Title
Synergy of Prediction and Control in Model-based Reinforcement Learning

Author
Lambert, Nathan Owen

Publication Date
2022

Peer reviewed|Thesis/dissertation

Synergy of Prediction and Control in Model-based Reinforcement Learning

by

Nathan Owen Lambert

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Kristofer S.J. Pister, Chair
Dr. Roberto Calandra
Professor Sergey Levine
Professor Claire Tomlin

Spring 2022

Synergy of Prediction and Control in Model-based Reinforcement Learning

Abstract

Synergy of Prediction and Control in Model-based Reinforcement Learning

by

Nathan Owen Lambert

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Kristofer S.J. Pister, Chair

Model-based reinforcement learning (MBRL) has often been touted for its potential to improve on the sample-efficiency, generalization, and safety of existing reinforcement learning algorithms. These model-based algorithms constrain the policy optimization during trial-and-error learning to include a structured representation of the environment dynamics. To date, the posited benefits have largely been left as directions for future work. This thesis attempts to illustrate the central mechanism in MBRL: how a learned dynamics model interacts with decision making. A better understanding of this interaction will point the field in the direction of enabling the posited benefits.

This thesis encompasses the interaction of model-learning with decision making with respect to two central issues: compounding prediction errors and objective mismatch. The compounding error challenge emerges from accumulating errors on recursive passes of any one-step transition model. Most dynamics models are trained for single-step accuracy, which often results in models with substantial long-term prediction error. Additionally, the model being trained for accurate transitions need not guarantee high-performance policies on the downstream task. The lack of correlation between model and policy metrics in separate optimization is coined and studied as Objective Mismatch.

These challenges are primarily studied in the context of sample-based model predictive control (MPC) algorithms, where the learned model is used to simulate trajectories and their resulting predicted rewards. To mitigate compounding error and objective mismatch, the trajectory-based dynamics model is a feedforward prediction parametrization containing a direct representation of time. This model represents one small, but important steps towards more useful dynamics models in model-based reinforcement learning. This thesis concludes with future directions on the synergy of prediction and control in MBRL, primarily focused on state-abstractions, temporal correlation, and future prediction methodologies.

To my family and dear friends.

Acknowledgments

Thank you to my advisor, Kris, for admitting me to this wildly exciting place of graduate study, U.C. Berkeley Electrical Engineering and Computer Sciences. Even though my research interests have consistently moved me away from your core research, you have never slowed in your support of me. You enabled me to take on something different in my Ph.D. and the result is successful beyond what we would have ever anticipated. This advising relationship hopefully helped you grow, and I look forward to see what we discover together in the future.

I was lucky to pair Kris with my mentor, colleague, co-advisor, and friend Roberto. When I started working with you, it was a tireless effort to prove myself in your field. Your support of me through this transition – ultimately acting as if I always belonged – has been central to my successes. I look forward to continuing to working together.

Beyond my advisors, I would like to thank some close collaborators within the Pister Group: First, thank you to Dan and Craig for doing the selfless mentorship work early in my Ph.D. that helped me get my feet underneath me. We are short a few celebrations, as our paths were artificially separated due to the pandemic. Thank you Lydia, for being a great desk-buddy – we even have published together and we never saw that coming! I was lucky to work closely with three other Berkeley Ph.D. students in the latter half of my Ph.D., Sarah, Thomas, and Tom, who formed our team doing formative work understanding how AI will integrate with society. Through a relatively independent Ph.D., you all made me remember some normal joys of being a collaborative graduate student and accepted me in a challenging time for myself, where we have now become close friends.

I do not know where I would be in my life without my family and specifically my brother Victor. V, Your compassion, patience, support, charisma, and love are just the beginning of my admiration for you. My parents Jean and Rob, you supported me through challenges that I did always admit I was having. I am glad that we got to expand on our parent-child relationships and become closer friends through my Ph.D. Cathy, your addition to our family is a unilaterally positive addition and you inspire me to follow my dreams. Max, while our future is unsure, I thank you for showing me how to love way more about this world. Our journey through COVID was always exciting and fearless; we are stronger from it. Thank you all for your patience.

My closest friends have contributed to this culminating work. Berkeley introduced me to Vikram, Saavan, Mau, Divya, and many others who now will make it even harder to leave the Bay. Thank you to Rohit for your never-ending willingness and joy in dealing with our unusual communication styles. Our friendship has taken on many forms during my Ph.D., but it has never dimmed and it will continue to flourish. Olav, Erik, Scott, and Lav – our friendships are not in close proximity, but we have never missed a beat.

This dissertation represents the closure on a long path of growth for me, and while I did not mention everyone here, I hold appreciation for all who helped me get to today.

Contents

Chapter 1

Introduction

Deep reinforcement learning (RL) has emerged as a powerful candidate for generating decision-making agents that can surpass that of human experts. Recent successes originated with research in games [SAH+20], but related techniques have been emerging in robotics [KVC+21], autonomous-vehicles [Ell21], efficient computing [MZR+22], energy technology [DFB+22], and likely many more projects not yet announced or shared. The core of reinforcement learning is a numerical methodology for open-ended trial-and-error learning. An agent acts in an environment, observes a state and a reward, and then uses that information to make decisions about the future. This framework is designed to allow consistent improvement in behavior by feedback over rewards and current parameters. To date, the most successful variants of these algorithms have been data- and compute-hungry, highly specialized, and operating in an end-to-end manner.

Model-based reinforcement learning (MBRL) is the conceptual counter to large end-to-end systems, where the learning agent includes the structured optimization of a dynamics model of the environment to make downstream decisions. The strengths of end-to-end learning, namely its power to optimize and scaling with data and compute, how been crucial to its recent successes. With these strengths come a series of downsides that are more difficult to address: at-scale model-free reinforcement learning is known to exploit the environment and be opaque to audits of decision reasoning. Learning a structured dynamics model of the environment has been shown to have benefits in sample-efficiency, and is touted for its potential to convey strengths in generalization, interpretability, and safety – positioning itself as a solution to large open problems in the field.

At the implementation level, most recent model-based reinforcement learning methods learn a model to represent discrete transitions in the environment. These transition approximations are used with a variety of decision making schemes to maximize reward, such as sample-based planning or actor-critic algorithms. This thesis aims to discuss and address critical issues limiting the pairing of learned dynamics models with a separately optimized control policy. There is a crucial difference between learning a model to accurately predict the dynamics of a system and learning a model to reliably convey high-performance policies. Developing this understanding is central to unlocking the posited benefits of model-based

learning.

This thesis begins with a review of existing methods for learning models for control in the context of model-based reinforcement learning, Chapter 2, with particular focus given to how techniques interface with different environments and controllers. The review is complemented with a brief case-study on applying model-based reinforcement learning to low-level control of a quadrotor in Chapter 3. Following, the thesis focuses on two issues limiting the advancement of MBRL techniques. One issue, the compounding error problem, emerges as prediction errors accumulating with each forward pass. This issue, addressed in Chapter 4 and based on our recent work [LPC22], focuses on how an understanding of one's application domain can be integrated into an understanding of how a model propagates trajectories.

On the other hand, an accurate model may not be shifting the agent's optimization space to be optimal in reward. Objective Mismatch arises when two separate optimization problems are executed sequentially in the hopes of achieving one downstream goal. This effect, described in Chapter 5 and our recent work [LAYC20], with respect to the optimization of an accuracy-based dynamics model inducing a policy with a task-based goal.

The popular algorithms using sample-based model predictive control (MPC), such as PETS [CCML18], POPLIN [WB20], and PDDM [NKLK20] suffer from both objective mismatch and compounding prediction errors. This thesis concludes with the study of a new class of dynamics model aimed at mitigating these challenges – the trajectory-based model. By adding an explicit notion of time, the trajectory-based model conveys benefits in long-term prediction accuracy, uncertainty quantification, and computational efficiency – shown in Chapter 6 [LWZPC21].

Model-based reinforcement learning has many open research and engineering challenges, noted by the challenges addressed in this thesis and the relative low-density of algorithms when compared to model-free or offline RL algorithms. This thesis concludes with future directions for the field, primarily interested in temporal abstractions and long-term predictions (Chapter 7).

Chapter 2

Learning Dynamics for Reinforcement Learning: A Brief Review

This chapter is a progression from the basic ideation of reinforcement learning, to the motivation of learning models for decision making, and a detailed discussion of which models we may want to learn.

2.1 Motivation & Preliminaries

Learning by Reinforcement

Machine Learning (ML) is the field of study focusing on using computation to make decisions in the presence of uncertainty. Classic notions of machine learning theory include trading off between the structural biases of a model and the inherent variance in observing partial data, developing model architectures for specific applications, understanding data with and without labels, and more. The general pipeline of a ML problems follows as inputs of a desired functionality and a dataset and outputs of a model to predict said data based on some underlying loss function.

The leading question with any machine learning method is: how do we utilize it to make decisions? A prediction is only a source of information that must be integrated into a broader framework to take action Countless products in modern life use the outputs of machine learning models to feed suggestions to users. Though, these systems largely are hand-engineered and did not converge on these actionable traits on their own. Here lies the modern motivation for learning by reinforcement: how do we deploy tools that can learn to make decisions autonomously?

Reinforcement Learning (RL) is a framework where an agent interacts with an environment to solve a task by trial and error. The framework which this thesis builds on is visualized in Fig. 2.1, and encompasses two entities: 1) the *agent*, that is often defined by its evolving decision making rule, called the policy π, and 2) the *environment*, that is often

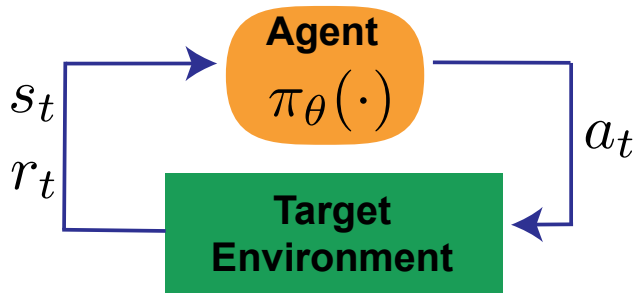


Figure 2.1: A basic visualization of the feedback inherent to reinforcement learning. The agent takes an action $\boldsymbol{a}_t$ corresponding to its policy π_θ and the current state of the environment $\boldsymbol{s}_t$. The environment outputs a scalar reward r_t based on the current state and action.

assumed to be a static playground by which the agent learns its abilities. The RL framework often combines multiple sub-components of traditional machine learning that are directly optimized for the purpose of taking actions. The central addition is the notion of an observed reward function by which the agent will tune its behavior.

The motivation and history of enabling autonomous learning agents comes from multiple directions. Broadly, the idea of learning via reinforcement has been of interest of scientists looking to imitate their mental processes. Though, a major driving force to progress in the computational methods for achieving this goal emerged from the field of optimal control. By enabling an agent with dynamic programming, a policy could converge to the optimal policy by modelling the expected return from a given state [Bel57]. These ideas, now described by the Bellman Operator are central to modern successes in deep reinforcement learning.

Why Learn a Model

In designing a RL agent, the design question with the largest impact on the downstream behavior is whether or not to include a structured dynamics model of the environment. This trade-off, described as model-free or model-based, changes how decisions are made and experience is stored. Those algorithms learning a structured representation of the dynamics, called model-based, are known to have better sample-efficiency and generalization capabilities than their model-free counterparts. Though, these methods are historically more difficult to implement, computationally intensive, and sometimes weaker performing.

Model-based reinforcement learning is well-motivated by a broader literature and is often touted for its potential to overcome these shortcomings. Leveraging local models of the world is widely accepted as a central mechanism of human cognition [Kaw99; DSD12], where different temporal abstractions of the world allow flexible decision making. Given that the field of reinforcement learning has long been motivated by realizing this understanding in decision making algorithms [SB18], MBRL is a natural continuation of mirroring human capabilities. Beyond these conceptual goals, learned models convey multiple structured

strengths that represent rich areas of recent work and important avenues for the future of model-based reinforcement learning. Here we discuss its potential for data-efficiency, generalization, safe decision making, interpretability, and exploration.

Data-efficiency Data-efficiency in RL is often characterized as the number of steps in the discrete environment that are need to solve a goal (in the real world, this directly links to clock time). Constraining the learning process to include a dynamics model induces a structural bias on the parameter space that substantially reduces the search-space over behaviors the agent can express [1]. This constrained search is the central idea behind the data-efficiency benefits of learning models for control. Data-efficiency is crucial for certain applications where every trial is even more costly, such as novel robotics [DLSP18; LSDP20] and healtcare [GJK+19].

Generalization Generalization in RL is the ability for an agent to move from one area in its environment to a not yet seen area without a drop in performance. A dynamics model learned for one task can be used for generalization when the agent behavior or state-space region needs to change, so long as the model extrapolates correctly between previous versions of the state-action space [KZGR21] [2]. Compared to model-free counterparts, a dynamics model can readily be used to induce a new policy for a new task with a simple change of the dynamics model. This ability to keep the same model and change the behavior of the reward function is the strength of entirely planning based approaches, where model-free agents are known to need substantial re-training or even re-collection to change the policy.

Safety Another advantage of a learned model over a policy is that of moving towards constrained safety by querying the model as offline knowledge. A safe agent is one which can avoid predetermined areas of a state-space that it should not operate in. Current methods integrate safety techniques from optimal control, such as system stability [BTSK17] or safe set constraints [KBTK18]. Future work in safety for MBRL will involve integrating these well-studied techniques from optimal control with advanced methods that can integrate with value functions or supervision [TBR+20].

Interpretability After multiple high-profile failures of machine learning algorithms to fairly integrate social and human values in optimizations, there is an increased appetite for the study of how machine learning algorithms can beneficially integrate with society. As these algorithms are given the freedom to act on many digital systems in an open-ended manner, as would be done with using reinforcement learning in the real-world, further effort should be taken to mitigate potential harms of these systems. The field of reinforcement learning

[1] The structural bias is the difference between needing to model all possible behaviors of an agent in the environment versus only the underlying dynamics (which is often simpler).

[2] Generalization can take many forms, such as within distribution or outside of distribution. For more information, see the referenced survey.

broadly is working to define a clear notion of what it means for a temporally evolving decision-making system to be interpretable [PV20], but a learned model allows more comprehensive investigation of safety, generalization, and sociotechnical specification [DGLZ21] [3].

Exploration Further study of knowledge representations in reinforcement learning is needed in order to enable agents capable of adapting to new situations and environments. Exploration is the concept for how an agent to autonomously collect the most relevant and interesting data in an environment without prior direction. A starting point will be to expand on recent work in knowledge-based exploration, where learned dynamics models are used as a intrinsic reward signal for exploration agents [BEP+18; LWW+22]. These models are often used only to generate reward signals for a model-free optimizer, but this intersection represents an opportunity to extend the usefulness of model-learning for control to new problem settings.

Technical Formulation

We utilize the setting where an agent's interactions with an environment to maximize reward r_t are modeled as a Markov Decision Process (MDP) [Bel57]. A MDP is defined by a state of the environment $\boldsymbol{s}$, an action $\boldsymbol{a}$ that is taken by an agent according to a policy $\pi_\theta(s_t)$, a transition function $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ governing the next state distribution, and a discount factor $\gamma \in [0, 1]$ weighting future rewards. With a transition in dynamics, the agent receives a reward r_t from the environment and stores the State-Action-Reward-(next)State tuple (SARS) data in a dataset $\mathcal{D} : \{\boldsymbol{s}_k, \boldsymbol{a}_k, r_k, \boldsymbol{s}_{k+1}\}$. The objective of an agent is often to maximize the cumulative future reward on a predetermined task by optimizing a set of parameters of the policy θ, shown in Eq. (2.1).

$$\mathbb{E}\big[\sum_{\tau=0}^{\infty} \gamma^\tau r_\tau | \boldsymbol{s}_0 = \boldsymbol{s}_t\big] \tag{2.1}$$

RL differs from other popular machine learning systems via the temporal variation. RL algorithms are designed to search for and optimize over dynamic datasets. Core to the process are two feedback loops [GDZL22]: 1) the agent acts in control feedback with the environment, where each decision is a function of the current state; and 2) the current parameters defining the policy and the past lifetime of an agent influence the future decisions in a form of behavioral feedback.

[3] As discussed in the safety section above, a model can be used to perform counterfactual queries on agent intent and understanding in a more principled manner than a policy. Where a policy will only suggest an action at a given state, a planning-based agent with a model can convey its action sequence with its expected outcomes – the user can study a mismatch between actions and plans that may be leaning on exploitative behavior.

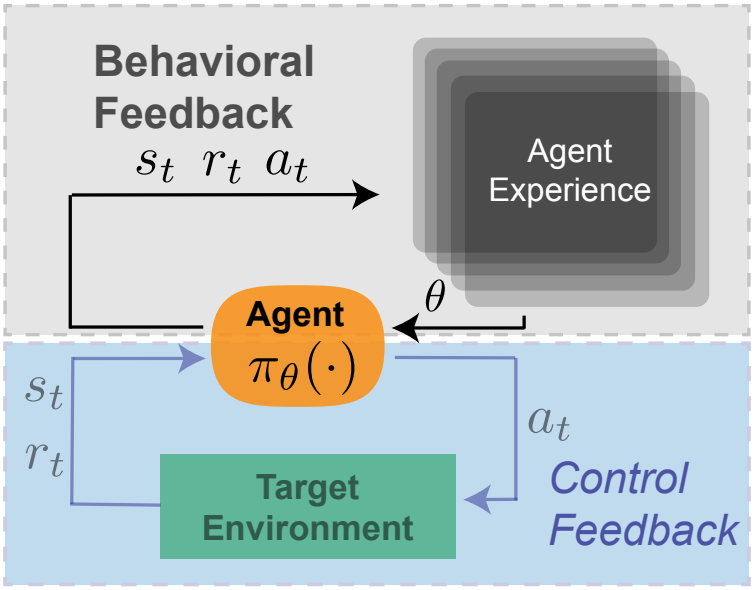


Figure 2.2: Reinforcement learning is the field of study for understanding how to make decisions in temporal processes. Here is an illustration of how the agent interacts with the environment and how different types of feedback emerge. The learning of a dynamics model would often occur in the Behavioral Feedback loop.

2.2 Model Types: What to Model

In this section we cover the different ways models are used and define terminology around using a learned dynamics model. The two most common dynamics models used are *Forward dynamics* and *Inverse dynamics*. An emerging category is that of *Reward models* where the model directly predicts the reward of a transition or a forward model is trained by proxy as a tool to predict rewards. All of these model varieties leverage the discrete nature of the environment, as is modelled by an MDP.

Forward Dynamics The forward dynamics aim at predicting the future states given the current state and possible actions to be performed. The forward model is a function to model the transition dynamics $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$. The model prediction is formulated as:

$$\boldsymbol{s}_{t+1} = f\left(\boldsymbol{s}_t, \boldsymbol{a}_t\right) . \tag{2.2}$$

Forward models are used in many domains and algorithms, including online-planning, offline value-estimation, and more. In the remainder of this chapter, the discussion is focused on forward dynamics models due to their prevalence (and they are often referred to as the dynamics model).

Inverse Dynamics Instead, inverse dynamics aim at predicting the actions to be performed given the current state, and the future desired (or measured) state. An inverse model is closely related to a policy π_θ, but is one that has the context of a desired state rather than the measured reward. The model prediction is formulated as:

$$\boldsymbol{a} = g(\boldsymbol{s}, \boldsymbol{s}_{\text{desired}}) \,. \tag{2.3}$$

These inverse models are useful when the agent knows the goal of a given task and this model can act as a policy [ET12; CSM+16].

Reward Models Reward is the central signal to an RL agent. When using MBRL, a common technique is to use a forward dynamics model to predict future states. With these predicted states, the agent needs a way to evaluate which trajectory is best. This can be done with a learned reward model that returns an estimate of the reward the real environment would return at that step (based on past tuples sampled from the environment):

$$r_t = f\left(\boldsymbol{s}_t, \boldsymbol{a}_t\right) \,. \tag{2.4}$$

Forward models taking in the state-action and predicting a future value of the environment have been used in parallel to dynamics models, expanding the prediction dimension by 1.

Related Works Model learning for control has a rich history in the fields of optimal control and state-estimation [NP11]. Recent work has seen a few in-depth reviews specific to dynamics models in MBRL. These focus on model-predictive agents [LHB+21], determining the correct metrics for dynamics models [KHT21], and studying the causes and effects of compounding error [LPC22].

Formulating Predictions

In this portion we focus on how the transitions in forward models can be formatted to improve on certain performance characteristics. The three types of models that we focus on are one-step models that directly model the environment transitions as a memoryless function, recurrent & latent models that learn models in a constrained representation of the state-space, and ensemble methods that augment any predictor with a more robust inference procedure.

One-step Models

The most common forward dynamics model is the single-step ahead of Eq. (2.2). The prediction is often formulated as a distribution to sample next state from, $\hat{\boldsymbol{s}}_{t+1} \sim f_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)$, and a propagation method is used to determine the predicted next state. To train a set of model parameters θ for accurate predictions, a log-likelihood formulation is used as

$$\arg\max_\theta \sum_{(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}) \in \mathcal{D}} \log f_\theta(\boldsymbol{s}_{n+1} | \boldsymbol{s}_n, \boldsymbol{a}_n). \tag{2.5}$$

Most other prediction formulations trade off one type of prediction accuracy (e.g. short or long-term predictions) in exchange for increased computation cost.

Here, we detail some popular modifications to the standard one-step formulation shown in Eq. (2.5).

Delta-state Consider the case of prediction two trajectories that have similar local dynamics (e.g. joint velocities), but they are off-set by a fixed positional delta (e.g. a different GPS reading), would the standard model parametrization succeed? In order to address this shortcoming, the one-step formulation is parametrized as a delta-state prediction for improved regularization of the training set, as

$$\boldsymbol{s}_{t+1} = \boldsymbol{s}_t + f_\theta\left(\boldsymbol{s}_t, \boldsymbol{a}_t\right) . \tag{2.6}$$

History Another situation where a modification is useful is one where limited state-data does not include crucial information such as velocities. In systems where the state-estimation is poorly observable, a common trick is to augment the state-space to contain the history of actions. For example, this can give the model equivalent information of a velocity measurement when only positions are available [LDY+19; KBM+20]. Given a desired history, H, the formulation becomes:

$$\boldsymbol{s}_{t+1} = f\left(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \ldots, \boldsymbol{s}_{t-H}, \boldsymbol{a}_{t-H}\right) . \tag{2.7}$$

This formulation is popular with neural network models due to their ability to handle large input dimensions. The history formulation can be modified to include any other contextual signals from the system that are not defined as the state (e.g. battery charge of a robot).

Autoregressive Prediction A final option for one-step models is to change the prediction so that each predicted state-index informs that remaining items because states are often strongly correlated (rather than predicting the entire state-vector in one pass). Autoregressive prediction directly addresses this by conditioning the prediction of each element of a vector on the previously predicted subcomponents at that time step, along with the current state and actions [LR85]. A motivating example is the prediction of each subsequent joint on a robotic arm given the previous joints' positions. This method recently has been re-examined in the context of MBRL due to its potential to represent each state dimension as a conditionally independent distribution [ZPN+21]. Predicting the states one by one constrains the distribution of each subsequent element, but the resulting predictive model is slower at inference than standard methods due to the number of forward passes scaling linearly with state dimension.

The autoregressive optimization extends the log-likelihood maximization in Eq. (2.5) to optimize for per-state predictions. Consider the i^{th} sub-element of a state vector as $\boldsymbol{s}_n^i$

$$\arg\max_\theta \sum_{(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}) \in \mathcal{D}} \Big[\log f_\theta(\boldsymbol{s}_{n+1} | \boldsymbol{s}_n, \boldsymbol{a}_n) + \sum_{i=1}^{d_s} \log f_\theta(\boldsymbol{s}_{n+1}^i | \boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}^1, \ldots, \boldsymbol{s}_{n+1}^{i-1}) \Big]. \tag{2.8}$$

[JLL21] also predicts the states in an autoregressive manner, though this Trajectory Transformer also predicts a policy and reward functions, differentiating itself from most feedforward models. Casting large language models as a predictor shifts the prediction problem from regression to classification.

Visual, Latent & Recurrent

Now we consider a different class of models focused on high-dimensional states and directly imbuing temporal feedback within the model as additional states [4]. The techniques of learning models from vision, optimizing in learned latent spaces, and incorporating recurrence have been growing concurrently in recent years. While each of these three methods can be found individually, recently they most frequently are found together.

Visual models use elements such as convolutional layers to use images directly in predictive models. These models can take many forms. A learned latent space is any state-space that is the output of a learned dynamics model. These latent spaces can be fine-tuned with gradients from many types of downstream components, such as gradients from the reward of a control policy. For example, [NYA+18] uses a visually conditioned feedforward model in a standard planning approach (*e.g.* without directly learning a latent state). While the most popular latent structure uses a recurrent neural network [HS18b], many examples exist for applications of learned latent spaces without recurrent models. Deep latent models have been used in online [RZN+21] and offline RL [RYRF21]. Additionally, latent spaces can be incorporated into other model modalities such as Gaussian Processes [SHD18], learned MDP's [GKBNB19], or for linear policy search [ZVS+19].

With these developments in visual reinforcement learning and learned latent spaces, recurrent models have been the most popular technique for advancements in pixel-based MBRL. These systems typically operate by combining a representation learning layer with a recurrent one-step dynamics model. This often involves three or more models – a recurrent state h_t, a latent state $\boldsymbol{z}_t$ from a learned representation, and predicted latents from a transition model $\hat{\boldsymbol{z}}_t$:

$$h_t = f_\phi(h_{t-1}, \boldsymbol{z}_{t-1}, \boldsymbol{a}_{t-1}) \,, \tag{2.9}$$

$$\boldsymbol{z}_t \sim q_\phi(\boldsymbol{z}_t | h_t, \boldsymbol{s}_t) \,, \tag{2.10}$$

$$\hat{\boldsymbol{z}}_t \sim p_\phi(\hat{\boldsymbol{z}}_t | h_t) \,. \tag{2.11}$$

This technique was pioneered by [HS18a], who uses a Variational Autoencoder (VAE) with a recurrent dynamics model and a linear control policy. This method has been extended as Recurrent state-space models (RSSM) in state-of-the-art pixel-based MBRL algorithms [HLNB21; HWS22]. Such recurrent, latent-space models have also been useful directly on planning for robotic control [EFD+18] and in Atari [KBM+20]. On recent baselines, recurrent models tend to outperform convolutional models and latent transition models

[4] It is important to distinguish this between flexible notions of time. These models are still single-step models with a fixed time-step, but they have a feedback-based state variable.

Model Type	Training Cost	Inference Cost	Parallel Option	Tooling Complexity	Domain Knowledge	State Setable
MLP						
↪ Single	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆
↪ Ensemble	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆
↪ Autoreg.	⋆⋆⋆⋆	⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆
Specialized MLP						
↪ Neural ODE	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆
↪ Physics-based	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
Locally Linear	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆
Gaussian Process	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆
Expert-designed						
↪ Env. Simulator	/	⋆	⋆	⋆⋆⋆	⋆	⋆
↪ Dynamical System	/	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆	⋆⋆⋆
Replay Buffer	/	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆

Table 2.1: Tooling and engineering considerations of different model types. *Training Cost* is the training time with a set dataset, *Inference Cost* is the cost of forward passes of a model, *Parallel Option* is the ability for a model to be accelerated, *Tooling Complexity* is the approximate dot product of the software maturity with the ease of use, *Domain Knowledge* is the required knowledge per evaluation environment, and *State Setable* is the ability for the model type to predict arbitrary state-action pairs (e.g. simulators often cannot).

for downstream reward prediction [BSH+20]. The final related class of models are those inspired by MuZero which utilize recurrent dynamics models, but they are only trained on reward signals (rather than for transition accuracy) [SAH+20].

