HYPERDYNAMICS: GENERATING EXPERT DYNAMICS MODELS BY OBSERVATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose HyperDynamics, a framework that conditions on an agent's interactions with the environment and optionally its visual observations, and generates the parameters of neural dynamics models based on inferred properties of the dynamical system. Physical and visual properties of the environment that are not part of the low-dimensional state yet affect its temporal dynamics are inferred from the interaction history and visual observations, and are implicitly captured in the generated parameters. We test HyperDynamics on a set of object pushing and locomotion tasks. It outperforms existing dynamics models in the literature that adapt to environment variations by learning dynamics over high dimensional visual observations, capturing the interactions of the agent in recurrent state representations, or using gradient-based meta-optimization. We also show our method matches the performance of an ensemble of separately trained experts, while also being able to generalize well to unseen environment variations at test time. We attribute its good performance to the multiplicative interactions between the inferred system properties—captured in the generated parameters—and the low-dimensional state representation of the dynamical system.

1 INTRODUCTION

Humans learn dynamics models that predict results of their interactions with the environment, and use such predictions for selecting actions to achieve intended goals (Miall & Wolpert, 1996; Haruno et al., 1999). These models capture intuitive physics and mechanics of the world and are remarkably versatile: they are expressive and can be applied to all kinds of environments that we encounter in our daily lives, with varying dynamics and diverse visual and physical properties. In addition, humans do not consider these models fixed over the course of interaction; we observe how the environment behaves in response to our actions and quickly adapt our model for the situation in hand based based on new observations. Let us consider the scenario of moving an object on the ground. We can infer how heavy the object is by simply looking at it, and we can then decide how hard to push. If it does not move as much as expected, we might realize it is heavier than we thought and increase the force we apply (Hamrick et al., 2011).

Motivated by this, we propose *HyperDynamics*, a framework that generates parameters for dynamics models (experts) dedicated to the situation in hand, based on observations of how the environment behaves. HyperDynamics has three main modules: i) an encoding module that encodes a few agent-environment interactions and the agent's visual observations into a latent feature code, which captures the properties of the dynamical system, ii) a hypernetwork (Ha et al., 2016) that conditions on the latent feature code and generates parameters of a dynamics model dedicated to the observed system, and iii) a target dynamics model constructed using the generated parameters that takes as input the current low-dimensional system state and the agent action, and predicts the next system state, as shown in Figure 1. We will be referring to this target dynamics model as an *expert*, as it specializes on encoding the dynamics of a particular scene at a certain point in time. HyperDynamics conditions on real-time observations and generates dedicated expert models on the fly. It can be trained in an end-to-end differentiable manner to minimize state prediction error of the generated experts in each task.

To place HyperDynamics in the context of existing literature on model learning and adaptation, we distinguish four groups of methods, depending on whether they explicitly distinguish between

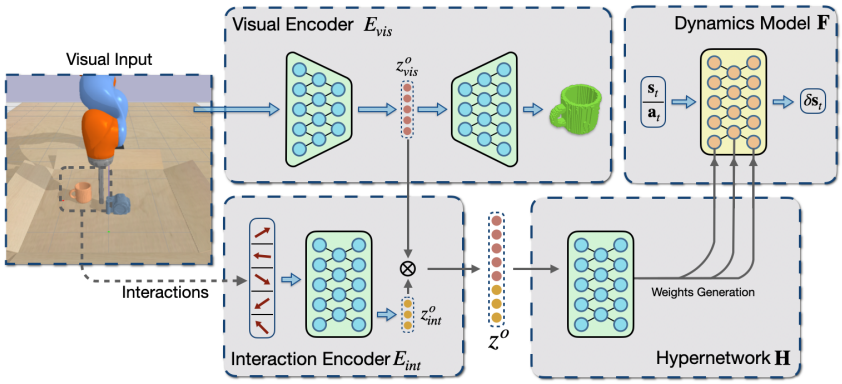

Figure 1: HyperDynamics encodes the visual observations and a set of agent-environment interactions and generates the parameters of a dynamics model dedicated to the current environment and timestep using a hypernetwork. HyperDynamics for pushing follows the general formulation, with a visual encoder for detecting the object in 3D and encoding its shape, and an interaction encoder for encoding a small history of interactions of the agent with the environment.

dynamic—such as joint configurations of an agent or poses of objects being manipulated—and static —such as mass, inertia, 3D object shape—properties of the system, and whether they update the dynamics model parameters or the static property values of the system. Though such exposition may not be exhaustive, it will help us put the proposed work in perspective with existing methods.

(i) No adaptation. These methods concern dynamics model over either low-dimensional states or high-dimensional visual inputs of a specific system, without considering potential changes in the underlying dynamics. These models tend to fail when the system behavior changes. Such *expert* dynamics models are popular formulations in the literature (Watter et al., 2015; Banijamali et al., 2018; Fragkiadaki et al., 2015; Zhang et al., 2019; Hafner et al., 2019). They can be adapted through gradient descent at any test system, yet that would require a large number of interactions.

(ii) Visual dynamics and recurrent state representations. These methods operate over high-dimensional visual observations of systems with a variety of properties (Finn & Levine, 2016; Ebert et al., 2018; Li et al., 2018a; Xu et al., 2019b; Mathieu et al., 2015; Oh et al., 2015a; Pathak et al., 2017), hoping the visual inputs could capture both the state and properties of the system. Some also attempt to encode a history of interactions with a recurrent hidden state (Xu et al., 2019a; Li et al., 2018a; Ebert et al., 2018; Sanchez-Gonzalez et al., 2018b), in order to implicitly capture information regarding the physical properties of the system. These methods use a single and fixed global dynamics model that takes system properties as input directly, together with its state and action.

