oquab, Julien Mairal, and Piotr Bojanowski (2023). "Vision transformers need registers." In: *Arxiv preprint arxiv:2309.16588*.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. (2020). "An image is worth 16x16 words: transformers for image recognition at scale." In: *Arxiv preprint arxiv:2010.11929*.

Duan, Yan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel (2016). "Benchmarking deep reinforcement learning for continuous control." In: *International conference on machine learning*. PMLR, pp. 1329–1338.

Ferraro, Stefano, Pietro Mazzaglia, Tim Verbelen, and Bart Dhoedt (2023). "Focus: object-centric world models for robotics manipulation." In: *Arxiv preprint arxiv:2307.02427*.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor." In: *International conference on machine learning*. PMLR, pp. 1861–1870.

Hafner, Danijar, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi (2019). "Dream to control: learning behaviors by latent imagination." In: *Arxiv preprint arxiv:1912.01603*.

Hafner, Danijar, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson (2019a). *Learning latent dynamics for planning from pixels*. URL: `https://planetrl.github.io/` (visited on 07/24/2024).

– (2019b). "Learning latent dynamics for planning from pixels." In: *International conference on machine learning*. PMLR, pp. 2555–2565.

Hafner, Danijar, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba (2020). "Mastering atari with discrete world models." In: *Arxiv preprint arxiv:2010.02193*.

Hafner, Danijar, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap (2023). "Mastering diverse domains through world models." In: *Arxiv preprint arxiv:2301.04104*.

Haramati, Dan, Tal Daniel, and Aviv Tamar (2024). "Entity-centric reinforcement learning for object manipulation from pixels." In: *Arxiv preprint arxiv:2404.01220*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition." In: *Proceedings of the ieee conference on computer vision and pattern recognition*, pp. 770–778.

Hendrycks, Dan and Kevin Gimpel (2016). "Gaussian error linear units (gelus)." In: *Arxiv preprint arxiv:1606.08415*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural computation* 9.8, pp. 1735–1780.

Jiang, Jindong, Sepehr Janghorbani, Gerard De Melo, and Sungjin Ahn (2019). "Scalor: generative world models with scalable object representations." In: *Arxiv preprint arxiv:1910.02384*.

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). "Planning and acting in partially observable stochastic domains." In: *Artificial intelligence* 101.1-2, pp. 99–134.

Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore (1996). "Reinforcement learning: a survey." In: *Journal of artificial intelligence research* 4, pp. 237–285.

Kingma, Diederik P. (2014). "Adam: a method for stochastic optimization." In: *Arxiv preprint arxiv:1412.6980*.

Kipf, Thomas, Gamaleldin F. Elsayed, Aravindh Mahendran, Austin Stone, Sara Sabour, Georg Heigold, Rico Jonschkowski, Alexey Dosovitskiy, and Klaus Greff (2021). "Conditional object-centric learning from video." In: *Arxiv preprint arxiv:2111.12594*.

Kirillov, Alexander, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. (2023). "Segment anything." In: *Proceedings of the ieee/cvf international conference on computer vision*, pp. 4015–4026.

Konda, Vijay and John Tsitsiklis (1999). "Actor-critic algorithms." In: *Advances in neural information processing systems* 12.

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016). "End-to-end training of deep visuomotor policies." In: *Journal of machine learning research* 17.39, pp. 1–40.

Li, Richard, Allan Jabri, Trevor Darrell, and Pulkit Agrawal (2020). "Towards practical multi-object manipulation using relational reinforcement learning." In: *Icra*.

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015). "Continuous control with deep reinforcement learning." In: *Arxiv preprint arxiv:1509.02971*.

Locatello, Francesco, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf (2020). "Object-centric learning with slot attention." In: *Advances in neural information processing systems* 33, pp. 11525–11538.

Loshchilov, Ilya and Frank Hutter (2016). "Sgdr: stochastic gradient descent with warm restarts." In: *Arxiv preprint arxiv:1608.03983*.

