Entity Abstraction in Visual Model-Based Reinforcement Learning

Rishi Veerapaneni[*,1], **John D. Co-Reyes**[*,1], **Michael Chang**[*,1], **Michael Janner**[1]
Chelsea Finn[2], **Jiajun Wu**[3], **Joshua Tenenbaum**[3], **Sergey Levine**[1]

Abstract: We present OP3, a framework for model-based reinforcement learning that acquires object representations from raw visual observations without supervision and uses them to predict and plan. To ground these abstract representations of entities to actual objects in the world, we formulate an interactive inference algorithm which incorporates dynamic information in the scene. Our model can handle a variable number of entities by symmetrically processing each object representation with the same locally-scoped function. On block-stacking tasks, OP3 can generalize to novel block configurations and more objects than seen during training, outperforming both a model that assumes access to object supervision and a state-of-the-art video prediction model.

Keywords: model-based reinforcement learning, objects, compositionality

1 Introduction

A powerful tool for modeling the complexity of the physical world is to frame this complexity as the composition of simpler entities and processes. For example, the study of classical mechanics in terms of macroscopic objects and a small set of laws governing their motion has enabled not only an explanation of natural phenomena like apples falling from trees but the invention of structures that never before existed in human history, such as skyscrapers. Paradoxically, the creative *variation* of such physical constructions in human society is due in part to the *uniformity* with which human models of physical laws apply to the literal building blocks that comprise such structures – the reuse of the same simpler models that apply to primitive entities and their relations in different ways obviates the need, and cost, of designing custom solutions from scratch for each construction instance.

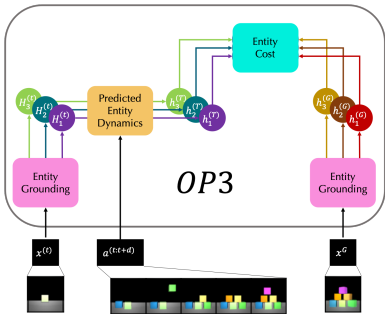

Figure 1: OP3 infers a set of entity variables $H_{1:K}^{(t)}$ from an observation x^t and predicts their future states given a sequence of actions $a^{(t:T)}$. We evaluate rollouts during planning by scoring these predicted states against inferred goal hidden states $h_k^{(G)}$.

The challenge of scaling the generalization abilities of learning robots follow a similar characteristic to the challenges of modeling physical phenomena: the complexity of the task space may scale combinatorially with the configurations and number of objects, but if all scene instances share the same set of objects that follow the same physical laws, then transforming the problem of modeling scenes into a problem of modeling objects and the local physical processes that govern their interactions may provide a significant benefit in generalizing to solving novel physical tasks the learner has not encountered before.

That is the central hypothesis of this paper, which we test by defining models for perceiving and predicting raw observations that are themselves compositions of simpler functions that operate locally on *entities* rather than globally on scenes. Importantly, the symmetry that all objects follow the same physical laws enables us to define these learnable entity-centric functions to take as input argument

* Equal contribution.

[1]University of California Berkeley. [2]Stanford University. [3]Massachusetts Institute of Technology.

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

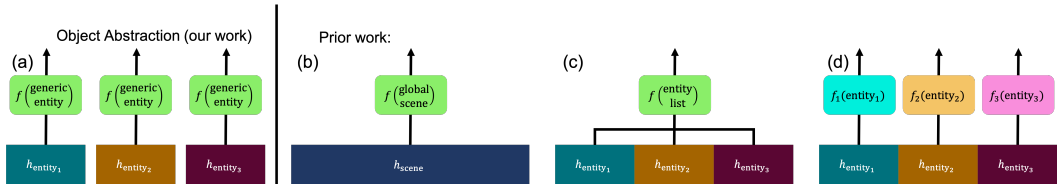

Figure 2: Our approach to model-based RL imposes the **entity abstraction**: **(a)** The hidden state is factorized into *local* entity states, each of which are symmetrically processed with the same function that takes in a *generic entity* as an argument. In contrast, prior work do not incorporate symmetric processing of the hidden state. **(b)** The hidden state represents the scene *globally* and is processed with a single function. **(c)** Different chunks of the hidden state represent different entities and the entire state is processed with a single function. **(d)** The hidden state is factorized into *local* entity states, each of which are processed with a different function.

a variable that represents a generic entity, the specific instantiations of which are all processed symmetrically be the same function. We use the term *entity abstraction* to refer to the abstraction barrier that isolates the abstract *variable*, which the entity-centric function is defined with respect to, from its concrete *instantiation*, which contains information about the appearance and dynamics of an object that modulates the function's behavior.

Defining the observation and dynamic models of a model-based reinforcement learner as neural network functions of abstract entity variables allows for symbolic computation in the space of entities, but the key challenge for realizing this is to ground the values of these variables in the world from raw visual observations. Fortunately, the language of partially observable Markov decision processes (POMDP) enables us to represent these entity variables as latent random state variables in a state-factorized POMDP, thereby transforming the variable binding problem into an inference problem with which we can build upon state-of-the-art techniques in amortized iterative variational inference [1–3] to use temporal continuity and interactive feedback to infer the posterior distribution of the entity variables given a sequence of observations and actions.

We present a framework for *object-centric perception, prediction, and planning* (OP3), a model-based reinforcement learner that predicts and plans over entity variables inferred via an *interactive inference* algorithm from raw visual observations. Empirically OP3 learns to discover and bind information about actual objects in the environment to these entity variables *without any supervision on what these variables should correspond to.* As all computation within the entity-centric function is *local in scope* with respect to its input entity, the process of modeling the dynamics or appearance of each object is *protected* from the computations involved in modeling other objects, which allows OP3 to generalize to modeling a variable number of objects in a variety of contexts with no re-training.

Contributions: Our conceptual contribution is the use of entity abstraction to integrate graphical models, symbolic computation, and neural networks in a model-based RL agent. This is enabled by our technical contribution: defining models as the composition of locally-scoped entity-centric functions and the interactive inference algorithm for grounding the abstract entity variables in raw visual observations without any supervision on object identity. Empirically, we find that OP3 achieves about three times greater accuracy than state of the art video prediction models solving novel single and multi-step block stacking tasks.

2 Related Work

Representation learning for visual model-based reinforcement learning: Prior works have proposed learning video prediction models [4–7] to improve exploration [8] and planning [9] in reinforcement learning. However, such works and others [10–13] that represent the scene with a single representation vector may be susceptible to the binding problem [14, 15] and must rely on data to learn that the same object in two different contexts can be modeled similarly. But processing a disentangled latent state with a single function [16–20] or processing each disentangled factor with a different function [6, 21] (1) assumes a fixed number of entities that cannot be dynamically adjusted for generalizing to more objects than in training and (2) has no constraints to enforce that multiple instances of the same entity in the scene be modeled in the same way. For generalization, often the particular arrangement of objects in a scene does not matter so much as what is constant across scenes – properties of individual objects and inter-object relationships – which the inductive biases of these prior works do not capture. The entity abstraction in OP3 enforces symmetric processing of entity representations, thereby overcoming the limitations of these prior works.

Unsupervised grounding of abstract entity variables in concrete objects: Prior works that model entities and their interactions often pre-specify the identity of the entities [22–28], provide additional

supervision [29–32], or provide additional specification such as segmentations [33], crops [34], or a simulator [35, 36]. Those that do not assume such additional information often factorize the entire scene into pixel-level entities [37–39], which do not model objects as coherent wholes. None of these works solve the problem of grounding the entities in raw observation, which is crucial for autonomous learning and interaction. OP3 builds upon recently proposed ideas in grounding entity representations via inference on a symmetrically factorized generative model of static [3, 14, 40] and dynamic [41] scenes, whose advantage over other methods for grounding [42–45] is the ability to refine the grounding with new information. In contrast to other methods for binding in neural networks [46–49], formulating inference as a mechanism for variable binding allows us to model uncertainty in the value of the variables.

3 Problem Formulation

Let x^* denote a physical scene and $h^*_{1:K}$ denote the objects in the scene. Let X and A be random variables for the image observation of the scene x^* and the agent's actions respectively. In contrast to prior works [10] that use a single latent variable to represent the state of the scene, we use a set of latent random variables $H_{1:K}$ to represent the state of the objects $h^*_{1:K}$. We use the term *object* to refer to h^*_k, which is part of the physical world, and the term *entity* to refer to H_k, which is part of our model of the physical world. The generative distribution of observations $X^{(0:T)}$ and latent entities $H^{(0:T)}_{1:K}$ from taking T actions $a^{(0:T-1)}$ is modeled as:

$$p\left(X^{(0:T)}, H^{(0:T)}_{1:K} \;\middle|\; a^{(0:T-1)}\right) = p\left(H^{(0)}_{1:K}\right) \prod_{t=1}^{T} p\left(H^{(t)}_{1:K} \;\middle|\; H^{(t-1)}_{1:K}, a^{(t-1)}\right) \prod_{t=0}^{T} p\left(X^{(t)} \;\middle|\; H^{(t)}_{1:K}\right) \quad (1)$$

where $p(X^{(t)} \mid H^{(t)}_{1:K})$ and $p(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, A^{(t-1)})$ are the observation and dynamics distribution respectively shared across all timesteps t. Our goal is to build a model that, from simply observing raw observations of random interactions, can generalize to solve novel compositional object manipulation problems that the learner was never trained to do, such as building various block towers during test time from only training to predict how blocks fall during training time.