(iii) System identification (Bhat et al., 2002). These methods model explicitly the properties of the dynamical system using a small set of semantically meaningful physics parameters, such as mass, inertia, damping, etc. (Ajay et al., 2019; Mrowca et al., 2018), and adapt by learning their values from data on-the-fly (Chen et al., 2018; Li et al., 2019; Sanchez-Gonzalez et al., 2018b; Battaglia et al., 2016; Sanchez-Gonzalez et al., 2018a).

(iv) Online model adaptation via meta-learning (Finn et al., 2017; Nagabandi et al., 2019; Clavera et al., 2018; Nagabandi et al., 2018a). These methods perform online adaptation on the parameters of the dynamics models through gradient descent. They are are optimized to produce a good parameter initialization that can be adapted to the local context with only a few gradient steps at test time.

The expert models in (i) work well for their corresponding systems but fail to handle system variations. A natural solution to address this is to train a separate expert for each dynamical system. However, this can hardly scale and doesn't transfer to systems with novel properties. Inspired by this need, HyperDynamics aims to infer a system's properties by simply observing how it behaves, and automatically generate an expert model that is dedicated to the observed system. Methods in (ii) and (iii), though attempting to infer system properties, optimize a single and fixed global dynamics model, which takes as input both static system properties and fast-changing states, with the hope that such a model can handle variations of systems and generalizes. We argue that a representation which encodes system information varies across different system variations, and each instance of this presentation ideally should correspond to a different dynamics model that needs to be used in the corresponding setting. These models are different, but also share a lot of information as similar

systems will have similar dynamics functions. HyperDynamics makes explicit assumptions about the relationships between these systems, and attempts to exploit this regularity and learn such commonalities across different system variations.

We evaluate HyperDynamics on single- and multi-step state predictions, as well as downstream model-based control tasks. Specifically, we apply it in a series of object pushing and locomotion tasks. Our experiments show that HyperDynamics is able to generate performant dynamics models that match the performance of separately and directly trained experts, while also enabling effective generalization to systems with novel properties in a few-shot manner, outperforming representative models from the categories discussed above. We attribute its performance gain to the multiplicative way of combining explicit factorization of encoded system features with the low-dimensional state representation of the system. Our code is included in the supplementary material.

2 Related Work

Model learning and model adaptation Learning dynamics models that can generalise across environment physical and visual variations is an open research problem in model-based reinforcement learning, the field that investigates action selection by temporal unrolling of dynamics (Nguyen-Tuong & Peters, 2011). The difficulty of traditional analytical models to capture the complexity of real world systems has led many researchers to pursue learning-based models of dynamics (Watter et al., 2015; Banijamali et al., 2018; Hafner et al., 2019; Zhang et al., 2019), or combine such models with analytic ones (Zeng et al., 2019; Ajay et al., 2019). Dynamics models that operate over high-dimensional images are often self-supervised (no need for annotation of the state) by predicting future frames. These methods infer a low-dimensional state of the system from a history of 2D images or 3D poinclouds (Li et al., 2018a; Xu et al., 2019b; Ebert et al., 2018; Agrawal et al., 2016) and predict a representation for the sensory input for the next time step, such as a future 2D image (Finn et al., 2016; Finn & Levine, 2016; Ebert et al., 2018; Oh et al., 2015a; Ebert et al., 2017; Oh et al., 2015b; Kaiser et al., 2019), or an image embedding of a future frame (Ha & Schmidhuber, 2018; Chiappa et al., 2017; Hafner et al., 2019; Agrawal et al., 2016; Pathak et al., 2017; Hafner et al., 2019). Models that factorize images into objects or particles in 2D or 3D and estimate their dynamics using relational biases implemented through graph message passing algorithms are harder to supervised since they need annotations regarding the objects in the scene and their trajectories over time Janner et al. (2018); Battaglia et al. (2016); Fragkiadaki et al. (2015); Chang et al. (2016). To infer physical properties of the system that can not be discerned from visual features, some works propose to encode a history of agent interactions (Li et al., 2018b; Xu et al., 2019a; Ebert et al., 2018). The meta-learning methods of dynamics in (Finn et al., 2017; Nagabandi et al., 2019; Clavera et al., 2018) learn a parameter initialization for a dynamics model that can best be adapted through gradient descent to the system properties in hand, by backpropagating through the optimization process. The idea of using multiple model initializations for efficient adaptation to novel test conditions are investigated in (Cully et al., 2015; Nagabandi et al., 2018a). Kaushik et al. (2020) proposes dynamics models that alongside a state and action condition (through concatenation) on a situation embedding, which is jointly trained with a dynamics model.

HyperNetworks Central to HyperDynamics is the notion of using one network to generate the parameters of another network. This idea has previously been studied in various domains including image classification (Ha et al., 2016), generative models (Ratzlaff & Fuxin, 2019), and neural machine translation (Ha et al., 2016; Platanios et al., 2018). These works are motivated by various intuitions, such as using the structural features of the original network to generate network layers for reduction in the number of parameters (Ha et al., 2016), generating an ensemble of networks for the same task to improve robustness to adversarial data (Ratzlaff & Fuxin, 2019), or generating language encoders conditioned on the language context for better weight sharing across languages in a machine translation setting (Platanios et al., 2018). To the best of our knowledge, this is the first work that applies the idea of generating model parameters to the domain of model-based RL, and proposes to generate expert models conditioned on system properties.

