Unsupervised Object-Based Transition Models For 3D Partially Observable Environments

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present a slot-wise, object-based transition model that decomposes a scene into objects, aligns them (with respect to a slot-wise object memory) to maintain a consistent order across time, and predicts how those objects evolve over successive frames. The model is trained end-to-end without supervision using transition losses at the level of the object-structured representation rather than pixels. Thanks to the introduction of our novel alignment module, the model deals properly with two issues that are not handled satisfactorily by other transition models, namely object persistence and object identity. We show that the combination of an object-level loss and correct object alignment over time enables the model to outperform a state-of-the-art baseline, and allows it to deal well with object occlusion and re-appearance in partially observable environments.

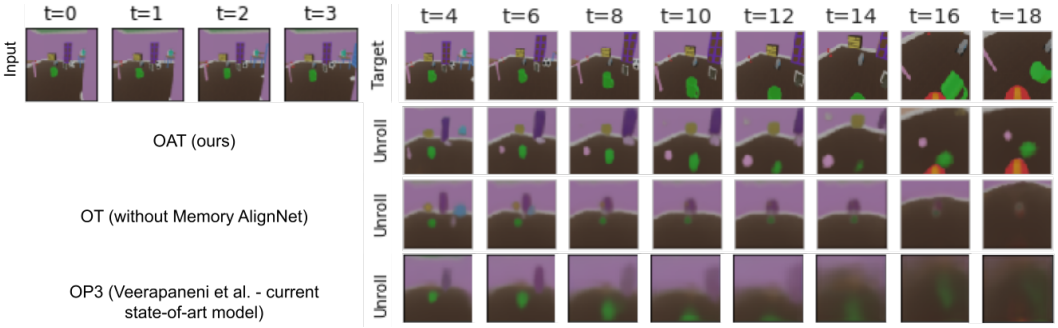

Figure 1: **Comparing our model to baselines.** Each model was trained with four input steps and to unroll for six steps, here we unroll for 15 steps. Our model, OAT, performs significantly better than OT (our model without the AlignNet) and current state-of-the-art model, OP3 [31]. See Figure 17 in the Appendix for additional OP3 roll-outs.

1 Introduction

In spite of their well-documented ability to learn complex tasks, today's deep reinforcement learning agents are still far from matching humans at out-of-distribution generalisation or few-shot transfer [6, 20, 22]. Two architectural features commonly proposed to remedy this are (1) transition models that enable the agent to internally explore paths through state space that it has never experienced [28, 24, 11], and (2) compositionally structured representations that enable the agent to represent meaningful states that it has never encountered [7]. These two features are not exclusive; transition models that operate on compositionally structured representations are a potent combination, and these are the subject of the present paper. Specifically, our focus is on transition models that operate at the

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

level of *objects*, which are the most obvious candidates for the structural elements of representations likely to be useful for artificial agents inhabiting 3D worlds such as our own [27].

While there has been progress in object-based transition models [33, 31, 34, 19], current models do not deal satisfactorily with *object persistence* (the concept that objects typically continue to exist when they are no longer perceptible [23]) or with *object identity* (the concept that a token object at one time-step is the same token object at a later time-step [1]). As we show, transition models that neglect object persistence tend to perform badly in complex, partially observable environments, while models that neglect object identity are unable to integrate information about a single object (and its interactions) over time in a way that generalises to future time-steps. By proposing and incorporating an novel module for aligning objects across time using a slot-based memory, our model handles both these concepts, and exhibits better performance as a result.

An important feature of our transition model is that it makes predictions and computes losses in a representation space that is divided into objects. This contrasts with existing models that make predictions in an unstructured representation space [12, 11]. Making predictions and computing losses in an object-structured representation space facilitates learning, not only because the representation space is lower dimensional than pixel space, but also because the model can exploit the fact that dynamics tend to apply to objects as a whole, which simplifies learning. However, to compute prediction losses directly over distinct object representations (rather than first mapping predictions back to pixels [33, 31]), the objects in a predicted representation must be matched with those in the target representation, which again requires a proper treatment of object persistence and identity. The result is a model that can roll-out accurate predictions for significantly more steps than those seen during training, outperforming state-of-the-art for comparable models.

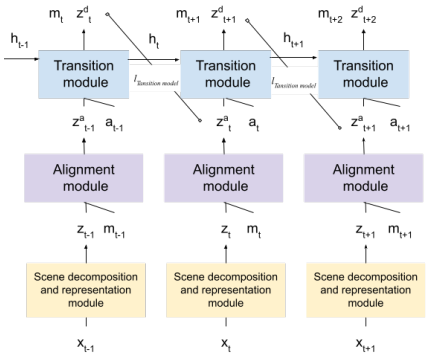

Figure 2: **Encoding steps.** The *scene decomposition and representation module* extracts K object representations, $z_t \in \Re^{K \times F}$, from images, $x_t \in \Re^{W \times H \times 3}$. These are aligned using the *alignment module* to obtain, $z_t^a \in \Re^{M \times F}$. Aligned objects are fed to the slot-wise *transition model* with the action, a_t, and hidden state, $h_t (= h_t^a) \in \Re^{M \times H}$, to predict the object representations at the next time-step, $z_t^d \in \Re^{M \times F}$ as well as the updated memory, $m_{t+1} \in \Re^{M \times F}$. On this figure we also demonstrate how the transition model loss is computed between z_t^d and z_t^a.

To achieve this, our model, Objects-Align-Transition (OAT), combines (1) a *scene decomposition and representation module*, MONet [2], that transforms a raw image into a slot-wise object-based representation, (2) an novel *alignment module* which, with the aid of a slot-wise memory, ensures that each object is represented in the same slot across time, even if it has temporarily disappeared from view, and (3) a novel slot-wise *transition model* that operates on the object representations to predict future states. All three components are differentiable, and the whole model is trained end-to-end without supervision. We evaluate the model on two sequential image datasets. The first is collected from a pre-trained agent moving in a simulated 3D environment, where we show that our model outperforms the current state-of-the-art object-based transition model (OP3 [31]). The second dataset is collected from a real robot arm interacting with various physical objects, where we demonstrate accurate roll-outs over significantly longer periods than used in training. An ablation study shows that the model's success depends on the combination of correct object alignment across time and the use of a transition loss over object-level representations instead of over pixels.

2 Our Model: Objects-Align-Transition

Our model, Objects-Align-Transition (OAT), combines a scene decomposition and representation module, in this case MONet, with our slot-wise transition module, via our alignment module, which ensures that each slot in the transition model receives objects corresponding to the same token object (or identity) across time. The whole model is trained end-to-end. We now provide details of each of these modules (see Figure 2).

2.1 Scene decomposition and representation module

The input to OAT is a sequence of RGB images, $x \in [0,1]^{T \times W \times H \times 3}$, with T time-steps, width, W, and height, H. We leave out the batch dimensions for simplicity. Each image in the sequence, x_t, is passed through a scene decomposition and representation module[1], in this case MONet, to obtain a slot-wise object representation, $[z_{t,k}]_{1:K}$, occupying K slots, where $z_{t,k} \in \Re^F$ is an object representation vector with F features.

MONet consists of an attention module (a U-net [26]) — which predicts object segmentation masks, $\mu_{t,k} \in [0,1]^{T \times W \times H \times 1}$, for each slot — and a slot-wise VAE. The encoder of the slot-wise VAE is fed the predicted object segmentation masks and the input image, x_t, and outputs object representations, $z_t \in \Re^{K \times F}$. The decoder reconstructs the masks, $\tilde{\mu}_{k,t}$, and each object's pixels, $\tilde{x}_{t,k}$. MONet's computations can be summarised by the following equations: $\mu_{k,t} = \text{Attention_Network}(x_t)$, $z_{k,t} = \text{MONet_encoder}(x_t, \mu_{k,t})$ and $\tilde{\mu}_{k,t}, \tilde{x}_{t,k} = \text{MONet_decoder}(z_k)$.

It may be tempting to feed the slot-wise object representations, $[z_{t,k}]_{k=1:K}$, at time, t, directly to a slot-wise transition model. However, MONet's representations [2] are *not stable* across time, meaning that a specific object may appear in different slots across time [34] (see Figure 3). This leads to two major problems when training slot-wise transition models on slot-wise object-based representations. Firstly, it is difficult to compute losses between predicted and target object slots. Computing a slot-wise *object-level loss* requires us to know how the previous object representations (and thus the predicted object representations) correspond with the target object representations. Secondly, if object representations do not occupy consistent slots it makes it harder to integrate information about a single object (and its interactions with other objects) across time [34], and makes it harder to predict the reappearance of that object (as show in Section 4.2). To address the issue of slot stability we introduce AlignNet, an alignment module.