When all tasks follow the same dynamics we can achieve such generalization with a planning algorithm if given a sequence of actions we could compute $p(X^{(T+1:T+d)} \mid X^{(0:T)}, A^{(0:T+d-1)})$, the posterior predictive distribution of observations d steps into the future. Approximating this predictive distribution can be cast as a variational inference problem (Appdx. 6.2) for learning the parameters of an approximate observation distribution $\mathcal{G}(X^{(t)} \mid H^{(t)}_{1:K})$, dynamics distribution $\mathcal{D}(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, A^{(t-1)})$, and a time-factorized recognition distribution $\mathcal{Q}(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, X^{(t)}, A^{(t-1)})$ that maximize the evidence lower bound (ELBO), given by $\mathcal{L} = \sum_{t=0}^{T} \mathcal{L}^{(t)}_{\mathrm{r}} - \mathcal{L}^{(t)}_{\mathrm{c}}$, where

$$\mathcal{L}^t_{\mathrm{r}} = \mathbb{E}_{h^t_{1:K} \sim q\left(H^t_{1:K} \mid h^{0:t-1}_{1:K}, x^{1:t}, a^{0:t-1}\right)} \left[\log \mathcal{G}\left(x^t \mid h^t_{1:K}\right)\right]$$

$$\mathcal{L}^t_{\mathrm{c}} = \mathbb{E}_{h^{t-1}_{1:K} \sim q\left(H^{t-1}_{1:K} \mid h^{1:t-2}_{1:K}, x^{1:t-1}, a^{0:t-2}\right)} \left[D_{KL}\left(\mathcal{Q}\left(H^t_{1:K} \mid h^{t-1}_{1:K}, x^t, a^{t-1}\right) || \, \mathcal{D}\left(H^t_{1:K} \mid h^{t-1}_{1:K}, a^{t-1}\right)\right)\right].$$

The ELBO pushes $\mathcal{Q}$ to produce states of the entities $H_{1:K}$ that contain information useful for not only reconstructing the observations via $\mathcal{G}$ in $\mathcal{L}^{(t)}_{\mathrm{r}}$ but also for predicting the entities' future states via $\mathcal{D}$ in $\mathcal{L}^{(t)}_{\mathrm{c}}$. Sec. 4 will next offer our method for incorporating entity abstraction into modeling the generative distribution and optimizing the ELBO.

4 Object-Centric Perception, Prediction, and Planning (OP3)

The *entity abstraction* is derived from an assumption about symmetry: that the problem of modeling a dynamic scene of multiple entities can be reduced to the problem of (1) modeling a single entity and its interactions with an *entity-centric* function and (2) applying this function to every entity in the scene. Our choice to represent a scene as a set of entities exposes an avenue for directly encoding such a prior about symmetry that would otherwise not be straightforward with a global state representation.

As shown in Fig. 2, a function F that respects the entity abstraction requires two ingredients. The first ingredient (Sec. 4.1) is that $F(H_{1:K})$ is expressed in part as the higher-order operation $\texttt{map}(f, H_{1:K})$ that broadcasts the same entity-centric function $f(H_k)$ to every entity variable H_k. This yields the benefit of automatically transferring learned knowledge for modeling an individual entity to all entities in the scene rather than learn such symmetry from data. As f is a function that takes in a single generic entity variable H_k as argument, the second ingredient (Sec. 4.2) is a mechanism that binds information from the raw observation X about a particular object h^*_k to the variable H_k.

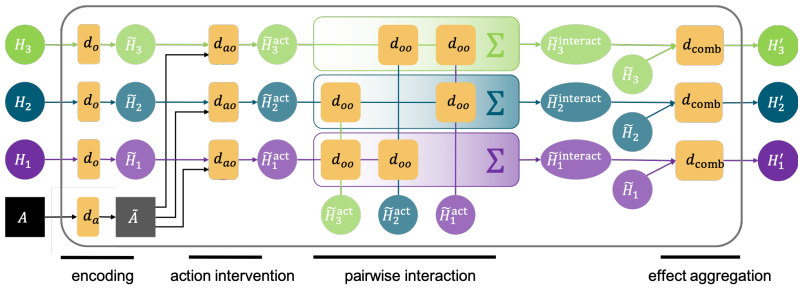

Figure 4: The **dynamics model** $\mathcal{D}$ models the time evolution of every object by symmetrically applying the function $\mathcal{d}$ to each object. For a given object, d models the individual dynamics of that object (d_o), embeds the action vector (d_a), computes the action's effect on that object (d_{ao}), computes each of the other objects' effect on that object (d_{oo}), and aggregates these effects together (d_{comb}).

4.1 Entity Abstraction in the Observation and Dynamics Models

The functions of interest in model-based RL are the observation and dynamics models $\mathcal{G}$ and $\mathcal{D}$ with which we seek to approximate the data-generating distribution in equation 1.

Observation Model: The observation model $\mathcal{G}(X \mid H_{1:K})$ approximates the distribution $p(X \mid H_{1:K})$, which models how the observation X is caused by the combination of entities $H_{1:K}$. We enforce the entity abstraction in $\mathcal{G}$ by applying the same entity-centric function $g(X \mid H_k)$ to each entity H_k, which we can implement using a mixture model at each pixel (i, j):

$$p\left(X_{(ij)} \mid H_{1:K}\right) = \sum_{k=1}^{K} m_{(ij)}\left(H_k\right) \cdot g\left(X_{(ij)} \mid H_k\right), \tag{2}$$

where g computes the mixture components that model how each individual entity H_k is independently generated, combined via mixture weights m that model the entities' relative depth from the camera, the derivation of which is in Appdx. 6.1.1.

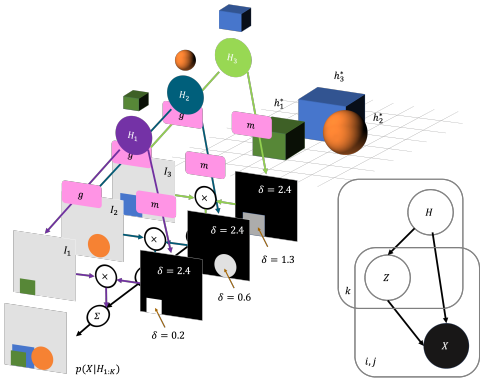

Figure 3: **(a)** The observation model $\mathcal{G}$ models an observation image as a composition of sub-images weighted by segmentation masks. The shades of gray in the masks indicate the depth δ from the camera of the object that the sub-image depicts. **(b)** The graphical model of the generative model of observations, where k indexes the entity, and i, j indexes the pixel. Z is the indicator variable that signifies whether an object's depth at a pixel is the closest to the camera.

Dynamics Model: The dynamics model $\mathcal{D}(H'_{1:K} \mid H_{1:K}, A)$ approximates the distribution $p(H'_{1:K} \mid H_{1:K}, A)$, which models how an action A intervenes on the entities $H_{1:K}$ to produce their future values $H'_{1:K}$. We enforce the entity abstraction in $\mathcal{D}$ by applying the same entity-centric function $\mathcal{d}(H'_k \mid H_k, H_{[\neq k]}, A)$ to each entity H_k, which reduces the problem of modeling how an action affects a scene with a combinatorially large space of object configurations to the problem of simply modeling how an action affects a single *generic* entity H_k and its interactions with the list of other entities $H_{[\neq k]}$. Modeling the action as an finer-grained intervention on a *single* entity rather than the entire scene is a benefit of using local representations of entities rather than global representations of scenes.

However, at this point we still have to model the combinatorially large space of *interactions* that a single entity could participate in. Therefore, we can further enforce the entity abstraction on $\mathcal{d}$ by applying the same *pairwise* function $\mathcal{d}_{oo}(H_k, H_i)$ to each entity pair (H_k, H_i), for $i \in [\neq k]$. Omitting the action to reduce clutter (the full form is written in Appdx. 6.1.2), the structure of the $\mathcal{D}$ therefore follows this form:

$$\mathcal{D}\left(H'_{1:K} \mid H_{1:K}\right) = \prod_{k=1}^{K} \mathcal{d}\left(H'_k \,\middle|\, H_k, H_k^{\text{interact}}\right), \text{ where } H_k^{\text{interact}} = \sum_{i \neq k}^{K} d_{oo}\left(H_i, H_k\right). \tag{3}$$

The entity abstraction therefore provides the flexibility to scale to modeling a variable number of objects by solely learning a function $\mathcal{d}$ that operates on a single generic entity and a function $\mathcal{d}_{oo}$ that operates on a single generic entity pair, both of which can be re-used for across all entity instances.