3 HyperDynamics

HyperDynamics generates expert dynamics models on the fly, conditioned on observed system features. We study applying HyperDynamics to a series of object pushing and locomotion tasks, but in

principle it is agnostic to the specific context of a dynamics learning task. In this section, we first present the general form of HyperDynamics, and then describe the task-specific modules we used.

Given a dynamical system o (which is typically comprised of an agent and its external environment) with a certain set of properties, HyperDynamics observes both how it behaves under a few actions (interactions), as well as its visual appearance when needed. Then, a visual encoder E_{vis} encodes the visual observation I into a latent visual feature code z^o_{vis}, and an interaction encoder E_{int} encodes a k-step interaction trajectory $\tau = (s_0, a_0, ...s_k, a_k)$ into a latent physics feature code z^o_{int}. We then combine these two features codes by concatenating them, $z^o = [z_{int}, z_{vis}]$, forming a single vector that captures the underlying dynamics properties of the system. A *hypernetwork* $\mathbf{H}$ maps z^o to a set of neural network weights for a forward dynamics model $\mathbf{F}^o$, which is a mapping from the state of the system and the action of the agent, to the state change at the subsequent time step $t+1$: $\delta s_t = \mathbf{F}^o(s_t, a_t)$. The hypernetwork is a fully-connected network and generates all parameters in a single-shot as the output of it's final layer. The detailed architecture of HyperDynamics for an object pushing example is shown in Figure 1.

We train HyperDynamics by backpropagating error gradients of the generated dynamics model's prediction, all the way to our visual and interaction encoders and the hypernetwork. Let $\mathcal{O}$ denote the pool of all systems used for training. Let T denote the length of the collected trajectories. Let N denote the number of trajectories collected per system. The final loss for our model is defined as:

$$\mathcal{L} = \frac{1}{|\mathcal{O}|TN} \sum_{o=1}^{|\mathcal{O}|} \sum_{t=1}^{T} \sum_{i=1}^{N} \|\delta s_t - \mathbf{F}(s_t, a_t; \mathbf{H}([E_{vis}(I;\phi), E_{int}(\tau;\psi)]);\omega)\|_2, \tag{1}$$

where ϕ, ψ and ω denote the learned parameters of E_{vis}, E_{int} and $\mathbf{H}$, respectively.

We consider a set of locomotion tasks for which only the interaction history is used as input, in accordance with prior work (e.g., Finn et al., 2017; Nagabandi et al., 2019; Clavera et al., 2018; Nagabandi et al., 2018a), and a pushing task on a diverse set of objects, where the visual encoder encodes the 3D shape of the object.

3.1 HyperDynamics for Object Pushing

For our object pushing setting, we consider a robotic agent equipped with an RGB-D camera to observe the world and an end-effector to interact with the objects. The agent's goal is to push objects to desired locations on a table surface. At time t, the state of our pushing system is $s^o_t =< s^{obj}_t, s^{rob}_t >$, where the object state s^{obj}_t is a 13-vector containing the object's position $\mathbf{p}^{obj}_t$, orientation $\mathbf{q}^{obj}_t$ (represented as a quaternion), as well as its translational and rotational velocities $\mathbf{v}^{obj}_t$, and the robot state s^{rob}_t is a 3-vector representing the position of its end-effector. For controlling the robot, we use position-control in our experiments, so the action $a_t = (\delta x, \delta y, \delta z)$ is a 3-vector representing the movement of the robot's end-effector.

Objects behave differently under pushing due to varying properties such as shape, mass, and friction relative to the table surface. For this task, HyperDynamics tries to infer object properties from its observations and generate a dynamics model specific to the object being manipulated. While the 3D object shape can be inferred by visual observations, physics properties such as mass and friction are hard to discern visually. Instead, they can be inferred by observing the motion of an object as a result to a few interactions with the robot (Xu et al., 2019b). To produce the code z^o, we use both E_{vis} for encoding shape information and E_{int} for encoding physics properties.

Given an object to be pushed, we assume its properties stay unchanged during the course of manipulation. To produce the interaction trajectory τ, the robot pushes the object by k actions sampled from random directions. E_{int} then maps z_{int} to a 1-dimensional code $z_{int} \in \mathbb{R}^2$. In practice we found $k = 5$ sufficient to yield informative z_{int} for pushing, and used it in all our experiments. Since this is a relatively short trajectory, we use a simple fully-connected network for E_{int}.

Given a single RGB-D image I of the scene, E_{vis} produces object-centric $z_{vis} \in \mathbb{R}^8$ to encode the shape information of the objects in the scene. We build our E_{vis} using the recently proposed Geometry-Aware Recurrent Networks (GRNNs) (Tung et al., 2019), which learn to map single 2.5D images to complete 3D feature grids $\mathbf{M} \in \mathbb{R}^{w \times h \times d \times c}$ that represent the appearance and the shape of the scene and its objects. GRNNs learn such mappings using differentiable unprojection (2.5D to 3D) and projection (3D to 2.5D) neural modules and can be optimized end-to-end for both 3D

object detection and view prediction, and complete missing or occluded shape information while optimized for such objectives. We refer readers to Tung et al. (2019) for more details.