2.2 Alignment module

Our alignment module, Memory AlignNet, must play two key roles. The first is to align objects across time, enabling us to compute slot-wise object-level losses for training. The second is to learn a slot-wise memory that encodes the history of each slot across time, allowing our transition model to operate effectively in partially observable environments by keeping tack of objects as they go in and out of view. This is especially important for embodied agents that take actions in 3D environments, where the agent's looking around frequently causes objects to move in and out of its field-of-view.

We propose Memory AlignNet, which takes a slot-wise memory, $m_t \in \Re^{M \times F}$, with $M \geq K$ slots and the (stacked) output of the scene decomposition and representation module, $z_t \in \Re^{K \times F}$, returning the aligned object representations, $z_t^a \in \Re^{M \times F}$. To perform alignment, the Memory AlignNet predicts an adjacency matrix, $A_t \in \Re^{M \times K}$, that aligns objects, z_t, with the memory, m_t. This adjacency matrix allows us to compute a *soft* alignment $z^{a,soft} = A_t z_t$ or a hard alignment, $z^a \triangleq z^{a,hard} = \text{Hungarian}(A_t) z_t$. We will refer to $z_t^{a,hard}$ as z_t^a throughout the rest of the paper. Hungarian$(\cdot)$ denotes the application of the Hungarian algorithm, a non-differentiable algorithm which computes a permutation matrix given an adjacency matrix. The soft version of the alignment is used to train the Memory AlignNet while the hard version of the alignment is passed to the slot-wise transition model, which has the same number of slots, M, as the memory. Further details in Section A.2 of the Appendix.

The hard aligned objects may also be used to update the memory using a recurrent slot-wise model. For simplicity we use our transition model, $\mathcal{T}_\theta(\cdot)$, to predict deltas for both the object representations, z_t^a, and the memory, m_t. We will detail this in the next section.

2.3 Slot-wise Transition module

The transition model operates in both an encoding and an unrolling phase. During the encoding phase the transition model is fed aligned, observed object representations, z_t^a, and actions, a_t, to predict aligned object representations at the next time-step, $z_{t+1}^d \in \Re^{M \times F}$. During the unroll phase, the

[1] We tried some variants where z_t depended on z_{t-1}. However these did not yield significantly beneficial results.

transition model is fed the predictions, z_t^d, and actions, a_t, from the current time-step to predict the object representations, z_{t+1}^d, at the next time-step.

More concretely in the **encoding** steps, our transition model, $\mathcal{T}_\theta(\cdot)$, takes aligned object representations, z_t^a, and a hidden state, $h_t^a \in \Re^{M \times H}$, and predicts deltas for both the object representations, z_t^a, and memory, m_t. Concretely, the output of our transition model is given by, $[\Delta_t^a, \Delta_t^m], h_{t+1}^a = \mathcal{T}_\theta(z_t^a, a_t, h_t^a)$. The memory and the object representations are then updated as follows: $m_{t+1} = m_t + \Delta_t^m$ and $z_{t+1}^d = z_t^a + \Delta_t^a$. In the **unroll** steps, next step predictions are given by $z_{t+1}^d = z_t^d + \Delta_t^d$ where $[\Delta_t^d, \Delta_t^m], h_{t+1}^a = \mathcal{T}_\theta(z_t^d, a_t, h_t^a)$. By using the transition model to predict deltas, z_t^d is aligned by default and does not need to be re-aligned when unrolling the model.

A key feature of our transition model is that weights are shared across object slots and can be instantiated in many different ways. One simple option is to use a *SlotLSTM*; an LSTM applied independently to each slot, sharing weights between slots. An alternative, novel instantiation which we found to work well is to first apply a transformer [32, 30] and then the SlotLSTM (see Figure 10 and Section A.3). This allows the model to capture interactions with other objects (via the transformer) and integrate that information over time (via the SlotLSTM). Further details in Section A.2 of the Appendix.

2.4 Training

When training OAT we jointly learn the parameters of the scene decomposition and representation module, the alignment module and the transition module. Each module has its own losses that contribute to downstream gradients and updates. For example, both the transition model and the alignment module losses can influence the scene decomposition and representation module's updates. Let us define the losses for each module.

Scene decomposition and representation module losses (MONet). We train MONet using standard MONet losses, l_{MONet}, see Equation 3 in [2]. The first term in the MONet loss is a scene reconstruction term. This is a spatial mixture loss parameterised by σ_{bg} and σ_{fg}, which are the standard deviation used for each slot's component likelihood (for the first slot and remaining slots, respectively) that go into the mixture loss. The remaining terms are regularisation terms that 1) induce a latent information bottleneck needed for good representation learning (scaled by β, the latent KL loss weight), and 2) ensure that predicted and target masks are similar (scaled by γ, the mask KL loss weight). MONet losses are computed only in the encoding phase and are not used in the unroll phase. Additionally, MONet loss gradients do not affect the transition module or the alignment module weights.

Alignment module losses (Memory AlignNet). The memory AlignNet consists of a reconstruction loss between the softly aligned object representations, $z_t^{a,soft}$ and the output of the transition model, z_t^d for the corresponding time-step. There are also regularisation losses including an entropy loss, $\mathbb{H}(\cdot)$, on the adjacency matrix, A_t, which encourages values towards zero and one; and a loss that penalises columns that sum to more that one, avoiding the case where multiple objects are assigned to the same memory slot. The AlignNet loss, l_{AlignNet} is defined as $l_{\text{AlignNet}} = \sum_{t=1}^{T} ||z_t^d - z_t^{a,soft}||_2^2 + \psi \mathbb{H}(A_t) + \sum_{j=1}^{M} \max(0, (\sum_{k=1}^{K} A_{t,k,j} - 1))$, where T is the total number of encoding and unroll steps. We use $\psi = 0.01$ for all experiments presented in this paper.

Slot-wise transition module losses. The slot-wise transition model is trained with a reconstruction loss between the outputs of the transition model, z_t^d, and the aligned observations for the same time-step, z_t^a. The slot-wise transition model loss, $l_{\text{Transition model}}$ is defined as $l_{\text{Transition model}} = \sum_{t=1}^{T} ||z_t^d - z_t^a||_2^2$. Importantly, we compute a loss directly between object representations without the need for decoding them to obtain pixels.

OAT is trained end-to-end to minimise $l_{\text{MONet}} + l_{\text{AlignNet}} + \zeta l_{\text{Transition model}}$, using $\zeta = 10$ for all experiments presented in this paper.

3 Related Work

Our work addresses the challenging topic of learning object-based transition models in 3D partially observable environments without supervision. We identify two main components missing from

current object-based transition models, an understanding of object persistence and identity meaning that current models are not able to perform well in **partially observable environments** and that models are often trained without **semantically meaningful losses.**

Firstly, we acknowledge previous work on temporally extend scene decomposition and representation models [34, 13], typically used for video representation learning, object-based transition models [19, 16] and action conditional object-based transition models such as OP3 [31] and C-SWM [18]. We consider OP3 [31] to be the current state-of-the-art object-based transition model since C-SWM requires privileged information about exactly which object an action was applied to, while OP3 (like our model) simply requires the action taken by the agent. OP3 alternates between refinement steps and dynamics (or prediction) steps. One drawback of OP3 is that its refinement steps requires access to the observation, which means refinement cannot be applied when rolling out the model to unobserved time-steps. For this reason [31] only demonstrate unrolls for a limited number of steps.

3.1 Current models are not designed for partially observable environments.

The world that we operate in, and which we intend our agent's to operate in is partially observable, a severe limitation of existing object-based transition models [31, 18] and temporally extended object scene decomposition and representation models [34, 10, 29, 13] is their inability to cope with partially observable environments. A promising approach proposed by [13] uses an external memory for object tracking (but do not propose a transition model), their mechanism is different to that of our proposed alignment module, AlignNet, which we use to perform slot alignment in OAT. [13] train their model, TBA, for reconstruction while AlignNet incorporates dynamics and is trained using prediction, meaning that AlignNet can resolve ambiguities using dynamics[2], while TBA cannot. Additionally, TBA can only cope with static backgrounds and so it not applicable here.

3.2 Current models are not trained using semantically meaningful losses.

Another problem with current approaches to learning object-based transition models is how they are trained. In most scenes the background accounts for most of the pixels while the objects account for only a small fraction of them. So while it may be tempting to train transition and video-representation models using a pixel-level loss [33, 34, 31, 19, 16, 13], we show that is is preferable to compute losses directly between predicted and target objects (see Section 4.2 and Figure 1). Further, pixel-level losses require the object representations to be decoded into an image [33, 34, 31] which is often computationally expensive.