4.2 Interactive Inference for Binding Object Properties to Latent Variables

For the observation and dynamics models to operate from raw pixels hinges on the ability to bind the properties of specific physical objects $h^*_{1:K}$ to the entity variables $H_{1:K}$. For latent variable models,

we frame this variable binding problem as an inference problem: binding information about $h^*_{1:K}$ to $H_{1:K}$ can be cast as a problem of inferring the parameters of $p(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T-1)})$, the posterior distribution of $H_{1:K}$ given a sequence of interactions. Maximizing the ELBO in Sec. 3 offers a method for learning the parameters of the observation and dynamics models while simultaneously learning an approximation to the posterior $q(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T-1)}) = \prod_{t=0}^{T} \mathcal{Q}(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, x^{(t)}, a^{(t)})$, which we have chosen to factorize into a per-timestep recognition distribution $\mathcal{Q}$ shared across timesteps. We also choose to enforce the entity abstraction on the process that computes the recognition distribution $\mathcal{Q}$ by decomposing it into a recognition distribution q applied to each entity:

$$\mathcal{Q}\left(H^{(t)}_{1:K} \mid h^{(t-1)}_{1:K}, x^{(t)}, a^{(t)}\right) = \prod_{k=1}^{K} q\left(H^{(t)}_{k} \mid h^{(t-1)}_{k}, x^{(t)}, a^{(t)}\right). \tag{4}$$

Whereas a neural network encoder is often used to approximate the posterior [10, 19, 50], a forward pass that computes q in parallel for each entity is insufficient to break the symmetry for dividing responsibility of modeling different objects among the entity variables [51] because the entities do not have the opportunity to communicate about which part of the scene they are representing.

We therefore adopt an *iterative inference* approach [1] to compute the recognition distribution $\mathcal{Q}$, which has been shown to break symmetry among modeling objects in static scenes [3]. Iterative inference computes the recognition distribution via a *procedure*, rather than a single forward pass of an encoder, that iteratively refines an initial guess for the posterior parameters $\lambda_{1:K}$ by using gradients from how well the generative model is able to predict the observation based on the current posterior estimate. The initial guess provides the noise to break the symmetry.

For scenes where position and color are enough for disambiguating objects, a static image may be sufficient for inferring q. However, in interactive environments disambiguating objects is more underconstrained because what constitutes an object depends on the goals of the agent. We therefore incorporate actions into the amortized varitional filtering framework [2] to develop an *interactive inference* algorithm (Appdx. 6.3.1 and Fig. 5) that uses temporal continuity and interactive feedback to disambiguate objects. Another benefit of enforcing entity abstraction is that preserving temporal consistency on entities comes for free: information about each object remains bound to its respective H_k through time, mixing with information about other entities only through explicitly defined avenues, such as in the dynamics model.

4.3 Training at Different Timescales

The variational parameters $\lambda_{1:K}$ are the interface through which the neural networks f_g, f_d, f_q that output the distribution parameters of $\mathcal{G}$, $\mathcal{D}$, and $\mathcal{Q}$ communicate. For a particular dynamic scene, the execution of interactive inference optimizes the variational parameters $\lambda_{1:K}$. *Across* scene instances, we train the weights of f_g, f_d, f_q by backpropagating the ELBO through the entire inference procedure, spanning multiple timesteps. OP3 thus learns at three different timescales: the variational parameters learn (1) across M steps of inference within a single timestep and (2) across T timesteps within a scene instance, and the network weights learn (3) across different scene instances.

Beyond next-step prediction, we can directly train to compute the posterior predictive distribution $p(X^{(T+1:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$ by sampling from the approximate posterior of $H^{(T)}_{1:K}$ with $\mathcal{Q}$, rolling out the dynamics model $\mathcal{D}$ in latent space from these samples with a sequence of d actions, and predicting the observation $X^{(T+d)}$ with the observation model $\mathcal{G}$. This approach to action-conditioned video prediction predicts future raw observations directly from raw observations and actions, but with a bottleneck of K time-persistent entity-variables with which the dynamics model $\mathcal{D}$ performs symbolic relational computation.

4.4 Object-Centric Planning

OP3 rollouts, computed as the posterior predictive distribution, can be integrated into the standard visual model-predictive control [9] framework. Since interactive inference grounds the entities $H_{1:K}$ in the actual objects $h^*_{1:K}$ depicted in the raw observation, this grounding essentially gives OP3 access to a *pointer* to each object, enabling the rollouts to be in the space of entities and their relations. These pointers enable OP3 to not merely predict in the space of entities, but give OP3 access to an *object-centric action space*: for example, instead of being restricted to the standard (`pick_xy`, `place_xy`) action space common to many manipulation tasks, which often requires biased picking with a scripted policy [52, 53], these pointers enable us to compute a mapping (Appdx. 6.5.2) between

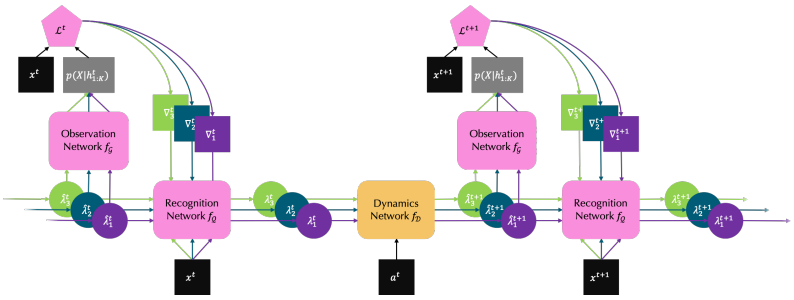

Figure 5: **Amortized interactive inference** alternates between refinement (pink) and dynamics (orange) steps, iteratively updating the belief of $\lambda_{1:K}$ over time. $\hat{\lambda}$ corresponds to the output of the dynamics network, which serves as the initial estimate of λ that is subsequently refined by $f_{\mathcal{Q}}$ and $f_{\mathcal{Q}}$. ∇ denotes the feedback used in the refinement process, which includes gradient information and auxiliary inputs (Appdx. 6.3.1).

`entity_id` and `pick_xy`, allowing OP3 to automatically use a `(entity_id, place_xy)` action space without needing a scripted policy.

4.5 Generalization to Various Tasks

We consider tasks defined in the same environment with the same physical laws that govern appearance and dynamics. Tasks are differentiated by goals, in particular goal configurations of objects. Building good cost functions for real world tasks is generally difficult [54] because the underlying state of the environment is always unobserved and can only be modeled through modeling observations. However, by representing the environment state as the state of its entities, we may obtain finer-grained goal-specification without the need for manual annotations [55]. Having rolled out OP3 to a particular timestep, we construct a cost function to compare the predicted entity states $H^{(P)}_{1:K}$ with the entity states $H^{(G)}_{1:K}$ inferred from a goal image by considering pairwise distances between the entities, another example of enforcing the entity abstraction. Letting S' and S denote the set of goal and predicted entities, we define the form of the cost function via a composition of the task specific distance function c operating on entity-pairs:

$$\mathcal{C}\left(H^{(G)}_{1:K}, H^{(P)}_{1:K}\right) = \sum_{a \in S'} \min_{b \in S} c\left(H^{(G)}_{a}, H^{(P)}_{b}\right), \tag{5}$$

in which we pair each goal entity with the closest predicted entity and sum over the costs of these pairs. Assuming a single action suffices to move an object to its desired goal position, we can greedily plan each timestep by defining the cost to be $\min_{a \in S', b \in S} c(H^{(G)}_a, H^{(P)}_b)$, the pair with minimum distance, and removing the corresponding goal entity from further consideration for future planning.

5 Experiments

Our experiments aim to study the various ways in which entity abstraction improves generalization, planning, and modeling. Sec. 5.1 shows that from only training to predict how objects fall, OP3 generalizes to solve various novel block stacking tasks with three times better accuracy than a state-of-the-art video prediction model. Sec. 5.2 shows that OP3 can plan for multiple steps in a difficult multi-object environment. Sec. 5.3 shows that OP3 learns to ground its abstract entities in objects from real world videos.

5.1 Combinatorial Generalization without Object Supervision

We first investigate how well OP3 can learn object-based representations without additional object supervision, as well as how well OP3's factorized representation can enable combinatorial generalization for scenes with many objects.

Domain: In the MuJoCo [56] block stacking task introduced by Janner et al. [33] for the O2P2 model, a block is raised in the air and the model must predict the steady-state effects of dropping the block on a surface with multiple objects, which implicitly requires modeling the effects of gravity and collisions. The agent is never trained to stack blocks, but is tested on a suite of tasks where it must construct block tower specified by a goal image. Janner et al. [33] showed that an object-centric model with access to *ground truth* object segmentations can solve these tasks with about 76% accuracy. We now consider whether OP3 can do better, but *without any supervision on object identity*.

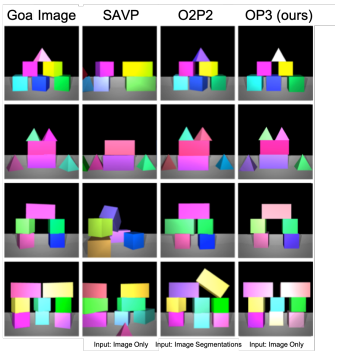

Figure 6: Respective results of our method in comparison to prior work. OP3 outperforms O2P2 even though it does not have access to any object segmentations.

SAVP	O2P2	OP3 (ours)
24%	76%	**82%**

Table 1: Accuracy (%) of block tower builds by the SAVP baseline, the O2P2 oracle, and our approach. O2P2 uses image segmentations whereas OP3 uses only raw images as input.

# Blocks	SAVP	OP3 (xy)	OP3 (entity)
1	54%	73%	**91%**
2	28%	55%	**80%**
3	28%	41%	**55%**

Table 2: Accuracy (%) of multi-step planning for building block towers. (xy) means `(pick_xy, place_xy)` action space while (entity) means `(entity_id, place_xy)` action space.