E_{vis} first detects all the objects in the scene, and then crops the scene map around the object we want to push to produce an object-centric 3D feature map $\mathbf{M}^{obj} = \mathrm{crop}(\mathbf{M})$, which is then used for shape encoding. We opt for such 3D representations since it allows a more realistic pushing setting: we do not require the camera viewpoint to be fixed throughout training and testing, and we can handle robot-object occlusions better since our encoder learns to complete missing (occluded) shape information in the map $\mathbf{M}^{obj}$. We train E_{vis} for object detection jointly with the prediction loss in Equation 1. We also add additional regularization losses by reconstructing the object's shape, where the visual code z_{vis} is passed into a decoder D_{vis} to reconstruct the object shape V^{obj}, represented as binary occupancy voxel grids. Note that the reconstruction loss can be applied from pure visual data, by observing objects from multiple viewpoints, without necessarily interacting with them. This is important because moving a camera around objects is cheaper than interacting with them through the robot's end-effectors. In addition, since our tasks concerns planar pushing, we found reconstructing a top-down 2D shape instead of the the full 3D one suffices for regularization, essentially predicting a top-down view of the object from any perspective viewpoint.

Object 3D orientation: context or state? The shape of rigid objects does not change over time; rather, its orientation changes. The framework discussed so far intuitively divides the input information among different modules: the hypernetwork generates a dynamics model conditioned on object properties, and the dynamics model takes care of low-dimensional object states. However, when there's only a few objects used for training, the hypernetwork will only strive to optimize for a discrete set of dynamics models, thus potentially leading to severe overfiting on the training data. Instead of producing z_{vis} to encode object shape information at its canonical pose, we propose to encode the object shape oriented at its current orientation at each time step, and discard the orientation information from states fed into the generated dynamics model, as their generated parameters would already encode the orientation information. Using oriented as opposed to canonical shape as input permits the hypernetwork $\mathbf{H}$ to have way more training examples, even from a small number of objects with distinct shapes. Intuitively, $\mathbf{H}$ now treats the same object with different orientations as distinct objects, and generates a different expert for each. This helps smoothen and widen the data distribution seen by the hypernetwork.

3.2 HyperDynamics for Locomotion

For locomotion tasks, we consider a legged robot walking in a dynamic environment with changing terrains, where their properties are only locally consistent. At timestep t, the state contains the joint velocities and joint positions of the robot, as well as the position of its center of mass. The dimension of the state depends on the actual morphology of the robot. Torque-control is used to send low-level torque commands to the robot.

Unlike the object pushing setting where the properties of the objects can be safely assumed to remain constant, here the system dynamics changes rapidly due to varying terrain properties. Hence, we adopt an online adaptation setting where at timestep t, the interaction data is taken to be the most recent trajectory $\tau = (s_{t-k}, a_{t-k}, ...s_{t-1}, a_{t-1})$. We use a recurrent network with Gated Recurrent Unit (GRU) (Cho et al., 2014) layers to encode the interaction trajectory, and in practice we found $k = 16$ (equal to a trajectory segment of around 0.3 seconds) to be informative enough for E_{int} to infer accurate terrain properties. The encoded $z_{int} \in \mathbb{R}^2$ is then used by $\mathbf{H}$ to generate a dynamics model for the local segment of the terrain in real-time.

3.3 Model Unrolling and Action Planning

Action-conditioned dynamics models can be unrolled forward in time for long-term planning and control tasks. We apply our approach with model predictive control (MPC) for action planning and control. Specifically, we use random shooting (RS) (Nagabandi et al., 2018b) for action planning, unroll our model forward with randomly sampled action sequences, pick the best performing one, and replan at every timestep using updated state information to avoid compounding errors.

The unrolling mechanism of HyperDynamics for pushing is straightforward: at each timestep t, a generated dynamics model $\mathbf{F}_t^o$ predicts δs_t^o for the system. Afterwards, we compute $s_{t+1}^o = s_t^o \oplus \delta s_t^o$,

where $\oplus$ represents simple summation for positions and velocities, and quaternion composition for orientations. We then re-orient the object-centric feature map $\mathbf{M}^{obj}$ using the updated object orientation $\mathbf{q}_{t+1}^{obj}$ and generate a new dynamics model $\mathbf{F}_{t+1}^{o}$. Finally, the updated full state s_{t+1}^{o}, excluding $\mathbf{q}_{t+1}^{obj}$, is used by $\mathbf{F}_{t+1}^{o}$ for motion prediction. For locomotion, at each timestep we obtain one z_{int}, generate a dynamics model $\mathbf{F}_{t}^{o}$ for the local context, and use it for all unrolling steps into the future, since future terrain properties are not predictable.

4 EXPERIMENTS

Our experiments aim to answer these questions: (1) Is HyperDynamics able to generate dynamics models across environment variations that perform as well as expert dynamics models, trained specifically on the dynamics of each environment variation? (2) Does HyperDynamics generalize to systems with novel properties? (3) How does HyperDynamics compare with methods that either use fixed global dynamics models or adapt their parameters during the course of interaction through meta-optimization? We test our proposed framework in the tasks of both object pushing and locomotion, and describe each of them in details below.

4.1 OBJECT PUSHING

Many prior works that learn object dynamics consider only quasi-static pushing or poking, where an object always starts to move or stops together with the robot's end-effector (Finn & Levine, 2016; Agrawal et al., 2016; Li et al., 2018a; Pinto & Gupta, 2017). We go beyond simple quasi-static pushing by varying the physics parameters of the object and the scene, and allow an object to slide by itself if pushed with a high speed. A PID controller controls the robot's torque in its joint space under the hood, based on position-control commands, so varying the magnitude of the action could lead to different pushing speed of the end-effector. We test HyperDynamics on its motion prediction accuracy for single- and multi-step object pushing, as well as its performance when used for pushing objects to desired locations with MPC.