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning." In: *International conference on machine learning*. PMLR, pp. 1928–1937.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing atari with deep reinforcement learning." In: *Arxiv preprint arxiv:1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning." In: *Nature* 518.7540, pp. 529–533.

Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar (2018). *Foundations of machine learning*. MIT press.

Mosbach, Malte and Sven Behnke (2024). "Grasp anything: combining teacher-augmented policy gradient learning with instance segmentation to grasp arbitrary objects." In: *Arxiv preprint arxiv:2403.10187*.

Pavlichenko, Dmytro and Sven Behnke (2023). "Deep reinforcement learning of dexterous pre-grasp manipulation for human-like functional categorical grasping." In: *2023 ieee 19th international conference on automation science and engineering (case)*. IEEE, pp. 1–8.

Press, Ofir, Noah A. Smith, and Mike Lewis (2021). "Train short, test long: attention with linear biases enables input length extrapolation." In: *Arxiv preprint arxiv:2108.12409*.

Rubinstein, Reuven Y. (1997). "Optimization of computer simulation models with rare events." In: *European journal of operational research* 99.1, pp. 89–112.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning internal representations by error propagation, parallel distributed process-

ing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986." In: *Biometrika* 71.599-607, p. 6.

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. (2020). "Mastering atari, go, chess and shogi by planning with a learned model." In: *Nature* 588.7839, pp. 604–609.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play." In: *Science* 362.6419, pp. 1140–1144.

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (2014). "Deterministic policy gradient algorithms." In: *International conference on machine learning*. PMLR, pp. 387–395.

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement learning: an introduction*. MIT press.

Sutton, Richard S., David McAllester, Satinder Singh, and Yishay Mansour (1999). "Policy gradient methods for reinforcement learning with function approximation." In: *Advances in neural information processing systems* 12.

Tassa, Yuval, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. (2018). "Deepmind control suite." In: *Arxiv preprint arxiv:1801.00690*.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "Mujoco: a physics engine for model-based control." In: *2012 ieee/rsj international conference on intelligent robots and systems*. IEEE, pp. 5026–5033.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need." In: *Advances in neural information processing systems* 30.

Villar-Corrales, Angel, Ismail Wahdan, and Sven Behnke (2023). "Object-centric video prediction via decoupling of object dynamics and interactions." In: *2023 ieee international conference on image processing (icip)*. IEEE, pp. 570–574.

Watkins, Christopher John Cornish Hellaby (1989). *Learning from delayed rewards*. King's College, Cambridge United Kingdom.

Watkins, Christopher John Cornish Hellaby and Peter Dayan (1992). "Q-learning." In: *Machine learning* 8, pp. 279–292.

Webber, J. Beau W. (2012). "A bi-symmetric log transformation for wide-range data." In: *Measurement science and technology* 24.2, p. 027001.

Williams, Ronald J. and Jing Peng (1991). "Function optimization using connectionist reinforcement learning algorithms." In: *Connection science* 3.3, pp. 241–268.

Yoon, Jaesik, Yi-Fu Wu, Heechul Bae, and Sungjin Ahn (2023). "An investigation into pre-training object-centric representations for reinforcement learning." In: *Arxiv preprint arxiv:2302.04419*.

Yu, Tianhe, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine (2020). "Meta-world: a benchmark and evaluation for multi-task and meta reinforcement learning." In: *Conference on robot learning*. PMLR, pp. 1094–1100.

Zadaianchuk, Andrii, Maximilian Seitzer, and Georg Martius (2020). "Self-supervised visual reinforcement learning with object-centric representations." In: *Arxiv preprint arxiv:2011.14381*.

Zhang, Biao and Rico Sennrich (2019). "Root mean square layer normalization." In: *Advances in neural information processing systems* 32.

Ziebart, Brian D. (2010). *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University.