Setup: We train OP3 on the same dataset and evaluate on the same goal images as Janner et al. [33]. While the training set contains up to five objects, the test set contains up to nine objects, which are placed in specific structures (bridge, pyramid, etc.) not seen during training. The actions are optimized using the cross-entropy method (CEM) [57], with each sampled action evaluated by the greedy cost function described in Sec. 4.5. Accuracy is evaluated using the metric defined by Janner et al. [33], which checks that all blocks are within some threshold error of the goal.

Results: The two baselines, SAVP [6] and O2P2, represent the state-of-the-art in video prediction and symmetric object-centric planning methods, respectively. SAVP models objects with a fixed number of convolutional filters and does not process entities symmetrically. O2P2 does process entities symmetrically but has access to ground truth object segmentations. As shown in Table 1, OP3 achieves better accuracy than O2P2, even without any ground truth supervision on object identity. OP3 achieves three times the accuracy of SAVP, which suggests that symmetric modeling of entities is enables the flexibility to transfer knowledge of dynamics of a single object to novel scenes with different configurations heights, color combinations, and numbers of objects than those from the training distribution. Fig. 11 and Fig. 10 in the Appendix show that, by grounding its entities in objects of the scene through inference, OP3's predictions isolates only one object at a time without affecting the predictions of other objects.

5.2 Multi-Step Planning

The goal of our second experiment is to understand how well OP3 can perform multi-step planning. The task in the previous section can be performed with a greedy planning algorithm: each block placement can be selected to maximally reduce the difference between the current and goal scene. In this next experiment, we modify the block stacking to require our model to reason over temporally extended action sequences on objects already present in the scene, by changing the action space to represent a picking and dropping location.

Goals are specified with a goal image, and the initial scene contains all of the blocks needed to build the desired structure. This task is more difficult because the agent may have to move blocks out of the way before placing other ones which would require multi-step planning. Furthermore, an action only successfully picks up a block if it intersects with the block's outline, which makes searching through the combinatorial space of plans a challenge. As stated in Sec. 4.4, having a pointer to each object enables OP3 to plan in the space of entities. We compare two different action spaces `(pick_xy, place_xy)` and `(entity_id, place_xy)` to understand how automatically filtering for pick locations at actual locations of objects enables better efficiency and performance in planning. Details for determining the `pick_xy` from `entity_id` are in the appendix.

Results: We compare with SAVP, which uses the `(pick_xy, place_xy)` action space. With this standard action space (Table 2) OP3 achieves between 1.5-2 times the accuracy of SAVP. This performance gap increases to 2-3 times the accuracy when OP3 uses the `(entity_id, place_xy)` action space. The low performance of SAVP with only two blocks highlights the difficulty of such combinatorial tasks for model-based RL methods, and highlights the both the generalization and localization benefits of a model with entity abstraction. Fig. 7 shows that with the same number of planning steps, SAVP is unable to find good enough plans to construct the goal configuration, whereas OP3 is able to plan more efficiently, suggesting that OP3 may be a more effective model

than SAVP in modeling combinatorial scenes. Fig. 8 shows the execution of interactive inference during training, where OP3 alternates between four refinement steps and one prediction step. Notice that OP3 infers entity representations that decompose the scene into coherent objects and that entities that do not model objects model the background. Notice also in the last column ($t = 2$) that OP3 predicts the appearance of the green block even though the green block was partially occluded in the previous timestep, which is expected because the latent entity variables in OP3 persist through time.

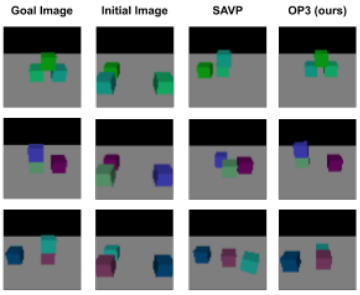

Figure 7: End result of multi-step planning.

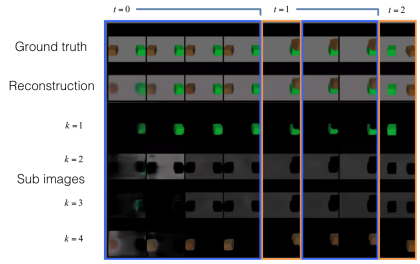

Figure 8: Visualization of multi-step planning. Blue boxes correspond to refinement steps within the same timestep while orange boxes correspond to predicting the future timestep.

5.3 Real World Evaluation

The previous tasks used simulated environments with monochromatic objects. Now we study how well OP3 scales to real world data with cluttered scenes, object ambiguity, and occlusions. We evaluate OP3 on the dataset from Ebert et al. [58] which contains videos of a robotic arm moving cloths and other deformable and multipart objects with varying textures.

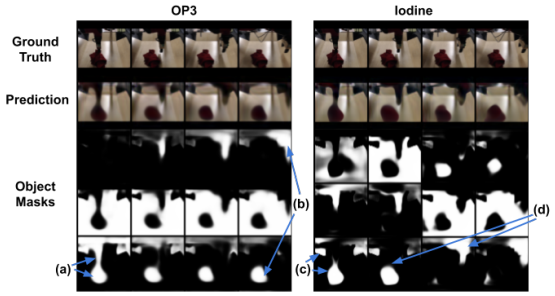

Figure 9: Qualitative results on learning object representations on real world data, showing learned object masks for IODINE and our method. OP3 initially segments the arm and object in the same mask (a), then separates them out after incorporating actions (b). IODINE initially segments multiple objects with the same mask (c) and also does not have temporal consistency (d).

We evaluate qualitative performance by visualizing the object segmentations and compare against vanilla IODINE, which does not incorporate a interaction-based dynamics model into the inference process. Fig. 9 highlights the strength of OP3 in preserving temporal continuity and disambiguating objects in real world scenes. While IODINE can disambiguate monochromatic objects in static images, we observe that it struggles to do more than just color segmentation on more complicated images where movement is required to disambiguate objects. In contrast, whereas OP3 also initially performed color segmentation by grouping the towel, arm, and dark container edges together, by observing the effects of moving the arm it separates these entities into different groups.

6 Discussion

In the physical world, objects can be viewed as variables that are symmetrically processed by the same physical laws. Modeling such entities as an abstraction on the state is a modeling choice, in the same way that rewards are an abstraction on the state that indicates performance, that actions are an abstraction on the state that indicates avenues for intervention. As we have shown with the various generalization, planning, and modeling benefits of OP3 in various novel compositional multi-object tasks, abstraction may provide benefit in grouping shared structure in ways that can be modeled generically rather specifically for each instance. What often makes abstraction difficult, especially that which enables symbolic computation like the relational computation OP3 performs, is not so much the lossiness of abstraction but the inability to continuously update the abstract representation with more raw data. Framing abstract variables as random variables in a graphical model whose posterior can be inferred and refined over time from raw data using neural networks, as OP3 does, can provide a potential bridge between the symbolic world and the noisy high dimensional physical world, opening a path to scaling robotic learning to more combinatorially complex tasks.

Acknowledgments

The authors would like to thank the anonymous workshop reviewers for their helpful feedback and comments. The authors would also like to thank Sjoerd van Steenkiste, Nalini Singh and Marvin Zhang for helpful discussions on the graphical model, Klaus Greff for help in implementing IODINE, Tom Griffiths, Karl Persch, and Oleg Rybkin for feedback on earlier drafts, Joe Marino for discussions on iterative inference, and Sam Toyer, Anirudh Goyal, and Peter Battaglia for insightful discussions. This research was supported in part by the National Science Foundation under IIS-1651843, IIS-1700697, and IIS-1700696, the Office of Naval Research, ARL DCIST CRA W911NF-17-2-0181, DARPA, Berkeley DeepDrive, Google, Amazon, and NVIDIA.

References

[1] J. Marino, Y. Yue, and S. Mandt. Iterative amortized inference. *arXiv:1807.09356*, 2018.

[2] J. Marino, M. Cvitkovic, and Y. Yue. A general method for amortizing variational filtering. In *Advances in Neural Information Processing Systems*, pages 7857–7868, 2018.

[3] K. Greff, R. L. Kaufmann, R. Kabra, N. Watters, C. Burgess, D. Zoran, L. Matthey, M. Botvinick, and A. Lerchner. Multi-object representation learning with iterative variational inference. *arXiv:1903.00450*, 2019.

[4] N. Wichers, R. Villegas, D. Erhan, and H. Lee. Hierarchical long-term video prediction without supervision. *arXiv:1806.04768*, 2018.

[5] E. L. Denton et al. Unsupervised learning of disentangled representations from video. In *Advances in neural information processing systems*, pages 4414–4423, 2017.

[6] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine. Stochastic adversarial video prediction. *arXiv:1804.01523*, 2018.

[7] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[8] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

[9] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE, 2017.

[10] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. *arXiv:1811.04551*, 2018.

[11] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine. Solar: Deep structured latent representations for model-based reinforcement learning. *arXiv:1808.09105*, 2018.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[13] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. *arXiv:1605.09128*, 2016.

[14] K. Greff, R. K. Srivastava, and J. Schmidhuber. Binding via reconstruction clustering. *arXiv:1511.06418*, 2015.

[15] F. Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[16] W. F. Whitney, M. Chang, T. Kulkarni, and J. B. Tenenbaum. Understanding visual concepts with continuation learning. *arXiv:1602.06822*, 2016.

[17] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[18] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015.