Ensemble Methods

Ensembles of bootstrapped models have gained popularity as a simple method to estimate epistemic uncertainty and improve accuracy of various models [OAC18; LLSA21]. For a given predictive model, we consider the ensemble to have B members, using θ_b to denote the b^{th} set of parameters for the model f_{θ_b}. Each model in the ensemble have the same composition (e.g. number of layers) and are trained with cross-validation to get different sets of parameters. Deep ensembles can be composed of both deterministic or probabilistic [LPB17] members, with the primary practical difference being in how they are used to propagate predictions.

Model Type	Accuracy	Uncertainty Management	Value Prediction	State Scaling	Data Scaling	Generalization
MLP						
↪ Single	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆
↪ Ensemble	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆
↪ Autoreg.	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆
Specialized MLP						
↪ Neural ODE	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆
↪ Physics-based	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆⋆⋆⋆
Locally Linear	⋆⋆	⋆⋆	⋆	⋆	⋆	⋆
Gaussian Process	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆⋆
Expert-designed						
↪ Env. Simulator	⋆⋆⋆⋆⋆	/	⋆⋆⋆⋆	/	/	⋆⋆⋆⋆⋆
↪ Dynamical System	⋆⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆⋆
Replay Buffer	⋆⋆⋆⋆	⋆	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆	⋆

Table 2.2: Prediction and control considerations of different model types. *Accuracy* is the ability for the model to predict held out evaluation sets, *Uncertainty Management* is the ability for a model to represent the empirical uncertainty in the training set, *Value Prediction* is the ability for the dynamics model to also prediction the reward received from a given transition, *State Scaling* is the ability for the model to maintain accuracy with increasing state-action dimension, *Data Scaling* is the ability for a model to improve accuracy with orders of magnitude more data, and *Generalization* is the ability for a model to handle generalization within and without the training distribution.

2.3 Tools for Prediction: How to Model

In this section, we detail the various tools that can be used to deploy predictive models. A summary of the strengths and weaknesses of various approaches is shown with respect to the engineering and tooling considerations in Tab. 2.1 and the decision-performance characteristics in Tab. 2.2.

Feedforward Neural Networks

The most popular tool for modeling predictions in recent MBRL algorithms are multi-layer perceptron (MLP) networks. Compared to recent advancements in natural language pro-

cessing and computer vision, these "deep" models are relatively shallow, with many state-of-the-art algorithms [CCML18; JFZL19; WB19] and open-source code [PAZLC21; MSA+21] utilizing 2-5 layer networks of 128-512 neurons.

The most common solution is to constrain the resulting model to take on the form of a Gaussian distribution with a diagonal covariance matrix $f_\theta(\boldsymbol{s}_n, \boldsymbol{a}_n) = \mathcal{N}(\mu_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n), \Sigma_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n))$, with a loss function that penalizes the relative prediction accuracy with a l_2 penalty on each variance:

$$\text{loss}_{\text{Gauss}}(\boldsymbol{\theta}) = \sum_{n=1}^{N} [\mu_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n) - \boldsymbol{s}_{n+1}]^\top \Sigma_{\boldsymbol{\theta}}^{-1}(\boldsymbol{s}_n, \boldsymbol{a}_n)[\mu_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n) - \boldsymbol{s}_{n+1}] + \log\det \Sigma_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n). \tag{2.12}$$

The mean squared error (MSE) loss can also be viewed as a reduction of the Gaussian loss to that of delta distributions centered around a given point prediction (the mean of the prediction in the formulation above):

$$\text{loss}_{\text{M.S.E.}}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \left(\boldsymbol{s}_{n+1} - \mu_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n)\right)^T \left(\boldsymbol{s}_{n+1} - \mu_{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n)\right) . \tag{2.13}$$

Linking these two losses simplifies the notation when moving to more advanced loss functions and propagation methods.

Given any standard loss function discussed above, a multi-step loss can be used during training to help optimize for long-term prediction accuracy [AN04; VHB15; AMKL19; HLF+19; LHB+21]. With a given propagation method for $\hat{\boldsymbol{s}}_{n+1} = f_\theta(\boldsymbol{s}_n, \boldsymbol{a}_n)$, the multi-step loss is shown with the MSE loss function at each step:

$$\text{loss}_{\text{multi.}}(\boldsymbol{\theta}) = \frac{1}{H} \sum_{i=0}^{H} \text{loss}_{\text{M.S.E.}}(\boldsymbol{\theta})(\hat{\boldsymbol{s}}_{n+i+1}, \boldsymbol{a}_{n+i+1}) . \tag{2.14}$$

This multi-step loss evenly averages the prediction accuracy of a simulated trajectory across each H predicted steps.

Strengths Ease of use, powerful scaling properties with both dimensionality and data.

Weaknesses Poor uncertainty quantification, un-interpretable.

Integrating with Control Can be used with decision-time and offline planning, provides gradients, required GPU for online methods.

Specialized MLPs

Given that many of the current evaluation benchmarks for MBRL are in robotic tasks, there is substantial potential in linking the decision making algorithms with progress in using physical knowledge to model these systems. To date, the advancements in this area can be separated into learned differential equation models and physics-based forward models. The goal of these models with control would be to provide a substantial improvement in both short- and long-horizon predictions when deployed on a suitable system. Substantial opportunity lays in integrating these new predictive methods with control.

Neural Differential Equations Neural Ordinary Differential Equations (Neural ODEs) were introduced to more accurately model discrete transitions between sequences of hidden states [CSA+18]. The discrete transitions follow a simple Euler discretization of a continuous transform, so they can be reconstructed by differentiation through an ODE solver. These methods solve the form of

$$\boldsymbol{s}_{t+1} = \boldsymbol{s}_t + \int_{t_0}^{t_0+h} F_\theta(\boldsymbol{s}(t))dt\,. \tag{2.15}$$

Physics-based Models On the other hand, physics-based models constrain the optimization of existing forward dynamics models [CGH+20; GMMK20]. Working from a physical system with state $\boldsymbol{s}_t = (q, \dot{q})$, a Euler-Lagrange equation can be applied as a constraint on a transition from $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1}$ as $\frac{d}{dt}\frac{d\mathcal{L}}{d\dot{q}_j} = \frac{d\mathcal{L}}{dq_j}$, which can be re-written in vector form to get the following solution for the acceleration, $\ddot{q}$:

$$\ddot{q} = (\nabla_{\dot{q}}\nabla_{\dot{q}}^\top \mathcal{L})^{-1}[\nabla_q \mathcal{L} - (\nabla_q \nabla_{\dot{q}}^\top \mathcal{L})\dot{q}]]\,, \tag{2.16}$$

which can be solved via integration to compute the dynamics of the system. These constrained optimizations have shown to be both sample-efficient and more accurate for suitable systems [5], but further advancements are needed for systems without the suitable low-level observations.

Strengths Potential to include better inductive biases for control, potential for interpretability.

Weaknesses Increased training complexity, designed for smooth systems (e.g. contacts are challenging).

Integrating with Control (Open questions) added model complexity difficult for online control.

[5] They have been studied for a variety of dynamical systems, but often in the context of a well-shaped training dataset. In RL problems, the dataset is multi-modal and lacking uniform coverage, which results in a more challenging modeling problem.

Locally Linear

Linear models are a strong candidate for simple systems or when agent behavior is restricted to a small, stable region of the state-space. Additionally, these models allow easy translation into popular linear control strategies such as iterative Linear Quadratic Regulator / Gaussian (iLQR / iLQG). Different methods for solving for the model exist, but the simplest is structuring the model as the result of solving the least squared problem after each episode, $\arg\min_\omega ||X\omega - b||^2$, where $b = (\boldsymbol{s}_{n+1} - \boldsymbol{s}_n), \omega = \begin{bmatrix}\hat{\boldsymbol{A}} & \hat{\boldsymbol{B}}\end{bmatrix}, X = \begin{bmatrix}S & U\end{bmatrix}$ (S and U are stacked state and action vectors). The next state is then predicted with a linear system as $\hat{s}_{n+1} = \hat{\boldsymbol{A}}\boldsymbol{s}_n + \hat{\boldsymbol{B}}\boldsymbol{a}_n$.

Additionally, linear Gaussian models can be learned that enable some of the sampling for propagation benefits discussed above. In this case, the model of the dynamics follows as $f_\theta(\boldsymbol{s}_{n+1}|\boldsymbol{s}_n, \boldsymbol{a}_n) = \mathcal{N}(f_{\boldsymbol{s}n}\boldsymbol{s}_n + f_{\boldsymbol{a}n}\boldsymbol{a}_n + f_{cn}, \boldsymbol{F}_n)$, which can be solved for directly via the statistical properties of the data [RG99]. This formulation can be used in guided policy search [LA14] and updated with linear updates [FLA16] and recursive least squares [YC14]. While simple, efficient, and often accurate, these models are less suited for environments with complex dynamics or when generalization is needed.

Strengths Simple, fast at training and inference.

Weaknesses Cannot capture nonlinear dynamics, low potential generalization, worse scaling with dimensionality and data.

Integrating with Control Easy to use with classical control techniques (e.g. iLQR), differentiable.

Gaussian Processes

The most popular non-parametric model used in MBRL is the Gaussian Process (GP) [Ras03]. A GP takes the form of a non-parametric transition function,

$$f \sim \mathcal{GP}(m, k)\,, \tag{2.17}$$

where m is the mean of the dynamics with a covariance function k. These can take on many forms, but a popular solution is that of a regular Gaussian distribution. Given samples x_i in $i = 1, \ldots, n$, the model takes the mean $\mu_i = m(x_i) = \frac{1}{4}x_i^2$ and covariance $\Sigma_{ij} = k(x_i, x_j) = \exp(-\frac{1}{2}(x_i - x_j)^2)$, resulting in the model as $f \sim \mathcal{N}(\mu, \Sigma)$.

Gaussian Processes have been studied extensively in the context of offline dynamics modeling [WHF06; WFT14; HK11] and for model-based policy search [DR11; BSWR14; GWH14].

Strengths Stable uncertainty quantification, strong model accuracy.

Weaknesses Does not scale well above low state-action dimension.

Integrating with Control Uncertainty estimates can be useful for control, inverting kernel can be slow for online decisions.

Hand-designed Tools

Environment Simulators Many papers make use of existing simulators that can simulate the dynamics for use in planning, such as examples in robotics [TET12] or games [KBM+20]. Using environment simulators can be powerful for demonstrating control for the first time, though serious limitations can preclude transferring these methods to the real world. For example, if a simulator cannot set arbitrary initial states, it is not suitable for online decision making algorithms planning from a measurement. Additionally, building simulators can be very expensive and is only attainable for relatively well understood systems.

Dynamical Systems An alternate to building a simulator of the entire world is for engineers to develop a suitable state-space model of the dynamical system. These can take the form of a control-affine system,

$$\boldsymbol{s}_t = \boldsymbol{A}(\boldsymbol{s}_t, \boldsymbol{a}_t)\boldsymbol{s}_t + \boldsymbol{B}(\boldsymbol{s}_t, \boldsymbol{a}_t)\boldsymbol{a}_t + \boldsymbol{c}(\boldsymbol{s}_t, \boldsymbol{a}_t)\,. \tag{2.18}$$

These methods build extensively on techniques of optimal control.

Strengths Can require little-to-no environmental data, fast and parallel inference, interpretable.

Weaknesses Requires extremely precise domain knowledge / engineering time and often limited to linear systems.

Integrating with Control Integrates well with most control types and allows use of optimal control techniques (e.g. iLQR).

Replay Memory

The replay buffer can be viewed as a statistical model of the environment by querying transitions. A replay buffer is a type of model [HHA19; VS15] with remarkable similarities to a tabular dynamics model, but it is limited in its ability to generalize to new states. Hindsight experience replay [AWR+17] actualizes the goal of using the replay buffer as a model, but this formulation lacks an output of a structured understanding of dynamics. Similarly, [ESL19] use the replay buffer as a graph to search over for control.

Other Models

There are many other methods for learning a model of a system to be use for control. Some include using a convolutional neural network to learn system parameters of a physical

system [WYLFT15], a graph neural network modeling over system parameters [SHS+18], tabular dynamics models [Tou97; Boo97], and more. These other methods often increase the specificity of the model in order to match the system. Such matching often improves accuracy, but comes at the cost of generalization and implementation complexity.

2.4 Propagating Trajectories

All methods for predicting trajectories with variants of the one-step model operate via composed function calls of the learned model, as:

$$\hat{\boldsymbol{s}}_{t+h} = f_\theta(\dots f_\theta(f_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t), \boldsymbol{a}_{t+1}) \dots, \boldsymbol{a}_{t+h}) \,. \tag{2.19}$$

In this, any dynamics model f carries a prediction error $\epsilon_t = \hat{s}_t - s_t$. The resulting error growth is referred to as *compounding error* [LPC22], which grows multiplicatively by the next prediction's input being subject to all past errors in the prediction, as

$$\hat{\boldsymbol{s}}_{t+h} = f_\theta(\dots f_\theta(f_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) + \epsilon_t, \boldsymbol{a}_{t+1}) + \epsilon_{t+1} \dots, \boldsymbol{a}_{t+h}) + \epsilon_{t+h} \,. \tag{2.20}$$

Different model types allow for different propagation modalities that can reduce compounding error, enable multi-modal predictions, and carry more stable uncertainty propagation.

There are different propagation methods for translating from a distribution to a prediction. The simplest, *expectation* propagation assumes takes $\boldsymbol{s}_t = \mu_{\boldsymbol{\theta}}$ (only option for MSE loss without an ensemble). Alternatively, a state can be propagated by *sampling* by selecting a value from a unit Guassian random vector, $\boldsymbol{\eta} \sim \mathcal{N}(0, 1)$ and predicting the next state as $\boldsymbol{s}_t = \mu_{\boldsymbol{\theta}} + \Sigma_{\boldsymbol{\theta}}\boldsymbol{\eta}$. This can also be denoted by sampling directly from the distribution induced by the model, but this notation can be confusing when discussing both probabilistic and deterministic models.

With ensembles, additional methods for propagating trajectories are available. The expectation propagation can be expended by taking the mean over the bootstrap ensemble members, though this method has no capability to address bias or utilize the trained uncertainty estimates. Moment matching can be used to re-sample the particles after each step corresponding to a smooth Gaussian distribution [GMR16]. This represents a gain on expressivity over the expectation propagation, but lacks the ability to express multi-modal trajectories.

[CCML18] proposes two methods to propagating trajectories with probabilistic ensembles, Trajectory Sampling (TS). Two variants of TS – TS1 and TS∞ – differ on how the ensemble members are used and which types of uncertainty are captured. TS1 samples a new ensemble member for each particle for each dynamics step in the unrolling process to mitigate the bias of any one model. The re-sampling per step can be incremented from TSi to TS∞, where one bootstrap member remains assigned to each particle to maintain the separation of aleoteric and epistemic uncertainty. The iterative stepping through distributions is similar to that of a particle filter without the re-shaping of distributions at each step. Each particle stepping through the models per-step allows the particles to capture multiple modes.

Chapter 3

Using MBRL: Case-study with Sample-based Control

This short chapter is based on our work to control novel robots and a Crazyflie quadrotor [LDY+19]. This section shows how a simple MBRL setup can be used to solve a real world problem.

3.1 Experimental Setup

In this section, we use as experimental hardware platform the open-source Crazyflie 2.0 quadrotor [Bit16]. The Crazyflie is 27 g and 9 cm^2, so the rapid system dynamics create a need for a robust controller; by default, the internal PID controller used for attitude control runs at 500 Hz, with Euler angle state estimation updates at 1 kHz. This section specifies the Robot Operating System (ROS) base-station and the firmware modifications required for external stability control of the Crazyflie.

All components we used are based on publicly available and open source projects. We used the Crazyflie ROS interface supported here: github.com/whoenig/crazyflie_ros [HA17]. This interface allows for easy modification of the radio communication and employment of the learning framework. Our ROS structure is simple, with a Crazyflie subscribing to PWM values generated by a controller node, which processes radio packets sent from the quadrotor in order to pass state variables to the model predictive controller (as shown in Fig. 3.2). The Crazyradio PA USB radio is used to send commands from the ROS server; software settings in the included client increase the maximum data transmission bitrate up to 2 Mbps and a Crazyflie firmware modification improves the maximum traffic rate from 100 Hz to 400 Hz.

In packaged radio transmissions from the ROS server we define actions directly as the pulse-width modulation (PWM) signals sent to the motors. To assign these PWM values directly to the motors we bypass the controller updates in the standard Crazyflie firmware by changing the motor power distribution whenever a CRTP Commander packet is received (see Fig. 3.2). The Crazyflie ROS package sends empty ping packets to the Crazyflie to ask

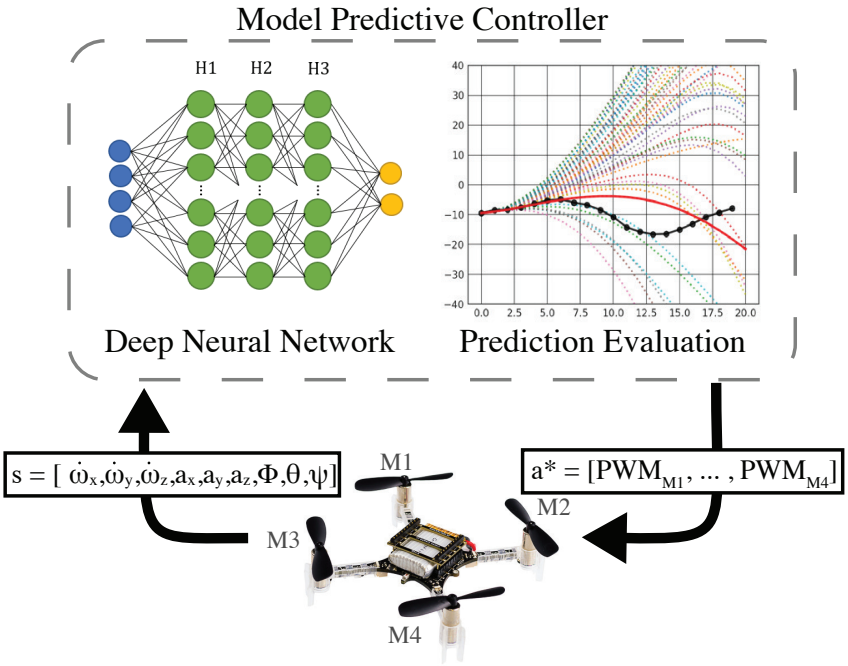


Figure 3.1: The reinforcement learning control loop used to stabilize the Crazyflie. Within the model predictive controller, the agent uses a neural network to unroll predictions and select an action. Using deep model-based reinforcement learning, the quadrotor reaches stable hovering with only 10,000 trained datapoints – equivalent to 3 minutes of flight.

for logging data in the returning acknowledgment packet; without decreasing the logging payload and rate we could not simultaneously transmit PWM commands at the desired frequency due to radio communication constraints. We created a new internal logging block of compressed IMU data and Euler angle measurements to decrease the required bitrate for logging state information, trading state measurement precision for update frequency. Action commands and logged state data are communicated asynchronously; the ROS server control loop has a frequency set by the ROS rate command, while state data is logged based on a separate ROS frequency. To verify control frequency and reconstruct state action pairs during rollouts we use a round-trip packet ID system.

3.2 Learning System

The foundation of a controller in MBRL is a reliable forward dynamics model for predictions. In this paper, we refer to the current state and action as s_t and a_t, which evolve according to the dynamics $f(s_t, a_t)$. Generating a dynamics model for the robot often consists of training a NN to fit a parametric function f_θ to predict the next state of the robot as a discrete

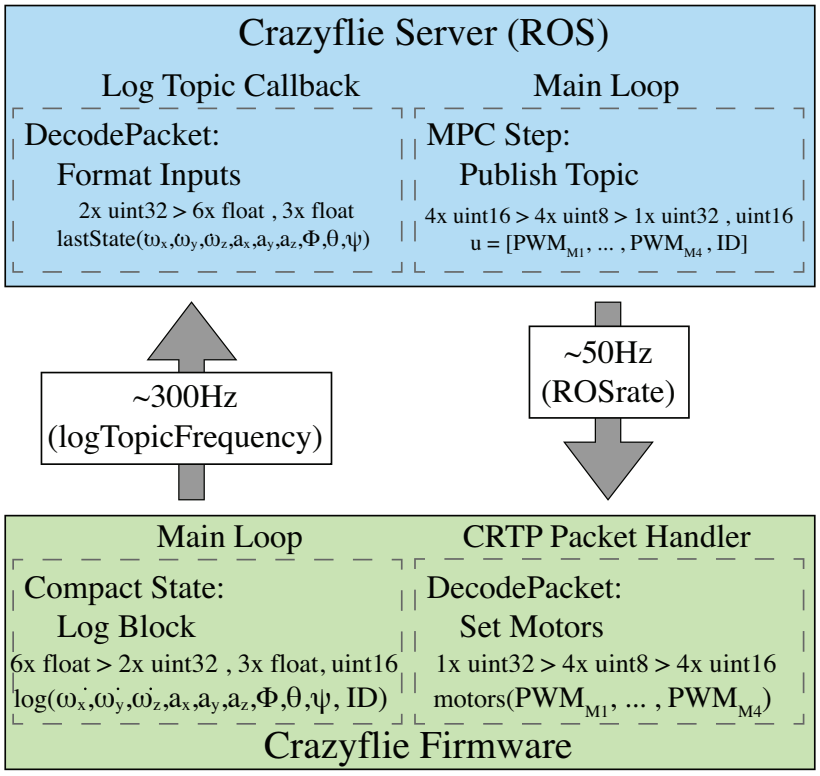


Figure 3.2: The ROS computer passes control signals and state data between the MPC node and the Crazyflie ROS server. The Crazyflie ROS server packages Tx PWM values to send and unpacks Rx compressed log data from the robot.

change in state $s_{t+1} = s_t + f_\theta(s_t, a_t)$. In training, using a probabilistic loss function with a penalty term on the variance of estimates, as shown in Eq. (4.11), better clusters predictions for more stable predictions across multiple time-steps [CCML18]. The probabilistic loss fits a Gaussian distribution to each output of the network, represented in total by a mean vector μ_θ and a covariance matrix Σ_θ

$$l = \sum_{n=1}^{N} [\mu_\theta(s_n, a_n) - s_{n+1}]^T \Sigma_\theta^{-1}(s_n, a_n)[\mu_\theta(s_n, a_n) - s_{n+1}] + \log \det \Sigma_\theta(s_n, a_n)\,. \tag{3.1}$$

The probabilistic loss function assists model convergence and the variance penalty helps maintain stable predictions on longer time horizons. Our networks implemented in Pytorch train with the Adam optimizer for 60 epochs with a learning rate of .0005 and a batch size of 32. Fig. 3.3 summarizes the network design. The network structure was cross validated offline for prediction accuracy verses potential control frequency. Initial validation of training parameters was done on early experiments, and the final values are held constant for each rollout in the experiments reported. The validation set is a random subset of measured (s_t, a_t, s_{t+1}) tuples in the pruned data.

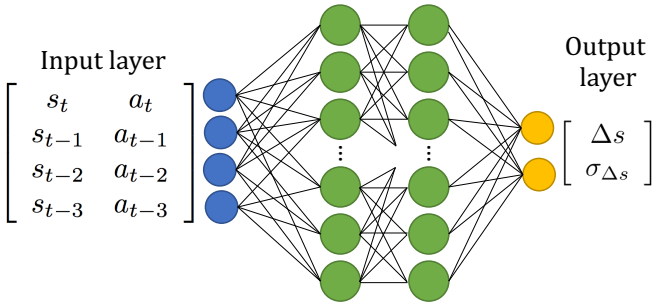


Figure 3.3: The NN dynamics model predicts the mean and variance of the change in state given the past 4 state-action pairs. We use 2 hidden layers of width 250 neurons.

Additional dynamics model accuracy could be gained with systematic model verification between rollouts, but experimental variation in the current setup would limit empirical insight and a lower model loss does not guarantee improved flight time. Our initial experiments indicate improved flight performance with forward dynamics models minimizing the mean and variance of state predictions versus models minimizing mean squared prediction error, but more experiments are needed to state clear relationships between more model parameters and flight performance.

Training a probabilistic NN to approximate the dynamics in this high-noise task requires pruning of logged data (e.g. dropped packets cause large time-steps and nonphysical dynamics that must be removed) and scaling of variables per-dimension to assist model convergence. Our state s_t is the vector of Euler angles (yaw, pitch, and roll), linear accelerations, and angular accelerations, reading

$$s_t = \left[\dot{\omega}_x,\ \dot{\omega}_y,\ \dot{\omega}_z,\ \phi,\ \theta,\ \psi,\ \ddot{x},\ \ddot{y},\ \ddot{z}\right]^T. \tag{3.2}$$

The Euler angles are from the an internal complementary filter, while the linear and angular accelerations are measured directly from the on-board MPU-9250 9-axis IMU. In practice, for predicting across longer time horizons, modeling acceleration values as a global next state rather than a change in state increased the length of time horizon in composed predictions before the models diverged. While the Euler angle predictions are stable, the raw accelerations are corrupter by high sensor noise and are extremely difficult to predict, so all the linear and angular accelerations are trained to fit the global next state (rather than a delta-state formulation shown in Eq. (2.6)).

We combine the state data with the four PWM values, $a_t = [m_1, m_2, m_3, m_4]^T$, to get the system information at time t. The NNs are cross-validated on a held out set to confirm using all state data (i.e., including the relatively noisy raw measurements from all flights, rather than a subset) improves prediction accuracy in the change in state.

While the dynamics for a quadrotor are often represented as a linear system, for a Micro Air Vehicle (MAV) at high control frequencies motor step response and thrust asymmetry

heavily impact the change in state, resulting in a heavily nonlinear dynamics model. The step response of a Crazyflie motor RPM from PWM 0 to max or from max to 0 is on the order of 250 ms, so our update time-step of 20 ms is short enough for motor spin-up to contribute to learned dynamics. To account for spin-up, we append past system information to the current state and PWMs to generate an input into the NN model that includes past time. From the exponential step response and with a bounded possible PWM value within $p_{eq} \pm 5000$, the motors need approximately 25 ms to reach the desired rotor speed; when operating at 50 Hz, the time step between updates is 20 ms, leading us to an appended states and PWMs history of length 4. This state action history length was validated as having the lowest test error on our data-set (lengths 1 to 10 evaluated). This yields the final input of length 52 to our NN, ξ, with states and actions combined to $\xi_t = \begin{bmatrix} s_t & s_{t-1} & s_{t-2} & s_{t-3} & a_t & a_{t-1} & a_{t-2} & a_{t-3} \end{bmatrix}^T$.

3.3 Control Optimization

This section explains how we incorporate our learned forward dynamics model into a functional controller. The dynamics model is used for control by predicting the state evolution given a certain action, and the MPC provides a framework for evaluating many action candidates simultaneously. We employ a 'random shooter' MPC, where a set of N randomly generated actions are simulated over a time horizon T. The best action is decided by a user designed objective function that takes in the simulated trajectories $\hat{X}(a, s_t)$ and returns a best action, a^*, as visualized in Fig. 3.4. The objective function minimizes the receding horizon cost of each state from the end of the prediction window to the current measurement.

The candidate actions, $\{a_i = (a_{i,1}, a_{i,2}, a_{i,3}, a_{i,4})\}_{i=1}^{N}$, are 4-tuples of motor PWM values centered around the stable hover-point for the Crazyflie. The candidate actions are constant across the prediction time horizon T. For a single sample a_i, each $a_{i,j}$ is chosen from a uniform random variable on the interval $[p_{eq,j} - \sigma, p_{eq,j} + \sigma]$, where $p_{eq,j}$ is the equilibrium PWM value for motor j. The range of the uniform distribution is controlled by the tuned parameter σ; this has the effect of restricting the variety of actions the Crazyflie can take. For the given range of PWM values for each motor, $[p_{eq} - \sigma, p_{eq} + \sigma]$, we discretize the candidate PWM values to a step size of 256 to match the future compression into a radio packet. This discretization of available action choices increases the coverage of the candidate action space. The compression of PWM resolution, while helpful for sampling and communication, represents an uncharacterized detriment to performance.