An alternative way to compute object-level losses is to compute a minimum assignment loss [4], employing the Hungarian algorithm, between predicted and target object representations. However, doing so can be problematic because you need to first compute a similarity matrix on which to apply the Hungarian algorithm. We show in Section 4.2 that computing a loss using the Hungarian algorithm (using L_2 loss between all object pairs as the similarity matrix), leads to poor generalisation in transition models when performing longer unrolls than those seen during training.

Furthermore, at the start of training comparing predictions with targets to compute a loss may not be meaningful since predictions will start off as essentially random. In our approach, we align the *inputs* with the targets, and condition the slot-wise prediction on the slot-wise history. This reinforces slot stability and allows us to accurately predict changes, Δ_t^a in each object representation.

Interestingly Lowe et al. [21] use Deep Sets [35] to encode predicted and target object sets and use a contrastive loss between encodings. Their approach has only been demonstrated for very simple datasets. C-SWM [18] do compute losses in their representation space, however they avoided the object correspondence problem because they extracted representations spatially, keeping the spatial ordering and using data where object movement was limited. Their approach is unlikely to scale well to partially observable environments with significant movement of objects across the field of view, especially when being trained to predict multiple steps in to the future.

OP3 [31] and other models [34] attempt to induce a weak, implicit object alignment by conditioning predicted object representations on those from the previous time-step. This does not guarantee

[2]For example when two visually similar objects collide with one another, the AlignNet can use dynamics to resolve which object is which after the collision.

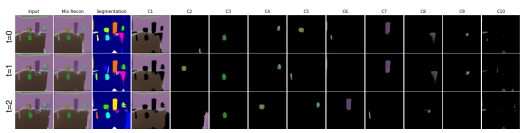

Figure 3: **Outputs of our scene decomposition and representation (MONet) module** across time. MONet predicts $K = 10$ latent object representation vectors $\{z_{t,i}\}_{i=1,\ldots,K}$ at each time-step. Here we visualise those vectors, in columns C1 to C10, using MONet's decoder. Notice that objects switch slots across time. This makes it difficult to (a) compute losses between predictions and targets and (b) integrate information about an object across time.

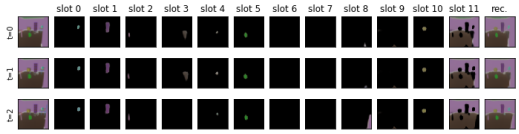

Figure 4: **Aligned inputs (first 4 time-steps) and targets (last 6 time-steps).** Our alignment module outputs object representation vectors which we visualise here using MONet's entity decoder. Most of the objects are now in consistent slots across time, making it easier to compute losses directly between object representations using a simple L_2 loss. Notice that while MONet outputs 10 slots, the AlignNet has 12 slots. Additional time-steps shown in Figure 11 in the Appendix.

alignment (or slots with consistent identity) over time, especially in partially observable environments and they do not use the alignment for computing losses.

Finally, we note that while there are existing models that learn to predict future states, given actions, directly from pixels [12, 11] we have focused primarily on related work that predicts the future states of objects, because we are particularly interested in developing models that may support future work towards object-based agents.

4 Experiments and Results

In this section we (1) demonstrate OAT's performance in the 3D Playroom environment [14, 15] and compare to the current state-of-the-art object-based transition model, OP3, [31] (Section 4.1), (2) investigate the benefit of alignment and object-level losses when training transition models (Section 4.2), (3) apply OAT in a real world robotics environment and show it accurately predicts both the motion of the robotic arm and its physical interaction with objects (Section 4.3).

4.1 Objects-Align-Transition Results in Playroom.

We train and test our model, OAT, on data collected by an agent taking actions in a Playroom environment [14, 15]. Each procedurally generated room in the dataset contains between 10 and 45 objects from 34 different classes in 10 different colours and three different sizes. A dataset of observation-action trajectories, $(x_t, a_t)_{t=0,1,\ldots,20}$, is generated by an agent taking actions according to a learned policy in a procedurally generated room. We collect $100,000$ trajectories with a $7:2:1$ train:validation:test split. The top row of Figure 5 shows an example trajectory.

We train OAT with four encoding steps (i.e. the model sees four observations for the first four time-steps) and six unrolling steps. MONet outputs K object representations, $\{z_{t,i}\}_{i=1,\ldots,K}$. We use $K = 10$ objects, with $F = 32$ features, and a memory with $M = 12$ slots. Figure 3 visualises the MONet outputs for the first 3 time steps. Our model achieves good segmentation (see the Adjusted Rand Index, ARI, in Table 1 for segmentation metrics). Importantly, notice that the slots are not stable across time. For example, the purple object in slot C7 at $t = 0$ switches slots at $t = 2$. Similarly the yellow object in slot C5 at $t = 0$ switches to slot C2 at $t = 1$ and slot C4 at $t = 2$.

The object representations, $z_t \in \Re^{[K\times F]}$, output by MONet are fed to the alignment module. The outputs of the alignment module, $z_t^a \in \Re^{[M\times F]}$, are visualised in Figure 4. The outputs of the alignment module are object representations, Figure 4 is a visualisation of these representations using MONet's decoder. Our alignment module successfully keeps objects in consistent slots across time.

The output of the alignment module is used to generate both inputs and targets for training the transition model. The transition model predicts latents, $z_t^d \in \Re^{[M\times F]}$. In Figure 5 we visualise roll-outs from our model using MONet's decoder; the reconstructed scene visualisations are generated as the mask-weighted sum of the each slot's pixels, $\sum_k \tilde{\mu}_{k,t}\tilde{x}_{t,k}$. While the model is trained to unroll for six steps, here we unroll for 15 steps. The transition model only sees the first four frames. In the top example we see that the model is able to predict the appearance of the avatar well and in general we notice that the model given the agents actions is able to predict the position of objects

well and without the representations degrading (for comparison to baselines see Figure 1). In some of the examples the targets appear to have more objects than those seen in the unroll: this is because the model has only seen the first four frames and has not seen those other objects in the room and therefore does not have enough information to predict where unseen objects will appear.

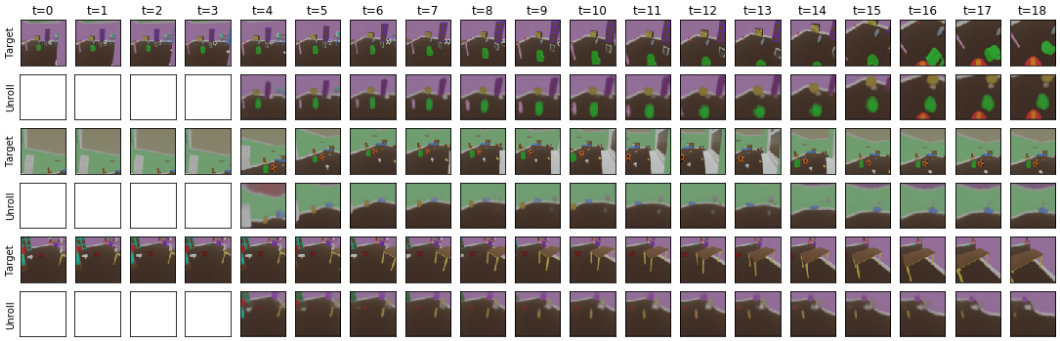

Figure 5: **Unrolling OAT for more time-steps than those seen during training.** The slot-wise transition model makes predictions in latent space (one latent per object), which we visualise using MONet's decoder. The slot-wise transition model was trained with 4 inputs steps and to unroll for 6 steps. Here the model takes the first 4 steps as input and is unrolling for 15. On the top row at $t = 16$, it is impressive that the model is able predict the appearance of the avatar's base (the red circle with the yellow stripe) when looking down. (For a per-object visualisation see Figure 13 in the Appendix.)

To conclude these experiments we compare our model to the current state-of-the-art object-based transition model, OP3 [31], and to our ablated model, OT, without the alignment module in Table 1 and Figure 1. Qualitatively, from Figure 1 we see that OAT significantly outperforms both baselines. Unrolls using OT, trained without alignment, lead to objects merging towards a grey cloud in the middle of the frame. Unrolls from OP3 lead to objects fading into the background (we explore the cause of each of these pathologies in the next Section 4.2). Results were consistent for each model across multiple runs.