Experimental Setup. We train our model with data generated using the PyBullet simulator (Coumans & Bai, 2016–2019). Our setup uses a Kuka robotic arm equipped with a single rod-shaped end-effector, as shown in Figure 1. Our dataset consists of only 31 different object meshes with distinct shapes, including 11 objects from the MIT Push dataset (Yu et al., 2016) and 20 objects selected from four categories (*camera*, *mug*, *bowl* and *bed*) in the ShapeNet Dataset (Chang et al., 2015). We split our dataset so that 24 objects are used for training and 7 are used for testing. The objects can move freely on a planar table surface of size $0.6m \times 0.6m$. For data collection, we randomly sample one object with randomized mass and friction coefficient, and place it on the table with a random starting position. Then, we instantiate the end-effector nearby the object and collect pushing sequences with length of 5 timesteps, where each action is a random horizontal displacement of the robot's end-effector ranging from $3cm$ to $6cm$, and each timestep is defined to be $800ms$. Our method relies on a single 2.5D image as input for object detection and motion prediction. Note that although collecting actual interactions with objects is expensive and time-consuming, data consisting of only varying shapes are cheap and easily accessible. To ensure z_{shape}^{o} produced by the shape encoder E_{shape} captures sufficient shape information for pushing, we train it on the whole ShapeNet Chang et al. (2015) for shape autoencoding as an auxiliary task, jointly with the dynamics prediction task. Please refer to Appendix A for more details of our experimental setup.

Baselines. We compare our model against these baselines: **(1)** *XYZ*: a model that only take as input the action and states of the object and the agent, without having access to its visual feature or physics properties, similar to Andrychowicz et al. (2020) and Wu et al. (2017). **(2)** *Direct*: a model uses the same E_{vis} and E_{int} as our model does. The only difference is that it directly optimizes a single global dynamics model instead of generating experts. This approach resembles the most common way of conditioning on object features used in the current literature (Finn & Levine, 2016; Agrawal et al., 2016; Li et al., 2018a; Xu et al., 2019a; Fragkiadaki et al., 2015; Zhang et al., 2019; Hafner et al., 2019), where a global dynamics model takes as input both the state and properties of the dynamical system. **(3)** Visual Foresight (*VF*) (Ebert et al., 2018) and *DensePhysNet* (Xu et al., 2019a): these two models rely on 2D visual inputs of the current frame, and predict 2D optical flows that transform the current frame to the future frame. These models also use a global dynamics model for varying system properties, and feed system features directly into the model, following the same

Table 1: Motion rediction error (in centimeters).

Model	Seen Objects		Novel Objects	
	$t = 1$	$t = 5$	$t = 1$	$t = 5$
Expert-Ens	**0.82**	**4.22**	1.68	5.83
XYZ	1.45	5.89	1.72	6.46
Direct	1.01	5.14	1.24	5.60
MB-MAML	1.68	8.91	1.76	9.05
HyperDynamics	**0.83**	**4.27**	**0.93**	**4.77**

Table 2: Success rate for object pushing with MPC.

Model	Seen Objects		Novel Objects	
	No Obst.	Obst.	No Obst.	Obst.
Expert-Ens	**0.92**	**0.72**	0.80	0.44
XYZ	0.80	0.42	0.74	0.44
Direct	0.88	0.62	0.84	0.58
MB-MAML	0.70	0.42	0.68	0.38
VF	0.62	—	0.52	—
DensePhysNet	0.70	—	0.58	—
HyperDynamics	**0.92**	**0.70**	**0.92**	**0.68**

philosophy of *Direct*. **(4)** *MB-MAML* (Nagabandi et al., 2019): a meta-trained model which applies model-agnostic meta-learning to model-based RL, and trains a dynamics model that can be rapidly adapted to the local context when given newly observed data. **(5)** *Expert-Ens*: we train a separate expert model for each distinct object, forming an ensemble of experts. These experts assume access to the *ground-truth* orientation, mass, and friction. This baseline serves as an oracle when tested on objects seen during training. For unseen objects, we select a corresponding expert from the ensemble using shape-based nearest-neighbour retrieval, essentially performing system identification.

We evaluate all the methods on prediction error for both single-step ($t = 1$) and multi-step unrolling ($t = 5$) (Table 1). We also test their performance for downstream manipulation tasks, where the robot needs to push the objects to desired target locations with MPC, in scenes both with and without obstacles. When obstacles are at present, the agent needs to plan a collision-free path. We report their success rates in 50 trials in Table 2. Red numbers in bold denote the best performance across all models, while black numbers in bold represent oracle performance (provided by *Expert-Ens*). *VF* and *DensePhysNet* are not tested for motion prediction since they work in 2D image space and do not predict explicit object motion, and are not applicable for collision checking and collision-free pushing for the same reason.