Our investigation focuses on controlled hovering, but other tasks could be commanded with a simple change to the objective function. The objective we designed for stability seeks to minimize pitch and roll, while adding additional cost terms to Euler angle rates. In the cost function, λ effects the ratio between proportional and derivative gains. Adding cost terms to predicted accelerations did not improve performance because of the variance of the

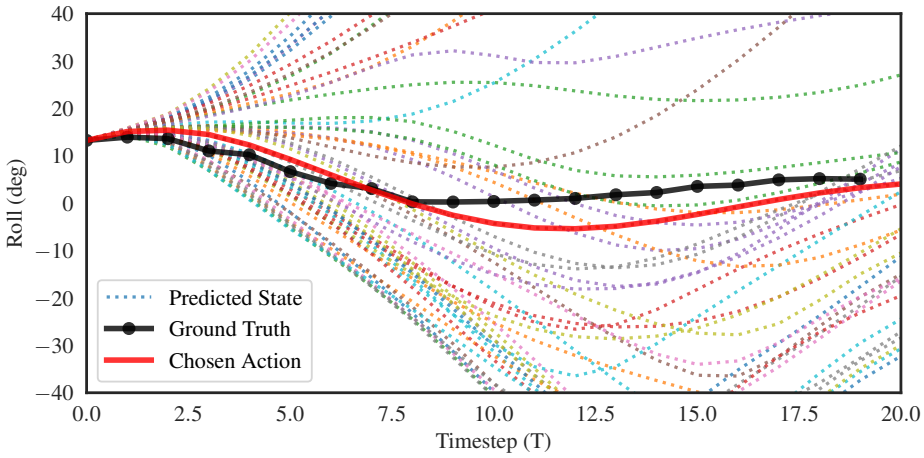


Figure 3.4: Predicted states for $N = 50$ candidate actions with the chosen "best action" highlighted in red. The predicted state evolution is expected to diverge from the ground truth for future t because actions are re-planned at every step.

predictions.

$$a^* = \arg\min_a \sum_{t=1}^{T} \lambda(\psi_t^2 + \theta_t^2) + \dot{\psi}_t^2 + \dot{\theta}_t^2 + \dot{\phi}_t^2 \,. \tag{3.3}$$

Our MPC operates on a time horizon $T = 12$ to leverage the predictive power of our model [1]. Higher control frequencies can run at a cost of prediction horizon, such as $T = 9$ at 75 Hz or $T = 6$ at 100 Hz. The computational cost is proportional to the product of model size, number of actions (N), and time horizon (T). At high frequencies the time spanned by the dynamics model predictions shrinks because of a smaller dynamics step in prediction and by having less computation for longer T, limiting performance. At 50 Hz, a time horizon of 12 corresponds to a prediction of 240 ms into the future. Tuning the parameters of this methodology corresponds to changes in the likelihood of taking the best action, rather than modifying actuator responses, and therefore its effect on performance is less sensitive than changes to PID or standard controller parameters. At 50 Hz, the predictive power is strong, but the relatively low control frequencies increases susceptibility to disturbances in between control updates. A system running with an Nvidia Titan Xp attains a maximum control frequency of 230 Hz with $N = 5000, T = 1$. For testing we use locked frequencies of 25 Hz and 50 Hz at $N = 5000,\ T = 12$.

[1] Experiments executed on NVIDIA 1080.

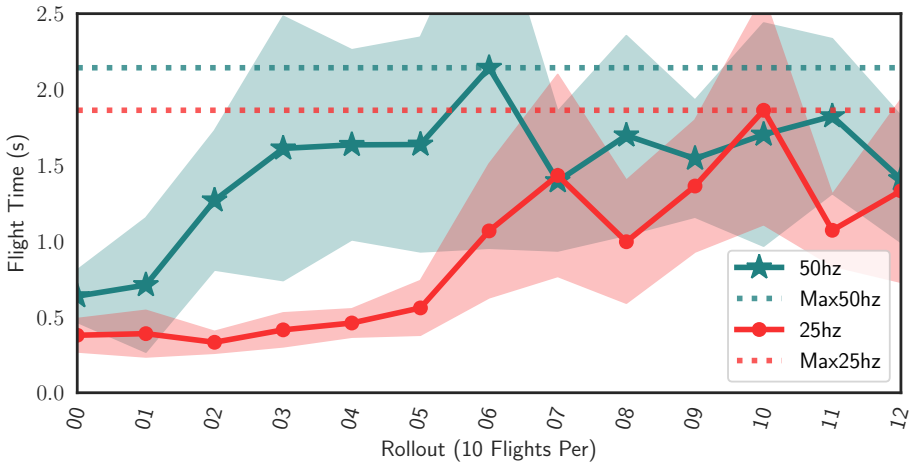


Figure 3.5: Mean and standard deviation of the 10 flights during each rollout learning at 25 Hz and 50 Hz. The 50 Hz shows a slight edge on final performance, but a much quicker learning ability per flight by having more action changes during control.

3.4 Results

The performance of our controller is measured by the average flight length over each roll-out. Failure is often due to drift induced collisions, or, as in many earlier roll-outs, when flights reach a pitch or roll angle over 40°. In both cases, an emergency stop command is sent to the motors to minimize damage. Additionally, the simple on-board state estimator shows heavy inconsistencies on the Euler angles following a rapid throttle ramping, which is a potential limiting factor on the length of controlled flight. Notably, a quadrotor with internal PIDs will still fail regularly due to drift on the same time frame as our controller; it is only with external inputs that the PID controllers will obtain substantially longer flights. The drift showcases the challenge of using attitude controllers to mitigate an offset in velocity.

Learning Process The learning process follows the RL framework of collecting data and iteratively updating the policy. We trained an initial model f_0 on 124 and 394 points of dynamics data at 25 Hz and 50 Hz, respectively, from the Crazyflie being flown by a random action controller. Starting with this initial model as the MPC plant, the Crazyflie undertakes a series of autonomous flights from the ground with a 250 ms ramp up, open-loop takeoff followed by on-policy control while logging data via radio. Each roll-out is a series of 10 flights, which causes large variances in flight time. The initial roll-outs have less control authority and inherently explore more extreme attitude orientations (often during crashes), which is valuable to future iterations that wish to recover from higher pitch and/or roll. The random and first three controlled roll-outs at 50 Hz are plotted in Fig. 3.6 to show the rapid improvement of performance with little training data.

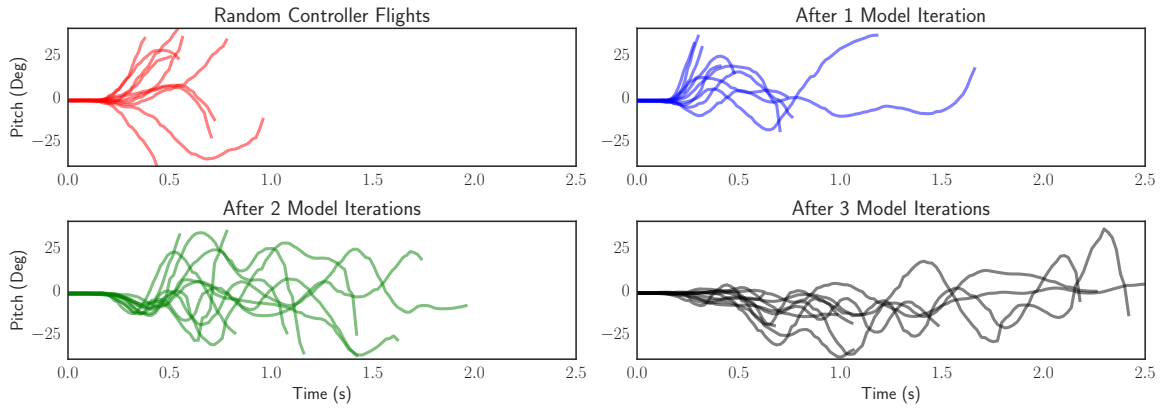


Figure 3.6: The pitch over time for each flight in the first four model training iterations of learning at 50 Hz showing the rapid increase in control ability on limited data. The models are trained once per ten flights, so each quadrant of this figure represents 10 flights with one learned dynamics model. The random and first controlled roll-out show little ability, but by roll-out 3 it flies for > 2 seconds.

The full learning curves are shown in Fig. 3.5. At both 25 Hz and 50 Hz the rate of flight improvement reaches its maximum once there is 1,000 trainable points for the dynamics model, which takes longer to collect at the lower control frequency. The improvement is after roll-out 1 at 50Hz and roll-out 5 at 25Hz. The longest individual flights at both control frequencies is over 5 s. The final models at 25 Hz and 50 Hz are trained on 2,608 and 9,655 points respectively, but peak performance is earlier due to dynamics model convergence and hardware lifetime limitations.

Performance Summary This controller demonstrates the ability to hover, following a "clean" open-loop takeoff, for multiple seconds (an example is shown in Fig. 3.8). At both 25 Hz and 50 Hz, once reaching maximum performance in the 12 roll-outs, about 30% of flights fail to drift. The failures due to drift indicate the full potential of the MBRL solution to low-level quadrotor control. An example of a test flight segment is shown in Fig. 3.7, where the control response to pitch and roll error is visible.

The basis of comparison, typical quadrotor controllers, achieve better performance, but with higher control frequencies and engineering design iterations leveraging system dynamics knowledge. With the continued improvement of computational power, the performance of this method should be re-characterized as potential control frequencies approach that of PID controllers. Beyond comparison to PID controllers with low computational footprints, the results warrant exploration of MBRL for novel dynamical systems. In less than 10 minutes of clock time, and only 3 minutes of training data, we present comparable, but limited, performance that is encouraging for future abilities to match and surpass basic controllers. Moving the balance of this work further towards domain specific control would likely improve

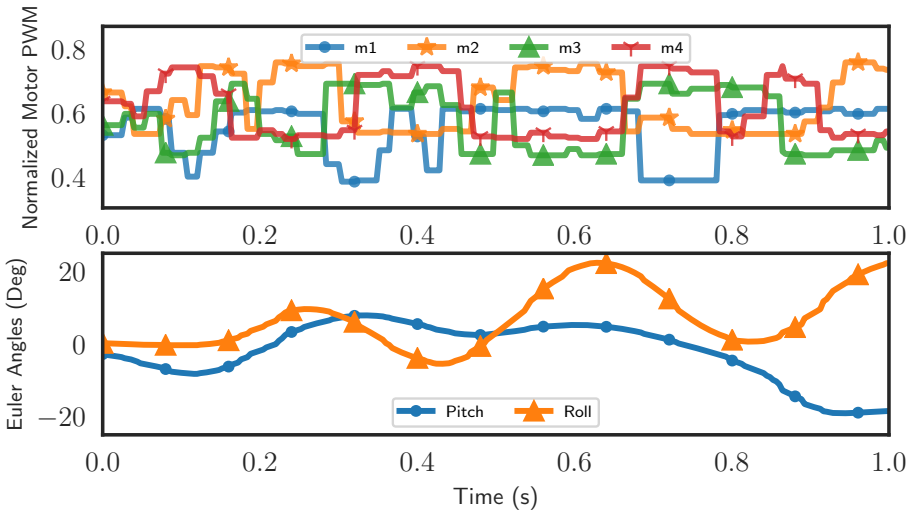


Figure 3.7: The performance of the 50 Hz controller. (*Above*) The controlled PWM values over time, which visibly change in response to angle oscillations. (*Below*) Pitch and roll.

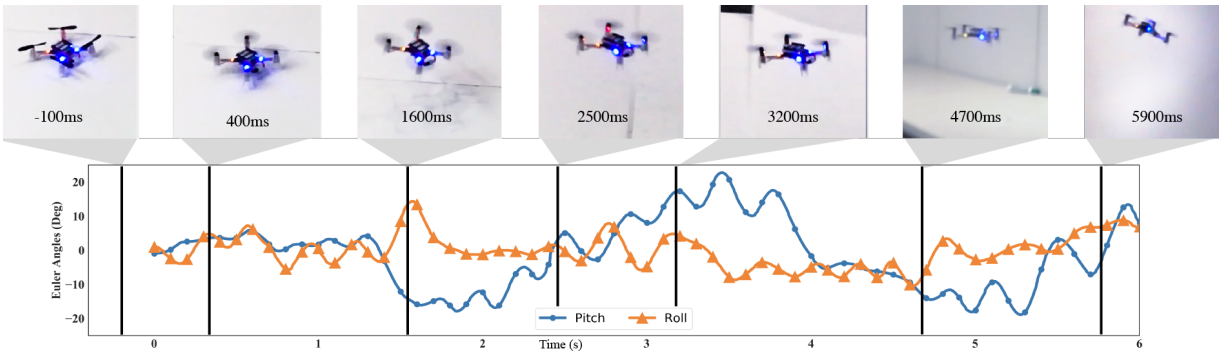


Figure 3.8: A full flight of Euler angle state data with frames of the corresponding video. This flight would have continued longer if not for drifting into the wall. The relation between physical orientation and pitch and roll is visible in the frames. The full video is online on the accompanying website.

performance, but the broad potential for applications to more and different robotic platforms compels exciting future use of MBRL.

Chapter 4

Compounding Prediction Errors in Learned Models

In this chapter, we study in detail the causes of compounding prediction errors in learned dynamics models. This chapter is primarily covered in our recent paper highlighting this issue [LPC22]. The issue, frequently referenced in state-of-the-art papers [CNF+18; WBC+19] and other tools [AMKL19; LWZPC21] for model-based reinforcement learning is not well studied from first principles even though it is referenced as a substantial limiting factor on performance. In this chapter we characterize the impact of three types of problem properties relevant to long-term prediction accuracy in MBRL: a) the *system* properties inherent to the environment and controller used; b) the *model* training and prediction linked with the optimization; and c) *other* factors at the interface of systems with models.

Sec. 4.3 focuses on how the environment and agent impact modelling accuracy. It starts by covering The Underlying Dynamics by observing and setting the eigenvalues of state-space systems. Then, this section covers often discussed properties of learning models such as Process Noise and State-space Dimension. The section concludes with observations of how the controller can impact accuracy, by studying Data Distribution & Density and Recomputing Actions for prediction.

Sec. 4.4 highlights how different modeling decisions, e.g. which type of model or loss function, impacts accuracy given a static dataset. This section covers Prediction Formulation & Training, Model Capacity, and Data Normalization.

Sec. 4.5 demonstrates how broader examples of prediction can impact long-term predictions. This section uses the Lorenz system to showcase the limitations of Predicting Chaotic Dynamics. Next, the section shows how different temporal abstractions influence accuracy with a discussion of Control Frequency & Signal to Noise Ratio.

The chapter concludes with real-world examples in Sec. 4.6 and tips for understanding compounding error in Sec. 4.7.

4.1 Problem Formulation

As discussed in Sec. 2.4, to the prediction of the long-term future, a one-step dynamic model is often recursively applied as

$$\hat{\boldsymbol{s}}_{t+h} = f_\theta(\dots f_\theta(f_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t), \boldsymbol{a}_{t+1}) \dots, \boldsymbol{a}_{t+h}) \,. \tag{4.1}$$

Any parametrization of the dynamics model f carries a prediction error $\epsilon_t = \hat{s}_t - s_t$. It is often observed that this error grows multiplicatively by the next prediction's input being subject to all past errors in the prediction, as

$$\hat{\boldsymbol{s}}_{t+h} = f_\theta(\dots f_\theta(f_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t) + \epsilon_t, \boldsymbol{a}_{t+1}) + \epsilon_{t+1} \dots, \boldsymbol{a}_{t+h}) + \epsilon_{t+h} \,. \tag{4.2}$$

The central metric we will use to quantify and visualize compounding error is the mean-squared prediction error (MSE) at each step. The action sequence used with a given trajectory, $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \dots, \boldsymbol{a}_h\}$ is provided when planning the trajectory and measuring its performance. To that end, we train a dynamics model f_θ, and use it to predict to a horizon h, steps into the future, generating a predicted trajectory $\hat{s}_i \forall i \in [1, h]$. Then, the predicted error is computed by summing across all of the state dimensions at the given time-step as:

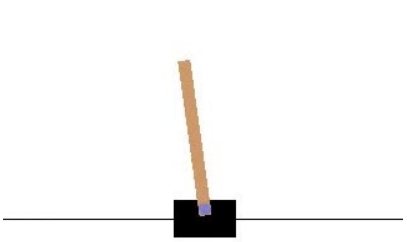
$$\text{MSE}_t = \sum_{d=0}^{d_s} \|\hat{\boldsymbol{s}}_{t,d} - \boldsymbol{s}_{t,d}\|_2^2 \,. \tag{4.3}$$

For each experiment and environment, the MSE is normalized per-state to $[0, 1]$ to make the calculated MSE represent average predictive accuracy proportional to the relative state error rather than the numerical error (e.g. to normalize across states of different state types like positions and velocities). The goal is to make the errors shown more intuitive – an mean squared error of 1 represents an error across each state averaging to 100%.

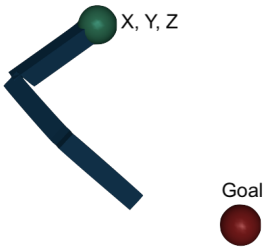
This chapter includes many models to evaluate predictions, as are discussed in Sec. 2.2.

One-step Dynamics Models We use the delta-state formulation that is popular for regularizing the prediction distribution, $\boldsymbol{s}_{t+1} = \boldsymbol{s}_t + f_\theta(\boldsymbol{s}_t, a_t)$, and compare to the true-state variant, $\boldsymbol{s}_{t+1} = f_\theta(\boldsymbol{s}_t, a_t)$. The true-state models are denoted with a -S, such as the true-state probabilistic ensemble *PE-S*. These formulations can be used with multiple loss functions, including Mean Squared Error (MSE) for deterministic models and Negative Log Likelihood (NLL) for probabilistic models. All model types can be used with an ensemble that weights predictions across multiple trained models, denoted with *E*.

In this work, the studies primarily compare the delta-state and true-state prediction formulations with simple deterministic models (D) and with rich probabilistic ensembles (PE). For the probabilistic models, the trajectories are propagated with expectation based propagation, more options for the PE models are detailed in [CCML18]. The chapter compares one-step models to *linear models* (LIN) based on least-squares learning of a linear predictor and *zero models* (ZERO) that return a predicted state of the zero vector at each step. Additional model training details are included in Sec. 4.2.



(a) Cartpole.



(b) Reacher.

(c) Quadrotor.

Figure 4.1: Experimental platforms used for studying compounding error.

4.2 Experimental Setting

Here, the systems evaluated in this chapter are explained.

State-space System

To test the possible causes of compounding error, we test the ability of deep one-step models to predict variations of a clearly defined system. Consider a state-space system defined with a state $s \in \mathbb{R}^3$ and an action $a \in \mathbb{R}$ as $\boldsymbol{s}_{t+1} = \boldsymbol{A}\boldsymbol{s}_t + \boldsymbol{B}\boldsymbol{a}_t + \omega_t$. Therein, we define $\boldsymbol{A}$ and $\boldsymbol{B}$ as follows to control the poles of the system:

$$\boldsymbol{A} = \begin{bmatrix} \rho & a_1 & a_2 \\ 0 & \rho & a_3 \\ 0 & 0 & \rho \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{4.4}$$

We set the desired eigenvalues, or poles, of the system to be ρ. The other parameters of the system are sampled randomly for each trial as $a_i, b_i \sim \mathcal{U}(-1, 1)$, which act as a source of uncertainty. The default process-noise in the environment $\omega_i \sim \mathcal{U}(-0.01, 0.01)$. All actions are chosen randomly from $\mathcal{U}(-1, 1)$ and act via the randomly generated $\boldsymbol{B}$ matrices.

For a discrete-time state-space system, the time evolution of the state over time can be solved for explicitly. This time evolution, shown in Eq. (4.5), is the goal of what these composed one-step systems attempt to model, yet result in compounding errors. The solution to the discrete transition dynamics takes the form an transient response from the initial state and a forced response corresponding to the applied action sequence:

$$s_t = \underbrace{\boldsymbol{A}^t \boldsymbol{s}_0}_{\text{Transient}} + \underbrace{\sum_{l=0}^{k-1} A^{k-l-1} B a(l)}_{\text{Forced Response}}. \tag{4.5}$$

In this case, the magnitude of the transient response at a given time is proportional to the

eigenvalues of the matrix:

$$\|s_t\| \leq \|\boldsymbol{A}^t\| \cdot \|\boldsymbol{s}_0\| + \sum_{l=0}^{k-1} \|A^{k-l-1}B\| \cdot \|a(l)\| \,. \tag{4.6}$$

For the state-space system, the time by which the prediction error converges to its minimum for a given pole is proportional to the number of steps by which the transient of the initial state will decay. The transient is proportional to powers of the dynamics matrix, $\|A\|^k$, which is proportional to the powers of the eigenvalues. For the poles in $\{0.01, 0.05, 0.1, 0.25, 0.5\}$ the number of time-steps until the steady state error is reached is proportional to the mean transient decay in Tab. 4.1. Example trajectories and one-step predictions are shown in Fig. 4.2. In this parametrization, the input at any time is randomly sampled, so the forced response becomes a source of noise.

Cartpole

We evaluate predictions of state and reward of cartpol ($d_s = 4$, $d_a = 1$) agents conditioned on a varied Linear Quadratic Regulator (LQR) [CD12] control laws (Fig. 4.4).

Reacher

The task associated with the reacher ($d_s = 15$, $d_a = 5$) environment is to maneuver the end-effector of the arm from an initial position state to an end position state. Our experiments control the agent using a Proportional-Integral-Derivative (PID) controller with randomly generated parameter vectors $\boldsymbol{K} \in \mathbb{R}^{15}$.

Quadrotor - Simulated

The quadrotor model ($d_s = 9$, $d_a = 4$) is based off the Crazyflie [GSWWK17] – an 27 g, open-source micro-aerial vehicle. The 12 state Euler-step simulation follows [MKC12]. The simulated controller is a linear, pitch and roll Proportional-Derivative controller with randomly sampled parameters.

Quadrotor - Real World

Due to the high noise on the accelerations measured by on-board sensors, we evaluate predicting a restricted state of Euler angles from direct motor voltages as:

$$s_t = \begin{bmatrix} \text{Yaw:}\psi & \text{Pitch:}\theta & \text{Roll:}\phi \end{bmatrix}, \quad a_t = \begin{bmatrix} V_1 & V_2 & V_3 & V_4 \,. \end{bmatrix} \tag{4.7}$$

When discretizing dynamics, lower sample rates yield more unstable eigenvalues, but the system can also gain by having relatively lower signal-to-noise ratio on the measured states.

Poles (ρ)	Transient decay $k : \|\boldsymbol{A}^k \boldsymbol{x}_0\| < 1 \times 10^{-4}$	Delta-state labels $\text{mean}(\|s_{t+1} - s_t\|) \pm \sigma(\cdot)$	True-state labels $\text{mean}(\|s_{t+1}\|) \pm \sigma(\cdot)$
0.01	4.7	0.019 ± 0.063	0.011 ± 0.021
0.05	6.1	0.020 ± 0.066	0.012 ± 0.020
0.10	7.4	0.018 ± 0.052	0.012 ± 0.026
0.25	11.2	0.017 ± 0.044	0.016 ± 0.048
0.50	21.8	0.020 ± 0.042	0.033 ± 0.087
0.75	55.5	0.029 ± 0.051	0.134 ± 0.299
0.90	168.3	0.086 ± 0.156	1.284 ± 2.610
0.95	N.A.	0.225 ± 0.351	8.511 ± 14.30
1.00	N.A.	7.442 ± 11.164	201.2 ± 450.5
1.10	N.A.	$6.5 \times 10^4 \pm 1.9 \times 10^5$	$7.0 \times 10^5 \pm 1.9 \times 10^6$

Table 4.1: Dataset properties for state-space systems with different eigenvalues. The transient decay is the number of discrete transitions on average by which the transient term in Eq. (4.5) decays below 1×10^{-4} (chosen based on the steady state prediction error that the most stable poles converge to on average). The mean and standard deviations of the data labels represent a relative challenge for the models – as the input and output normalizers need to compress a wider range of data to $\mathcal{N}(0, 1)$, the more sensitive the learning process becomes (for more on normalization, see Sec. 4.4).

Quadruped - Real World

The state, $s_t \in \mathbb{R}^{52}$, corresponds to the following:

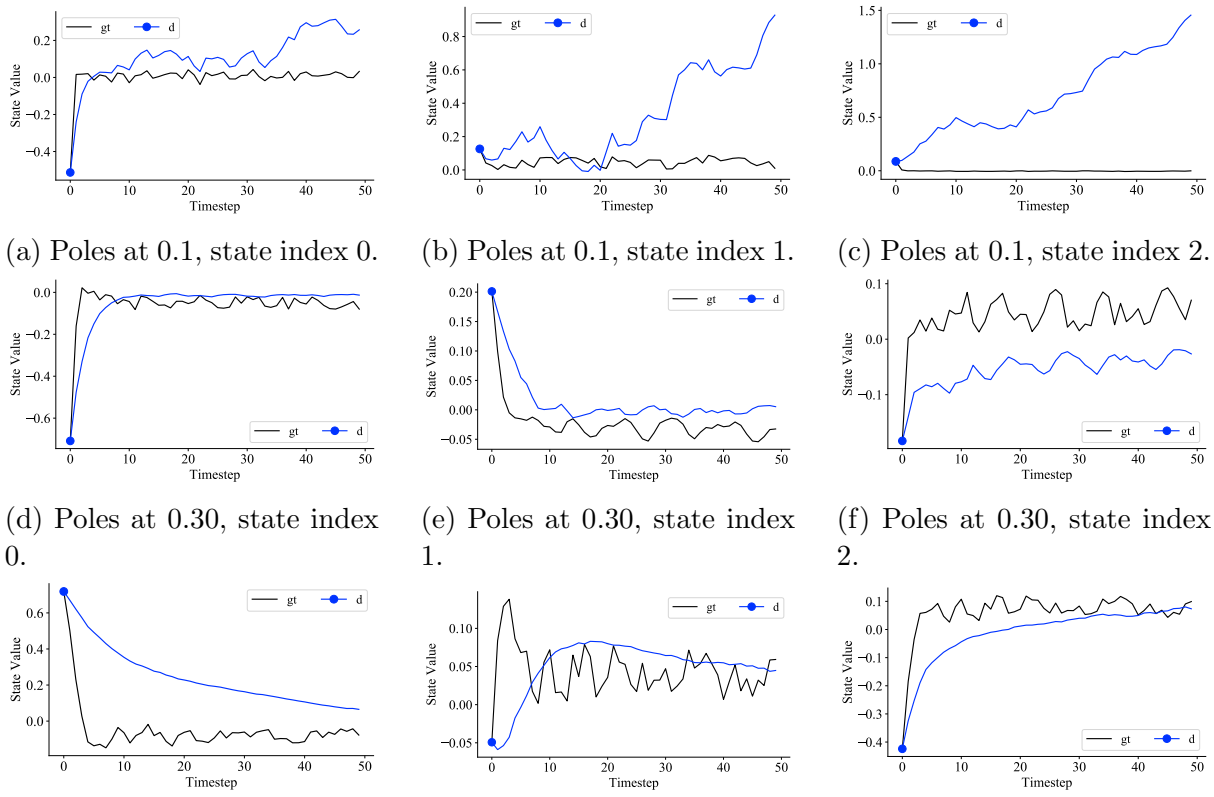
$$s_t = \begin{bmatrix} \boldsymbol{x} & \dot{\boldsymbol{x}} & \boldsymbol{\omega} & \boldsymbol{x}_k^f & \psi & \theta & \phi & \alpha_i & \dot{\alpha}_i & c_k^f \,. \end{bmatrix} \tag{4.8}$$

Here, $\boldsymbol{x} \in \mathbb{R}^3$ is the position of the robot base, $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular rates of the robot base, $\boldsymbol{x}_k^f \in \mathbb{R}^3$ is the position of the kth foot, ψ, θ, ϕ are the Euler angles, α_i is the joint angle for each of the 12 motors, and c^f is an indicator if each foot is in contact with the ground. The action, $a_t \in \mathbb{R}^{60}$, is a bimodal input, where for each of the 12 motors on the robot has 5 dimensional action space of desired joint position and velocities $(\alpha_i^*, \dot{\alpha}_i^*)$, low-level PID control coefficients (K_i^p, K_i^d), and set torque (τ_i):

$$a_t = \begin{bmatrix} \alpha_i^* & \dot{\alpha}_i^* & K_i^p & K_i^d & \tau_i \,. \end{bmatrix} \tag{4.9}$$

Model Training

To learn a model of the dynamics, we use a feedforward neural network with two hidden layers of width 256. Ensemble models use $E = 5$ members. The models are trained on 100



(a) Poles at 0.1, state index 0.