To quantitatively compare models we consider three metrics: **Encoding ARI**, **Unroll Pixel Error** and **Unroll ARI**. The Encoding ARI (Adjusted Rand Index, [17, 25]), measures the accuracy of the object segmentation masks learned by the scene decomposition model. The Unroll Pixel Error and Unroll ARI evaluate the quality of the transition models' unrolls. To compute each of these we decode the latents, z_t^d, predicted by unrolling the transition model, to produce images, x_t^d, and masks, $\mu_{k,t}^d$. The Unroll Pixel Error is the mean-squared-error between the ground truth images and x_t^d for the unroll steps only. The Unroll ARI is the accuracy of the decoded masks $\mu_{k,t}^d$ compared with the ground-truth masks. For both ARI scores we exclude background pixels from the score since accurate decomposition of the objects is the main concern here. The Unroll ARI is a more meaningful evaluation of the unrolls than the Unroll Pixel Error since it is not affected by the background which often accounts for most of the pixels. Note that we only use ground-truth masks for evaluation purposes.

The results in Table 1 further demonstrate the crucial role that alignment (AlignNet) plays when performing unrolls. Without alignment the Unroll ARI is significantly higher because we are not able to compute a semantically meaningful object-level loss for training. Additionally, we see that our model significantly out performs OP3 on all metrics (see Figure 1). In light of the results in Table 1, in the next section we will more concretely look at the role of alignment and the object-level loss (between object representations) when training transition models.

	Encoding ARI	Unroll Pixel Error	Unroll ARI
OAT (ours)	**0.62**	**0.0121**	**0.42**
OT (ours, no AlignNet)	0.59	0.0143	0.12
OP3	0.32	0.0132	0.33

Table 1: **Comparing our model OAT to baselines.** We trained three OAT and OT models and report the ARI score, Unroll Pixel error and Unroll ARI for the model with the best Unroll ARI score. Similar for OP3 except we did 10 runs since we saw more variance in the OP3 results (see Appendix D for details).

Reporting object-level error for models trained using:	Object-level loss	Pixel-level loss
Aligned inputs	**0.0929 ± 0.00469**	14.4 ± 18.4
Unaligned inputs	$\geq 20.3 \pm 0.757$	$\geq 3312 \pm 4050$

Table 2: **The role of alignment and object-level loss when training transition models.** This table reports the mean±std latent error between predicted and ground truth latents (for 5 seeds). For models trained using unaligned input latents we compute the object-level loss using the Hungarian algorithm which is a lower bound on the actual value. Models trained with pixel-level loss can become very unstable resulting in high variance between runs. Individual seeds are shown in Figure 9.

4.2 What Matters For Object-centric Transition Models?

In this section we demonstrate the need for both (1) **alignment**, which ensures that each slot in the transition model receives the same object consistently across time and (2) **object-level loss**, computed between predicted and target object representations, rather than a pixel-level loss. We also compare different transition model cores and find a transformer followed by a slot-wise LSTM to be best.

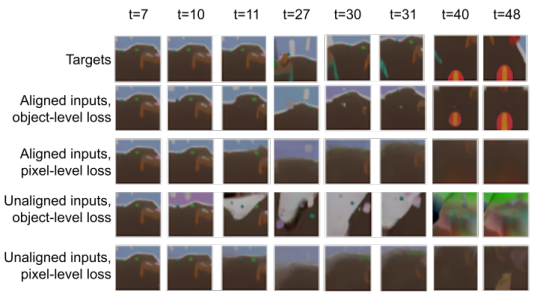

Figure 6: **Comparing slot-wise transition models trained with aligned vs. unaligned input latents and pixel-level vs. object-level loss.** These models were trained with 8 inputs steps and 12 unroll steps. Here they are being unrolled for 40 time-steps[3] The output of the transition model is a slot-wise object-based representation for each time-step. Here, we visualise the object representation vectors using MONet's decoder. Only models trained with aligned inputs were able to predict the reappearance of the chair at $t = 30$. Only the model trained with aligned inputs and object-level loss is able to predict the appearance of the avatars "base". In models trained with pixel-level loss predictions becomes very blurred.

In this section (Section 4.2) only, we use a MONet model that is trained using ground-truth masks instead of learning its own masks. We do this here to directly analyse the benefits of aligned object representations and object-level losses for the transition model without other confounds. Using ground-truth masks allows us to know each objects true identity across time and to directly measure the role of alignment in transition models without confounding errors from AlignNet or MONet's segmentation quality. We use a pre-trained MONet (with fixed weights) to compute object representations to keep the model similar to the full end-to-end model described in the rest of the paper.

We train slot-wise transition models under four different conditions; feeding aligned or unaligned object representations (and target) object representations to the transition model[4] and train the model with a pixel-level or object-level loss. Results in Table 2 clearly demonstrate the benefits of (1) using aligned object representations for training transition models and (2) training transition models with a object-level loss. These results are critical for future development of transition models and demonstrates the need for alignment in transition models. Note that for the experiments using the unaligned inputs we computed the object-level loss using the Hungarian algorithm which is a lower bound estimate of the true object-level loss (since the minimum assignment in L_2 may not be the correct assignment).

Additionally, Figure 6 compares models trained under the four conditions listed above, we see that models trained using aligned inputs (and targets) are able to predict re-appearance of objects while those without aligned latents are not (Figures 6, $t = 30$). We also see that models trained with an object-level loss and without alignment deteriorate quickly. Additionally, for models trained with a pixel-level loss we notice a "ghosting" effect where objects fade into the background across time, similar to the effects seen in the OP3 results (Figure 17). Pixel-level loss leads to this "ghosting" effect because most pixels in the observation are background pixels and so the background pixels dominate the loss.

[4]for the unaligned inputs we shuffle the order of the ground-truth masks $\in \Re^{[K,W,H,3]}$ that are fed to MONet along the K axis, for aligned inputs we ensure that each slot contains a consistent object across time.

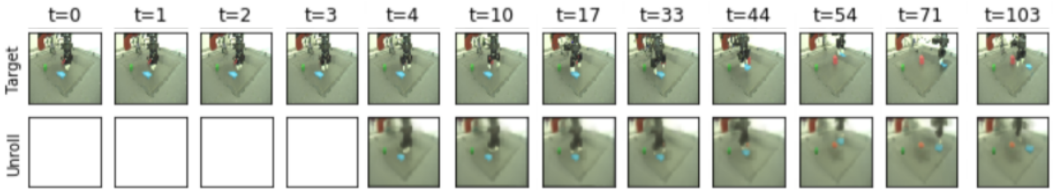

Figure 7: **Unrolling for many more steps than those seen during training.** OAT is trained to take four input steps and to unroll for six time-steps, here we demonstrate the model unrolling for 100 time-steps. Impressively, we see that our model learns both to predict the behaviour of the robotics arm accurately and how the arm interacts with the objects. We also see that the model is able to unroll for significantly more steps that those seen during training. Note that our model makes prediction in latent space; here we are visualising those latents using MONet's decoder. Additional results in Figure 15 in the Appendix.

We found these results to be consistent across multiple runs (see Figure 9 in the Appendix) and different choices for slot-wise transition model architectures. For the results shown in this section we used a transformer with a slot-wise LSTM. Figure 10, in the Appendix, compares object and pixel errors for different transition module cores. Training with object-level loss and aligned inputs, the transformer with slot-wise LSTM worked best.

4.3 Application to Robotics

Here we demonstrate the application of OAT to a real world dataset [3] of robot trajectories. These trajectories involve a robot arm interacting with three coloured objects of varying shapes and colours. The dataset is particularly challenging because our model must learn to predict not just the motion of the robot arm, given the arm actions, but it must also learn to predict how the arm interacts with objects requiring some understanding of intuitive physics.

We train OAT with four input steps and to unroll for six steps. Our model achieves excellent segmentation results, shown in Figure 14 of the Appendix. Figure 7 shows impressive results obtained when unrolling our model, OAT, for significantly more steps than those seen during training. What is more in the 3rd sample from the top, we see that the model correctly predicts the reappearance of the red objects after it had been fully occluded at $t = 3$ for 51 time-steps, reappearing fully at $t = 54$ in both the prediction and the target frame. Our model, OAT, is able to accurately predict the reappearance of objects, even through longer term occlusion, because the model explicitly models the history of each object endowing the model with a notion of object persistence.

5 Conclusion

We presented Objects-Align-Transition (OAT), an object-centric transition model that combines a scene decomposition and object representation module (MONet) with a slot-wise transition module, via an alignment module. The alignment module plays two key roles. Firstly, it ensures that the slot-wise transition model receives slot-consistent object representations across time. Secondly, it allows us to compute an object-level loss rather than a pixel-level loss which is commonly used when training transition models.