[19] T. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih. Unsupervised learning of object keypoints for perception and control. *arXiv:1906.11883*, 2019.

[20] V. Goel, J. Weng, and P. Poupart. Unsupervised video object segmentation for deep reinforcement learning. *arXiv:1805.07780*, 2018.

[21] Z. Xu, Z. Liu, C. Sun, K. Murphy, W. T. Freeman, J. B. Tenenbaum, and J. Wu. Unsupervised discovery of parts, structure, and dynamics. *arXiv:1903.05136*, 2019.

[22] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv:1612.00341*, 2016.

[23] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

[24] J. B. Hamrick, A. J. Ballard, R. Pascanu, O. Vinyals, N. Heess, and P. W. Battaglia. Metacontrol for adaptive imagination-based optimization. *arXiv:1705.02670*, 2017.

[25] M. Janner, K. Narasimhan, and R. Barzilay. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics*, 6:49–61, 2018.

[26] K. Narasimhan, R. Barzilay, and T. Jaakkola. Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874, 2018.

[27] V. Bapst, A. Sanchez-Gonzalez, C. Doersch, K. L. Stachenfeld, P. Kohli, P. W. Battaglia, and J. B. Hamrick. Structured agents for physical construction. *arXiv:1904.03177*, 2010.

[28] A. Ajay, M. Bauza, J. Wu, N. Fazeli, J. B. Tenenbaum, A. Rodriguez, and L. P. Kaelbling. Combining physical simulators and object-based networks for control. *arXiv:1904.06580*, 2019.

[29] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[30] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[31] D. Wang, C. Devin, Q.-Z. Cai, F. Yu, and T. Darrell. Deep object centric policies for autonomous driving. *arXiv:1811.05432*, 2018.

[32] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi. Visual semantic navigation using scene priors. *arXiv:1810.06543*, 2018.

[33] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu. Reasoning about physical interactions with object-oriented prediction and planning. *arXiv:1812.10972*, 2018.

[34] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. *arXiv:1511.07404*, 2015.

[35] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017.

[36] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv:1706.04317*, 2017.

[37] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *arXiv:1706.01427*, 2017.

[38] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, et al. Deep reinforcement learning with relational inductive biases. 2018.

[39] Y. Du and K. Narasimhan. Task-agnostic dynamics priors for deep reinforcement learning. *arXiv:1905.04819*, 2019.

[40] K. Greff, S. van Steenkiste, and J. Schmidhuber. Neural expectation maximization. 2017.

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

[42] G. Zhu, J. Wang, Z. Ren, and C. Zhang. Object-oriented dynamics learning through multi-level abstraction. *arXiv:1904.07482*, 2019.

[43] S. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, G. E. Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pages 3225–3233, 2016.

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

[45] A. R. Kosiorek, H. Kim, I. Posner, and Y. W. Teh. Sequential attend, infer, repeat: Generative modelling of moving objects. *arXiv:1806.01794*, 2018.

[46] S. D. Levy and R. Gayler. Vector symbolic architectures: A new building material for artificial general intelligence. In *Conference on Artificial General Intelligence*, 2008.

[47] P. Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 2009.

[48] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

[49] 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.

[50] Z. Xu, Z. Liu, C. Sun, K. Murphy, W. T. Freeman, J. B. Tenenbaum, and J. Wu. Modeling parts, structure, and system dynamics via predictive learning. 2018.

[51] Y. Zhang, J. Hare, and P.-B. Adam. Deep set prediction networks. *arXiv:1906.06565*, 2019.

[52] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[53] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[54] J. Fu, A. Singh, D. Ghosh, L. Yang, and S. Levine. Variational inverse control with events: A general framework for data-driven reward definition. In *Advances in Neural Information Processing Systems*, pages 8538–8547, 2018.

[55] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv:1812.00568*, 2018.

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

[57] R. Y. Rubinstein and D. P. Kroese. The cross-entropy method. In *Information Science and Statistics*, 2004.

[58] F. Ebert, S. Dasari, A. X. Lee, S. Levine, and C. Finn. Robustness via retrying: Closed-loop robotic manipulation with self-supervised learning. *arXiv:1810.03043*, 2018.

[59] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[60] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[61] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[62] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2): 270–280, 1989.

[63] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[64] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *ArXiv*, abs/1211.5063, 2012.

Supplementary Material

6.1 Generative Model

Here we describe in detail the functional form of the observation and dynamics models.

6.1.1 Observation Model

The observation model $\mathcal{G}$ models how the objects $H_{1:K}$ cause the image observation $X \in \mathbb{R}^{N \times M}$. Each object H_k is rendered independently as the sub-image I_k and the resulting K sub-images are combined to form the final image observation X. To combine the sub-images, each pixel $I_{k(ij)}$ in each sub-image is assigned a depth $\delta_{k(ij)}$ that specifies the distance of object k from the camera at coordinate (ij). of the image plane. Thus the pixel $X_{(ij)}$ takes on the value of its corresponding pixel $I_{k(ij)}$ in the sub-image I_k if object k is closest to the camera than the other objects, such that

$$X_{(ij)} = \sum_{k=1}^{K} Z_{k(ij)} \cdot I_{k(ij)}, \tag{6}$$

where $Z_{k(ij)}$ is the indicator random variable $\mathbb{1}[k = \operatorname{argmin}_{k \in K} \delta_{k(ij)}]$, allowing us to intuitively interpret Z_k as segmentation masks and I_k as color maps. In reality we do not directly observe the depth values, so we must construct a probabilistic model to model our uncertainty:

$$p\left(X | H_{1:K}\right) = \prod_{i,j=1}^{N,M} \sum_{k=1}^{K} m_{(ij)}(H_k) \cdot g\left(X_{(ij)} \mid H_k\right), \tag{7}$$

where every pixel (ij) is modeled through a set of mixture components $g\left(X_{(ij)} \mid H_k\right) := p\left(X_{ij} | Z_{k(ij)} = 1, H_k\right)$ that model how pixels of the individual sub-images I_k are generated, as well as through the mixture weights $m_{ij}(H_k) := p\left(Z_{k(ij)} = 1 | H_k\right)$ that model which point of each object is closest to the camera.

6.1.2 Dynamics Model

The dynamics model $\mathcal{D}$ models how each object H_k is affected by action A and the other objects $H_{[\neq k]}$. It applies the same function $\mathcal{d}(H'_k \mid H_k, H_{[\neq k]}, A)$ to each state, composed of several functions illustrated and described in Fig. 4:

$$\begin{aligned}
&\tilde{H}_k = d_o(H_k) \qquad \tilde{A} = d_a(A^t) \qquad \tilde{H}_k^{\text{act}} = d_{ao}(\tilde{H}_k \tilde{A}) \\
&H_k^{\text{interact}} = \sum_{i \neq k}^{K} d_{oo}(\tilde{H_i}^{\text{act}}, \tilde{H_k}^{\text{act}}) \qquad H_k^{'} = d_{\text{comb}}(\tilde{H}_k^{\text{act}}, H_k^{\text{interact}}),
\end{aligned}$$

where for a given object k, $d_{ao}(\tilde{H}_k \tilde{A}) := d_{\text{act-eff}}(\tilde{H}_k, \tilde{A}) \cdot d_{\text{act-att}}(\tilde{H}_k, \tilde{A})$ computes how $(d_{\text{act-eff}})$ and to what degree $(d_{\text{act-att}})$ an action affects the object and $d_{oo}(\tilde{H_i}^{\text{act}}, \tilde{H_k}^{\text{act}}) := d_{\text{obj-eff}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}}) \cdot d_{\text{obj-att}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}})$ computes how $(f_{\text{obj-eff}})$ and to what degree $(d_{\text{obj-att}})$ other objects affect that object. $d_{\text{obj-eff}}$ and $d_{\text{obj-att}}$ are shared across all object pairs. The other functions are shared across all objects.

6.2 Problem Formulation

Here we provide a derivation of the evidence lower bound and the posterior predictive distribution for dynamic latent variable model with multiple latent states.