Our model outperforms all baselines by a margin on both prediction and control tasks. When tested on objects seen during training, while our model needs to infer orientation information and physics properties via encoding visual input and interactions, it performs on par with the *Expert-Ens* oracle, which assumes access to ground-truth orientation, mass and friction coefficient. This shows that the dynamics model generated by HyperDynamics on the fly is as performant as the separately and directly trained dynamics experts. The *XYZ* baseline depicts the best performance possible when shape and physics properties of the objects are unknown. When tested with novel objects unseen during training, our model shows a performance similar to seen objects, outperforming *XYZ* by a great margin, demonstrating that our model transfers acquired knowledge to novel objects well. *Expert-Ens* on novel objects performs similarly to *XYZ*, suggesting that with only a handful of distinct objects in the training set, applying these dedicated models to novel objects, though after nearest-neighbour searches, yields poor generalization. Performance gain of our model over *Direct* suggests our hierarchical way of conditioning on system features and generating experts outperforms the common framework in the literature, which uses a global model with fixed parameters for varying system dynamics. Our model also shows a clear performance gain over *MB-MAML*, when only 5 data samples of interaction are available for online adaptation. This indicates that our framework, which directly generates dynamics model given the system properties, is much more sample-efficient than the meta-trained dynamics model prior that needs to adapt to the context via extra tuning.

4.2 Locomotion

Experimental Setup. We set up our environment using the MuJoCo simulator (Todorov et al., 2012). In particular, we consider two types of robots: a planar cheetah and a quadrupedal "ant" robot. For each robot, we consider two environments where the terrains are changing: (1) *Slope*: the robot needs to walk over a terrain consisting of multiple slopes with changing steepness. (2) *Pier*: the robot needs to walk over a series of blocks floating on the water surface, where damping and friction properties vary between the blocks. As a result, we consider these 4 tasks in total: *Cheetah-Slope*, *Ant-Slope*, *Cheetah-Pier* and *Ant-Pier*. Figure 2 shows two of them. Unlike the setting of object pushing, in the context of locomotion it's unrealistic to assume arbitrary resetting of robot states for data collection. As a result, on-policy data collection is performed: the latest model is used for action selection to collect best-performing trajectories, which are then used to update the dataset for model training. For *Slope*, the environment is comprised of a series of consecutive terrain

Table 3: Comparison of average total return for locomotion tasks.

Model	Cheetah-Slope		Cheetah-Pier		Ant-Slope		Ant-Pier	
	Seen	Novel	Seen	Novel	Seen	Novel	Seen	Novel
Expert-Ens	**104.1**	76.9	**354.6**	321.4	**142.1**	124.2	**219.0**	179.3
Recurrent	64.4	76.6	292.5	279.7	**133.9**	**142.1**	176.4	170.5
MB-MAML	55.3	62.7	291.3	297.1	104.5	110.4	180.2	171.0
HyperDynamics	**107.9**	**124.5**	**349.4**	**337.2**	**138.8**	**140.1**	**216.6**	**208.4**

segments, each of which has a randomly sampled steepness. During test time, we also evaluate on unseen terrains with either higher of lower steepness. For *Pier*, each of the blocks floating on the water has randomly sampled friction and damping properties, and moves vertically when the robot runs over it, resulting in a rapidly changing terrain. Similarly, during test time we also evaluate using blocks with properties outside of the training distribution. See Appendix A for more details.

Baselines. We compare our model against these baselines: **(1)** *Recurrent* (Sanchez-Gonzalez et al., 2018b): A recurrent model that also uses GRUs (Cho et al., 2014) to encode a history of interactions to infer system dynamics and perform system identification implicitly. **(2)** *MB-MAML* (Nagabandi et al., 2019): a model meta-trained using MAML, same as the one used for object pushing. It uses the same 16-step interaction trajectory for online adaptation. **(3)** *Expert-Ens*: An ensemble of separately trained experts, similar to the one used for object pushing. The only difference is that in pushing, we train one expert for each distinct object, while now we have a continuous distribution of terrain properties used for training. Therefore, for each task we pick several environments with properties evenly sampled from the continuous range, and train an expert for each of them.

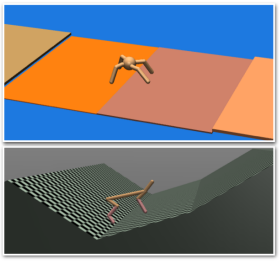

Figure 2: We consider two types of robots and two environments for locomotion. Top: Ant-Pier. Down: Cheetah-Slope.

We apply all the methods with MPC for action selection and control, and report in Table 3 the average return computed over 500 episodes. Again, red numbers denote the best performance and the black ones represent the oracle performance. In all tasks, HyperDynamics is able to infer accurate system properties and generate corresponding dynamics models that match the oracle *Expert-Ens* on seen terrains, and shows a great advantage over it when tested on unseen terrains. Our model also outperforms both *MB-MAML* and *Recurrent* on most of the tasks. Note that here *Recurrent* can be viewed as serving the same sole of the *Direct* baseline in pushing, since it feeds system features together with the states to a fixed global dynamics model. The results suggest that our explicit factorization of system features and the multiplicative way of aggregating it with low-dimensional state provide a clear advantage over methods that either train a fixed global model or perform online parameter adaptation on a meta-trained dynamics prior.