In an ablation study, we demonstrated the essential role that alignment and object-level losses play when training transition models. Additionally, we significantly out performs existing state-of-the-art object-centric transition models in a 3D partially observable environment, and we applied our model to a real-world robotics dataset, predicting many steps further into the future than those seen during training.

There is room to improve our model in the future, for example by making stochastic predictions about the future, and by better modelling the uncertainty about the objects in the environment. Meanwhile, our work paves the way for future object-centric agent research, for example, enabling agents to plan over future trajectories in object representation space.

References

[1] R. Baillargeon, E. S. Spelke, and S. Wasserman. Object permanence in five-month-old infants. *Cognition*, 20(3):191–208, 1985.

[2] C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner. MONet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

[3] S. Cabi, S. Gómez Colmenarejo, A. Novikov, K. Konyushkova, S. Reed, R. Jeong, K. Zolna, Y. Aytar, D. Budden, M. Vecerik, et al. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv*, pages arXiv–1909, 2019.

[4] J. D. Co-Reyes, R. Veerapaneni, M. Chang, M. Janner, C. Finn, J. Wu, J. Tenenbaum, and S. Levine. Discovering, predicting, and planning with objects. *International Conference on Machine Learning Workshop*, 2019.

[5] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[6] M. Garnelo, K. Arulkumaran, and M. Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[7] M. Garnelo and M. Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23, 2019.

[8] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2008.

[9] K. Greff, R. L. Kaufman, R. Kabra, N. Watters, C. Burgess, D. Zoran, L. Matthey, M. Botvinick, and A. Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pages 2424–2433. PMLR, 2019.

[10] K. Greff, A. Rasmus, M. Berglund, T. Hao, H. Valpola, and J. Schmidhuber. Tagger: Deep unsupervised perceptual grouping. In *Advances in Neural Information Processing Systems*, pages 4484–4492, 2016.

[11] D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

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

[13] Z. He, J. Li, D. Liu, H. He, and D. Barber. Tracking by animation: Unsupervised learning of multi-object attentive trackers. *arXiv preprint arXiv:1809.03137*, 2018.

[14] F. Hill, A. Lampinen, R. Schneider, S. Clark, M. Botvinick, J. L. McClelland, and A. Santoro. Environmental drivers of systematicity and generalization in a situated agent. In *International Conference on Learning Representations*, 2020.

[15] F. Hill, S. Mokra, N. Wong, and T. Harley. Human instruction-following with deep reinforcement learning via transfer-learning from text. *arXiv preprint arXiv:2005.09382*, 2020.

[16] J.-T. Hsieh, B. Liu, D.-A. Huang, L. F. Fei-Fei, and J. C. Niebles. Learning to decompose and disentangle representations for video prediction. In *Advances in Neural Information Processing Systems*, pages 517–526, 2018.

[17] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[18] T. Kipf, E. van der Pol, and M. Welling. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

[19] A. Kosiorek, H. Kim, Y. W. Teh, and I. Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. In *Advances in Neural Information Processing Systems*, pages 8606–8616, 2018.

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

[21] S. Löwe, K. Greff, R. Jonschkowski, A. Dosovitskiy, and T. Kipf. Learning object-centric video models by contrasting sets. *arXiv preprint arXiv:2011.10287*, 2020.

[22] G. Marcus and E. Davis. *Rebooting AI: Building Artificial Intelligence We Can Trust*. Ballantine Books Inc., 2019.

[23] Z. W. Pylyshyn. The role of location indexes in spatial perception: A sketch of the finst spatial-index model. *Cognition*, 32:65–97, 1989.

[24] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems 30*, pages 5690–5701, 2017.

[25] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[26] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[27] M. Shanahan, B. Beyret, M. Crosby, and L. Cheke. Artificial intelligence and the common sense of animals. *Trends in Cognitive Sciences*, 24(11):862–872, 2020.

[28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[31] R. Veerapaneni, J. D. Co-Reyes, M. Chang, M. Janner, C. Finn, J. Wu, J. B. Tenenbaum, and S. Levine. Entity abstraction in visual model-based reinforcement learning, 2019.

[32] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018.

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

[34] M. A. Weis, K. Chitta, Y. Sharma, W. Brendel, M. Bethge, A. Geiger, and A. S. Ecker. Unmasking the inductive biases of unsupervised object representations for video sequences, 2020.

[35] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In the Conclusions section we discuss how our model may be improved.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] One of our datasets is available via [3] which we reference in the paper. The code and Playroom data [14] have not been made publicly available.
   (b) Did you specify all the training details (e.g., data splits, hyper-parameters, how they were chosen)? [Yes] This is detailed in Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Where we compute averages over seeds, e.g. Table 6, we report mean and std additionally in the appendix when we show training plots we show all seeds. For Table 1 we show the best value for each model.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In Section A.1.1 we give details of the GPUs and training times.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] We are not releasing any assets.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Model Details

A.1 Objects-Align-Transition Implementation Details

A.1.1 Training Hyperparameters

MONet and AlignNet each have their own set of hyper-parameters. OAT introduces only one additional parameter, ζ, which weights the transition model loss. While we used $\zeta = 10$ for all the results in this paper, we found that the model performed similarly well for smaller values of ζ, such as $\zeta = 1.0$. For MONet, we use the same hyper-parameters as those detailed in the Appendix 1.B of [2], except we use $\gamma = 0.05$ for the mask KL weight rather than $\gamma = 0.5$. Qualitatively, our model

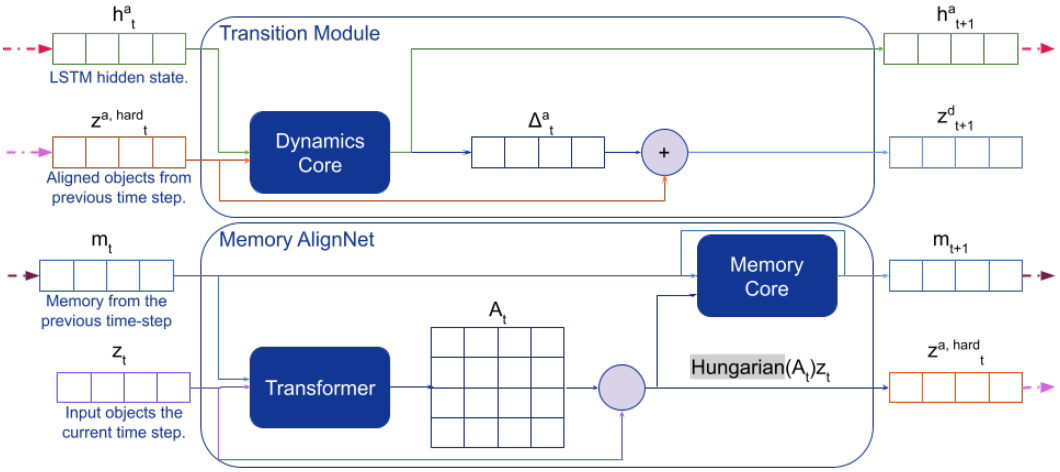

Figure 8: **Details of the Memory AlignNet and its interactions with the Transition module.**

still performed well with $\gamma = 0.5$. We train OAT for 2 million steps (30 days, though the model appears to have converged after 1M steps) with an effective batch size of 32 (batch size of four spread over eight NVIDIA V100 GPUs) and a learning rate of 3×10^{-4}. We have to use a small batch size because MONet is processing a whole sequence of images and this requires more memory (for the activations) than processing a single image.

For the results on the robotics dataset we used the same hyper-parameters as above, except we use $\gamma = 0.5$. The model is trained for 1.6 million steps (16 days, however the model is close to convergence after 200k steps) using $K = 7$ object slots, $M = 8$ memory slots and a MONet feature size, $F = 32$.

A.2 Memory AlignNet Implementation Details

In Section 2.2 of the main paper we introduced the Memory AlignNet. Here we provide further implementation details and additional details in Figure 8.

The memory, m_t, is initialised with zeros. At each time-step the memory, m_t and unaligned latents, z_t, from MONet are fed to a transformer to produce an adjacency matrix, A_t. We append slot indices to the memory, $1, ..., M$ and set indices, 0 to the memories and 1 to the unaligned latents. We feed the memory and the latents to the transformer (treating them as the ‘inputs‘ and ‘outputs‘ respectively in the notation of [30]). Transformers typically output a value, instead we take the final attention output to be A_t. We use three transformer layers, one head and embedding sizes of 128.