(b) Poles at 0.1, state index 1.

(c) Poles at 0.1, state index 2.

(d) Poles at 0.30, state index 0.

(e) Poles at 0.30, state index 1.

(f) Poles at 0.30, state index 2.

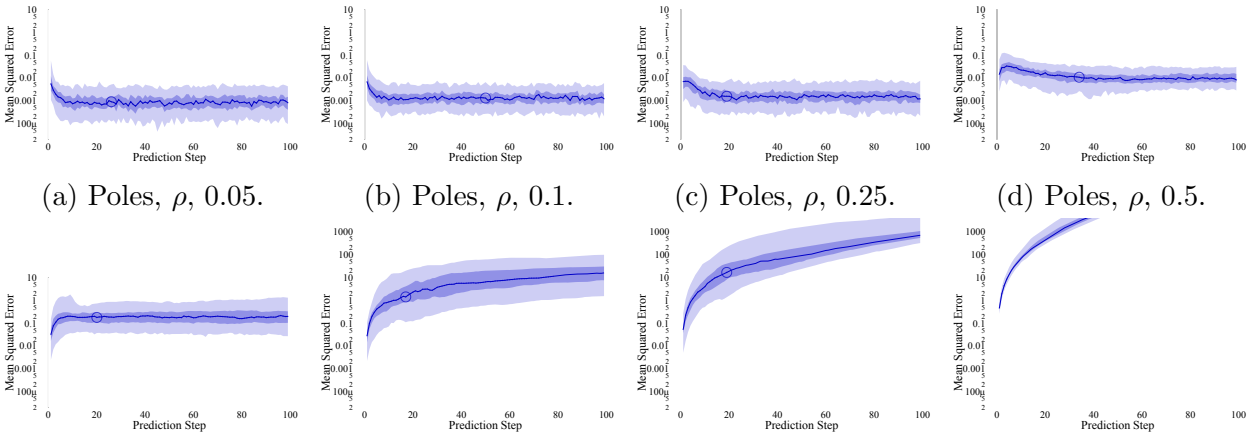
(g) Poles at 0.5, state index 0.

(h) Poles at 0.5, state index 1.

(i) Poles at 0.5, state index 2.

Figure 4.2: Example trajectories for the state-space system in three dimensions showcasing the predictions of deterministic models for $\rho = 0.1, 0.30, 0.5$. The three states for each pole correspond to one example trajectory, and the matrices and initial states are different for each of the representative poles shown. *gt* is the true state dynamics and *d* is the one-step model.

trajectories – all with different matrices $\boldsymbol{A}, \boldsymbol{B}$ with the same poles for the state-space systems. For the other environments, control policies are randomly sampled to create diverse training and testing data. The models are trained for 20 epochs with a learning rate of 0.0003 for deterministic and 0.000025 for probabilistic models with the Adam optimizer. Deterministic models use a batch size of 32 and probabilistic models use a batch size of 64. All state data is normalized to a unit Gaussian and action data is normalized to $[-1, 1]$ for training. The



(a) Poles, ρ, 0.05. (b) Poles, ρ, 0.1. (c) Poles, ρ, 0.25. (d) Poles, ρ, 0.5.

(e) Poles, ρ, 0.75. (f) Poles, ρ, 0.90. (g) Poles, ρ, 0.95. (h) Poles, ρ, 1.0.

Figure 4.3: Showing the compounding errors, formally the per-step MSE (median, 65^{th}, and 95^{th} percentiles), of state-space systems shown in Eq. (4.4) with different poles, ρ. Compounding errors vary substantially with the underlying poles of the environment and diverge when the poles approach instability. All models are trained and evaluated on separate datasets of 100 trajectories.

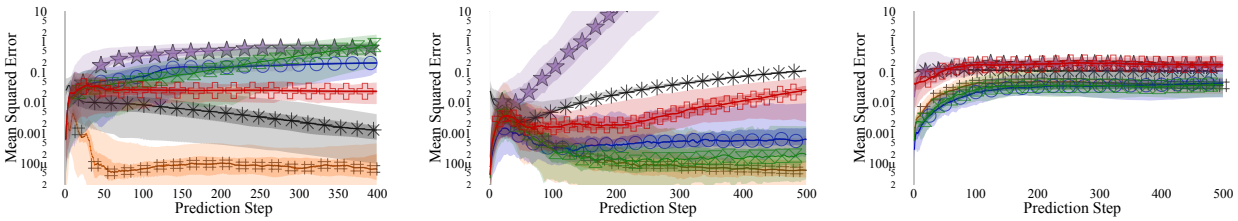
equations for computing the loss during training are shown for MSE and NLL:

$$l_{\text{MSE}} = \sum_{n=1}^{N} \| f_\theta(\boldsymbol{s}_n, \boldsymbol{a}_n) - \boldsymbol{s}_{n+1} \|_2^2 \,, \tag{4.10}$$

$$l_{\text{NLL}} = \sum_{n=1}^{N} [\mu_\theta(s_n, a_n) - s_{n+1}]^T \Sigma_\theta^{-1}(s_n, a_n)[\mu_\theta(s_n, a_n) - s_{n+1}] + \log \det \Sigma_\theta(s_n, a_n) \,. \tag{4.11}$$

Important to the convergence of supervised learning is the shape and magnitude of the training data. In Tab. 4.1, we compare the training data shape for the different state-space system eigenvalues when using true- and delta-state labels for the one-step dynamics model. As the eigenvalues become more unstable, the variation in the training labels grows exponentially. This effect can be counteracted with normalization if it is uniform, but the model loses the ability to differentiate between elements at fine scales, which could render the usefulness of the model low.

The linear model is the result of solving the least squared problem, $\arg\min_\omega ||X\omega - b||^2$, where $b = (\boldsymbol{s}_{t+1} - \boldsymbol{s}_t), \omega = \begin{bmatrix}\hat{\boldsymbol{A}} & \hat{\boldsymbol{B}}\end{bmatrix}, X = \begin{bmatrix}S & U\end{bmatrix}$ (S and U are stacked state and action vectors). The next state is then predicted with $\hat{s}_{t+1} = \hat{\boldsymbol{A}}\boldsymbol{s}_t + \hat{\boldsymbol{B}}\boldsymbol{a}_t$.



(a) Cartpole. $d_s + d_a = 5$. (b) Quadrotor. $d_s + d_a = 13$. (c) Reacher. $d_s + d_a = 20$.

LIN (#) ZERO (∗) D (●) D-S (⋆) PE (⋈) PE-S (+)

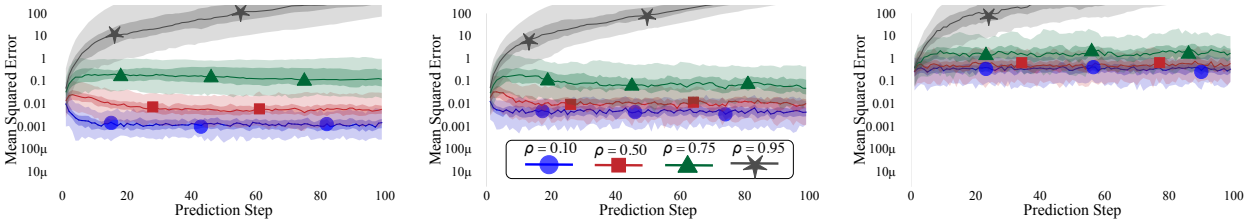
Figure 4.4: Comparing the MSE of prediction error per-step (median, 65^{th}, and 95^{th} percentiles) on common model types and parametrizations on simulated robotic tasks of different dynamics, simulators, and dimension. There is a trend of error of predictions increasing with the task difficulty, but there is high variability in the performance of any one model type when comparing across platforms. All model types are trained and evaluated on the same datasets, maintaining separate datasets of 100 trajectories for test-train split.

4.3 The Effects of System Properties

Underlying Dynamics

The underlying dynamics of a system to be modelled controls the relative complexity of the prediction landscape. One way to characterize the behavior of a system's behavior is through the poles, often computed as the eigenvalues for linear systems. To continue the state-space example, a discrete-time system is *stable* when the eigenvalues are within the unit-circle, $\|\rho\| < 1$. We vary the eigenvalues of our state-space system shown in Eq. (4.4) across a range of mostly stable and psuedo-stable values to compare the compounding error.

The results in Fig. 4.3 indicate that as eigenvalues approach instability, prediction accuracy quickly degrades. For stable systems, the error growth over time does not compound, but decreases over a short horizon before reaching steady state. Those systems with truly unstable poles, $\|\rho\| > 1$, diverge so rapidly that plotting and computing the magnitude of error is computationally intractable. It is also interesting that when the poles are notably stable, the relative *stable-ness* of the system does not have a large baring on prediction accuracy, as shown by the similarity in error between poles of $\rho < 0.25$. Examples of the compounding error on different simulated robotic tasks is shown in Fig. 4.4, which show substantial variation in compounding errors. With complex robotic systems it is often difficult to directly identify the eigenvalues, further increasing the difficulty of cross-platform comparisons. The Cartpole and Quadrotor environments represent stabilization tasks, which have similar error profiles when compared to the Reacher maniuplation task. The variation between platforms in both the magnitude and shape of compounding error motivates a deeper study of the causes, which could be revealed in other properties of the system such as the dimension or noise.



(a) 0× noise. (b) 10× noise. (c) 100× noise.

Figure 4.5: Comparing prediction accuracy when increasing the levels of process noise in the system above and below the default of $\omega_t \sim \mathcal{U}(-0.01, 0.01)$ on all dimensions (median, 65^{th}, and 95^{th} percentiles). The error between a system with default (shown in Fig. 4.3) and zero process noise shows that the default noise from the random action matrices determines the resulting prediction accuracy. An interesting feature is that when increasing the process noise from 10× to 15×, the modelling accuracy degrades by a factor of 15.

Process Noise

The underlying noise within the dynamics has a substantial impact on measurement and evolution of any dynamical system. The default state-space system has only process noise, sampled uniformly $\omega_i \sim \mathcal{U}(-0.01, 0.01)$. To measure the effects of this noise, we measure the prediction accuracy with the following multiples of the original noise: $0\times, 10\times, 100\times$. The results, shown in Fig. 4.5 indicate that noise can control the maximum accuracy. This floor is a primary contributor to model inaccuracy for stable poles $\rho = \{0.1, 0.5, 0.75\}$, but when the poles approach instability $\rho = 0.95$, the compounding error is similar across all noise levels.

An interesting observation of the learning process is the relation between the random actions, which the networked is informed of, and that of the process noise. By default, the input matrices $\boldsymbol{B}$ are randomized along with the control policy in each trajectory, so they computationally impossible for the general dynamics model to learn. In Fig. 4.6, the actions are zeroed so they no longer contribute as a disturbance on system dynamics, and the prediction accuracy shape is similar, but improves performance by about 100× when compared to Fig. 4.3.

For the robotic tasks shown in Fig. 4.4, the default noise varies dramatically. The Reacher has an unreported and low noise level (hidden within the Mujoco simulator), the Cartpole has uniform noise $\mathcal{U}(-0.1, 0.1)$ on all states, and the Quadrotor has a noise sampled from $\mathcal{N}(0, 0.0001)$. None of these simulators vary the level of noise relative to the type of the state variables (for example, a position in meters can take much lower magnitudes than an angle in degrees). The learned models on the quadrotor system converged to substantially lower levels, potentially indicating that deep models continue to improve with substantial noise reduction.

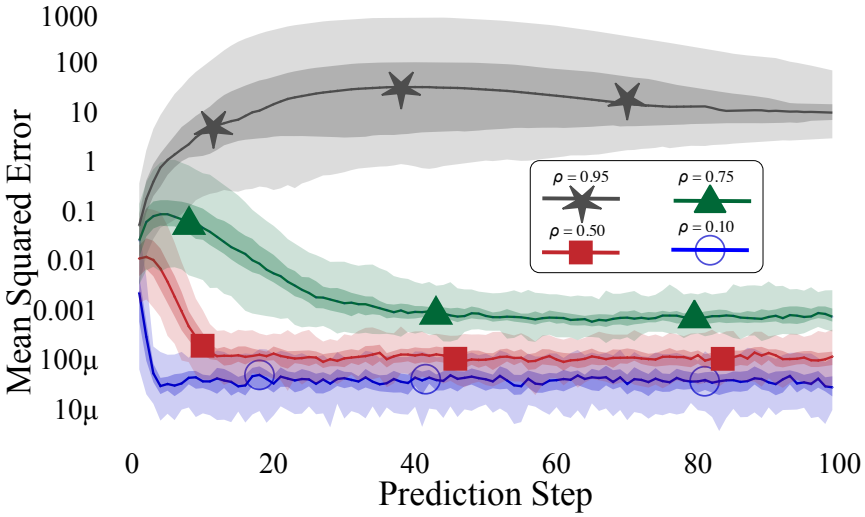
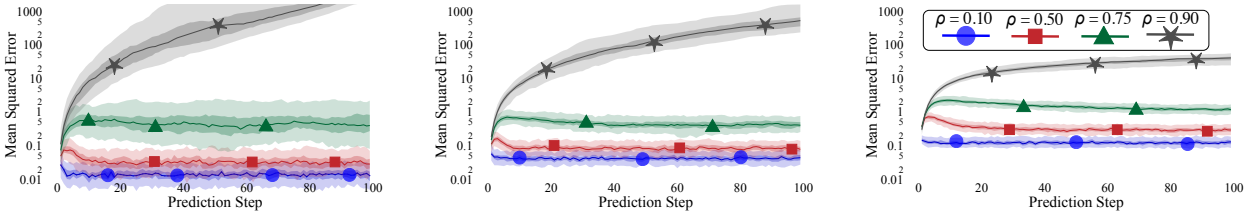


Figure 4.6: Showing how the randomly sampled input matrices, $\boldsymbol{B}$, and actions affect the per-step MSE with different eigenvalues (median, 65$^{\text{th}}$, and 95$^{\text{th}}$ percentiles) by collecting new data and evaluate newly trained models with $\boldsymbol{B} = 0$. The random actions are the second leading cause of prediction error behind the unstable eigenvalues.



(a) $d_s = 9$. (b) $d_s = 27$. (c) $d_s = 81$.

Figure 4.7: Comparing compounding error with different state dimensions to see if input-output prediction size challenges the models when the underlying dynamics are regularized (shown is the MSE with median, 65$^{\text{th}}$, and 95$^{\text{th}}$ percentiles). State size does not have a substantial effect on the modeling error (the decreased variance of the error at each step in the state-sizes could be due to averaging over more state dimensions).

State-space Dimension

A large motivation to using deep models for predicting dynamics is the ability to extend to higher dimensional tasks. While early work has shown that deep networks are useful for high dimensional tasks (such as [NKLK20; LDY+19] with state-action spaces over 40 dimensional), given the difficulty of comparing across different systems more controlled studies of prediction dimensionality are needed. Learning one-step models scale the input and output dimensions with respect to the environment state. To evaluate this, we scale the dimension-

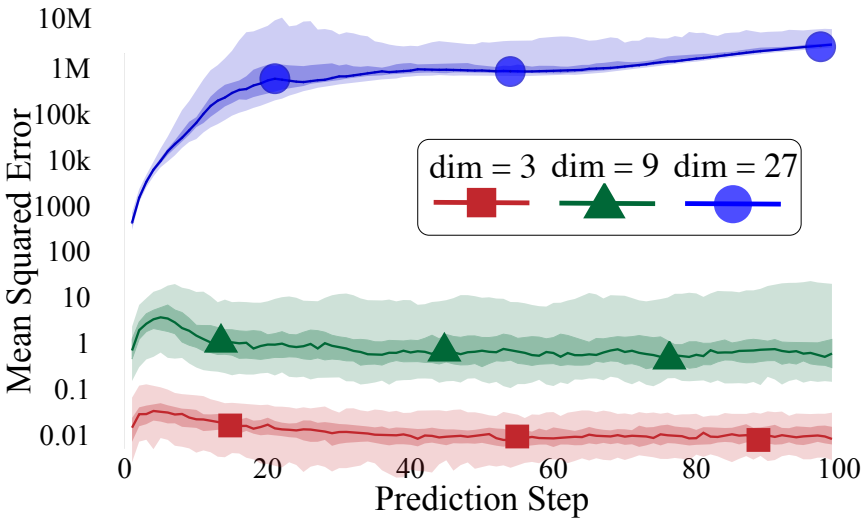


Figure 4.8: The prediction accuracy versus *unregularized* state-dimension growth given a set pole at $\rho = 0.5$ for all systems predictions. Specifically, in this figure the matrix norm, $\|\boldsymbol{A}\|_\infty$, of the state-dynamics grows with dimension.

ality of the state-space system from 3 to 9, 27, and 81, which is shown in Fig. 4.8. The effect of increasing the state without normalizing the underlying dynamics, $\boldsymbol{A}$, is a rapid increase in the compounding error because the matrix norm grows with state-dimension.

The underlying trajectories act more unstable when increasing the dimension of the state-space system because each state is a weighted sum of the current states. Without normalization the summation representing a linear transition continues to grow with state dimension. As a second experiment, the maximum row norm of the state-dynamics matrix, $\boldsymbol{A}_\infty$, is bounded to isolate the effects of prediction-dimension from the strong effects of system stability. As the dimension increases in this subsection, the dynamics are regularized so that $\|\boldsymbol{A}\|_\infty \leq 3$, as in the default system. With such normalization, the standard model types suffer from an increase in baseline prediction error, but the rate of error compounding does not grow, shown in Fig. 4.7. Increasing the dimension of the state also reduces the variance of the compounding MSE, which is likely due to averaging over more states rather than a change in prediction dynamics.

Data Distribution & Density

Understanding how model accuracy relates to an underlying training distribution is crucial to advancements in deep learning. Scaling the state-dimension of a system effectively reduces the density of data. Another axis for comparing the density of training data for a learned model is to compare model accuracy along trajectories understanding the relative density with respect to time. For stable systems, the data will likely be more dense at higher time

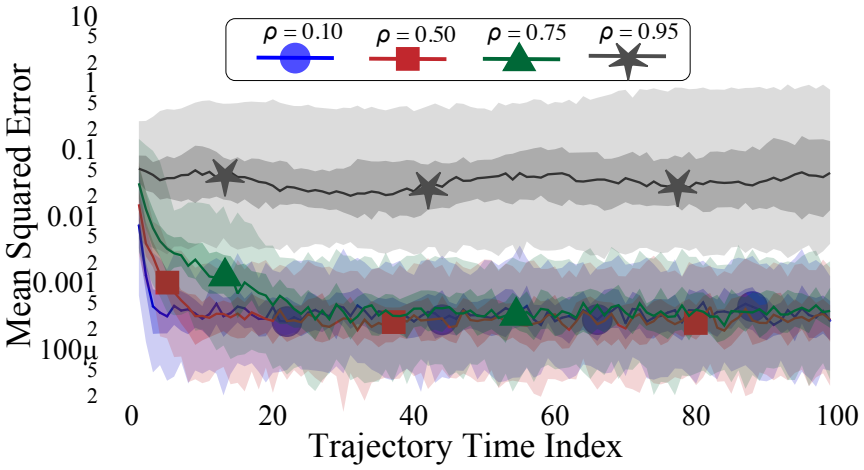
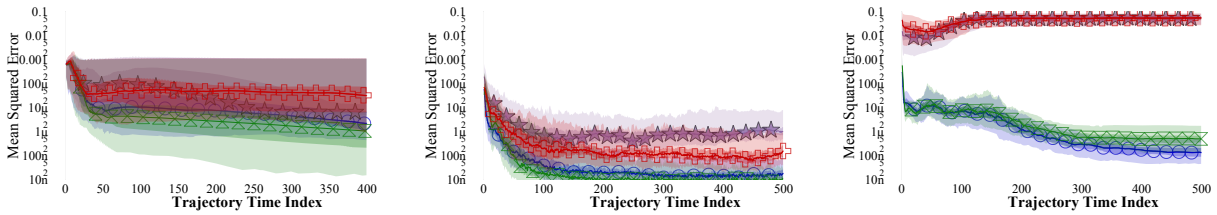


Figure 4.9: The per time index error for varying state-space systems. Again, unexpectedly, the error does not worsen further into the trajectories where state-space coverage is less dense.

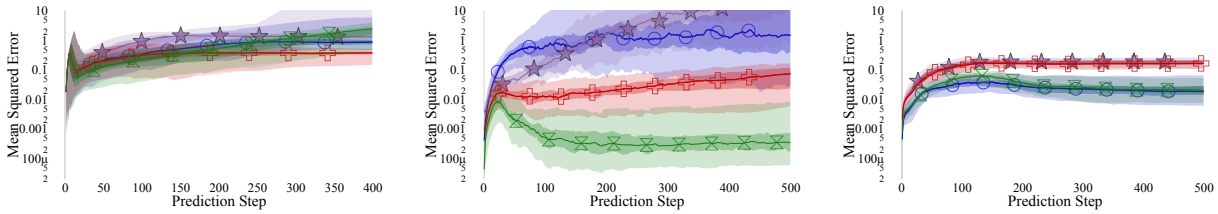


(a) Cartpole. (b) Crazyflie. (c) Reacher.

D (●) D-S (⋆) PE (⋈) PE-S (+)

Figure 4.10: The per-step model prediction error, rather than accumulated prediction error, (median, 65$^{\text{th}}$, and 95$^{\text{th}}$ percentiles) across trajectories in simulated robotic experiments. This highlights the relative error of a true state-action pair at a given time index in a trajectory. In these examples, the per-step error decreases as the controllers stabilize the robots towards the stationary target points.

indices (as is the case for the Cartpole and the Crazyflie), but for other systems the data-distribution over time in trajectories can take complex forms. As a proxy to density, we observe the per-step prediction error when predicting from the true state and action to the next state along each trajectory from the initial state $\boldsymbol{s}_0$ and at each intermediate state $\boldsymbol{s}_t$ (rather than the composed predictions as in most of this work). The results of the across-time one-step errors are shown in Fig. 4.10 for the simulated robots and Fig. 4.9 for the state-space system.



(a) Cartpole. $d_s + d_a = 5$. (b) Quadrotor. $d_s + d_a = 13$. (c) Reacher. $d_s + d_a = 20$.

D (●) D-S (★) PE (⋈) PE-S (+)

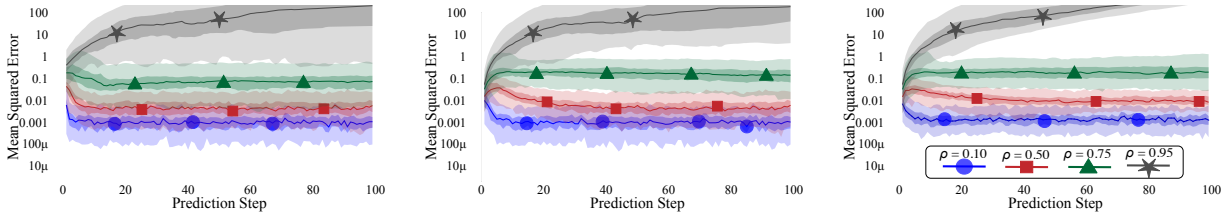
Figure 4.11: Showing how re-computing actions has a dramatically different effect on prediction MSE depending on the policy type (median, 65$^{\text{th}}$, and 95$^{\text{th}}$ percentiles). We hypothesize that these diverging predictions could be worsened when coupled with feed-forward control policies. The data used is the same as in Fig. 4.4 where the policy is computed based on the predicted state, rather than mimicking the original action. Note, the predictions for the different environments are across different horizons.

Re-computing Actions

When planning into the future there are two potential actions sequences: a logged action sequence to compare the model accuracy of a learned model to measured data and a generated action sequence to evaluate the potential usefulness of a simulated trajectory. The effect of re-computing the actions passed into the predictive model as $\boldsymbol{a}_t = \pi(\hat{\boldsymbol{s}}_t)$ instead of the original action sequence being provide by an oracle can be crucial to if a model will be useful under a certain controller. The action on a predicted state will take the form of $\boldsymbol{a} = \pi(\hat{\boldsymbol{s}}) = \pi(\boldsymbol{s} + \epsilon_t)$, so the action returned varies in most the model accuracy and policy robustness to perturbation. Depending on the problem formulation, long horizon prediction is often done with an action sequence passed into the model (representing the ground truth), but model accuracy can be dramatically different if the actions are re-computed from the predicted state as computed action sequences will also exhibit compounding error.

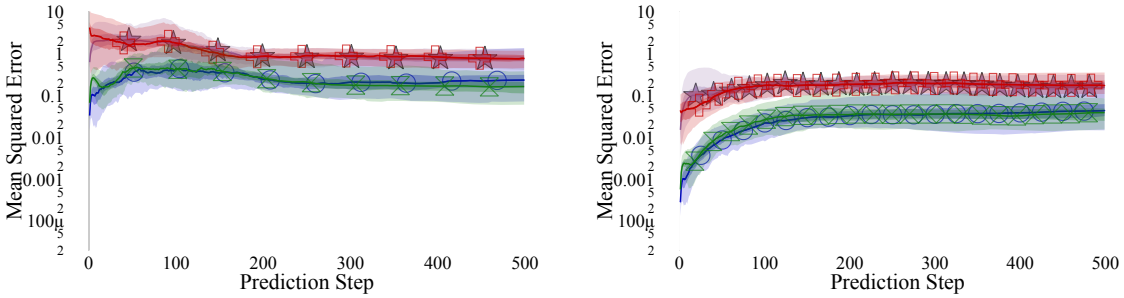
The original results of the models on the simulated robotic tasks are shown in Fig. 4.4 and the results where the oracle no longer provides the original action sequence are shown in Fig. 4.11. In this case, all environments show approximately a 10× increase in error when not given the action sequence from an oracle. Crucially, this type of planning without a real action sequence is how most MPC algorithms compute the action with a learned model. One may expect that reflexive policies recomputing actions would diverge faster because they compute the control off only the current state, while controllers with built in damping or slew limiting can predict more accurately (such as the integral or derivative terms in a PID controller), but this trend is not clear in our simulated results.

Generated action sequences are closely linked to using the models for control, but generally model training is only evaluated on accuracy with logged data. To date, there are no methods to evaluate the potential accuracy of randomly generated actions leaving future work to understand this relationship – for example, by evaluating the bootstrapped



(a) All predictions 0. (b) Linear model. (c) Default Neural Network.

Figure 4.12: Comparing the compounding error of the state-space system with simple linear and zero prediction models (shown is the MSE, 65$^{\text{th}}$, and 95$^{\text{th}}$ percentiles per pole). On the simple state-space system, the simple models perform comparably to the neural network, but this does not indicate they would be as useful for control.