6.2.1 Evidence Lower Bound

We begin with the log probability of the observations $X^{(1:T)}$ conditioned on a sequence of actions $a^{(0:T-1)}$:

$$\begin{aligned}
\log p\left(X^{(0:T)} \;\middle|\; a^{(0:T-1)}\right) &= \log \int_{h_{1:K}^{(0:T)}} p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right)\, dh_{1:K}^{(0:T)}. \\
&= \log \int_{h_{1:K}^{(0:T)}} p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right) \frac{q\left(h_{1:K}^{(0:T)} \;\middle|\; \cdot\right)}{q\left(h_{1:K}^{(0:T)} \;\middle|\; \cdot\right)}\, dh_{1:K}^{(0:T)}. \\
&= \log \mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(h_{1:K}^{(0:T)} \,\middle|\, \cdot\right)} \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right)}{h_{1:K}^{(0:T)} \sim q\left(h_{1:K}^{(0:T)} \;\middle|\; \cdot\right)}\right] \\
&\geq \mathbb{E}_{q\left(h_{1:K}^{(0:T)} \,\middle|\, \cdot\right)} \log \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right)}{q\left(h_{1:K}^{(0:T)} \;\middle|\; \cdot\right)}\right]. \qquad (8)
\end{aligned}$$

We have freedom to choose the approximating distribution $q\left(H_{1:K}^{(0:T)} \;\middle|\; \cdot\right)$ so we choose it to be conditioned on the past states and actions, factorized across time:

$$q\left(H^{(0:T)} \;\middle|\; x^{(0:T)}, a^{(0:T)}\right) = q\left(H_{1:K}^{(0)} \mid x^{(0)}\right) \prod_{t=1}^{T} q\left(H_{1:K}^{(t)} \;\middle|\; H_{1:K}^{(t-1)}, x^{(t)}, a^{(t-1)}\right)$$

With this factorization, we can use linearity of expectation to decouple Equation 8 across timesteps:

$$\mathbb{E}_{q\left(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T)}\right)} \log \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \;\middle|\; a^{(0:T-1)}\right)}{q\left(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T)}\right)}\right] = \sum_{t=0}^{(t)} \mathcal{L}_{\mathrm{r}}^{(t)} - \mathcal{L}_{\mathrm{c}}^{(t)},$$

where at the first timestep

$$\begin{aligned}
\mathcal{L}_{\mathrm{r}}^{(0)} &= \mathbb{E}_{h_{1:K}^{(0)} \sim q\left(H_{1:K}^{(0)} \mid X^{(0)}\right)} \left[\log p\left(X^{(0)} \;\middle|\; h_{1:K}^{(0)}\right)\right] \\
\mathcal{L}_{\mathrm{c}}^{(0)} &= D_{KL}\left(q\left(H_{1:K}^{(0)} \;\middle|\; X^{(0)}\right) || \, p\left(H_{1:K}^{(0)}\right)\right)
\end{aligned}$$

and at subsequent timesteps

$$\begin{aligned}
\mathcal{L}_{\mathrm{r}}^{(t)} &= \mathbb{E}_{h_{1:K}^{(t)} \sim q\left(H_{1:K}^{(t)} \mid h_{1:K}^{(0:t-1)}, X^{(0:t)}, a^{(0:t-1)}\right)} \left[\log p\left(X^{(t)} \;\middle|\; h_{1:K}^{(t)}\right)\right] \\
\mathcal{L}_{\mathrm{c}}^{(t)} &= \mathbb{E}_{h_{1:K}^{(t-1)} \sim q\left(H_{1:K}^{(t-1)} \mid h_{1:K}^{(0:t-2)}, X^{(1:t-1)}, a^{(0:t-2)}\right)} \left[D_{KL}\left(q\left(H_{1:K}^{(t)} \;\middle|\; h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)}\right) || \, p\left(H_{1:K}^{(t)} \;\middle|\; h_{1:K}^{(t-1)}, a^{(t-1)}\right)\right)\right].
\end{aligned}$$

By the Markov property, the marginal $q(H_{1:K}^{(t)} \mid h_{1:K}^{(0:t-1)}, X^{(0:t)}, a^{(0:t-1)})$ is computed recursively as

$$\mathbb{E}_{h^{(t-1)} \sim q\left(H_{1:K}^{(t-1)} \mid h_{1:K}^{(0:t-2)}, X^{(0:t-1)}, a^{(0:t-2)}\right)} \left[q\left(H_{1:K}^{(t)} \;\middle|\; h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)}\right)\right]$$

whose base case is $q\left(H^{(0)} \mid X^{(0)}\right)$ when $t = 0$.

Implementation: We approximate observation distribution $p(X \mid H_{1:K})$ and the dynamics distribution $p(H'_{1:K} \mid H_{1:K}, a)$ by learning the parameters of the observation model $\mathcal{G}$ and dynamics model $\mathcal{D}$ respectively as outputs of neural networks. We approximate the recognition distribution $q(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)})$ via an inference procedure that refines better estimates of the posterior parameters, computed as an output of a neural network. To compute the expectation in the marginal $q(H_{1:K}^{(t)} \mid h_{1:K}^{(0:t-1)}, X^{(0:t)}, a^{(0:t-1)})$, we follow standard practice in amortized variational inference by approximating the expectation with a single sample of the sequence $h_{1:K}^{(0:t-1)}$ by sequentially sampling the latents for one timestep given latents from the previous timestep, and optimizing the ELBO via stochastic gradient ascent [59–61].

6.2.2 Posterior Predictive Distribution

Section 6.2.1 described how we compute the distributions $p(X \mid H_{1:K})$, $p(H'_{1:K} \mid H_{1:K}, a)$, $q(H^{(t)}_{1:K} \mid h^{(t-1)}_{1:K}, X^{(t)}, a^{(t-1)})$, and $q(H^{(0:T)}_{1:K} \mid x^{(1:T)}, a^{(1:T)})$. Here we show that these distributions can be used to approximate the predictive posterior distribution $p(X^{(T+1:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$ by maximizing the following lower bound:

$$\begin{aligned}
\log p\left(X^{(T+1:T+d)} \,\middle|\, x^{(0:T)}, a^{(0:T+d)}\right) &= \int_{h^{(0:T+d)}_{1:K}} p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \,\middle|\, x^{(0:T)}, a^{(0:T+d)}\right) \, dh^{(0:T+d)}_{1:K} \\
&= \int_{h^{(0:T+d)}_{1:K}} p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \,\middle|\, x^{(0:T)}, a^{(0:T+d)}\right) \frac{q\left(h^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right)}{q\left(h^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right)} \, dh^{(0:T+d)}_{1:K} \\
&= \log \mathbb{E}_{h^{(0:T+d)}_{1:K} \sim q\left(h^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right)} \left[\frac{p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \,\middle|\, x^{(0:T)}, a^{(0:T+d)}\right)}{q\left(h^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right)}\right] \\
&\geq \mathbb{E}_{h^{(0:T+d)}_{1:K} \sim q\left(h^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right)} \log \left[\frac{p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \,\middle|\, x^{(0:T)}, a^{(0:T+d)}\right)}{q\left(h^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right)}\right].
\end{aligned} \tag{9}$$

The numerator $p(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)})$ can be decomposed into two terms, one of which involving the posterior $p(h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)})$:

$$p\left(X^{(T+1:T+d)}, h^{(0:T+d)}_{1:K} \,\middle|\, x^{(0:T)}, a^{(0:T+d)}\right) = p\left(X^{(T+1:T+d)} \,\middle|\, h^{(0:T+d)}_{1:K}\right) p\left(h^{(0:T+d)}_{1:K} \,\middle|\, x^{(0:T)}, a^{(0:T+d)}\right),$$

This allows Equation 9 to be broken up into two terms:

$$\mathbb{E}_{h^{(0:T+d)}_{1:K} \sim q\left(h^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right)} \log p\left(X^{(T+1:T+d)} \,\middle|\, h^{(0:T+d)}_{1:K}\right) - D_{KL}\left(q\left(H^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right) || \, p\left(H^{(0:T+d)}_{1:K} \,\middle|\, x^{(0:T)}, a^{(0:T+d)}\right)\right)$$

Maximizing the second term, the negative KL-divergence between the variational distribution $q(h^{(0:T+d)}_{1:K} \mid \cdot)$ and the posterior $p(h^{(0:T+d)}_{1:K} \mid x^{(0:T)}, a^{(0:T+d)})$ is the same as maximizing the following lower bound:

$$\mathbb{E}_{h^{(0:T)}_{1:K} \sim q\left(h^{(0:T)}_{1:K} \,\middle|\, \cdot\right)} \log p\left(x^{(0:T)} \,\middle|\, h^{(0:T)}_{1:K}, a^{(0:T-1)}\right) - D_{KL}\left(q\left(H^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right) || \, p\left(H^{(0:T+d)}_{1:K} \,\middle|\, a^{(0:T+d)}\right)\right) \tag{10}$$

where the first term is due to the conditional independence between $X^{(0:T)}$ and the future states $H^{(T+1:T+d)}_{1:K}$ and actions $A^{(T+1:T+d)}$. Note that Equation 10 is not the same as the ELBO in Equation 8 because the KL divergence term is with respect to distributions over $H^{(0:T+d)}_{1:K}$, not $H^{(0:T)}_{1:K}$. We choose to express $q\left(H^{(0:T+d)}_{1:K} \,\middle|\, \cdot\right)$ as conditioned on past states and actions, factorized across time:

$$q\left(H^{(0:T+d)} \,\middle|\, x^{(0:T+d)}, a^{(0:T+d-1)}\right) = q\left(H^{(0)}_{1:K} \mid x^{(0)}\right) \prod_{t=1}^{T+d} q\left(H^{(t)}_{1:K} \,\middle|\, H^{(t-1)}_{1:K}, x^{(t)}, a^{(t-1)}\right)$$

and thus we can maximize Equation 9 using the same techniques as maximizing Equation 8.

6.2.3 Optimization

Whereas approximating the ELBO in Equation 9 can be implemented by rolling out OP3 to predict the next observation via teacher forcing [62], approximating the posterior predictive distribution in Equation 9 can be implemented by rolling out the dynamics model d steps beyond the last observation and using the observation model to predict the future observations.

6.3 Algorithms

This section provides details for the interactive inference algorithm 6.3.1, the training algorithm, and the planning algorithm.