MONet outputs K object representations, z_t, even when there may be $< K$ objects in a scene. This means that some slots in the object representation do not correspond to an object, we refer to these slots as empty slots. Empty slots may be encoded differently which means that they could be treated as different objects, leading to a memory filled with empty slots. To avoid this we compute which slots in the latent input, z_t, are empty using MONet predicted masks, placing a threshold on the sum of pixels, we set this threshold to 4.0 and found this to work well. In soft alignment, we then mask out those slots to avoid assigning empty objects to slots that are already occupied and to ensure that all empty slots are treated in the same way (rather than creating many new slots for empty slots that may be encoded in different ways). In the hard alignment empty slots are assigned to an $M + 1^{th}$ slot and are not used to update the memory. Additionally, in the hard alignment we ensure that all non-empty slots are assigned to a unique memory slot so that no objects are "lost" in the process of alignment.

A.3 Transformer + SlotLSTM Transition Module Implementation Details

In Section 2.3 we introduce our slot-wise transition model, Transformer + SlotLSTM. We combine a transformer [30] with a slot-wise LSTM. Our transition model must play two roles (1) At each

time-step the model must consider how each object may interact with each other object, (2) the model must be able to integrate information about a single object over time.

For (1) we use transformers. While transformers are often used to process sequences of text, here we use transformers to encode the interactions between an object and all others at that time-step. The input to the transformer are the aligned object latents, $z_t^a \in \Re^{M \times F}$ and the action, a_t, taken at that time-step. The action is append to each of the object latents and this is fed to the transformer along with the states of the SlotLSTM (which we detail below).

For (2) we use SlotLSTM this is an LSTM with shared weights applied to each of the M slots in z_t^a. Since z_t^a are aligned over time, each LSTM is fed information about a single object over time and each has its own hidden state (which is passed to the transformer as described above) allowing the model to integrate information about a single object over time. We found that our transition model performed better than regular transformers (Figure 10) we attribute this to the fact that our model has an explicit state for each object which we were able to update over time.

A.4 Additional Details and Results for the Ablation.

Recall that in the ablation study **only**, we train MONet using ground-truth masks. This means that rather than using an attention network (U-net) to predict masks, $\mu_{k,t}$, we use ground-truth masks and feed these to the slot-wise VAE.

Figure 9 compares models trained using aligned vs. unaligned inputs and pixel-level vs. object-level loss, showing 5 runs per model configuration. Models trained using latent loss can do $10\times$ the number of updates per second compared to models trained using the pixel loss. Additionally, models trained with pixel-level loss tend to become unstable during training leading to high variance between seeds.

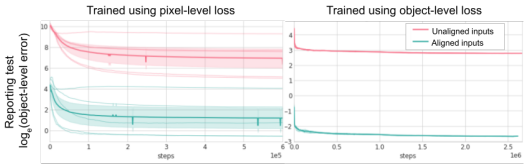

Figure 9: **Comparing models trained using aligned vs. unaligned inputs and pixel-level vs object-level loss.** We see that, across 5 runs, training transition models using aligned inputs and latent-level loss obtains best results.

Figure 10 shows the effect of using different slot-wise transition model cores. We see that a transformer followed by a slot-wise LSTM achieves the best results when training using an object-level loss and aligned inputs.

B Additional Results on the Playroom Dataset.

In the main body of the paper we demonstrated the alignment modules ability to align object representations over three time-steps. In Figure 11, we demonstrate outputs from the alignment module over 10 steps. The alignment module is able to keep track of objects across time.

Figure 12 shows additional unrolls using OAT, trained under the same conditions as those described in Section A.1. The figure shows consistently good results across many samples.

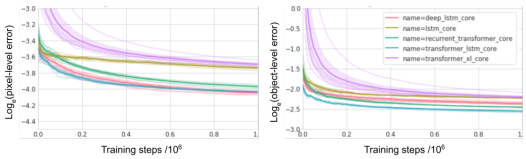

Figure 10: **Comparing transition module cores.** We show five runs for each model configuration. We find that a transformer with slot LSTM outperforms the Transformer XL [5], as well as LSTMs [8], Deep LSTMs and a recurrent transformer that predicts both a state and outputs at each time-step and appends that state to the input to make predictions at the next time-step.

Figure 11: **Aligned inputs (first 4 time-steps) and targets (last 6 time-steps).** Our alignment module outputs object representation vectors which we visualise here using MONet's entity decoder. Most of the objects are now in consistent slots across time, making it easier to compute semantically meaningful losses directly between object representations using a simple L2 loss. Refer to Figure 2 to see how the losses are computed. Notice that while MONet outputs 10 slots, the AlignNet has 12 slots.

In the main text we visualise OAT's unrolls by passing each predicted object representation to MONet's decoder function to reconstruct objects, $\tilde{x}_{t,k}$ and masks $\tilde{\mu}_{t,k}$, and combining these into a single image for each time-step, $\sum_{k=1}^{K} \tilde{\mu}_{t,k}\tilde{x}_{t,k}$. In Figure 13 rather than combining the component objects, we show each of the reconstructed, masked objects, $\tilde{x}_{t,k}\tilde{\mu}_{t,k}$ across time.

C Additional Results on the Robotics Dataset.

Figure 14 visualises the MONet outputs on the Robotics dataset. We see that the arm, gripper and objects are each represented in their own object slot, C1 to C7.

Figure 15 shows additional unrolls on the Robotics dataset.

D Reproducing OP3 Results

OP3 [31] is a slot-wise scene segmentation and dynamics model. It applies a refinement network (based on IODINE, [9]) to an initial estimate of the scene's slot-wise object features, followed by an action-conditioned prediction model to predict the features at the next time step. The refinement and prediction steps are interlaced through a sequence of steps, making OP3 the closest baseline to OAT.

We tested the correctness of our implementation on the `pickplace_multienv_10k` dataset used in the original OP3 paper and made available by the authors of OP3. The model was trained and