(a) Without sine & cosine state processing. (b) With preprocessing.

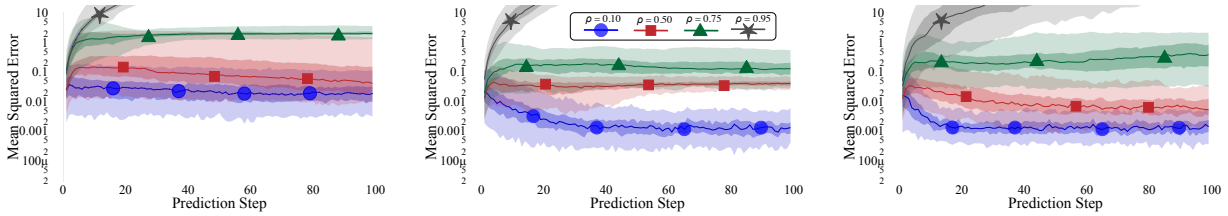
D (●) D-S (★) PE (⋈) PE-S (+)

Figure 4.13: Comparing the prediction accuracy (MSE median, 65$^{\text{th}}$, and 95$^{\text{th}}$ percentiles) on the Reacher environment with (*right*) and without (*left*) transforming the joint angles from radians θ_i to an expanded state for each joint $(\cos\theta_i, \sin\theta_i)$ to account for angle wrap-around the interval $[0, 2\pi]$. The joint angle transformation, while increasing the state dimension from 10 to 15 improves the prediction accuracy substantially on short horizons and at convergence.

uncertainty estimate of a probabilistic ensemble across a planned trajectory.

4.4 The Effects of Model Training and Parametrization

In this section we detail the effects of model training decisions on long-term prediction accuracy.



(a) 1 Training Trajectory. (b) 5 Training Trajectories. (c) 10 Training Trajectories.

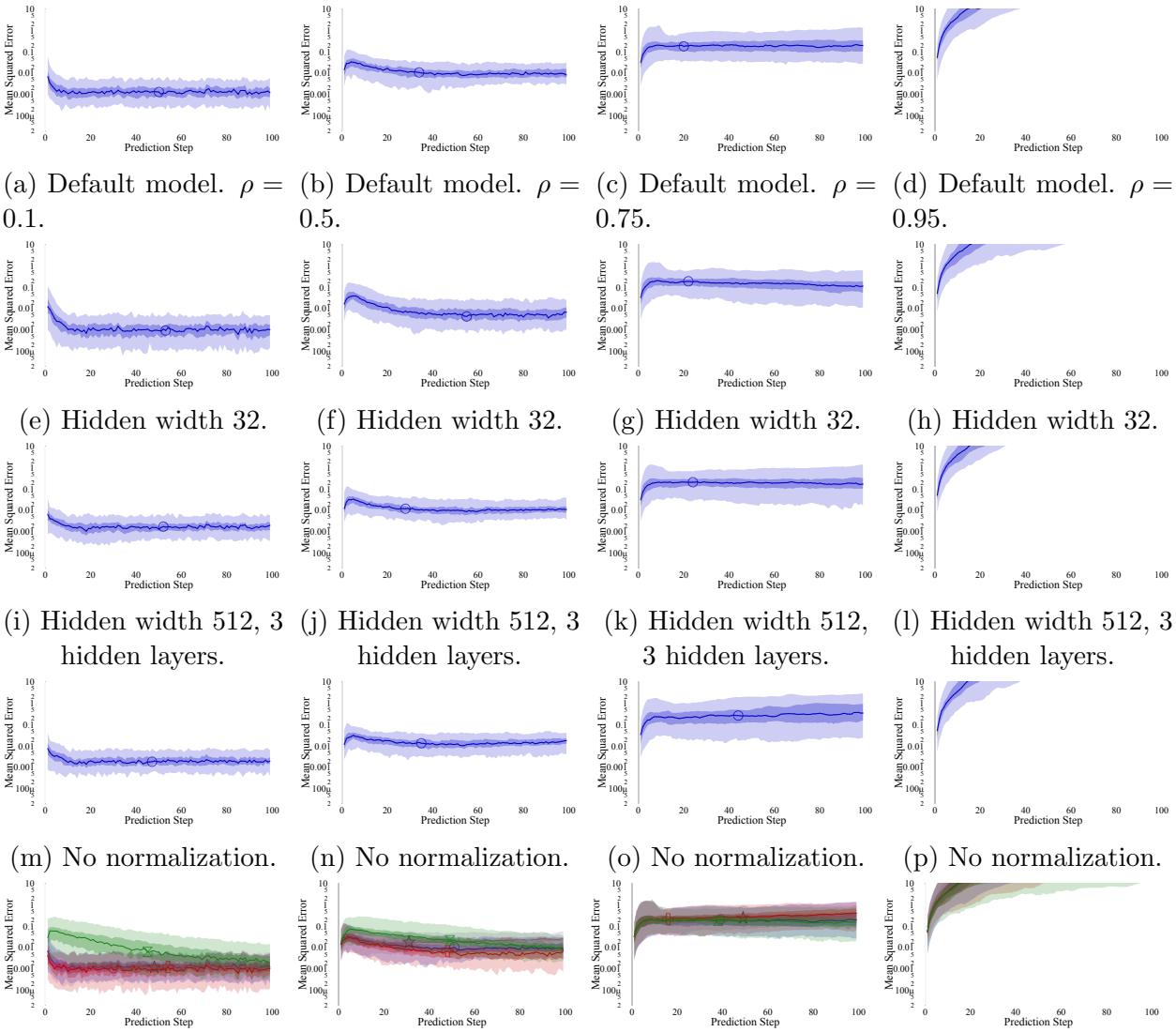
Figure 4.14: Prediction error across representative poles for different training set sizes given a constant training set of 100 trajectories for each pole. The models quickly converge with only 10 training trajectories.

Prediction Formulation & Training

Different model parametrizations, particularly ensembles and probabilistic loss functions, have been shown to improve the peak performance of MBRL algorithms [CCML18; JFZL19]. It is important to identify if these models are uniformly more accurate in predictions, or if their integration with controllers is important to the performance gains. As additional model comparisons, we include two *simple model* baselines often omitted in recent MBRL work: a linear model (LIN) and a model predicting 0 (ZERO) to provide context for the prediction errors presented. These simple models performances are highlighted in Fig. 4.12 for the state-space system and in Fig. 4.4 for the other simulated environments. These simple models are extremely strong baselines in terms of compounding error, but our work does not study their usefulness for control (e.g. the zero prediction model would be useless for control).

Another popular modelling tool is shift from a true one-step prediction to that of a delta-state formulation. For the simulated environments in Fig. 4.4, there *can* be an improvement by using the delta-state parametrization or an probabilistic ensemble, but it is not constant across all systems. Importantly, especially when deploying on real systems, is that the ranking of prediction accuracy per-model is not consistent across environment. Another implementation trick used in MBRL and other applications of model-learning for control is to map angles and other state-variables that may have discontinuities to smooth representations. For example, with angles, the state-space can be expanded to be the sine and cosine of each angle, such as done by default in the Reacher environment. The increase in state dimension for smooth state-space improves the prediction accuracy notably at short horizons ($h < 50$) and at convergence in Fig. 4.13 – confirming results presented in Sec. 4.3 that it is not a crucial factor for prediction accuracy when the underlying dynamics are constant.

We tested numerous other model types and neural network parametrizations (e.g. depth, layer size, parameter tuning, normalization, training set size, etc.) on the state-space system, but they had minimal effect on the prediction accuracy. The additional results can be found in the Appendices.



(a) Default model. $\rho = 0.1$. (b) Default model. $\rho = 0.5$. (c) Default model. $\rho = 0.75$. (d) Default model. $\rho = 0.95$.

(e) Hidden width 32. (f) Hidden width 32. (g) Hidden width 32. (h) Hidden width 32.

(i) Hidden width 512, 3 hidden layers. (j) Hidden width 512, 3 hidden layers. (k) Hidden width 512, 3 hidden layers. (l) Hidden width 512, 3 hidden layers.

(m) No normalization. (n) No normalization. (o) No normalization. (p) No normalization.

(q) More model types. (r) More model types. (s) More model types. (t) More model types.

Figure 4.15: Comparing the effects of common modeling tools on MSE (median, 65$^{\text{th}}$, and 95$^{\text{th}}$ percentiles) – changing models from top to bottom with increasing poles from right to left. Reducing the model capacity by lowering layer width from 256 to 32 units or by removing data normalization does not substantially affect prediction accuracy on the simple state-space system. The probabilistic ensemble is able to improve on prediction accuracy, though contrary to common practices, only when using the state-based predictions.

Model Capacity

Given recent advancements in deep learning driven by large datasets with evolving model architectures, one-step dynamics models operate with simpler and smaller models and datasets.

The results shown in Fig. 4.3 show the model prediction accuracy with a field-standard model capacity of 2 hidden layers and 250 neurons. To further examine the effects of model capacity, we also test the rate of divergence for deep predictive models on state-space systems with models with substantially fewer or greater parameters. The results of model predictive error with models of a hidden layer of size 32, shown in Fig. 4.15(a-d), and of a model with 3 hidden layers of width 512, shown in Fig. 4.15(e-h) show that for a simple task, changing the model size has little impact on prediction accuracy. The smaller model has slightly higher prediction error and variance among errors and the larger model has slightly improved prediction accuracy, though the effect is substantially less than the effects of system properties studied in Sec. 4.3. Additionally, the model accuracy with different training set sizes is shown in Fig. 4.14, where there is not a substantial effect beyond the first few trajectories.

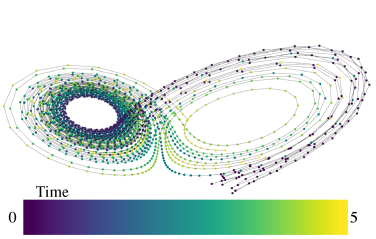
Data Normalization

Tools for designing and optimizing neural networks are designed to work on data centered around unit normal distributions – *i.e.* identical and independently distributed data closely centered around 0. In robotics data where one-step models are deployed, this is often not the case, which leaves it up the user to maintain data cleaning practices for dynamics model training.

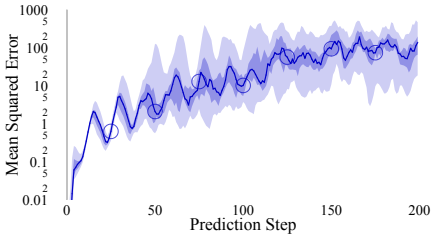
Normalization techniques map state and action variables over different ranges (bounds) and shapes (relative density) to well-behaved distributions to aid model training. The model normalizes the inputs and targets at training, and at prediction time utilizes these distributions to map new inputs to the latent space of the model and then back into the true distribution. Such mappings can also contribute to compounding error by pushing both inputs and targets of some validation data further outside of a training distribution. In this work, we map continuous variables to a normal distribution $\mathcal{N}(0, 1)$ and bounded variables (such as actions) to a uniform distribution $\mathcal{U}(-1, 1)$. The effects of turning this normalization off is a small increase in prediction error, shown in Fig. 4.15(i-l). Normalization is heavily sensitive to outliers because if some training points are substantially outside the distribution, it will further concentrate the data of interest onto a small region of the input space, resulting in a harder learning problem and one that is more sensitive to model bias.

4.5 Other Factors Impacting Compounding Error

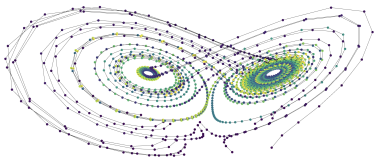
In this section, we build upon our study of system and model properties to show how some of these variables can interweave in complex manners, resulting in difficultly to forecast model performance.



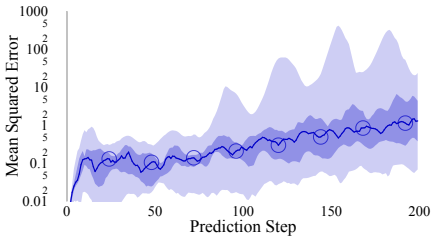
(a) Lorenz system, $x_0, y_0, z_0 \in [5, 10]$.



(b) Compounding prediction error, $x_0, y_0, z_0 \in [5, 10]$.



(c) Lorenz system, $x_0, y_0, z_0 \in [-10, 10]$.



(d) Compounding prediction error, $x_0, y_0, z_0 \in [-10, 10]$.

Figure 4.16: Highlighting the dynamics of the of the Lorenz system (*a,c*) as a challenging dynamics problem due to its chaotic dynamics that result in multi-modal behavior. The chaotic system also happens to be stable, which is reflected by its bounded prediction error per-step (*b,d*: median, 65^{th}, and 95^{th} percentiles) and a testing set of new trajectories. (*a,b*) are trajectories sampled from a more restricted initial condition, where the initial x, y, z coordinates fall in $[5, 10]$. (*c,d*) is a more diverse training and testing set, where the initial states x, y, z are sampled from $[-10, 10]$, though the dynamics model generalizes better to new previously unseen data.

Predicting Chaotic Dynamics

A fundamental limit of prediction can be posed as how to predict chaotic systems. A chaotic system is defined by the idea that a small perturbation in state can grow to an exponential difference over time. As a case study, we include prediction errors for the Lorenz system [Lor63], shown in Eq. (4.14). The canonical parameter η is often ρ, but we have replaced it to avoid overloading our symbol for pole. The equations governing the system follow,

$$\dot{x} = \sigma(y - x)\,, \tag{4.12}$$
$$\dot{y} = x(\eta - z) - y\,, \tag{4.13}$$
$$\dot{z} = xy - \beta z\,. \tag{4.14}$$

In this work, the initial states for the Lorenz system are constrained to two different distributions: $x_0, y_0, z_0 \in [5, 10]$ or $x_0, y_0, z_0 \in [-10, 10]$, resulting in a comparison between how

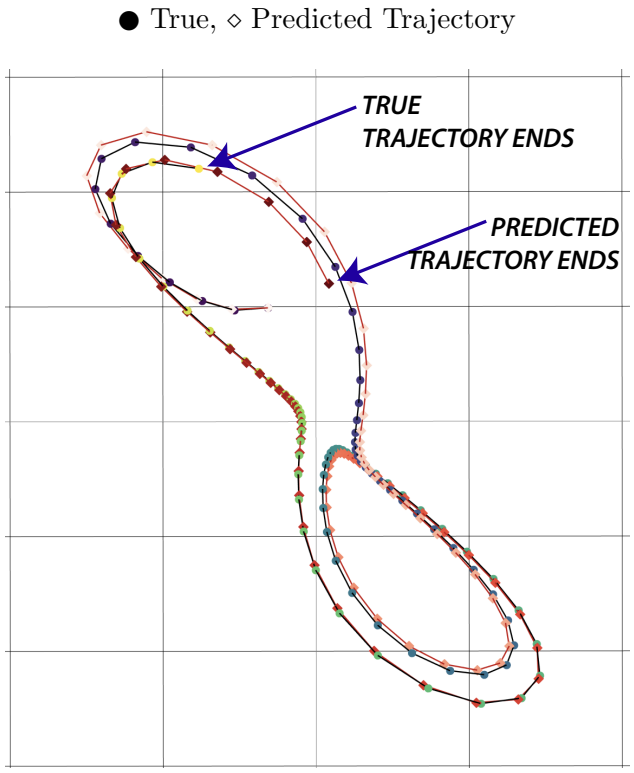
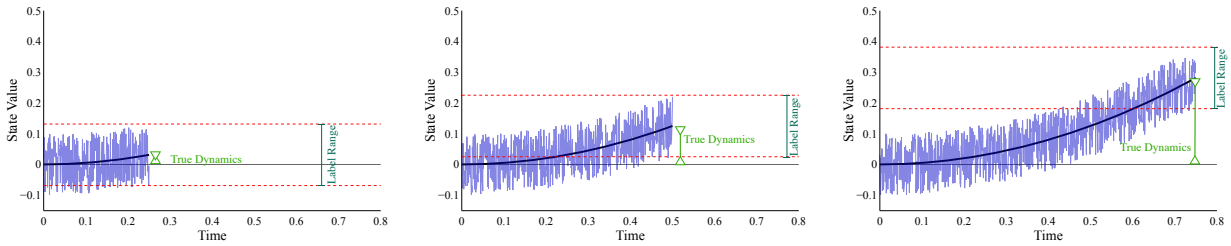


Figure 4.17: Example showing the prediction on the Lorenz system, where the predictions tend to advance in time and do not diverge rapidly.

training and test data distribution can affect dynamics model performance. Both training sets include 100 trajectories of length 500 that are evaluated on 100 previously unseen trajectories of length 200. While in practice learning to identify the three governing parameters of the dynamics could result in more accurate predictions, learning models for systems by which the analytical equations are unknown poses a problem of great interest for the field. As the simulated version of system has no noise and stable dynamics, the prediction error do not growth to infinity, but rather proportional to the separation of the two stable points, shown by the oscillations in Fig. 4.16.

Control Frequency & Signal to Noise Ratio

The signal to noise ratio (SNR) [SDDS88] is a metric for evaluating the relative strength of a signal that one wishes to measure to the noise that will be present in measurements, commonly deploying in digital signal processing. A related topic emerges with any dynamical system, where changing the sampling frequency of a system with uniform observation noise can implicitly change the SNR of the transition labels for supervised learning – a shorter sample time leads to higher impact of noise.



(a) 250 ms step size. SNR $<< 1$.

(b) 500 ms step size. SNR ≈ 1.

(c) 750 ms step size. SNR $>> 1$.

Figure 4.18: Showing how signal to noise ratio and step size are important for robotic tasks. The difference between the potential measurement regions is the magnitude of signal present in the labelled data. Crucially, a slower sampling frequency can increase the resolution of the labelled data, improving downstream prediction accuracy.

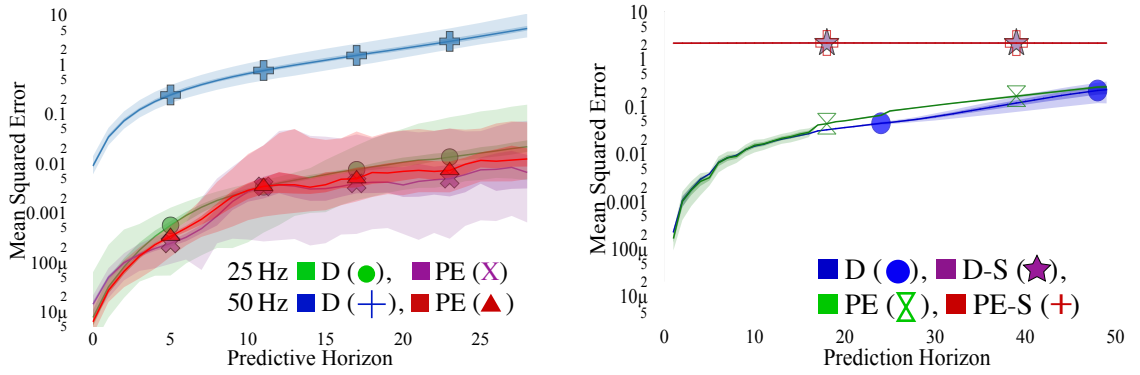
Consider a canonical control system, the double integrator, shown in Eq. (4.15), that is the underlying dynamics of Newtownian systems:

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{x}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t), \quad \boldsymbol{o}(t) = \boldsymbol{x}(t) + \omega^m(t)\,. \tag{4.15}$$

With a constant input – corresponding to a constant force – solutions to this equation take the form of a quadratic function. With a set measurement noise level the dynamics of a given system when sampled at different frequencies results in modeling problems of varying difficult, which we propose understanding as a signal to (measurement) noise ratio. The true change in state, $s_{t+1} - s_t$ is corrupted by some noise from the current and previous measurements, ω_t^m and ω_{t+1}^m, acting on the current and past observations $\boldsymbol{o}$, where the relative size of the true dynamics can be described as a signal-to-noise ratio (SNR):

$$\text{SNR} \approx \frac{\|s_t - s_{t-1}\|}{\|s_t - s_{t-1} + \omega_t^m + \omega_{t-1}^m\|}\,. \tag{4.16}$$

Crucial to accurately modelling dynamics is for the sampling rate to be slow enough by which the noise is a minor contribution to the targets, SNR $>> 1$. An illustration of this example is shown in Fig. 4.18, where a constant noise interval illustrates the possible data labels with different sample rates. All three simulators in this work do not have measurement error, though every real system's measurement error is determined by the quality of on-board or external state measurement. In the real quadrotor system that follows in Sec. 4.6, we evaluate model accuracy for two sampling rates. Finally, in real systems noise distributions often take on asymmetric and complex distributions, which can be measured and understood as specific detriments to learned model accuracy.



(a) Crazyflie Quadrotor.

(b) Unitree A1 Quadruped.

Figure 4.19: Looking at the real-world prediction divergence on a flying (a) and walking robot (b) two training sets for a Euler angles of a flying robot (median, 65^{th}, and 95^{th} percentiles).

4.6 Case Studies of Compounding Error with Real World Data

In this section we showcase the prediction accuracy of models trained on real-world dynamics data from a quadrotor and a quadruped. The quadrotor is a high-speed, high-noise system with datasets from two different control frequencies (showcasing the potential prediction challenge with low state-signal to noise ratio, which is studied further in Sec. 4.5). The quadruped shows how high-dimensional state-action spaces can reduce the prediction accuracy of true-state models.

Quadrotor: The Crazyflie is a micro-aerial vehicle that masses only 27 g, is 9 cm^2 and performs on-board sensor fusion with an MPU-9250 inertial measurement unit ($\boldsymbol{s} \in \mathbb{R}^3$, $\boldsymbol{a} \in \mathbb{R}^4$). The dataset used corresponds primarily to episodes of flights from 1 to 5 s attempting to stabilize the Euler angles of the robot (data is from [LDY+19]). Most of these sequences are unstable and end with failed control where the Euler angles diverge or the robot collided with a wall due to drift of unmeasured states. The prediction accuracy is shown for a training set in Fig. 4.19(a). Note that a prediction of the same horizon in steps translates to a longer prediction in *time* when the model is trained on data with a lower frequency. The results show that there is a clear gain in prediction accuracy with the lower frequency.

Quadruped: As another evaluation of compounding data, we have randomly created episodes of length 50 from a batch of 3800 points of state-action data from the Unitree A1 Quadruped ($a_t \in \mathbb{R}^{60}$, $s_t \in \mathbb{R}^{52}$). This data comes from non-episodic data of the quadruped walking with a trot gait. The action space is a multi-modal controller representing a unstudied problem in MBRL for control.When a leg corresponding to one of the 12 motors (3 per leg) is in contact with the ground, all actions are set to 0 except torque indicator, and vice-versa when the leg is in the air. The error shown when predicting a high-dimensional

input-output relation is shown in Fig. 4.19(b).

4.7 Understanding Compounding Error

Here, we outline a few key observations that should be considered when understanding the compounding prediction error on a new system. These should be the points of focus when model-learning for control is applied on new systems:

- **"*No Free Lunch*" Applies to Model Accuracy**: Given a fixed dataset, changing between different models will shift where error is present in the state-action space and over different predictive horizons. There will be no model that is perfect for one task, so designers should match their model to the desired controller.
- **Dynamics Dominates the Model Accuracy**: The properties of the dynamic system being modeled often has substantially greater effect on the prediction accuracy compared to model parametrization or training parameters. This point covers the accepted optimization procedures in the literature, but more complex optimizations, such as Automatic Machine Learning of dynamics models for MBRL [ZRP+21], is an exception that shifts modeling from an accuracy problem to one of maximization of reward.
- **Long-horizon Errors Can Level-off**: The results show that for many simulated applications, the error only compounds over an initial horizon h after which the error levels off or grows slowly. In most cases, this levelling happens when predictions are already useless for control purposes. However, we can postulate that might exist cases when the levelling off of error is sufficiently low that long-horizon predictions could be leveraged to solve sparse tasks.
- **Simple Models for Simple Systems**: In our low-dimensional experiments, simple linear models and deterministic neural networks provide a strong baseline that should be considered in real-world applications.
- **Low-to-Zero Noise is not a Accuracy Guarantee**: Transitioning from moderate to low to zero noise has diminishing returns on prediction accuracy, indicating that even simulated environments with no noise can still be difficult modelling tasks.

4.8 Future Work

Other Prediction Modalities Many other model types and trajectory propagation techniques exist that are well suited to a more narrow spectrum of problems than deep neural networks. Linear models are suited to linear systems [FLA16; BCXLT17b], Gaussian processes are useful for lower dimensionalities and dataset sizes [WHF06; DR11; MS17],

trajectory-based neural networks are useful with closed form control laws [LWZPC21], and new physics-based neural networks are yet to be deployed for control [KGZKM21; JHF21]. When dynamics models learn distributions instead of specific transitions, the method for propagation of the imagined trajectory can heavily impact both compounding error and downstream control. In this work, only expectation-based propagation is used for probabilistic models and probabilistic ensembles. Crucially, careful understanding of the various model types' strengths and weaknesses with respect to compounding error will yield improved performance when paired with a suitable controller.

Dynamics Modeling with Distribution Shift In model-based reinforcement learning, the data used to train the model changes with each step. This can take two forms: the amount of data in the replay buffer and the relative shape of the data distribution. There are complicated relationships in model-based reinforcement learning between model accuracy, task-performance, and data-distribution that are not studied in this chapter. Recent work suggests that optimizing solely for prediction accuracy does not result in maximum task-performance [LAYC20; ZRP+21]. These data-properties are very difficult to quantify but crucial for performance – for example, the relative density of labeled data points and the underlying difficulty of a transition to model both effect prediction accuracy and are not well understood.

Chapter 5

Objective Mismatch in Reinforcement Learning

In this chapter, we highlight a fundamental problem in the MBRL learning scheme: the *objective mismatch* issue. This work is primarily sourced from our paper describing Objective Mismatch [LAYC20]. The learning of the forward dynamics model is decoupled from the subsequent controller through the optimization of two different objective functions – prediction accuracy or loss of the single- or multi-step look-ahead prediction for the dynamics model, and task performance for the policy optimization. While the use of log-likelihood (LL) for system identification is an historically accepted objective, it results in optimizing an objective that does not necessarily correlate to controller performance. This chapter covers the following: 1) identify and formalize the problem of objective mismatch in MBRL; 2) examine the signs of and the effects of objective mismatch on simulated control tasks; 3) propose an initial mechanism to mitigate objective mismatch; 4) discuss the impact of objective mismatch and outline future directions to address this issue.