5 Conclusion

We presented HyperDynamics, a framework that conditions on system features inferred from observations and interactions with the environment to generate parameters for dynamics models dedicated to the observed system on the fly. We evaluate our framework in the context of object pushing and locomotion. Our experimental evaluations show that dynamics models generated by HyperDynamics perform on par with an ensemble of directly trained experts in the training environments, while other baselines fail to do so. In addition, our framework is able to transfer knowledge acquired from training to novel systems with unseen properties, even when only 24 distinct objects are used for training in the object pushing task. Our method presents explicit factorizations of system properties and state representations, and provides a multiplicative way for them to interact, resulting in a performance gain over both methods that employ a global yet fixed dynamics models, and methods using gradient-based meta-optimization. Handling more varied visual tasks and predicting both the structure and the parameters of the target dynamics model are interesting avenues for future work. The proposed framework is agnostic to the specific form of dynamics learning tasks, and we hope this work could stimulate future research into weight generation for adaptation of dynamics models in more versatile tasks and environments.

REFERENCES

Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in neural information processing systems*, pp. 5074–5082, 2016.

Anurag Ajay, Maria Bauza, Jiajun Wu, Nima Fazeli, Joshua Tenenbaum, Alberto Rodriguez, and Leslie Kaelbling. Combining physical simulators and object-based networks for control, 04 2019.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Ershad Banijamali, Rui Shu, Hung Bui, Ali Ghodsi, et al. Robust locally-linear controllable embedding. In *International Conference on Artificial Intelligence and Statistics*, pp. 1751–1759, 2018.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016.

Kiran S. Bhat, Steven M. Seitz, and Jovan Popovic. Computing the physical parameters of rigid-body motion from video. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen (eds.), *ECCV*, volume 2350 of *Lecture Notes in Computer Science*, pp. 551–565. Springer, 2002. ISBN 3-540-43745-2. URL http://dblp.uni-trier.de/db/conf/eccv/eccv2002-1.html#BhatSP02.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *CoRR*, abs/1612.00341, 2016. URL http://arxiv.org/abs/1612.00341.

Tao Chen, Adithyavairavan Murali, and Abhinav Gupta. Hardware conditioned policies for multi-robot transfer learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 9355–9366. Curran Associates Inc., 2018.

Silvia Chiappa, Sébastien Racanière, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *CoRR*, abs/1704.02254, 2017. URL http://arxiv.org/abs/1704.02254.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Ignasi Clavera, Anusha Nagabandi, Ronald Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt: Meta-learning for model-based control. 03 2018.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521:503–507, 05 2015. doi: 10.1038/nature14422.

Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017. URL http://arxiv.org/abs/1710.05268.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. *CoRR*, abs/1610.00696, 2016. URL http://arxiv.org/abs/1610.00696.

Chelsea Finn, Ian J. Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 64–72, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *CoRR*, abs/1511.07404, 2015.

David Ha and Jürgen Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018. URL http://arxiv.org/abs/1803.10122.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565, 2019.

Jessica Hamrick, Peter Battaglia, and Joshua B Tenenbaum. Internal physics models guide probabilistic judgments about object dynamics. In *Proceedings of the 33rd annual conference of the cognitive science society*, pp. 1545–1550. Cognitive Science Society Austin, TX, 2011.

Masahiko Haruno, Daniel M Wolpert, and Mitsuo Kawato. Multiple paired forward-inverse models for human motor learning and control. In M. J. Kearns, S. A. Solla, and D. A. Cohn (eds.), *Advances in Neural Information Processing Systems 11*, pp. 31–37. MIT Press, 1999.

Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. *CoRR*, abs/1812.10972, 2018. URL http://arxiv.org/abs/1812.10972.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

Rituraj Kaushik, Timothée Anne, and Jean-Baptiste Mouret. Fast online adaptation in robotics through meta-learning embeddings of simulated priors, 2020.

Jue Kun Li, Wee Sun Lee, and David Hsu. Push-net: Deep planar pushing for objects with unknown physical properties. In *Robotics: Science and Systems*, 2018a.

Juekun Li, Wee Sun Lee, and David Hsu. Push-net: Deep planar pushing for objects with unknown physical properties. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018b. doi: 10.15607/RSS.2018.XIV.024.

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgbSn09Ym.

Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015. URL http://arxiv.org/abs/1511.05440.

R. C. Miall and D. M. Wolpert. Forward models for physiological motor control. *Neural Netw.*, 9 (8):1265–1279, November 1996. ISSN 0893-6080. doi: 10.1016/S0893-6080(96)00035-4. URL http://dx.doi.org/10.1016/S0893-6080(96)00035-4.

Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems*, 2018.

Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl, 12 2018a.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018b.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019.

Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: A survey. *Cognitive processing*, 12:319–40, 04 2011. doi: 10.1007/s10339-011-0404-1.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. *arXiv preprint arXiv:1507.08750*, 2015a.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2863–2871. Curran Associates, Inc., 2015b.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *CoRR*, abs/1705.05363, 2017. URL `http://arxiv.org/abs/1705.05363`.

Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2161–2168. IEEE, 2017.

Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. Contextual parameter generation for universal neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 425–435, 2018.

Neale Ratzlaff and Li Fuxin. Hypergan: A generative model for diverse, performant neural networks. In *International Conference on Machine Learning*, pp. 5361–5369, 2019.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4470–4479, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018a. PMLR.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479, 2018b.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2595–2603, 2019.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pp. 2746–2754, 2015.

Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pp. 153–164, 2017.

Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B. Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *CoRR*, abs/1906.03853, 2019a. URL `http://arxiv.org/abs/1906.03853`.

Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. *arXiv preprint arXiv:1906.03853*, 2019b.

Kuan-Ting Yu, Maria Bauzá, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed: A high-fidelity experimental data set of planar pushing. *CoRR*, abs/1604.04038, 2016. URL `http://arxiv.org/abs/1604.04038`.

Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. 2019.

Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. Solar: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, pp. 7444–7453, 2019.

A EXPERIMENTAL DETAILS

A.1 OBJECT PUSHING

Our dataset consists of 31 object meshes with distinct shapes, including 11 objects from the MIT Push dataset (Yu et al., 2016) and 20 objects selected from four categories (*camera*, *mug*, *bowl* and *bed*) in the ShapeNet Dataset (Chang et al., 2015). 24 objects are used for generating the training data and 7 objects are used for testing. The size of the objects ranges from $7cm$ to $12cm$. Our training set contains $50,000$ trajectories of object motions, each with a length of 5 timesteps. Our testing set contains $1,000$ trajectories. For each trajectory, we randomly sample an object with randomized mass and friction coefficient, and place it on a $0.6m \times 0.6m$ table surface at a uniformly sampled random position. The mass is uniformly sampled from $[300, 1000]$, the friction coefficient of the object material is uniformly sampled from $[8e^{-4}, 12e^{-4}]$, and the friction coefficient of the table surface is set to be 10, all using the default units in PyBullet. At the begining of each trajectory, we instantiate the robot's end-effector nearby the object, with a random distance from the object's center ranging from $6cm$ to $10cm$. When sampling pushing actions, with a probability of 0.2 the agent pushes towards a completely random direction, and with a probability of 0.8 it pushes towards a point randomly sampled in the object's bounding box. Each action is a displacement with a magnitude sampled from $[3cm, 6cm]$.

Our method is viewpoint-invariant and relies on a single 2.5D image as input. We place cameras at 9 different views uniformly with different azimuth angles ranging from the left side to the right side of the robot, and the same elevation angles of 60 degrees. All cameras are looking at $0.1m$ above the center of the table, and $1m$ away from the look-at point. At test time, our model is tested with a single and randomly selected view. All images are 128×128.

For object pushing with MPC, we place a randomly selected object in the scene with a random initial position, and sample a random target location for it. At each step, we evaluate 30 random action sequences with the model before taking an action. For pushing without obstacle, the maximum distance of the target to the initial position for each object is capped at $0.25m$, and a planning horizon of 1 step is used since greedy action selection suffices for it. For pushing with obstacles, we randomly place 2 additional obstacles in the scene, selected from the same pool of available objects. We place the first obstacle between the starting and goal positions with a small perturbation sampled from $[-2cm, 2cm]$, so that there exists no straight collision-free path from the starting position to the goal. The second obstacle is placed randomly with a distance from the first one ranging in $[15cm, 25cm]$. The distance from the target to the initial position is uniformly sampled from the range $[0.24m, 0.40m]$. The model needs to plan a collision free path and thus a planning horizon of 10 steps is used. For both tasks, we run 50 trials and evaluate the success rate, where it is considered successful if the object ends up within $4cm$ from the target position, without colliding with any obstacles along the way.

We aim for simplicity of the architecture of each modules in our model. The interaction encoder is a fully-connected network with one hidden layer of size 8. The hypernetwork is a fully-connected network with one hidden layer of size 32. The visual encoder follows the architecture of Tung et al. (2019) and its encoder uses 3 convolutional layers, each followed by a max pooling layer, with a fully-connected layer at the end. It uses kernels of size 5, 3 and 3, with 2, 4, and 8 channels respectively. Its decoder uses 3 transposed convolutional layers, with 16, 8, and 1 channels and the same kernel size of 4. The generated dynamics model is a fully-connected network with 4 hidden layers, each of size 32. We use leaky ReLU as activations for all the modules. We use batch size of 8 and a learning rate of $1e-3$. All models are trained till convergence for 500K steps.

A.2 LOCOMOTION

The *Slope* environment consists of a series of consecutive upward slopes with length of 15m. For training, the height of each slope is randomly sampled from $[0.5m, 3.5m]$, and for testing, the height is randomly sampled from $[0m, 0.25m] \cup [0.75m, 4m]$. The *Pier* environment consists of a series of consecutive blocks with length of 4m. For training, the damping coefficient of each block is randomly sampled from $[0.2, 0.8]$, and for testing, the damping coefficient is randomly sampled from $[0, 0.1] \cup [0.9, 1.0]$. 5 experts are trained to form the expert ensemble for each task. *cheetah* has 6 controllable joints while *ant* has 10. The reward function for *cheetah* is defined to be $\frac{x_{t+1}-x_t}{0.01} -$

$0.05\|a_t\|_2^2$, and for *ant* it is $\frac{x_{t+1}-x_t}{0.02} + 0.05$, where x_t refers to the x-coordinate of the agent at time t.

Our encoder is a recurrent network with 2 GRU layers, first one with hidden size equal to the dimension of the agent state, and second one with hidden size of 32, followed by a ReLU activations and a fully-connected layer which outputs the encoded z_{int}. The hypernetwork is a fully-connected network with one hidden layer of size 16. The generated dynamics model is a fully-connected network with two hidden layers, each of size 128. During planning and control, 500 random action sequences are sampled with a planning horizon of 20 steps. Data collection is performed in an on-policy manner. We start the first iteration of data collection with randomly sampled actions, collecting 10 rollouts, each with 500 steps. We iterate over the data for 100 epochs before proceeding to next round of data collection. At each iteration, the size of training data is capped at 50000 steps, randomly sampled from all available trajectories collected. Here we use batch size of 128 and a learning rate of $1e-3$. All models are trained for 150 iterations till convergence. In addition, early stopping is applied if an rolling average of total return decreases.