6.3.1 Interactive Inference

Algorithms 1 and 2 detail M steps of the interactive inference algorithm at timestep 0 and $t \in [1, T]$ respectively. Algorithm 1 is equivalent to the IODINE algorithm described [3]. Recalling that $\lambda_{1:K}$ are the parameters for the distribution of the random variables $H_{1:K}$, we consider in this paper the case where this distribution is an isotropic Gaussian (e.g. $\mathcal{N}(\lambda_k)$ where $\lambda_k = (\mu_k, \sigma_k)$), although OP3 need not be restricted to the Gaussian distribution. The *refinement network* f_q produces the parameters for the distribution $q(H_k^{(t)} \mid h_k^{(t-1)}, x^{(t)}, a^{(t)})$. The *dynamics network* f_∂ produces the parameters for the distribution $\partial(H_k^{(t)} \mid h_k^{(t-1)}, h_{[\neq k]}^{(t-1)}, a^{(t)})$. To implement q, we repurpose the dynamics model to transform $h_k^{(t-1)}$ into the initial posterior estimate $\lambda_k^{(0)}$ and then use f_q to iteratively update this parameter estimate. β_k indicates the auxiliary inputs into the refinement network used in [3]. We mark the major areas where the algorithm at timestep t differs from the algorithm at timestep 0 in blue.

Algorithm 1 Interactive Inference: Timestep 0

1: **Input:** observation $x^{(0)}$
2: **Initialize:** parameters $\lambda^{(0,0)}$
3: **for** $i = 0$ **to** $M - 1$ **do**
4: Sample $h_k^{(0,i)} \sim \mathcal{N}\left(\lambda_k^{(0,i)}\right)$ for each entity k
5: Evaluate $\mathcal{L}^{(0,i)} \approx p\left(x^{(0)} \mid h_{1:K}^{(0,i)}\right) - D_{KL}\left(\mathcal{N}\left(\lambda_{1:K}^{(0,i)}\right) || \mathcal{N}(0, I)\right)$
6: Calculate $\nabla_{\lambda_k} \mathcal{L}^{(0,i)}$ for each entity k
7: Assemble auxiliary inputs β_k for each entity k
8: Update $\lambda_k^{(0,i+1)} \leftarrow f_{\text{refine}}\left(x^{(0)}, \nabla_\lambda \mathcal{L}^{(0,i)}, \lambda^{(0,i)}, \beta_k^{(0,i)}\right)$ for each entity k
9: **end for**
10: **return** $\lambda^{(0,M)}$

Algorithm 2 Interactive Inference: Timestep t

1: **Input:** observation $x^{(t)}$, action $a^{(t)}$, previous entity states $h_{1:K}^{(t-1)}$
2: **Predict** $\lambda_k^{(t,0)} \leftarrow f_\partial\left(h_k^{(t-1)}, h_{[\neq k]}^{(t-1)}, a^{(t)}\right)$
3: **for** $i = 0$ **to** $M - 1$ **do**
4: Sample $h_k^{(t,i)} \sim \mathcal{N}\left(\lambda^{(t,i)}\right)$ for each entity k
5: Evaluate $\mathcal{L}^{(t,i)} \approx \log \mathcal{G}\left(x^{(t)} \mid h_{1:K}^{(t)}\right) - D_{KL}\left(\mathcal{N}\left(\lambda_{1:K}^{t,(i)}\right) || \mathcal{N}\left(\lambda_{1:K}^{(t,0)}\right)\right)$
6: Calculate $\nabla_{\lambda_k} \mathcal{L}^{(t,i)}$ for each entity k
7: Assemble auxiliary inputs β_k for each entity k
8: Update $\lambda_k^{(t,i+1)} \leftarrow f_q\left(x^{(t)}, \nabla_{\lambda_k} \mathcal{L}^{(t,i)}, \lambda_k^{(t,i)}, \beta_k^{(t,i)}\right)$ for each entity k
9: **end for**
10: **return** $\lambda^{(t,M)}$

6.3.2 Training

We can train the entire OP3 system end-to-end by backpropagating through the entire inference procedure, using the ELBO at every timestep as a training signal for the parameters of $\mathcal{G}$, $\mathcal{D}$, $\mathcal{Q}$ in a similar manner as [41]. However, the interactive inference algorithm can also be naturally be adapted to predict rollouts by using the dynamics model to propagate the $\lambda_{1:K}$ for multiple steps, rather than just the one step for predicting $\lambda_{1:K}^{(t,0)}$ in line 2 of Algorithm 2. To train OP3 to rollout the dynamics model for longer timescales, we use a curriculum that increases the prediction horizon throughout training.

6.3.3 Cost Function

To compute the cost we use a distance function between hidden states, $D(H_a, H_b)$. For the first environment with single-step planning we use L_2 distance of the corresponding subimages.

$D(H_a, H_b) = L_2(I(H_a), I(H_b))$ where the masked sub-image of an object is the mask times the pixel means $I(H_k) = m_{(ij)}(H_k) \cdot g(X_{(ij)} \mid H_k)$. For the second environment with multi-step planning we a different distance function since the previous one may care more about if a shape matches than if the color matches. We instead use a form of intersection over union but that counts intersection if the mask aligns and pixel color values are close $D(H_a, H_b) = 1 - \frac{\sum_{i,j} m_{ij}(H_a)>0.01 \text{ and } m_{ij}(H_b)>0.01 \text{ and } L2(g(H_a)_{(ij)}, g(H_b)_{(ij)})<0.1}{\sum_{i,j} m_{ij}(H_a)>0.01 \text{ or } m_{ij}(H_b)>0.01}$. We found this version to work better since it will not give low cost to moving a wrong color block to the position of a different color goal block.

6.4 Architecture and Hyperparameter Details

We use similar model architectures as in [3] and so have rewritten some details from their appendix here. Differences include the dynamics model, inclusion of actions, and training procedure over sequences of data. Like [10], we define our latent distribution of size R to be divided into a deterministic component of size R_d and stochastic component of size R_s. We found that splitting the latent state into a deterministic and stochastic component (as opposed to having a fully stocahstic representation) was helpful for convergence. The posterior distribution $p(h|x)$ is a diagonal Gaussian. The output distribution $p(x|h)$ is also a diagonal Gaussian with means μ and global scale $\sigma = 0.1$. The decoder outputs the means μ and mask m_k.

Training: All models are trained with the ADAM optimizer [63] with default parameters and a learning rate of 0.0003. We use gradient clipping as in [64] where if the norm of global gradient exceeds 5.0 then the gradient is scaled down to that norm.

Inputs: For all models, we use the following inputs to the refinement network, where LN means Layernorm and SG means stop gradients. The following image-sized inputs are concatenated and fed to the corresponding convolutional network:

Description	Formula	LN	SG	Ch.
image	$\boldsymbol{x}$			3
means	$\boldsymbol{\mu}$			3
mask	$\boldsymbol{m}_k$			1
mask-logits	$\hat{\boldsymbol{m}}_k$			1
mask posterior	$p(\boldsymbol{m}_k \mid \boldsymbol{x},)$			1
gradient of means	$\nabla_{\boldsymbol{\mu}_k} \mathcal{L}$	✓	✓	3
gradient of mask	$\nabla_{\boldsymbol{m}_k} \mathcal{L}$	✓	✓	1
pixelwise likelihood	$p(\boldsymbol{x} \mid \boldsymbol{h})$	✓	✓	1
leave-one-out likelih.	$p(\boldsymbol{x} \mid \boldsymbol{h}_{i \neq k})$	✓	✓	1
coordinate channels				2
			total:	17

The posterior parameters $\boldsymbol{h}$ and their gradients are flat vectors, and we concatenate them with the output of the convolutional part of the refinement network and use the result as input to the refinement LSTM:

Description	Formula	LN	SG
gradient of posterior	$\nabla_{\boldsymbol{h}_k} \mathcal{L}$	✓	✓
posterior	$\boldsymbol{h}_k$		

6.4.1 Architecture

All models use the ELU activation function and the convolutional layers use a stride equal to 1 and padding equal to 2 unless otherwise noted.

Observation Model Decoder

Type	Size/*Ch.*	Act. Func.	Comment
Input: H_i	R_s		
Broadcast	R_s+*2*		+ coordinates
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*4*	Linear	RGB + Mask

Refinement Network

Type	Size/*Ch.*	Act. Func.	Comment
MLP	128	Linear	
LSTM	128	Tanh	
Concat $[H_i, \nabla_{H_i}]$	$2R$		
MLP	128	ELU	
Avg. Pool	R		
Conv 3×3	R	ELU	
Conv 3×3	*32*	ELU	
Conv 3×3	*32*	ELU	
Inputs	*17*		

Dynamics Network: The dynamics network takes in a sampled state and outputs the parameters of the posterior distribution. All models including the final layer uses ELU activations unless otherwise stated. MLP(N) would denote a multilayer peceptron with a hidden layer of size N.