5.1 The Origin of Objective Mismatch

The Subtle Differences between MBRL and System Identification

Many ideas and concepts in model-based RL are rooted in the field of optimal control and system identification [Sut91; Ber95; ZDG+96; Kir12; Bry18]. In system identification (SI), the main idea is to use a two-step process where we first generate (optimal) elicitation trajectories $\boldsymbol{s}_{t+h} = f_\theta(\boldsymbol{s}_t, h, \theta_\pi)$ to fit a dynamics model (typically analytical), and subsequently we apply this model to a specific task. This particular scheme has several assumptions: 1) the elicitation trajectories collected cover the entire state-action space; 2) the presence of virtually infinite amount of data; 3) the global and generalizable nature of the model resulting from the SI process. With these assumptions, the theme of system identification is effectively to collect a large amount of data covering the whole state-space to create a

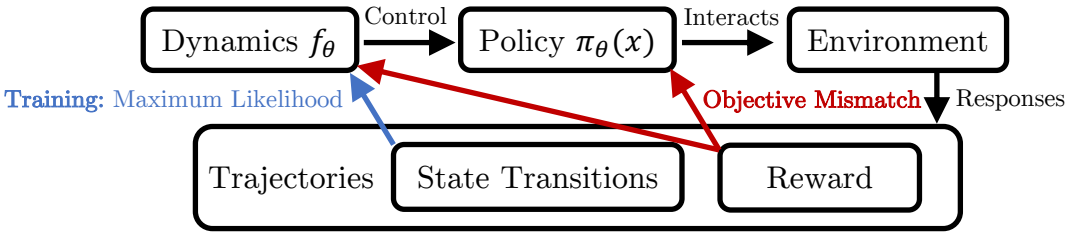
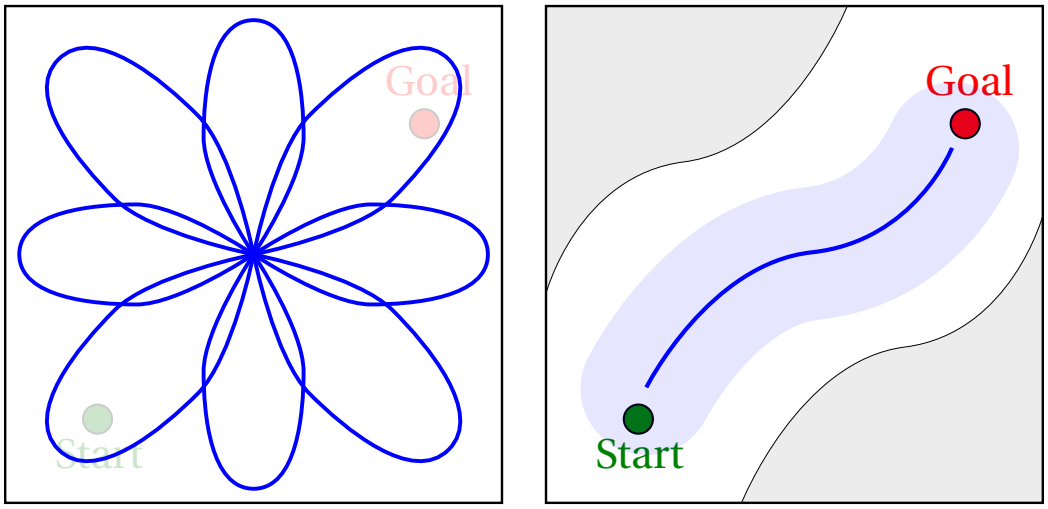


Figure 5.1: Objective mismatch in MBRL arises when a model is trained to maximize the likelihood but then used for control to maximize a reward signal not considered during training.



(a) Elicitation Trajectories. (b) On-policy Collection.

Figure 5.2: Sketches of state-action spaces. (*Left*) In system identification, the elicitation trajectories are designed off-line to cover the entire state-action space. (*Right*) In MBRL instead, the data collected during learning is often concentrated in trajectories towards the goal, with other parts of the state-action space being largely unexplored (grey area).

sufficiently accurate, global model that we can deploy on any desired task, and still obtain good performance. If these assumptions are true, using the closed-loop of MBRL should further improve performance over traditional open-loop SI [HGD96].

When adopting the idea of learning the dynamics model used in optimal control for MBRL, it is important to consider if these assumptions still hold. The assumption of virtually infinite data is visibly in tension with the explicit goal of MBRL which is to reduce the number of interactions with the environment by being "smart" about the sampling of new trajectories. In fact, in MBRL the offline data collection performed via elicitation trajectories is largely replaced by on-policy sampling to explicitly reduce the need to collect large amount

of data [CCML18]. Moreover, in the MBRL setting the data will not usually cover the entire state-action space, since they are generated by optimizing one task. In conjunction with the use of non-parametric models, this results in learned models that are strongly biased towards capturing the distribution of the locally accurate, task-specific data. Nonetheless, this is not an immediate issue since the MBRL setting rarely tests for generalization capabilities of the learned dynamics. In practice, we can now see how the assumptions and goals of system identification are in contrast with the ones of MBRL. Understanding these differences and the downstream effects on algorithmic approach is crucial to design new families of MBRL algorithms.

Definition

During the MBRL process of iteratively learning a controller, the reward signal from the environment is diluted by the training of a forward dynamics model with a independent metric, as shown in Fig. 5.1. In our experiments, we highlight that the minimization of some network training cost does not hold a strong correlation to maximization of episode reward. As dynamic environments becoming increasingly complex in dimensionality, the assumptions of collected data distributions become weaker and over-fitting to different data poses an increased risk.

Formally, the problem of objective mismatch appears as two de-coupled optimization problems repeated over many cycles of learning, shown in Eq. (5.1a,b), which could be at the cost of minimizing the final reward. This loop becomes increasingly difficult to analyze as the dataset used for model training changes with each experimental trial – a step that is needed to include new data from previously unexplored states. We characterize the problems introduced by the interaction of these two optimization problems, but, for simplicity, we do not consider the interactions added by the changes in the dynamics-data distribution during the learning process. In addition, we discuss potential solutions, but do not make claims about the best way to do so, which is left for future work.

$$\textbf{Training:} \quad \arg\max_{\theta} \sum_{i=1}^{N} \log p_\theta(s_i'|s_i, a_i), \qquad \textbf{Control:} \quad \arg\max_{a_{t:t+T}} \mathbb{E}_{\pi_\theta(s_t)} \sum_{i=t}^{t+T} r(s_i, a_i) \tag{5.1a,b}$$

Manifestations of Mismatch

Model mismatch between fitting the likelihood and optimizing the task's reward manifests itself in many ways. Here we highlight two of them and in Sec. 5.7 we discuss how related work connects in with these issues.

Long-horizon roll-outs of the model may be unstable and inaccurate. Time-series or dynamics models that are unrolled for long periods of time easily diverge from the true prediction and can easily step into predicting future states that are not on the manifold of reasonable trajectories. Taking these faulty dynamics models and using them as a smaller

part of a controller that optimizes some cost function under a poor approximation to the dynamics. Issues can especially manifest if, *e.g.*, the approximate dynamics do not properly capture stationarity properties necessary for the optimality of the true physical system being modeled.

Non-convex and non-smooth models may make the control optimization problem challenging The approximate dynamics might have bad properties that make the control optimization problem much more difficult than on the true system, even when the true optimal action sequence is optimal under the approximate model. This is especially true when using neural network as they introduce non-linearities and non-smoothness that make many classical control approaches difficult.

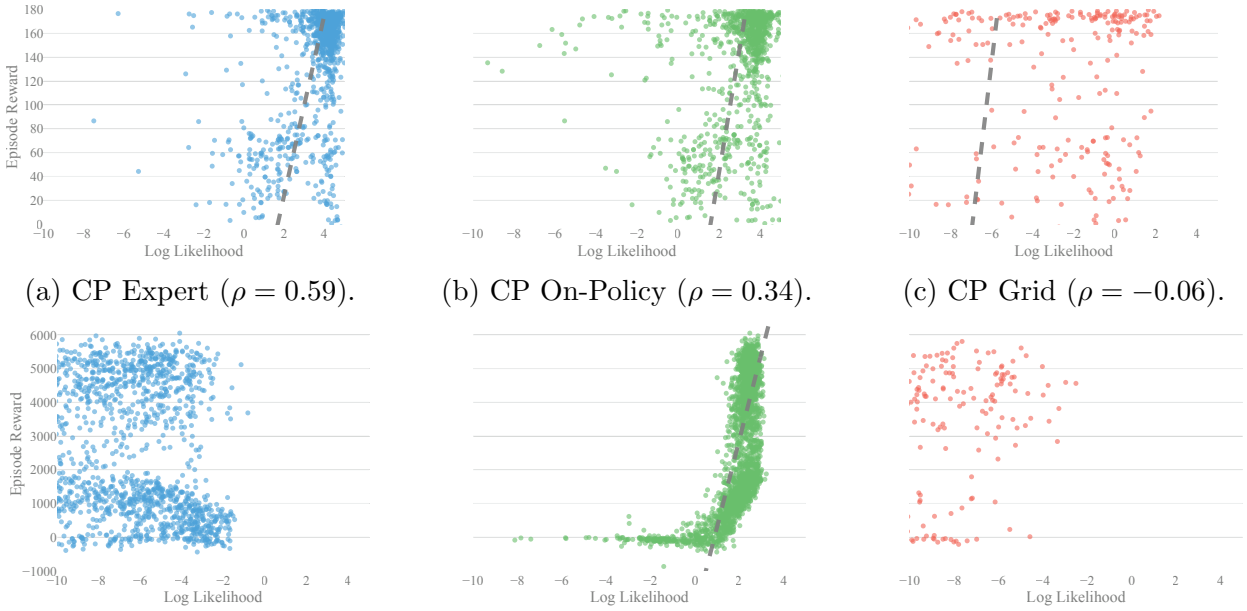
5.2 Experimental Setting

We now experimentally study the issue of objective mismatch to answer the following: 1) Does the distribution of models obtained from running a MBRL algorithm show a strong correlation between LL and reward? 2) Are there signs of sub-optimality in the dynamics models training process that could be limiting performance? 3) What model differences are reflected in reward but not in LL?

In our experiments, we use two popular RL benchmark tasks: the cartpole (CP) and half cheetah (HC). For more details on these tasks, model parameters, and control properties see [CCML18]. We use a set of 3 different datasets to evaluate how assumptions in MBRL affect performance. We start with high-reward, expert datasets (cartpole $r > 179$, half cheetah $r > 10000$) to test if on-policy performance is linked to a minimal, optimal exploration. The two other baselines are datasets collected on-policy with the PETS algorithm and datasets of sampled tuples representative of the entire state space. The experiments validate over a) many re-trained models and b) many random seeds, to account for multiple sources of stochasticity in MBRL.

5.3 Correlating Model Loss and Episode Reward

The MBRL framework assumes a clear correlation between model accuracy and policy performance, which we challenge even in simple domains. We aggregated $M_{cp} = 1000$ cartpole models and $M_{hc} = 2400$ half cheetah models trained with PETS. The relationships between model accuracy and reward on data representing the full state-space (grid or sampled) show no clear trend in Fig. 5.3c,f. The distribution of rewards versus LL shown in Fig. 5.3a-c shows substantial variance and points of disagreement overshadowing a visual correlation of increased reward and LL. This bi-model distribution on the half cheetah expert dataset, shown in Fig. 5.3d, relates to a unrecoverable failure mode in early half cheetah trials. The contrast between Fig. 5.3e and Fig. 5.3d,f shows a considerable per-dataset variation in the



(a) CP Expert ($\rho = 0.59$). (b) CP On-Policy ($\rho = 0.34$). (c) CP Grid ($\rho = -0.06$).

(d) HC Expert ($\rho = 0.07$). (e) HC On-Policy ($\rho = 0.46$). (f) HC Sampled ($\rho = 0.19$).

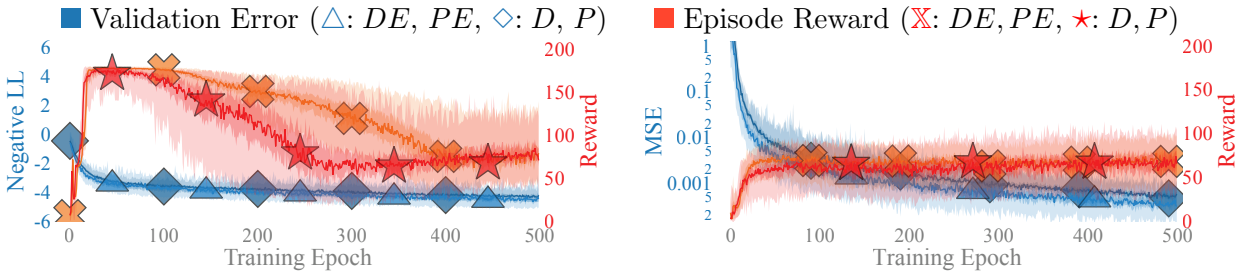
Figure 5.3: The distribution of dynamics models ($M_{models} = 1000, 2400$ for cartpole, half cheetah) from our experiments plotting in the LL-Reward space on three datasets, with correlation coefficients ρ. Each reward point is the mean over 10 trials. There is a trend of high reward to increased LL that breaks down as the datasets contain more of the state-space than only expert trajectories.

state-action transitions. The grid and sampled datasets, Fig. 5.3c,f, suffer from decreased likelihood because they do not overlap greatly with on-policy data from PETS.

If the assumptions behind MBRL were fully valid, the plots should show a perfect correlation between LL and reward. Instead these results confirm that there exists an objective mismatch which manifests as a decreased correlation between validation loss and episode reward. Hence, there is no guarantee that increasing the model accuracy (i.e., the LL) will also improve the control performance.

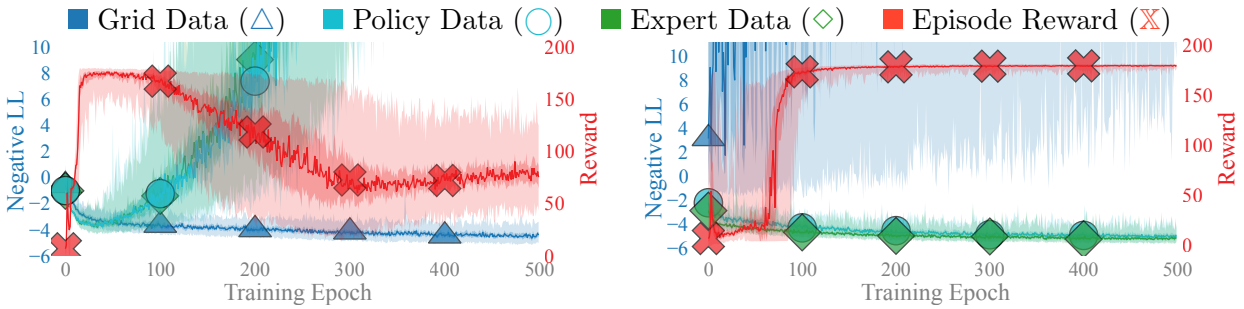
5.4 Examining Model Loss vs Episode Reward Per Training Epoch

This section explores how model training impacts performance at the per-epoch level. These experiments shed light onto the impact of the strong model assumptions outlined in Sec. 5.1. As a dynamics model is trained, there are two key inflection points - the first is the training epoch where episode reward is maximized, and the second is when error on the validation set



(a) P, PE models. (b) D, DE models.

Figure 5.4: The reward when re-evaluating the controller at each dynamics model training epoch for different dynamics models, $M = 50$ per model type. Even for the simple cartpole environment, D, DE fail to achieve full performance, while P, PE reach higher performance but eventually over-fit to available data. The validation loss is still improving slowly at 500 epochs, not yet over-fitting.



(a) **Trained:** grid **Tested:** expert, on-policy. (b) **Trained:** on-policy **Tested:** expert, grid.

Figure 5.5: The effect of the dataset choice on model (P) training and accuracy in different regions of the state-space, $N = 50$ per model type. (*Left*) when training on the complete dataset, the model begins over-fitting to the on-policy data even before the performance drops in the controller. (*Right*) A model trained only on policy data does not accurately model the entire state-space. The validation loss is still improving slowly at 500 epochs in both scenarios.

is optimized. These experiments highlight the disconnect between three practices in MBRL a) the assumption that the on-policy dynamics data can express large portions of the state-space, b) the idea that simple neural networks can satisfactorily capture complex dynamics, c) and the practice that model training is a simple optimization problem disconnected from reward. Note that in the figures of this section we use Negative Log-Likelihood (NLL) instead of LL, to reduce visual clutter.

For the grid cartpole dataset, Fig. 5.4 shows that the reward is maximized at a drastically different time than when validation loss is minimized for P, PE models. Fig. 5.5 highlights

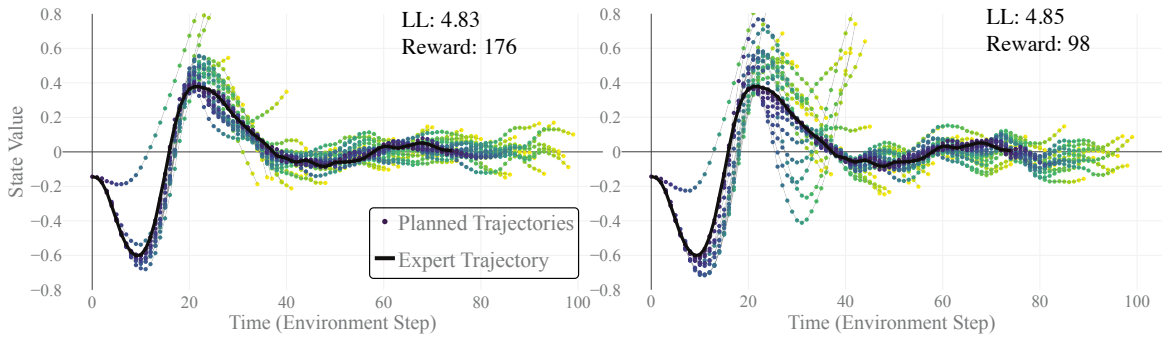


Figure 5.6: Example of planned trajectories along the expert trajectory for (*left*) a learned model and (*right*) the adversarially generated model trained to lower the reward. The planned control sequences are qualitatively similar except for the peak at $t = 25$. There, the adversarial attack applies a small nudge to the dynamics model parameters that significantly influences the control outcome with minimal change in terms of LL.

how the trained models are able to represent other datasets than they are trained on (with additional validation errors). Fig. 5.5b shows that on-policy data will not lead to a complete dynamics understanding because the grid validation data rapidly diverges. When training on grid data, the fact that the on-policy data diverges in Fig. 5.5a before the reward decreases is encouraging as objective mismatch may be preventable in simple tasks. Similar experiments on half cheetah are omitted because models for this environment are trained incrementally on aggregated data rather then fully on each dataset [CCML18].

5.5 Decoupling Model Loss from Controller Performance

We now study how differences in dynamics models – *even if they have similar LLs* – are reflected in control policies to show that a accurate dynamics model does not guarantee performance.

Adversarial attack on model performance

We performed an adversarial attack [SZS+13] on a deep dynamics model so that it attains a high likelihood but low reward. Specifically, we fine-tune the deep dynamics model's last layer with a zeroth-order optimizer, CMA-ES, (the cumulative reward is non-differentiable) to lower reward with a large penalty if the validation likelihood drops. As a starting point for this experiment we sampled a P dynamics model from the last trial of a PETS run on cartpole. This model achieves reward of 176 and has a LL of 4.827 on it's on-policy validation dataset. Using CMA-ES, we reduced the on-policy reward of the model to 98,

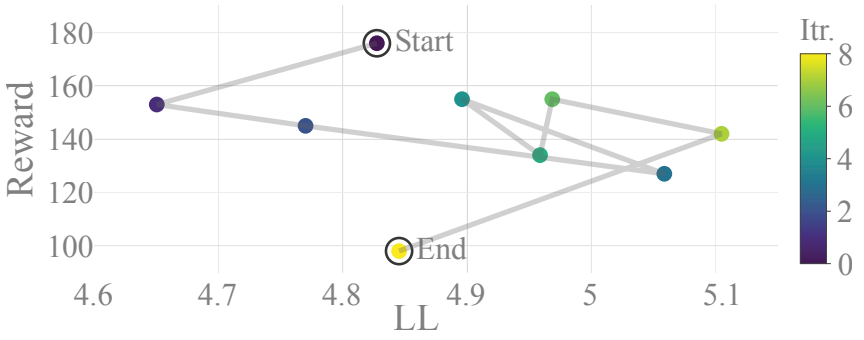


Figure 5.7: Convergence of the CMA-ES population's best member.

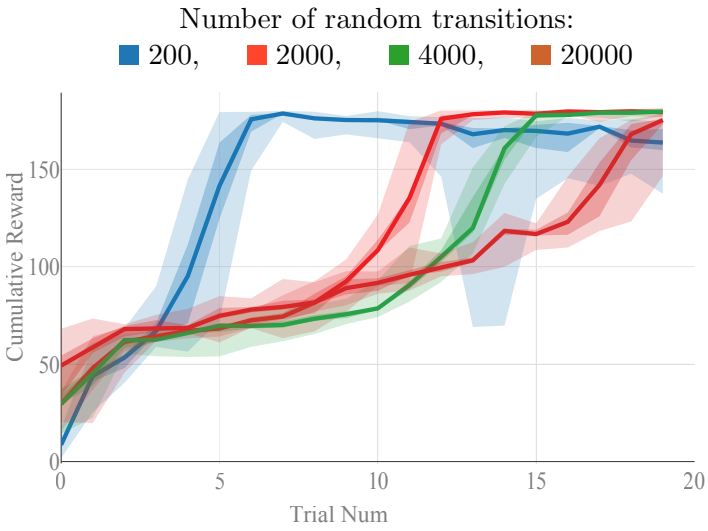


Figure 5.8: Cartpole (Mujoco simulations) learning efficiency is suppressed when additional data not relevant to the task is added to the dynamics model training set. This effect is related to the issue of objective mismatch because model training needs to account for potential off-task data.

on 5 trials, while slightly improving the LL; the CMA-ES convergence is shown in Fig. 5.7 and the difference between the two models is visualized in Fig. 5.6. Fine tuning of all model parameters would be more likely to find sub-optimal performing controllers because the output layer consists of about 1% of the total parameters. This experiment shows that the model parameters that achieve a low model loss inhabit a broader space than the subset that also achieves high reward.

Effect of Dataset Distribution when Learning

Learning speed can be slowed by many factors in dataset distribution, such as adding additional irrelevant transitions. When extra transitions from a specific area of the state space are included in the training set, the dynamics model will spend increased expression on these transitions. LL of the model will be biased down as it learns this data, but it will reduce the learning speed as new, more relevant transitions are added to the training set.

Running cartpole random data collection with a short horizon of 10 steps (while forcing initial babbling state to always be 0), for 20, 200,400 and 2000 babbling roll-outs (that sums up to 200, 2000, 4000 and 20000 transitions in the dataset finally shows some regression in the learning speed for runs with more useless data in the motor babbling. This data highlights the importance of careful exploration vs exploitation trade-offs, or changing how models are trained to be selective with data.

Finding the Optimal Data for Training a Model

The previous experiment showed that having too much data can slow MBRL performance, which leads to the natural question of how to mitigate it? In the context of micro-data RL for novel robotics, we proposed using clustering to better balance model capacity over the dynamics of interest [LTL+20]. By clustering on the training data for a learned model, we can actually improve prediction accuracy on a held-out validation set. Fig. 5.9 shows an improvement in validation set accuracy when training on a k-means clustered, uniformly representative subset of the training data on a real-world quadrotor dataset. The question of optimal data in reinforcement learning has been studying in the context of offline RL, where static inference procedures are used to generate policies [LWW+22; YBL+22]. Translating these ideas to model training will improve the data-efficiency of model-based RL algorithms.

5.6 Mitigating Mismatch During Training

Tweaking dynamics model training can partially mitigate the problem of objective mismatch. Taking inspiration from imitation learning, we propose that the learning capacity of the model would be most useful when accurately modeling the dynamics along trajectories that are relevant for the task at hand, while maintaining knowledge of nearby transitions for robustness under a stochastic controller. Intuitively, it is more important to model accurately the dynamics along the optimal trajectory, rather than modeling part of the state-action space that might never be visited to solve the task. For this reason, we now propose a model loss aimed at alleviating this issue.

Given a element of a state space (s_i, a_i), we quantify the distance of any two tuples, $d_{i,j}$. With this distance, we re-weight the loss, $l(y)$, of points further from the optimal policy to be lower, so that points in the optimal trajectory get a weight $\omega(y) = 1$, and points at the edge of the grid dataset used in Sec. 5.3 get a weight $\omega(y) = 0$. Using the expert

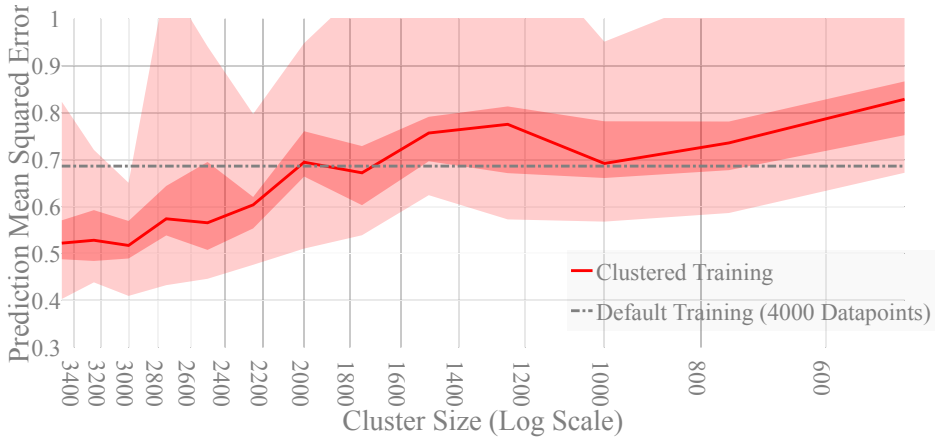


Figure 5.9: Removing redundant datapoints from quadrotor dynamics data can improve validation set accuracy while reducing stored data by over 50%. We trained 25 models at each dataset size, and evaluated them on the same validation set. From left to right is increasing the filtering (smaller training sets), which improves the accuracy on an 800 point validation set by up to 25%.

dataset discussed in Sec. 5.3 as a distance baseline, we generated $25e6$ tuples of (s, a, s') by uniformly sampling across the state-action space of cartpole. We sorted this data by taking the minimum orthogonal distance, d^*, from each of the points to the 200 elements in the expert trajectory. To create different datasets that range from near-optimal to near-global, we vary the distance bound ϵ, and number of training points, S. This simple form of re-weighting the neural network loss , shown in Eq. (5.2a,b,c), demonstrated an improvement in sample efficiency to learn the cartpole task, as seen in Fig. 5.10. Unfortunately, this approach is impractical when the optimal trajectory is not known in advance. However, future work could develop an iterative method to jointly estimate and re-weight samples in an online training method to address objective mismatch.

$$\textbf{Weighting} \;\; \omega(y) = ce^{-d^*(y)} \qquad \textbf{Standard} \;\; l(\hat{y}, y) \qquad \textbf{Re-weight} \;\; l(\hat{y}, y) \cdot \omega(y) \tag{5.2a,b,c}$$

Using simple reward as re-weight

An alternative to re-weighting w.r.t. the optimal trajectory could be re-weighting w.r.t. the reward of each state. The compelling advantage of this would be the easy availability of the reward without access to additional information (e.g., the optimal trajectory). However, the reward does not topologically have the desired shape compared to the optimal trajectory. In fact, for many rewards (e.g., distance to the target) the isocurves of reward are orthogonal to the optimal trajectory. This means that the resulting re-weighting would concentrate the dynamics to model accurately the part of the state-action space closer to the target, but

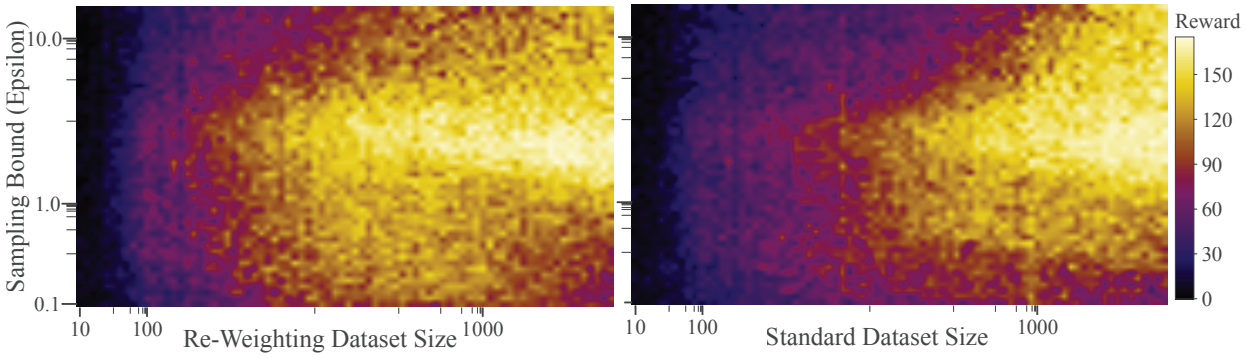


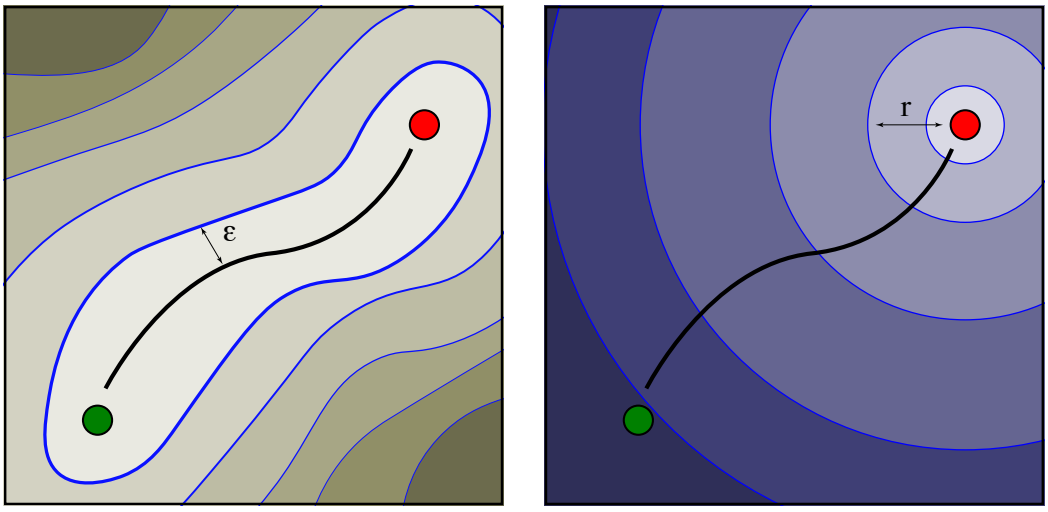
Figure 5.10: Mean reward of PETS trials ($N_{trials} = 100$), with (*left*) and without (*right*) model re-weighting, on a log-grid of dynamics model training sets with number of points $S \in [10, 2500]$ and sampling optimal-distance bounds $\epsilon \in [.28, 15.66]$. The re-weighting improves performance for smaller dataset sizes, but suffers from increased variance in larger set sizes. The performance of PETS declines when the dynamics model is trained on points too near to the optimal trajectory because the model lacks robustness when running online with the stochastic MPC.

it would ignore the dynamics that lead us to the reward in the first space (e.g., along the optimal trajectory). Intuitively, this is undesirable, as it might decrease performance in the initial stages of the trajectory. More research will be necessary to fully study alternatives forms of re-weighting.