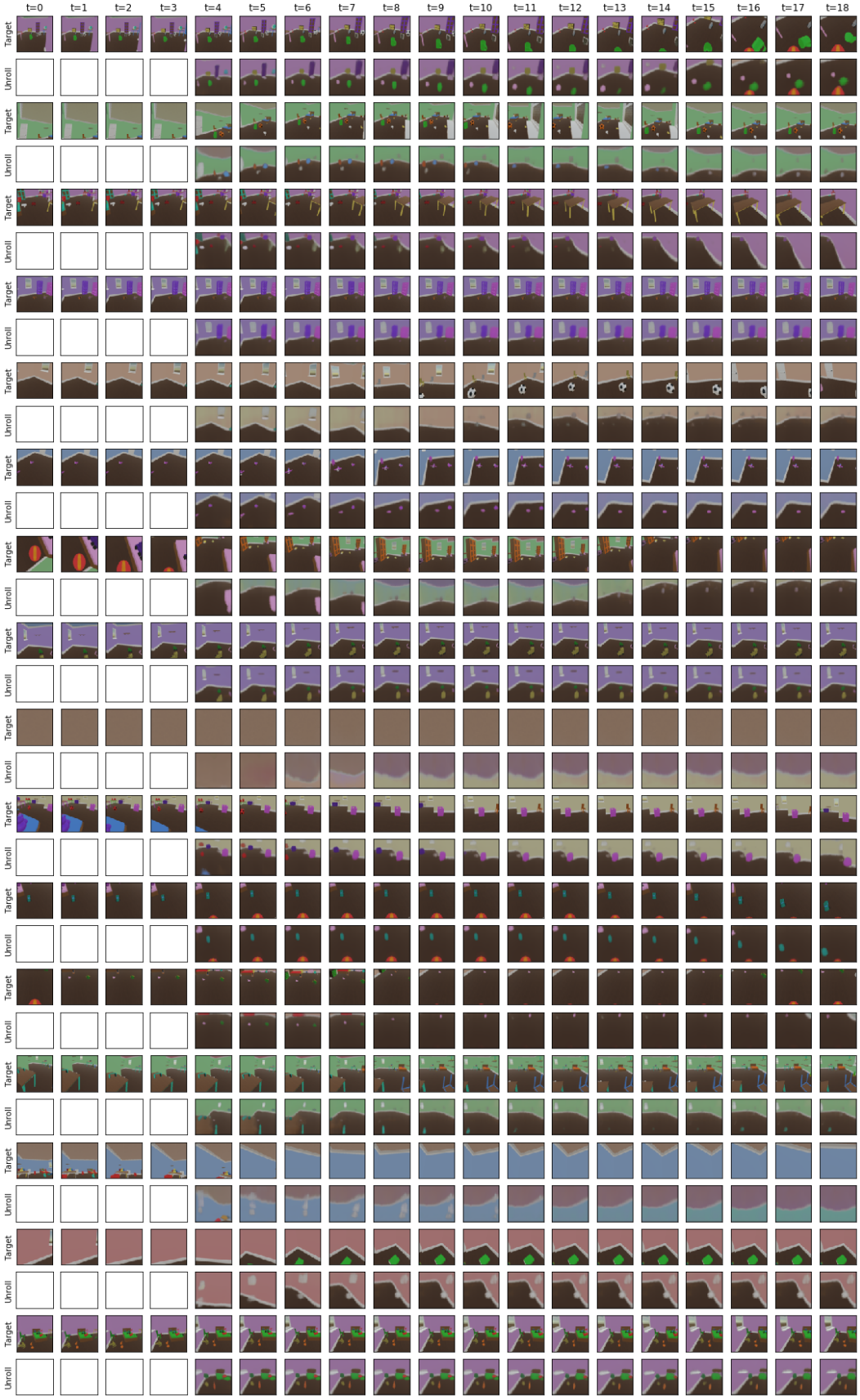

Figure 12: **Additional OAT unrolls.** OAT was trained to take four input steps and unroll for six time-steps, here we demonstrate OAT unrolling for 15 time-steps. Note, this model was trained using the exact same training set-up (detailed in Section A.1 in the Appendix) as results shown in Figure 5 but is a different seed. We see that we get similarly good results across multiple seeds.

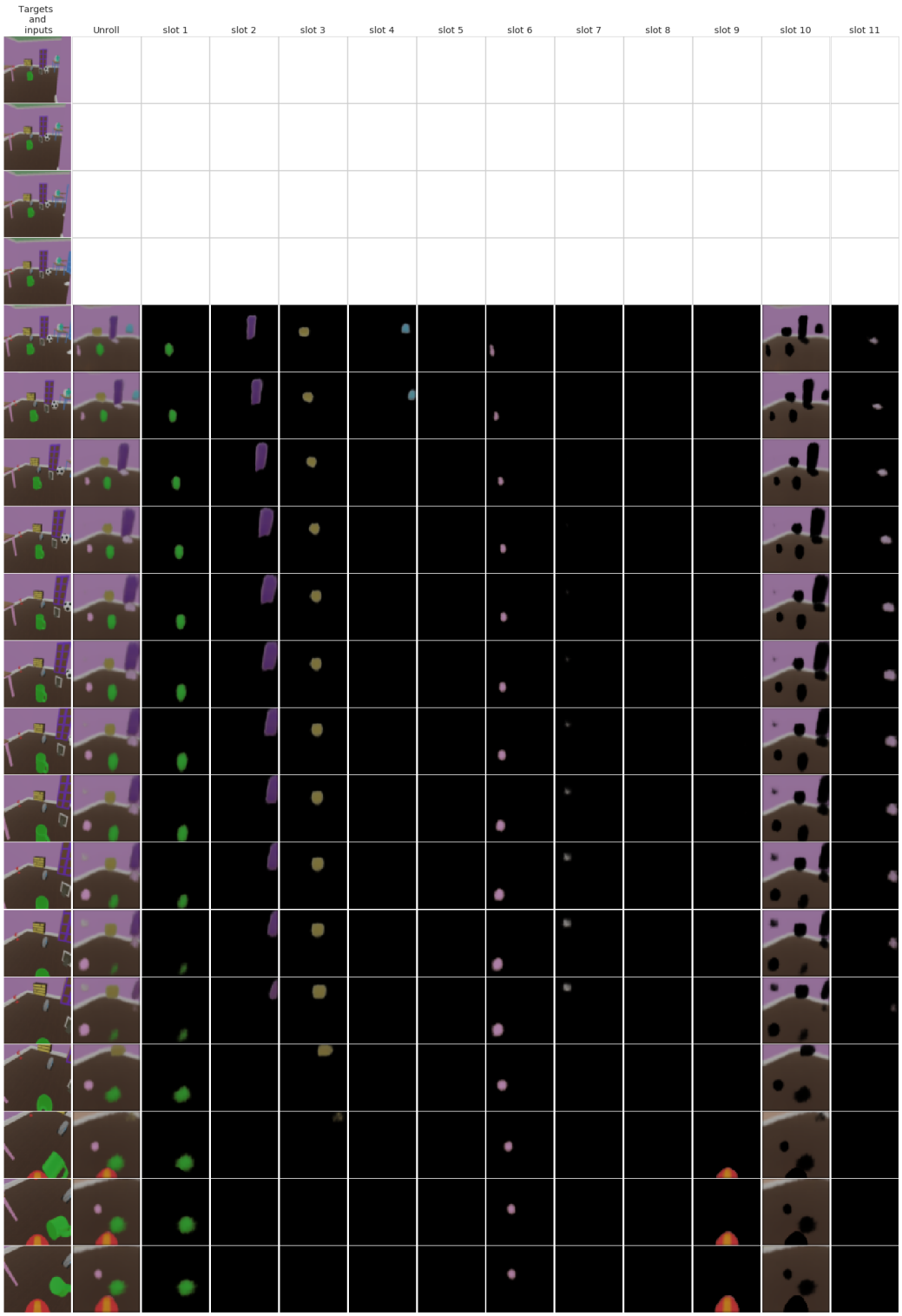

Figure 13: **Showing the components predicted during the unrolls.** Rows correspond to time-steps. Slot one to 11 show the objects predicted during the unroll. OAT sees the first four time-steps and unrolls for the next 15 time-steps.

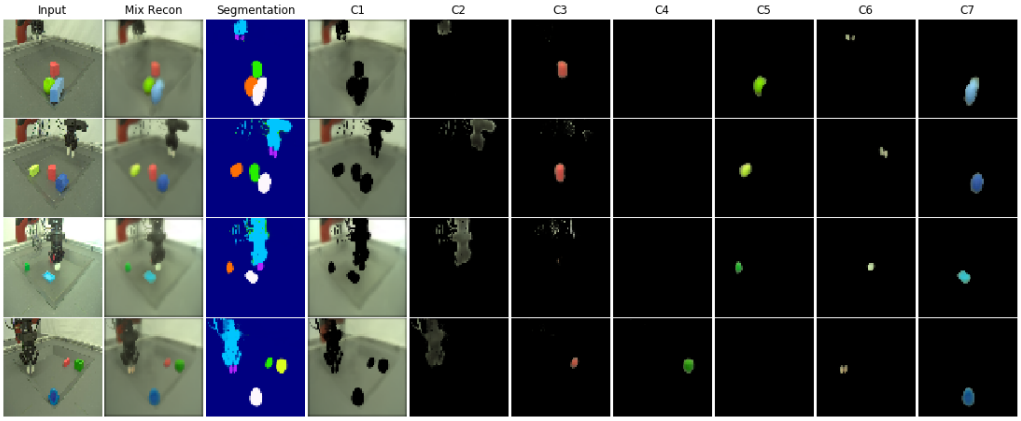

Figure 14: **Scene decomposition and representation module outputs for the real world robotics data.** Each object is correctly placed in its own column.