- f_o: MLP(R) with input H_i^t of size R and output $\tilde{H}_i$ of size R.
- f_a: MLP(R) with input a^t of size A and output $\tilde{a^t}$ of size 32.
- $f_{\text{obj-action-eff}}$: MLP($R$) with inputs $\tilde{H}_i$ and $\tilde{a^t}$ and output of size R.
- $f_{\text{obj-action-att}}$: MLP($R$) with inputs $\tilde{H}_i$ and $\tilde{a^t}$ and output of size 1.
- Multiply outputs of $f_{\text{obj-action-eff}}$ and $f_{\text{obj-action-att}}$ to get output $\tilde{H}_i$.
- $f_{\text{obj-obj-eff}}$: MLP($2R$) with inputs $\tilde{H}_i, \tilde{H}_j$ each of size R and output $\tilde{H}_{ij}$ of size R.
- $f_{\text{obj-obj-att}}$: MLP($2R$) with inputs $\tilde{H}_i, \tilde{H}_j$ each of size R and output $\tilde{Att}_{ij}$ of size 1 with sigmoid applied.
- Let $\tilde{H}_{eff} = \sum_{j \neq i} \tilde{H}_{ij} * \tilde{Att}_{ij}$.
- f_{comb}: MLP($2R$) with inputs $\tilde{H}_i, \tilde{H}_eff$ each of size R and output $\tilde{H}_i$ of size R.
- f_{det}: MLP(R) with input $\tilde{H}_i$ and output H_{det}^{t+1} of size R_d.
- $f_{\text{sto},\mu}$: MLP(R) with input $\tilde{H}_i$ and output H_{sto}^{t+1} of size R_s.
- $f_{\text{sto},\sigma}$: MLP(R) with input $\tilde{H}_i$ and output H_{det}^{t+1} of size R_s.

6.5 Experiment Details

6.5.1 Single-Step Block-Stacking

The training dataset has 60,000 trajectories each containing before and after images of size 64x64 from [33]. Before images are constructed with actions which consist of choosing a shape (cube, rectangle, pyramid), color, and an (x, y, z) position and orientation for the block to be dropped. At each time step, a block is dropped and the simulation runs until the block settles into a stable position. The model takes in an image containing the block to be dropped and must predict the steady-state effect. Models were trained on scenes with 1 to 5 blocks with $K = 7$ slots. CEM begins from a uniform distribution on the first iteration, uses a population size of 1000 samples per iteration, and uses 10% of the best samples to fit a Gaussian distribution for each successive iteration.

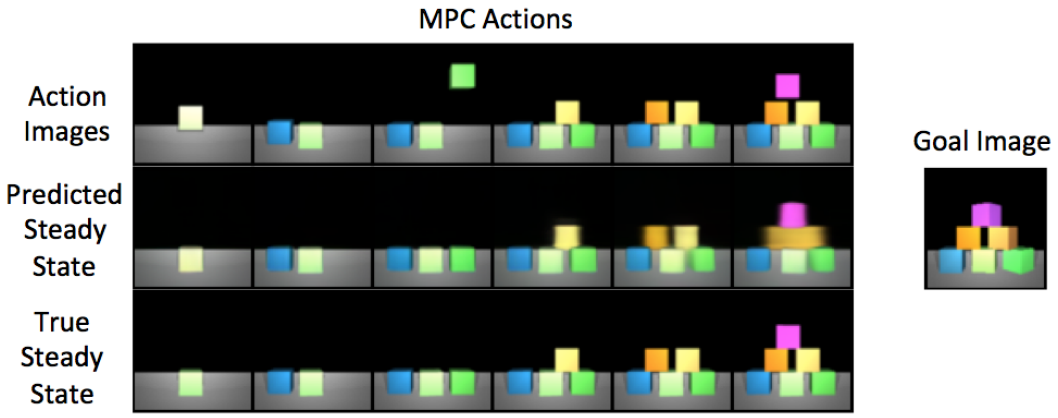

Figure 10: Qualitative results on building a structure. We see how our method is able to accurately and consistently predict the outcome of the action image, successively capturing the effect of inertial dynamics (gravity) and interactions with other objects.

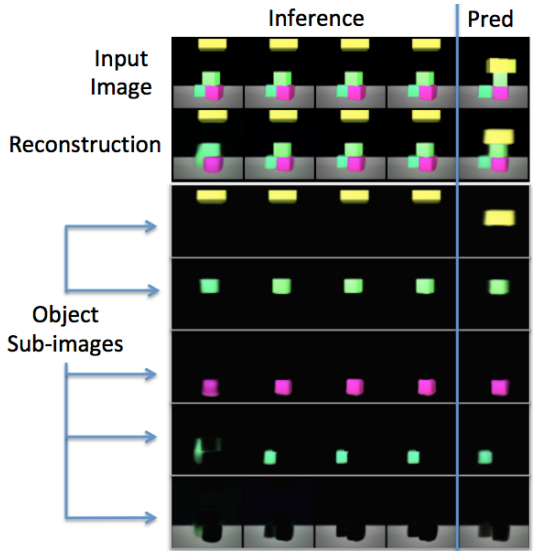

Figure 11: We show a demonstration of a rollout. The first four columns show inference iterations on the single input image, while the last column shows the predicted results using the dynamics module on the learnt hidden states (top right image is not given as input and shows the true outcome). The bottom 5 rows show $I(H_i)$ at each iteration, demonstrating how the model is able to capture individual objects, and the dynamics afterwards.

6.5.2 Multi-Step Block-Stacking

The training dataset has 10,000 trajectories each from a separate environment with two different colored blocks. Each trajectory contains five frames (64x64) of randomly picking and placing blocks. We bias the dataset such that 30% of actions will pick up a block and place it somewhere randomly, 40% of actions will pick up a block and place it on top of a another random block, and 30% of actions contain random pick and place locations. Models were trained with K = 4 slots. We optimize actions using CEM but we optimize over multiple consecutive actions into the future executing the sequence with lowest cost. For a goal with n blocks we plan n steps into the future, executing n actions. We repeat this procedure $2n$ times or until the structure is complete. Accuracy is computed as $\frac{\text{\# blocks in correct position}}{\text{\# goal blocks}}$, where a correct position is based on a threshold of the distance error.

For MPC we use two difference action spaces:

Coordinate Pick Place: The normal action space involves choosing a pick (x,y) and place (x,y) location.

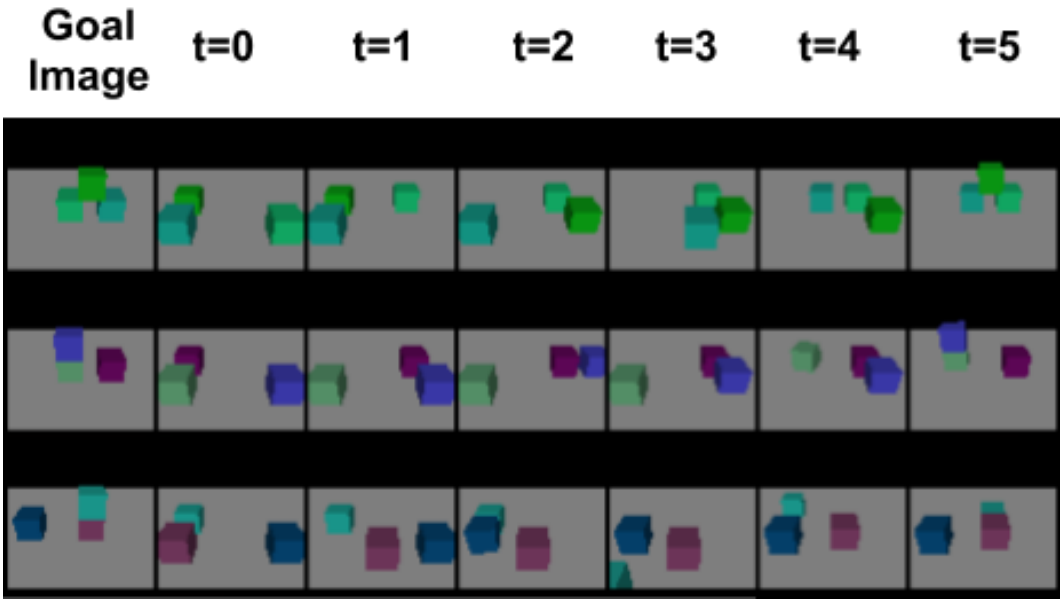

Figure 12: Demonstration of our method on several goals. t=0 denotes the initial scene that must be reconfigured to match the goal image. $t = 1...5$ show the executed action.

Entity Pick Place: A concern with the normal action space is that successful pick locations are sparse (2%) given the current block size. Therefore, the probability of picking n blocks consecutively becomes 0.02^n which becomes improbable very fast if we just sample pick locations uniformly. However, we create an action spaces that involves choosing one of the latent entities to move and then a place (x, y) location. This allows us to easily pick blocks consecutively if we can successfully map a latent `entity_id` of a block to a corresponding successful pick location. In order to determine the pick (x, y) from an `entity_id` k, we sample coordinates uniformly over the pick (x, y) space and then average these coordinates weighted by their attention coefficient on that latent:

$$\texttt{pick\_xy}|h_k = \frac{\sum_{x',y'} p(h_k|x,y) * \texttt{pick\_x'y'}}{\sum_{x',y'} p(h_k|x',y')}$$

where $p(h_k|x, y)$ are given by the attention coefficients produced by the dynamics model given h_k and the pick location (x, y) and x', y' are sampled from a uniform distribution.

6.5.3 Ablations

We perform ablations on the block stacking task from [33] examining components of our model. Table 3 shows the effect of non-symmetrical models or cost functions. The "Unfactorized Model" and "No Weight Sharing" follow b) and d) from Figure 2 and are unable to sufficiently generalize. We additionally see that even with the same OP3 model, the mean-square-error unfactorized cost function between the composite prediction and goal image significantly under performs our factorized cost function.

No Weight Sharing	Unfactorized Model	Unfactorized Cost
0 %	0 %	5%

Table 3: Accuracy of ablations. The no weight sharing model did not converge during training.