5.7 Discussion

Objective mismatch impacts the performance of MBRL – our experiments have gone deeper into this fragility. Beyond the re-weighting of the LL presented in Sec. 5.6, here we summarize and discuss other relevant works in the community.

Learning the dynamics model to optimize the task performance Most relevant are research directions on controllers that directly connect the reward signal back to the controller. In theory, this exactly solves the model mismatch problem, but in practice the current approaches have proven difficult to scale to complex systems. One way to do this is by designing systems that are fully differentiable and backpropagating the task reward through the dynamics. This has been investigated with differentiable MPC [AJSBK18], differentiable CEM [AY20], differentiable simulators [LSWP21], and Path Integral Control [ORA17], Universal Planning Networks [SJALF18] propose a differentiable planner that unrolls gradient descent steps over the action space of a planning network. [BCXLT17a] use a zero-order optimizer to maximize the controller's performance without having to compute gradients explicitly.



(a) Expert Distance Weight. (b) Reward Weight.

Figure 5.11: We propose to re-weight the loss of the dynamics model w.r.t. the distance ϵ from the optimal trajectory. Re-weighting of the loss of the dynamics model w.r.t. the reward, shown *right*, does not have the desired topology for optimal behavior as an accurate value function would.

Add heuristics to the dynamics model structure or training process to make control easier If it is infeasible or intractable to shape the dynamics of a controller, an alternative is to add heuristics to the training process of the dynamics model. These heuristics can manifest in the form of learning a latent space that is locally linear, *e.g.*, in Embed to Control and related methods [WSBR15], by enforcing that the model makes long-horizon predictions [KST+19], ignoring uncontrollable parts of the state space [GGL18], detecting and correcting when a predictive model steps off the manifold of reasonable states [Tal17], adding reward signal prediction on top of the latent space [GKBNB19], or adding noise when training transitions [MLJ+19]. [FBN17; Far18; VLGF22] also attempts to re-frame the transitions to incorporate a notion of the downstream decision or reward. Finally, [SRSSP21] proposes stabilizability constraints to regularize the model and improve the control performance. None of these papers formalize or explore the underlying mismatch issue in detail.

Continuing Experiments Our experiments represent an initial exploration into the challenges of objective mismatch in MBRL. Sec. 5.4 is limited to cartpole due to computational challenges of training with large dynamics datasets and Sec. 5.5 could be strengthened by defining quantitative comparisons in controller performance. Additionally, these effects should be quantified in other MBRL algorithms such as MBPO [JFZL19] and POPLIN [WB19].

Chapter 6

Trajectory-based Dynamics Model

In this chapter, we propose and study a new class of feed-forward dynamics models focused on capturing not the behavior of single steps, but the long-term time dependant evolution of a trajectory as a whole. This work is derived from our paper introducing the trajectory-based model [LWZPC21]. The main intuitions behind this model are that: 1) sequences of actions and states are usually strongly correlated with their neighbors across time (i.e., across a trajectory) and 2) in quantifying the quality of long-horizon predictions, it might be preferable to have higher uncertainty over the single steps, if the predication of the trajectory as a whole is more accurate. This is particularly crucial when the dynamics models are used for planning, where the relative ranking of the trajectories will directly impact the decision making (e.g., it might be relatively unimportant to know how we reach a location as long as we know that we can reach it accurately).

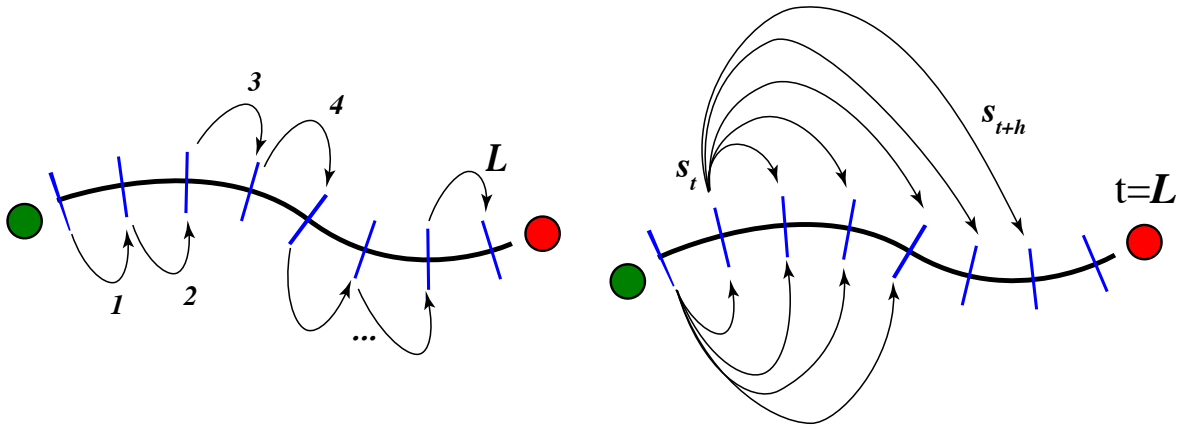
6.1 A New Prediction Formulation

We now describe our new trajectory-based dynamics models (which we refer in the rest of the manuscript as T) which focus on modeling trajectories over time rather than individual steps. The are two main intuitions behind the adoption of this type of model: 1) for control purposes it is often more valuable to have an accurate overall trajectory prediction compared to accurately predict single steps (which might compound error over long-term). This is even more important when planning, since for planning the relative ranking of the trajectories is what determines the eventual actions applied by control scheme such as MPC. 2) trajectories are often strongly correlated in space and time; however, single-step models do not have efficient mechanism to enforce that.

To address the error compounding, an idea would be to replace the recursive call of Eq. (4.1) with a n^{th} step prediction

$$\boldsymbol{s}_{t+h} = f_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{a}_{t+1} \ldots, \boldsymbol{a}_{t+h}) \,, \tag{6.1}$$

which does not requires recursion, and is thus more likely to produce stable long-term predictions. However, here we can observe how the dimensionality of the model to be learned



(a) One-step models:
$\boldsymbol{s}_{t+1} = f_\theta(\boldsymbol{s}_t, a_t)$

(b) Trajectory-based models:
$\boldsymbol{s}_{t+h} = f_\theta(\boldsymbol{s}_t, h, \theta_\pi)$

Figure 6.1: The model formulation for one-step models (a) and our new **trajectory-based models** (b).

depends on the length of the prediction into the future h, and that in addition, this model is generally only capable of predicting the resulting n^{th} step ahead prediction, but not its intermediate steps (this is not true for RNNs, but their formulation is more similar to Eq. (4.1)). A first variant of this n^{th} step ahead formulation would be to observe that the sequence of action $\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \ldots \boldsymbol{a}_{t+h}$ is typically generated by a generic, but single controller $\pi(\cdot)$ determined by parameters θ_π, and thus we can rewrite as

$$\boldsymbol{s}_{t+h} = f_\theta(\boldsymbol{s}_t, \theta_\pi)\,. \tag{6.2}$$

As long as the dimensionality of θ_π is smaller than the dimensionality of $\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \ldots \boldsymbol{a}_{t+h}$, this would result in an effective reduction of the dimensionality of our dynamics model and thus improved data-efficiency. However, once again this model only allows us to predict the final state but not the trajectory that led us there. Adding the notion of dynamic time-prediction is the final conceptual change to attempt to accurately predict long-term system dynamics in a data-efficient manner, by which we index the starting state at time t and directly predict to the future, variable horizon of h steps with one forward pass. The trajectory-based models predict the evolution from a starting state $\boldsymbol{s}_t$, subject to the control parameters θ_π, to a future-time index $t + h$, as

$$s_{t+h} = f_\theta(\boldsymbol{s}_t, h, \theta_\pi)\,. \tag{6.3}$$

Compared to the traditional recursive one-step ahead formulation of Eq. (6.1), this formulation provides several benefits that we now detail.

6.2 Benefits of Trajectory-based Models

Data-efficiency

One advantage of this formulation is that we can perform a re-labeling trick over the dataset of collected trajectories, to significantly augment the dataset used to train the trajectory-based model. We assume a dataset $\mathcal{D}$ of n collected trajectories $\mathcal{D} = \{\tau^n\}$, each of fixed length L. For each collected trajectory $\tau^j = [s_0, \ldots, s_L]$ we can now extract $L-1$ subtrajectories $\tau_i^j = [s_i, \ldots, s_L]$ for $i = 0 \ldots L-1$, and use them all for training the trajectory-based model. By training on all sub-trajectories, the model gains two strengths: 1) it can predict into the future from any state, not just those given as initial states from a environment, and 2) the number of training points grows proportional to the square of trajectory length, as

$$N_{\text{train}} = n \sum_{t=1}^{L} t = n \frac{(L)(L-1)}{2} \approx nL^2 \,. \tag{6.4}$$

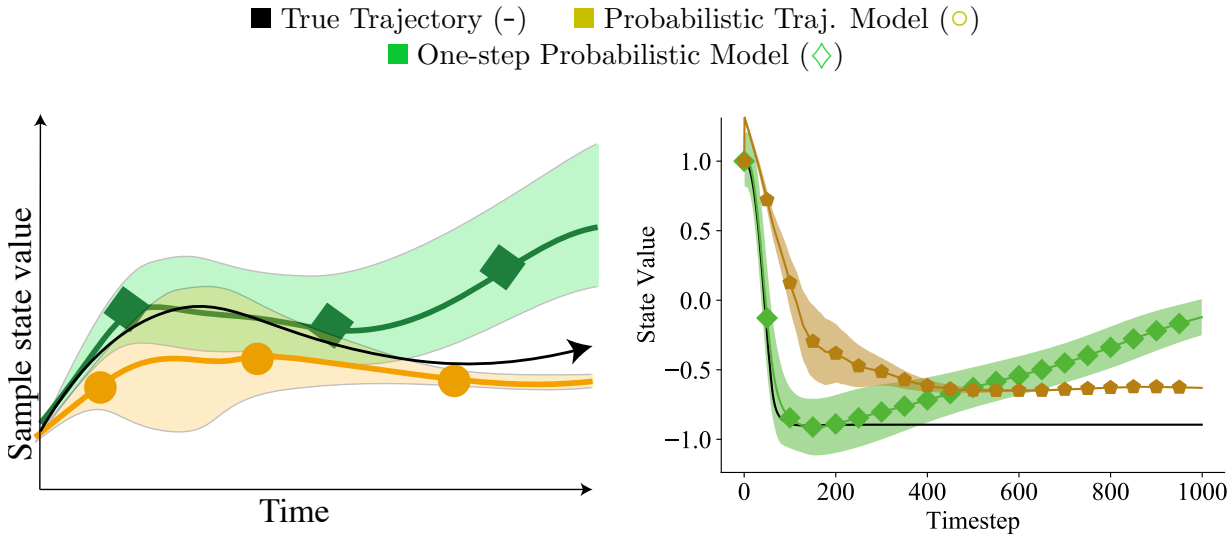
This results in models that better exploit the temporal structure of the systems, while using less data.

Computationally Efficient Planning

The trajectory-based models have a useful property of directly predicting entire trajectories instead of imagined roll-outs composed of repeated model evaluations. For prediction propagation, by only passing in a vector of time horizons $\boldsymbol{h}$ from a current state, a planner can evaluate the future with one forward pass, alleviating the computational burden (as well as the compounding, multiplicative error) associated with evaluating sequentially many steps of one-step models. In our model the predictions in a trajectory do not depend on the prediction at the previous step, which can dramatically increase the control frequency when planning online.

Capturing Empirical Distribution over Trajectories

One-step models commonly suffer from the issue of uncertainty explosion, where the predicted uncertainty over a trajectory typically keep increasing, and does not match the empirical uncertainty from the data. By propagating time directly, our probabilistic trajectory-based model can instead capture the uncertainty of variation in dynamics in the training set (*i.e.*, the model is more uncertain in areas of rapid movement and can become confident when motion converges), and the empirical uncertainty over trajectories. The uncertainty propagation is drawn in Fig. 6.2a and an example experiment is shown in Fig. 6.2b; both are compared to one-step models that have diverging uncertainty as the predicted states leave the training distribution. Stable uncertainty estimates convey promise when planning on robotic hardware, where action choices are balanced against model uncertainty due to high cost-per-test.



(a) Uncertainty sketch.

(b) Experimental uncertainty.

Figure 6.2: The trajectory-based models have prediction uncertainty proportional to the epistemic uncertainty in the training dataset. (*Left*) a sketch of the uncertainty mechanism. Trajectory-based models have uncertainty that can shrink when more confident in the dynamics, which is opposed to one-step models that have predicted uncertainties that diverge at long horizons. (*Right*) an example trial of a robotic prediction (from the reacher task in Sec. 6.3) highlights this uncertainty propagation with a probabilistic trajectory-based model and an one-step probabilistic model (P).

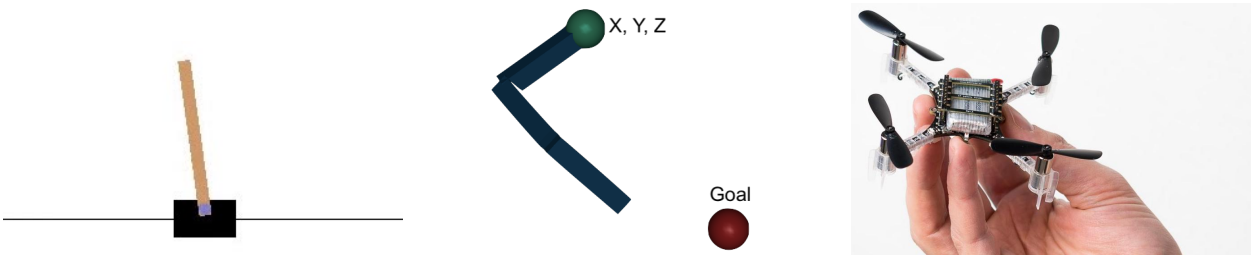
Continuous Time

Traditional one-step ahead models assume a discrete quantization of time such that the sampling frequency is constant. Instead, our model is agnostic to the use of discrete or continuous time, since the model can make use of data collected at arbitrary h and explicitly interpolate between them. While this property is not employed in the following experiments, this is a very desirable property that we aim to explore in future work.

6.3 Experimental Setting

We now evaluate the proposed trajectory-based models. In particular, we investigate the long term prediction accuracy, the ability for the trajectory-based model to predict unstable or periodic data, the sample efficiency benefit of the new parameterization, and using the new model for predicting experimental reward.

Model Training Using the same notation and model training formulations in [CCML18], we use four model types: D, P, DE, PE. The deterministic model, D, and deterministic



(a) Cartpole. (b) Reacher. (c) Quadrotor.

Figure 6.3: Experimental platforms used for studying the Trajectory-based model.

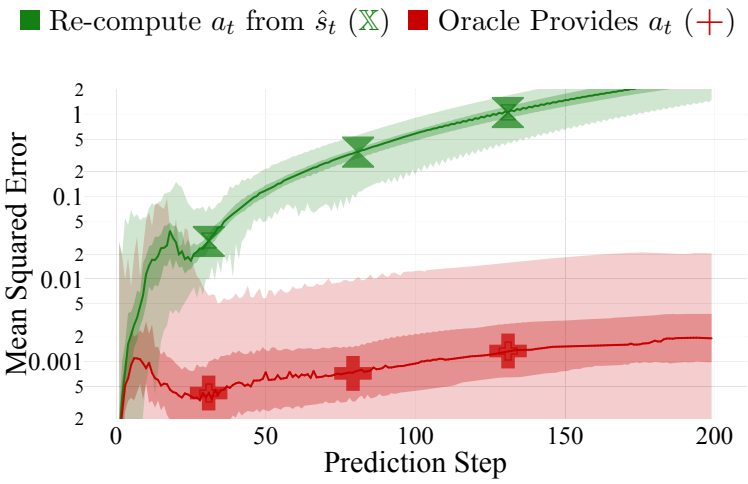


Figure 6.4: The prediction accuracy of a PE on Cartpole with and without re-computing the action at each step ($N_{\text{traj}} = 100$) shows the increased error from re-computing actions.

ensemble, DE, minimize the mean squared error (MSE) of predictions. The probabilistic model, P, and probabilistic ensemble, PE, minimize the negative log likelihood (NLL) of a Gaussian distribution of state transitions.

All models normalize the input states and actions to a standard normal distribution $\mathcal{N}(0, 1)$ in each dimension, and bounded control parameters are mapped to $[-1, 1]$. The feedforward models have two hidden layers of width 250, are optimized with Adam [KB14], with batch sizes of 32 for D,P and 64 for T, and learning rates of 2.5×10^{-5} for P models, 5×10^{-5} for D and 8×10^{-4} for T. Due to the rapid accruing of labeled data for the trajectory-based models, we cap the training set size at 1×10^5 by random downsampling. The LSTMs are trained with Adam with a learning rate of 0.1, with batches of sequences matching the trajectory length, L, and with normalization following [SVL14].

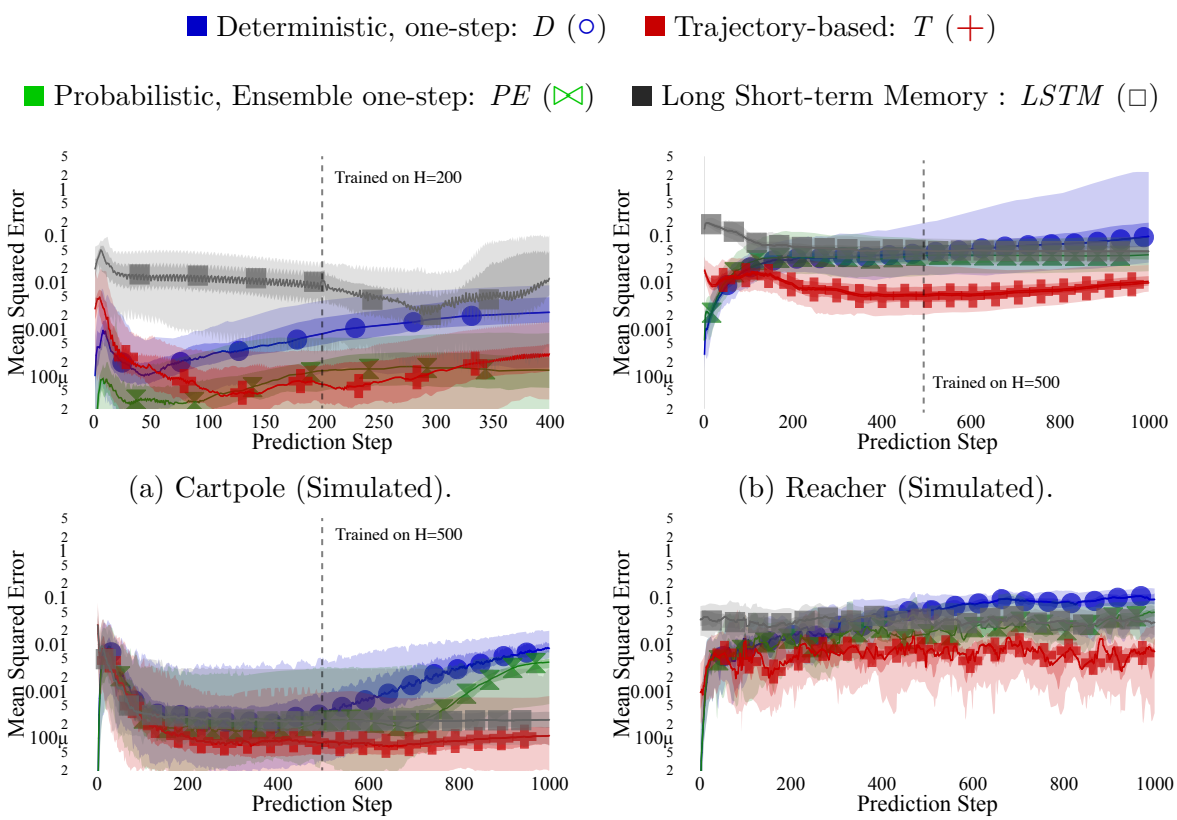
Cartpole (Simulated) We evaluate our models thoroughly on the cartpole task where the goal is to balance a mass over a sliding cart ($d_s = 4$, $d_a = 1$), shown in Fig. 6.3a. To have a continuous reward for prediction, we introduce a new reward function $r(s_t, a_t) = -(x^2 + \theta^2)$ and the remaining details are outlined in [CCML18]. We evaluate predictions of state and reward of cartpole agents conditioned on a Linear Quadratic Regulator (LQR) control policy. LQR solves the optimization $\min_u J(u) = \int_0^\infty \boldsymbol{s}^\top Q\boldsymbol{s} + \boldsymbol{a}^\top R\boldsymbol{a}\ \mathrm{d}t$ for the dynamics $\dot{\boldsymbol{s}} = \widetilde{A}\boldsymbol{s} + \widetilde{B}\boldsymbol{u}$. LQR control minimizes the expected cost based on a linearized dynamical system, $\tilde{A}, \tilde{B}$. The control policy, $\pi(\cdot)$, of LQR takes the form of state feedback, $a = -\boldsymbol{K}\boldsymbol{s}$, where $\boldsymbol{K} \in \mathbb{R}^{d_a \times d_s}$. For all experiments we use the following cost matrices to generate a optimal controller, $\boldsymbol{K}^*$: $Q = \mathrm{diag}(.5,\ .05,\ 1,\ .05)$, $\quad R = [1]$, and then sample a random vector $\boldsymbol{m} \in \mathbb{R}^{4\times 1}$ uniformly from the interval $[0.5, 1.5]$ to create a variety of controllers $\boldsymbol{K}_i = \boldsymbol{m}_i \cdot \boldsymbol{K}^* \forall i$.

Reacher (Simulated) For a higher dimensional task, we examine the 5 joint, three-dimensional, reacher manipulation task ($d_s = 15$, $d_a = 5$) in the Mujoco, OpenAI Gym environment [TET12; BCP+16], shown in Fig. 6.3b. The task associated with the environment is to maneuver the end-effector of the arm from an initial position state to an end position state. To create a diverse set of data for prediction, our experiments control the agent using a Proportional-Integral-Derivative (PID) controller with randomly generated parameter vectors $\boldsymbol{K} \in \mathbb{R}^{15}$. The parameters of a PID control are defined by a vector of joint angle targets, $\boldsymbol{z}_d \in \mathbb{R}^5$, proportional constants, $\boldsymbol{K}_p \in \mathbb{R}^5$, integrative constants, $\boldsymbol{K}_I \in \mathbb{R}^5$, and derivative constants, $\boldsymbol{K}_D \in \mathbb{R}^5$, for each rotatory joint. We set $\boldsymbol{K}_I = 0$ for all experiments. Given the joint angle z_i and the current error $e_i = z_i - z_d$, the control command at the i^{th} joint is $u_i = K_P \cdot e_i + K_D \cdot \dot{e}_i$.

Quadrotor (Simulated & Real Hardware) We validated the prediction accuracy of trajectory-based models on a simulated and experimental low-level attitude control of a quadrotor ($d_s = 5$, $d_a = 4$). The quadrotor model is based off the Crazyflie [GSWWK17], shown in Fig. 6.3c, an 27 g, open-source micro-aerial vehicle. The 12 state Euler-step simulation follows [MKC12] and has uniform Gaussian noise on all state variables sampled from $\sigma \sim \mathcal{N}(0, 0.01)$. The simulated controller is a linear, pitch and roll PD controller with randomly sampled parameters. For experimental data, we collected 180 s of aggressive flight data with default PID rate-controllers. This data was broken down into trajectories of length 1000 randomly, which we used to validate our prediction mechanism.

6.4 Long-term Prediction Accuracy

We now demonstrate the ability of the trajectory-based models (T) to more accurately predict long-horizon robotic dynamics by measuring the mean-squared error of the predicted trajectory versus the measured state, $\sum_{t=1}^{H} \|\hat{s}_t - s_t\|^2$. We evaluate the ability to predict horizons of over 100 steps and trajectories longer than the original training distributions. An advantage of trajectory-based models over one-step models is that T models lack a need to be given a time-series of actions from an oracle or to compute a new action from the current



(a) Cartpole (Simulated).

(b) Reacher (Simulated).

(c) Quadrotor (Simulated)

(d) Quadrotor (Real Hardware)

Figure 6.5: The log-scale, mean-squared prediction error for the tested environments. The simulated environments are tested on 250 validation trajectories and the experimental data is tested on 16 training trajectories due to experimental restrictions. Highlighted is the median error with the 65^{th} and 95^{th} percentile of errors. These figures highlight two properties: 1) $5\times$ to $10\times$ gain in long term prediction accuracy via trajectory-based models for $h > 50$ and 2) the uncertainty in one-step prediction continues to growth with the prediction step while trajectory-based error remains stable. The vertical line indicates the length of trajectories in the training distribution.

state. When predicting to long horizons with one-step models, the error can compound and diverge rapidly if the predicted state is used to re-compute the action, shown in Fig. 6.4. For a more competitive baseline in the remainder of our experiments, the one-step models predict the next state given the original action sequences, $\{a_t\}_{t=0}^{L}$, and the trajectory-based models given only θ_π.