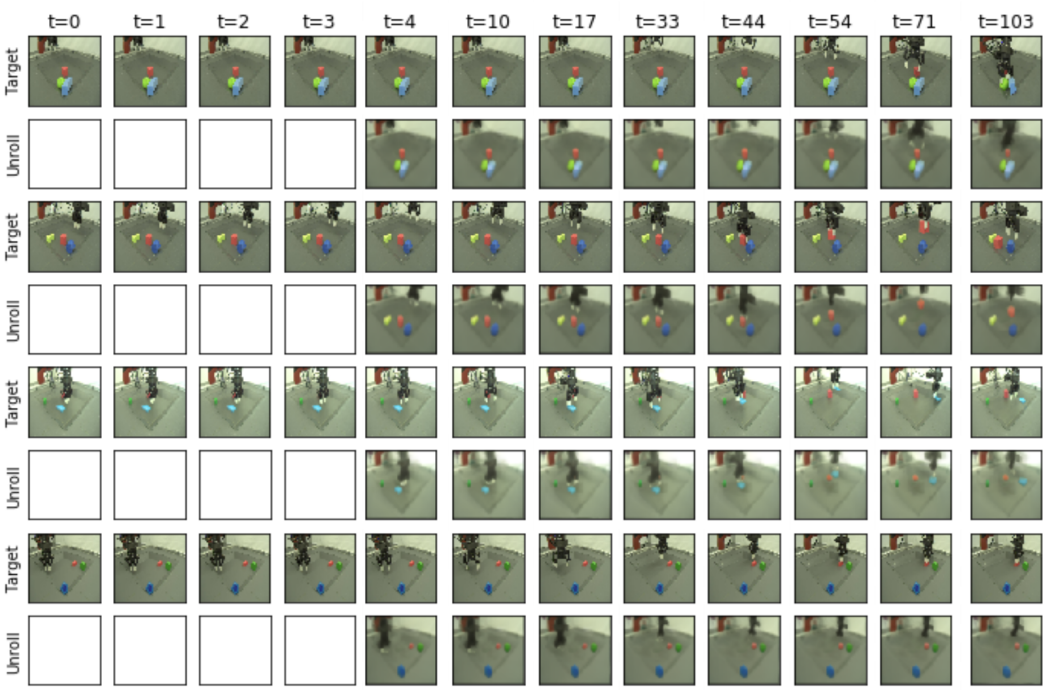

Figure 15: **Unrolling for many more steps than those seen during training.** OAT is trained to take four input steps and to unroll for six time-steps, here we demonstrate the model unrolling for 100 time-steps. Impressively, we see that our model learns both to predict the behaviour of the robotics arm accurately and how the arm interacts with the objects. We also see that the model is able to unroll for significantly more steps that those seen during training. Note that our model makes prediction in latent space; here we are visualising those latents using MONet's decoder.

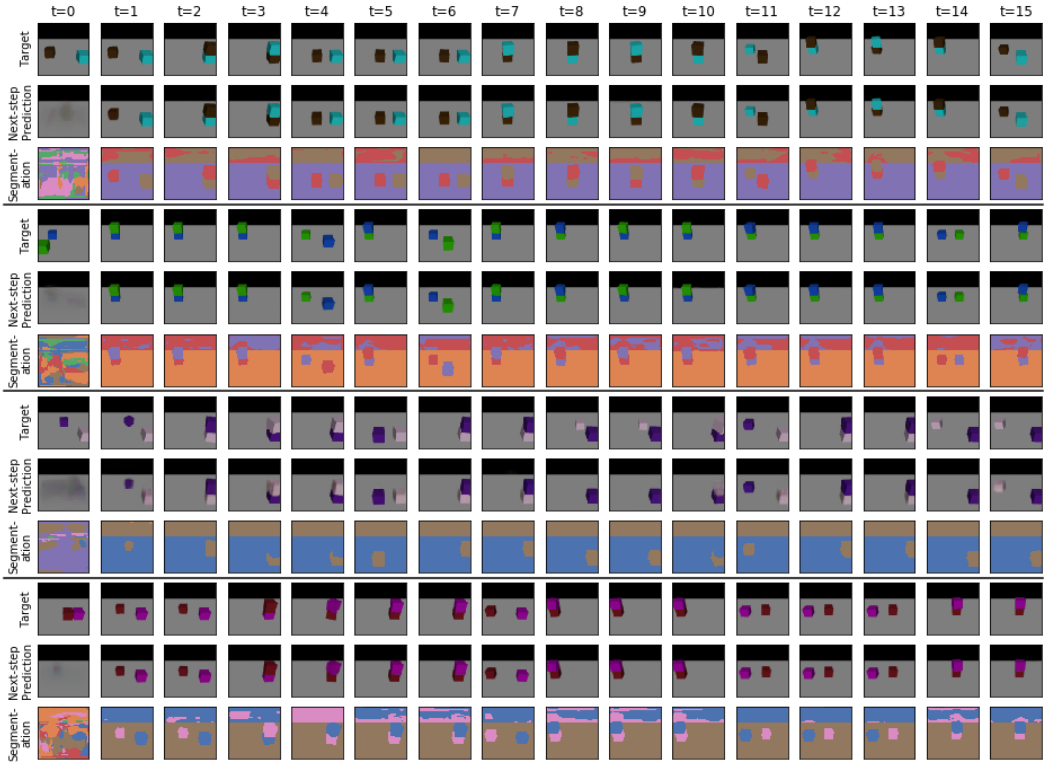

Figure 16: OP3 [31] results on the `pickplace_multienv_10k` dataset released by the OP3 authors. This model was trained as per OP3's standard regime with one dynamics step and one refinement step at each time-step (i.e. no unroll). We've plotted the decoded outputs post-dynamics but pre-refinement at each time-step to show the strength of the transition model. Though the segmentations are imperfect (partly because we used 7 component slots), and the model occasionally drops an object (e.g. sequence 3), it has learnt the dataset's jumpy object transitions perfectly.

evaluated using one refinement step and a next-step prediction in a loop. The results in Figure 16 show that our implementation has learnt the dynamics well enough to cope with the dataset's jumpy object transitions and we are able to reproduce the results similar to those shown in Figure 6(b) of the OP3 paper [31]. Note that we show the result of multiple discrete actions taken over many steps, while they shown only a single discrete action applied at one time step[5].

Following this validation of our implementation, we trained OP3 on the Playroom dataset in a regime that mirrors ours. Four burn-in steps with refinement and next-step prediction allow the model to build its initial estimate of the scene. These are followed by six unroll steps during training, where the refinement is disabled. Here the slot parameters are only updated via the prediction core. For evaluation, we roll-out for 15 steps instead of six to test the dynamics model at long-range prediction. Figure 17 shows eight such roll-out sequences from the model with the best ARI score over the rollout steps.

D.1 OP3 Hyperparameters

For the results shown in the main section of this paper, we substituted OP3's original relation net-based transition model with a transformer module plus SlotLSTM (transformer + SlotLSTM). This is identical to OAT's setup and allows the fairest comparison. The SlotLSTM's hidden size is also 128. Indeed, results shown in Figure 18 demonstrate that we obtain better performance training OP3 with the transformer + SlotLSTM (introduced in Section 2.3 of our paper), than using OP3's original relation net-based transition model. The transformer uses 2 layers, 4 heads, and embedding size 128.

[5] In Figure 7(a) of the OP3 paper [31] the authors do demonstrate predicting one step into the future, without refinement.

Figure 17: **Baseline OP3 [31] rollouts.** This model was trained with the same number of input steps (four) and unroll steps (six) as our model, OAT (see Figure 5). Here we picked the model with the best ARI score from 10 independent runs. The results are noteworth at t=4 (following the burn-in refinement steps) for accurately placing objects. But for t>4, the prediction core quickly begins to accumulate errors, distorting the size and position of objects and in some cases the floor edges. It further fails to predict the appearance of the avatar (for instance, toward the end of the first sequence), which is an easy-to-learn and predictable consequence of the action space.

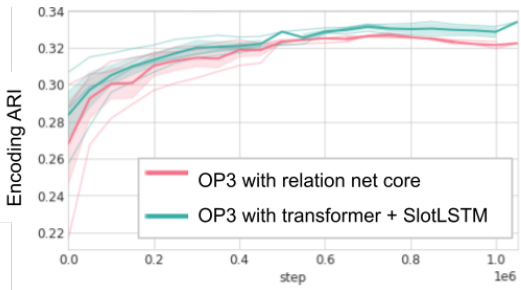

Figure 18: **Comparing OP3 trained with a relation net core vs. a transformer + SlotLSTM core.**

We train OP3 with the refinement and prediction schedule described above, again similar to OAT. The refinement encoder is a convolutional network with five layers with [64, 128, 128, 256, 256] output channels, kernel shape 5, and stride 1, followed by an MLP with [256, 256] hidden units. The slots have 64 latents each (and hence the MLP outputs 128 posterior parameters). We use stochastic latents only, avoiding OP3's deterministic latents without loss of generality.

The encoder, as in IODINE, is applied slot-wise across the following refinement inputs: the input image, the log likelihood of the image with respect to the predicted output distribution, the current estimate of the slot parameters, the gradient of the log likelihood with respect to the slot parameters, the logits of the decoded object masks, the masks themselves, the gradient of the log likelihood with respect to the masks, and a counter-factual (as in [9]).

The decoder is a broadcast decoder comprising transpose convolutions with [64, 64, 64, 64, 64, 4] output channels, kernel size 5, and stride 1. The decoded mask logits are activated with a tanh scaled by 10.0.

We used a KL loss scale of 0.5 and a fixed output distribution scale of 0.1. To stabilise training, we also clipped gradients to a norm of 5.0. For the refinement inputs, we clipped gradients to a slightly higher norm of 10.0. Finally, we used an effective batch size of 32, the RMSProp optimizer, and a learning rate of 1e-5. We trained all OP3 models for 4 million steps (15 days).