The prediction accuracy for D, PE, $LSTM$, and T models with error 65^{th}, 95^{th} percentiles (tested on 100 trajectories) is shown for the cartpole trained on 100 trials of 200 time-steps, Fig. 6.5a, 100 reacher trials of 500 time-steps, Fig. 6.5b, 100 simulated quadrotor trials of

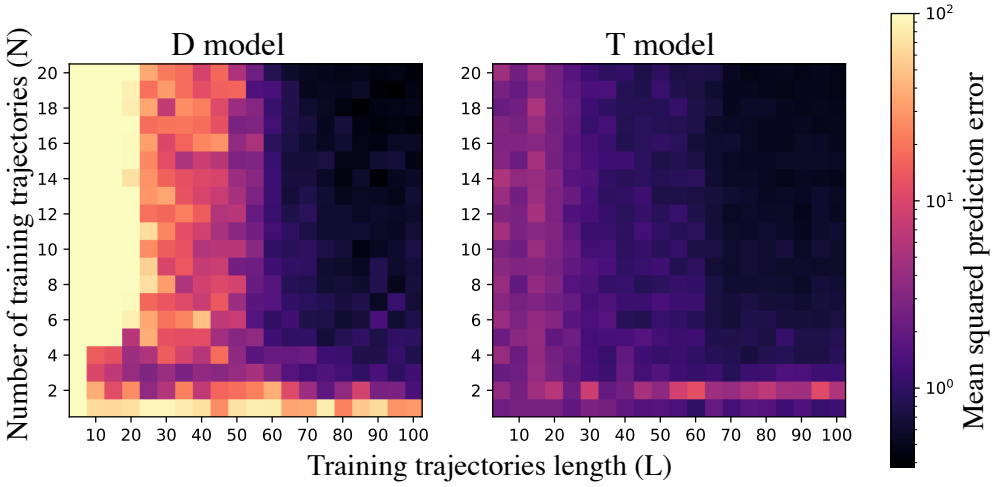


Figure 6.6: The median, cumulative prediction error of 5 models sweeping over the number N and length L of reacher training trajectories given on a constant validation set of trajectories of length 100. The trajectory-based model achieves substantially better prediction accuracy with both shorter and fewer trajectories in the training set. In the context of MBRL, the trajectory based model has better *sample efficiency* by having lower cumulative prediction error when trained on fewer trajectories N (a slice at a specific length L).

500 time-steps, Fig. 6.5c, and 16 experimental quadrotor trajectories of 1000 time-steps, Fig. 6.5d. The experimental quadrotor trajectories all have the same control parameters, which the model could use to better generalize across trajectories, showing that adding time-dependence alone can improve long-term prediction accuracy. All states are normalized to a range of $[0, 1]$ before computing the MSE to match error across different states (*i.e.*, the scale of a velocity is matched to the scale of an angle). The trajectory-based models are less accurate for short horizons ($h < 25$), but converge to an improvement of up to $10\times$ reduction in MSE for long horizons both in simulation and experiment. In practice, it is expected that some testing trajectories will extend beyond the expected length, which we evaluate by testing on trajectories of greater length than the training set. Even with out of distribution time indices, the trajectory-based models maintain their improvement in accuracy over one-step models, removing any need for T models to be trained on equal-length trajectories.

6.5 Accelerated Data Efficiency

Having sufficient labeled data is frequently a limiting factor in deep-learning tasks. Recall from Sec. 6.2 that the labelled data for trajectory-based models grows at a rate of the trajectory length squared, L^2. Leveraging this accelerated accruing of data, we evaluate the ability of trajectory-based models to predict accurately in the low-data regime. In

Prediction Mechanism	Prediction Mapping	Cartpole MSE $\pm\sigma$	Reacher MSE$\pm\sigma$
Direct mapping (GP)	$(\boldsymbol{K}, s_0) \mapsto \hat{\boldsymbol{r}}$	0.88 ± 1.87	0.13 ± 0.19
Direct mapping (NN)	$(\boldsymbol{K}, s_0) \mapsto \hat{\boldsymbol{r}}$	0.45 ± 1.19	0.07 ± 0.11
One-step; oracle (D)	$\left(f_\theta, \boldsymbol{a}_{t=0:L}, s_0\right) \mapsto \hat{\boldsymbol{r}}$	0.06 ± 0.10	0.46 ± 2.05
One-step; pred. actions (D)	$\left(f_\theta, \boldsymbol{K}, s_0\right) \mapsto \hat{\boldsymbol{r}}$	0.98 ± 1.73	0.07 ± 0.09
Trajectory-based (T)	$\left(f_\theta, \boldsymbol{K}, s_0\right) \mapsto \hat{\boldsymbol{r}}$	**0.01 ± 0.03**	**0.01 ± 0.01**

Table 6.1: The mean-squared predicted reward error across 100 simulated trajectories show the strength of learning long term dynamics for predicting episode reward. The 100 trajectories have different initial states s_0, and control parameters $\boldsymbol{K}$. The rewards are normalized per the number of trial step – 200 steps for the cartpole task and 500 for the reacher task.

the reacher environment, we trained models on a grid of trajectory lengths, $L \in [5, 100]$, and number of trajectories, $N \in [1, 20]$, to predict a validation set of 10 trajectories of length 100. Given an initial state, s_t, we can predict a set horizon, h, into the future to obtain a simulated trajectory of states, $\hat{\tau} = \{\hat{s}_i\}_{i=t}^{t+h}$, and measure the mean-squared error (MSE) across all normalized state dimensions. In Fig. 6.6, the trajectory-based models show improved performance for both a) shorter trajectories (L) (another link to predicting beyond the initial training length in Sec. 6.4) and b) fewer samples (N). The regime of low number of training trajectories (N) represents an area of high value to robotic tasks via its potential for reduced evaluations on a real robot.

6.6 Predictive Episode Reward

Predicting reward is tied to planning actions for robotic systems because if one can accurately predict rewards for an action set, then one can correctly rank actions. The trajectory-based models predict rewards in a simulated task by coupling reward prediction to stable long-term predictions. In this section we compare the predicted reward, $\hat{\boldsymbol{r}}$, of an initial state, $\boldsymbol{s}_0$, and control parametrization $\boldsymbol{K}$ to the true cumulative reward, $\boldsymbol{r} = \sum_{t=0}^{L} r(s_t)$.

We consider five methods for predicting the reward of an episode from an initial state and control parameters: 1) the trajectory-based models, 2) the one-step models given the action sequence from true-states (oracle), 3) the one-step models with actions computed from predicted states (predicted-actions), 4) Gaussian Processes (GPs), and 5) neural networks (NN) predicting directly from the initial state and control parameters to sum of reward. Each candidate method is a different mapping in the following function-space: $h : (\boldsymbol{s}_0, \boldsymbol{K}, \theta) \mapsto \boldsymbol{r}$, where θ carries different model formulations. The dynamics models are used to predict future states via $\hat{s}_{t+1} = f_\theta(\cdot)$, with the one-step models taking in the previous predicted state and the trajectory-based models updating the time index. The predicted trajectory reward uses

the environment-defined reward function $r(s_t, a_t)$ summed over time (the reward functions are action independent). A Gaussian process (GP), defined by a mean vector, $\mu_\theta(\boldsymbol{x})$, and a covariance matrix, $k(\boldsymbol{x}_1, \boldsymbol{x}_2)$, or a neural network (NN) can predict the reward with no structured dynamics model. for the GP and the NN, the target rewards are normalized uniformly to $[-1, 1]$ before prediction to aid in model training.

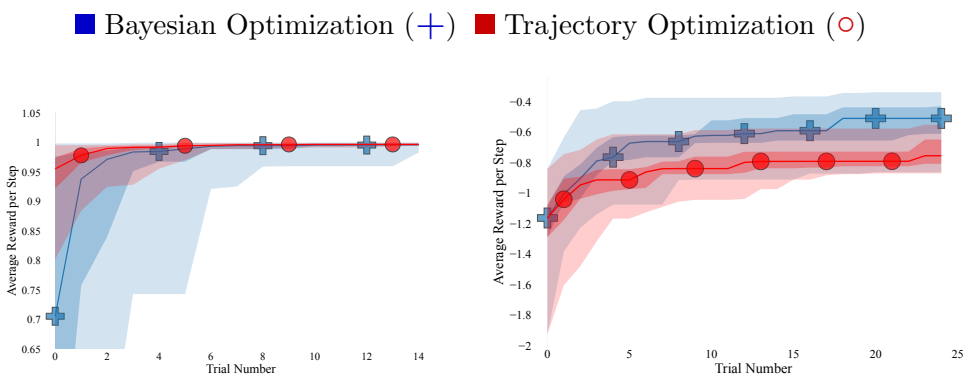
For both the cartpole and the reacher tasks, the trajectory-based model outperforms other methods in predicted reward accuracy. The mean-squared predicted reward error across 100 trajectories is shown in Tab. 6.1. We hypothesize that the *structured* learning of a dynamics model accurate across a trajectory improves reward prediction with knowledge of each step over predicting directly to the cumulative reward.

6.7 Iterative Learning of Control Parameters

We now compare the trajectory-based model against black-box optimization algorithms for the iterative learning of control parameters. At each iteration, we retrain the trajectory-based model and then use it to simulate rollouts of different control parameters. Finally, we execute the best control parameters found on the real system, and a new iteration starts. Ideally, a more accurate dynamics model over the trajectory will result in faster convergence and better performance. In our experiments, we indicate this approach as *Trajectory Optimization* and use covariance matrix adaptation evolution strategy (CMA-ES) to optimize the simulated rollouts. However, CMA-ES can be replaced with any optimizer that does not require gradients on the reward function. Trajectory Optimization generates new control parameters within the population of CMA-ES and simulates a trajectory of the trial length 200 for cartpole. The simulated reward $\hat{r}$ is the sum over the predicted states, as in Sec. 6.6. We compare this approach to the data-efficient black-box optimization algorithm Bayesian Optimization (BO) which iteratively optimize the control parameters without knowledge of the dynamics [MHBST16; CSPD16; BCXLT17a]. The results of the experiments in Fig. 6.7 show that on the toy cartpole benchmark task Trajectory Optimization converge to excellent performance faster than Bayesian Optimization. This demonstrates that we can learn in an iterative manner trajectory dynamics model, and that they can successfully be applied for control.

6.8 Model Predictive Control with Trajectory-based Models

We now demonstrate how to use the trajectory-based model in a common control architecture – model predictive control (MPC) [SKS03; Wie06]. MPC is a common tool for model-based reinforcement learning [CCML18; WWG+17; NYA+18], and originated in the study of optimal control leveraging predictions to make decisions [GPM89; Kir12]. MPC with



(a) Cartpole (1 is the maximum reward per step).

(b) Reacher (0 is the maximum reward per step).

Figure 6.7: The cumulative maximum reward for the iterative learning task on cartpole. The trajectory-based search with CMA-ES shows impressive performance, consistently solving the cartpole task in 2-4 trials. The 66$^{\text{th}}$ and 95$^{\text{th}}$ percentiles over 25 trials are shaded. , highlighting the consistency of the approach compared to Bayesian Optimization.

learned one-step dynamics models is formulated as

$$a_t^* = \arg\max_{u_{t:t+\tau}} \sum_{i=0}^{\tau} r(\hat{s}_{t+i}, a_{t+i}), \quad \hat{s}_{t+1} = f_\theta(\hat{s}_t, a_t). \tag{6.5}$$

With the trajectory-based model, the control formulation needs a modification in how the candidate solutions are selected. Sampling over control policies and computing the action from the current state, the new MPC formulation is

$$\theta_{\pi,t}^* = \arg\max_{\theta_{\pi,t:t+\tau}} \sum_{i=0}^{\tau} r(\hat{s}_{t+i}, a_{t+i}) \,, \tag{6.6}$$

$$\hat{s}_{t+\tau} = f_\theta(s_t, \theta_{\pi,t}, t+\tau) \,, \quad a_t^* = \theta_\pi^*(t) \,. \tag{6.7}$$

MPC is known to be computationally intensive – where some robots lack the computing infrastructure to run recent MBRL methods online – so we compare leveraging the trajectory-based optimization only from the first state, and running that policy through the remaining of the trial. This is a starting point, and re-planning online with varying update frequencies (holding the chosen policy for T steps) would allow flexibility in control.

As a comparison to iterative learning of one set of control parameters, as in Sec. 6.7, we compare the performance of MPC to that if the optimization is only run on the first time-step. In this case we maintain random sampling to mirror common applications of MPC in MBRL, while the Trajectory Optimization in Sec. 6.7 leveraged the CMA-ES optimizer. The preliminary results for the planning methods on the cartpole task are shown in Fig. 6.8.

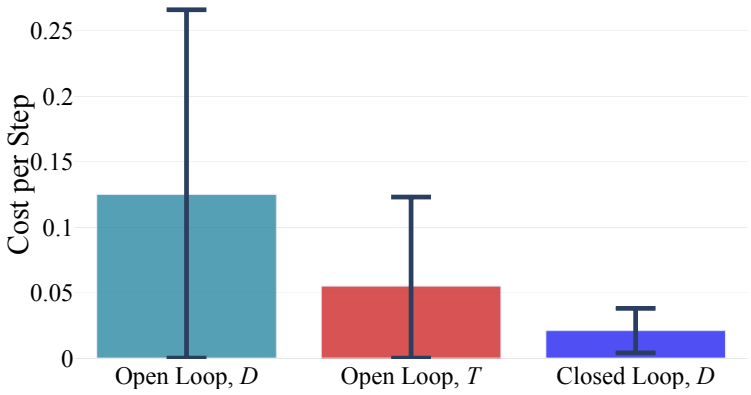


Figure 6.8: Comparison between trajectory-based and one-step ahead models planning from the initial state (i.e., open-loop) and the one-step model re-planning after each step (i.e., closed loop). The trajectory-based planning improves performance compared to the one-step ahead in open-loop. However, the one-step model used in closed-loop performs the best, at the expense of increased computational complexity for re-planning. The mean and standard deviation of the reward per step in the cartpole task are reported for 25 trials.

The trajectory-based model is limited by aggregating labelled data with MPC because it requires the sub-trajectories that it is labelled on to have constant control parameters. This can be partially overcome by re-planning at a lower frequency, but is an important direction for future work to better integrate the new models into existing MBRL literature.

6.9 Predicting Unstable and Periodic Dynamics

Important to the application of dynamics models to robotic tasks is the ability to accurately model dynamics when a) trained on imperfect data (e.g. data with noise and divergent modes) and b) evaluated on different modes of data. In this section, the evaluation of predicting stable dynamics is extended onto unstable and periodic dynamics. Representative stable, unstable, and periodic trajectories are shown in Fig. 6.9a,b,c. Unstable dynamics are designed to be diverging through the trajectory and periodic dynamics have consistent cyclic motion of various frequencies.

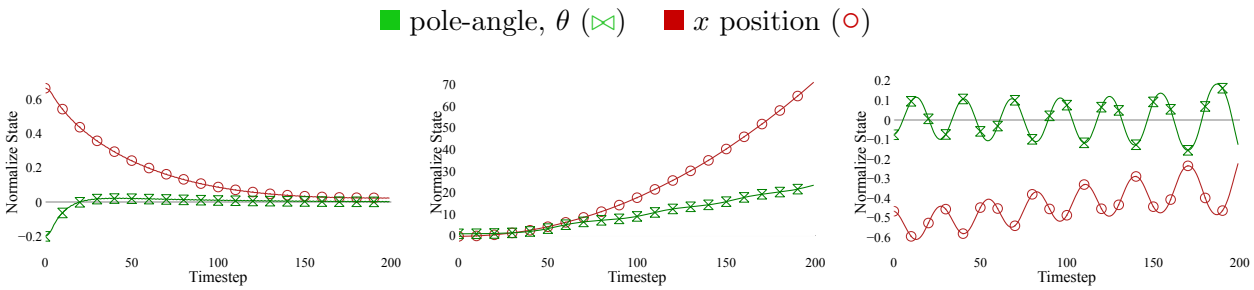
To test this, we collect 3 training and testing datasets ($N_{\text{traj}} = 100$) for each datatype above in the cartpole environment via different tunings of LQR control. The stable data is the same used in Sec. 6.4 where the majority of trajectories converge towards $\boldsymbol{s} = 0$. The one-step models maintain similar performance to trajectory-based models when trained on stable data, shown in Fig. 6.9d,g,j. In the unstable-data trained models (Fig. 6.9e,h,k) and the periodic-data trained models (Fig. 6.9f,i,l), the trajectory-based models demonstrate an impressive performance in modelling dynamics motion across long horizons. Except when trained and tested on unstable data, the one-step models diverge rapidly – suggesting two

potential modes of training for the one-step models: a) they may be memorizing the unstable trajectories and b) when training on stable data, the majority of the delta-state predictions are around 0, so when predicting out of distribution data the model may predict no change, slowing divergence. Conversely, when trained on unstable or periodic data, and testing elsewhere, the one-step models diverge rapidly due to the constant change in state in the training set, shown in Fig. 6.9e,f,i,k,l. The ability for trajectory-based models to generalize from periodic data to stable dynamics confers a substantial improvement over the one-step model.

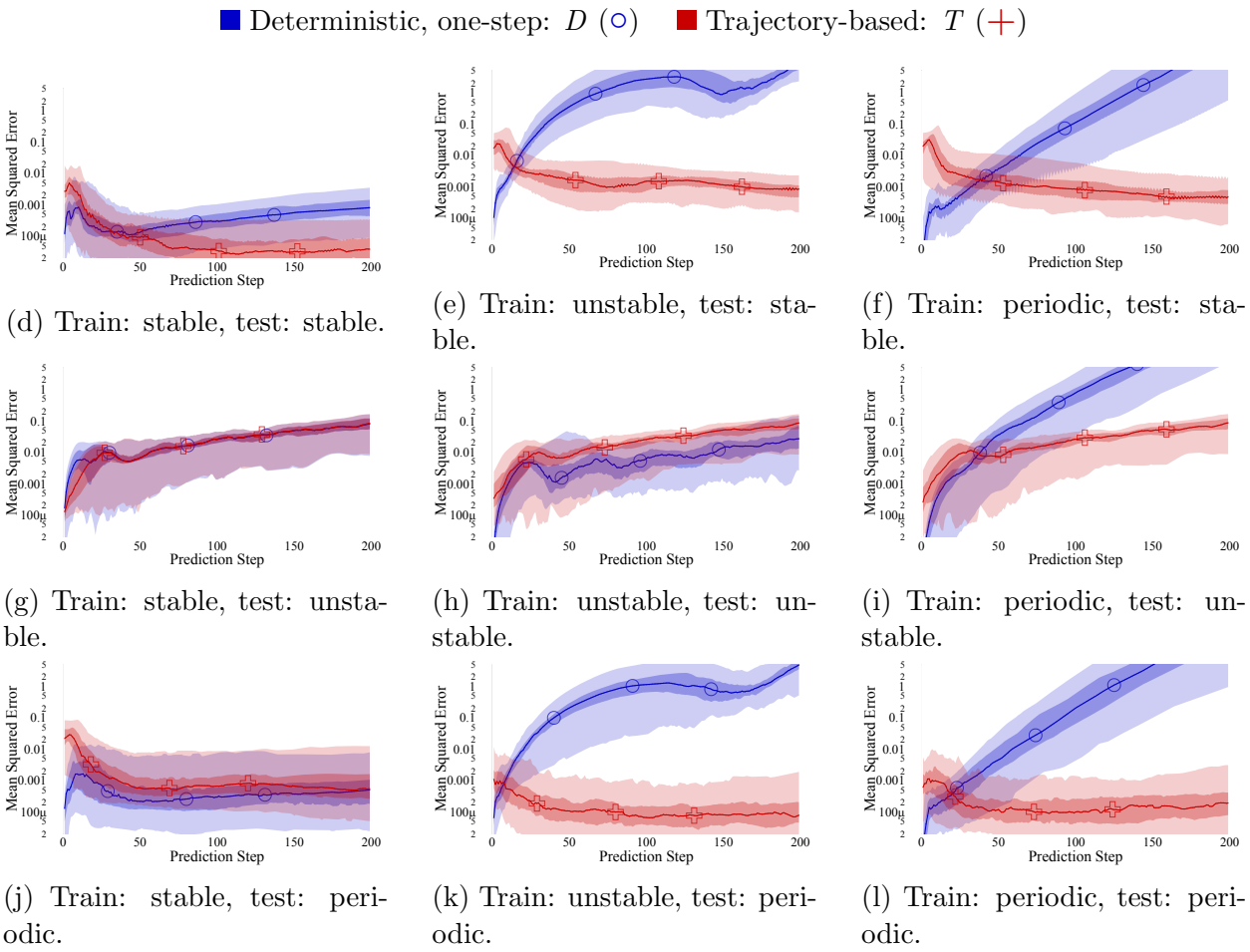
6.10 Related Work

Predicting long-term trajectories has a long-history, dating back to autoregressive-moving-average models (ARMAX) [NP11], and has been a growing area of research as robotic complexity outgrew the performance of historical approaches. A common approach is to modify one-step prediction training to account for correlation across a trajectory with NNs or GPs. Specifically, [NXD20] proposes a time-weighted loss function to learn unstable state-space systems, but does not apply it to high-dimensional, nonlinear systems. [LAYC20] showed increased correlation between model-accuracy and downstream reward when training with batches sampled from a single-trajectory instead of random transitions, but predictions still suffer from compounding errors.

Other related methods have attempted to create direct, structural links between models and trajectory training data. With high dimensional images (rather than states) [KST+19] uses an auto-regressive, recurrent network to predict observations in a latent state-space. [DDN+17] proposes a multi-step Gaussian Process for learning robotic control and the approach is studied further using the correlation between prediction steps in [HAFZ20]. [WFT14] suggests using a new kernel in GPs to correlate across trajectory data and using simulated model rollouts to predict reward as a prior GP mean function. These methods all leverage one-step models to create long-term dynamics, which we differentiate from by including time-dependence. Other exciting avenues of long-term prediction of dynamical systems are with neural ordinary differential equations [CRBD18] and long short-term memory blocks (LSTMs) [HS18b], but both are yet to be successfully applied to prioperceptive (*i.e.,* not vision based) robotics control tasks. We baseline our method against LSTMs which have implicit time dependance, where our method includes it explicitly in the input.



(a) A stable trajectory. (b) An unstable trajectory. (c) A periodic trajectory.



(d) Train: stable, test: stable.

(e) Train: unstable, test: stable.

(f) Train: periodic, test: stable.

(g) Train: stable, test: unstable.

(h) Train: unstable, test: unstable.

(i) Train: periodic, test: unstable.

(j) Train: stable, test: periodic.

(k) Train: unstable, test: periodic.

(l) Train: periodic, test: periodic.

Figure 6.9: *Above:* representative trajectories for the stable, unstable, and periodic datasets. *Below:* the evaluation error when using a model trained exclusively on one-variety of data (e.g. unstable) to predict all types of data. This figure represents 6 models (D and T trained on 3 different datasets) and 18 evaluations (the 6 models evaluated on the 3 testing sets).

Chapter 7

Future Directions and Open Challenges

7.1 Filling the Gap Between Model-based and Model-free

The gap between canonical model-based and model-free algorithms is shrinking. Some of the most notable advancements in recent years have been algorithms that would easily fit the taxonomy of model-based *and* model-free. Prominent examples of this are the Model-based Policy Optimization (MBPO) [JFZL19] that learns via an actor-critic algorithm in a model simulator and MuZero [SAH+20] that learns a dynamics model that is not trained explicitly for accuracy. Moving forward, it is likely that model-free will be the narrow classification – where any algorithm that *does not employ a model* is model-free. This classification leaves model-based methods encompassing a large spectrum of algorithms.

A more informative taxonomy is needed for model-based reinforcement learning algorithms. The current algorithms can be classified into a few distinctive areas. Online planning algorithms rely on repeated unrolling of action candidates [CCML18; WB20; WWG+17], model-sim algorithms train the agent in a learned simulator [JFZL19], and implicit model-based algorithms train a model by proxy that is not explicitly constrained to the dynamics [SAH+20; ASYW21]. This represents an initial direction for a taxonomy, though it is far from complete.

7.2 Accurate, Generalizable, and Transferable Predictions

Further improvements to the accuracy and transfer of learned dynamics models is central to the continued progress in MBRL. The primary avenue of development of MBRL algorithms has been on simulated, continuous control tasks where relatively shallow neural networks

can learn the dynamics with relatively low data in comparison to Q-functions. The next generation of algorithms will likely change both the prediction formulation and the tools for doing so. Past work uses multiple models to predict multiple horizons [AMKL19], a multi-step loss [VHB15], or more parametrization tweaks on the same variety of feedforward, discrete step model [LWZPC21].

A series of recent works have taken more substantial steps to reparametrize the predictions in MBRL. The recent trajectory transformer uses pre-trained language models to cast the prediction problem as classification [JLL21], the trajectory-based model adds direct temporal dependency to the standard feedforward dynamics models for gains in long-term predictions [LWZPC21], and the Gamma model re-frames prediction as a generative model [JML20]. Continuing to integrate these lines of work into effective decision making routines is an open challenge for the field.

These works on new forward predictive models are pointing MBRL in the same direction of task-focused accurate models, while dynamics models have more structural benefits, such as the transfer of knowledge in a closed form or generalization to new tasks. The current state-of-the-art in MBRL is to re-train the dynamics model for every trial on every task. New infrastructure to allow model transfer (within or across domains) for continual learning and faster iteration would facilitate research progress. The field of MBRL also lacks well-adopted tools for evaluating their models on other notions of performance than single-domain, single-task accuracy. There is need for isolated model evaluation, where new model training procedures can be evaluated given a set decision making scheme. These direction could move the dynamics models from a per-agent training paradigm to a large model paradigm where representation learning and data aggregation allow for the capture of more complex behavior across domains.

7.3 Spatio-temporal Abstractions

In real-world control problems, the notion of time is often both flexible and inconsistent. Due to most of the progress of RL research being driven by accessible and fast simulated environments with static notions of time (that the user often cannot modify), the opportunities entailed by a richer notion of time in decision making have not been explored in depth. The goal of a generally intelligent and practical agent is one that can dynamically vary its notion of time to solve tasks that span from reflexes to strategies.

Recent works on spatio-temporal abstraction in reinforcement learning have largely fallen under the themes of options frameworks and hierarchical RL. Options frameworks cast the decision making problem at a higher level of abstraction where an agent decides on a behavior for a period of time, rather than a sequence of actions [SPS99; Pre00]. Options have been used in many ways in RL, such as in exploration [ŞB04] or skill transfer [KB07], but they lack the multi-level or even continuous notion of time that would most mirror the diversity of internal models used by humans [DSD12].

One step closer to arbitrary notions of time lays hierarchical model-based reinforcement learning, which promises to help scale solutions by leveraging natural layered structures of the environment. These methods either learn models [JS08], policies [KNST16], or values [Die+98] focusing on different temporal horizons. For each sub-version of the problem, the agent will use a separate decision making rule. Ideally, with more flexible notions of temporal abstraction one decision rule can be used to solve all potential tasks in an abstract domain.

Model-based reinforcement learning has an exciting opportunity and challenge to directly incorporate flexible notions of time into the model, and then progress on the decision making can follow. Recent work has been investigating the performance of machine learning systems to model chaotic systems [Gil20; JHF21] or continuous physics [KGZKM21]. These problem spaces can push developments in temporal abstraction due to their grounding in systems that are often characterized across a spectrum of timescales. MBRL will benefit from research on the integration problem of new model classes with existing decision making tools.

7.4 Computational Efficiency: From Planning to Reactive Policies

Translating MBRL algorithms to the real world is often limited by the computational intensity of the recent methods. Given the prevalence of simulators in driving algorithmic progress, little focus is given to the clock time of the algorithms in comparison to the focus on sample complexity.Model predictive control (MPC) based algorithms suffer most from this because of the thousands of forward passes through a model at decision time. Efforts have begun to reduce MPC-based decision rules to policy via better proposals for action candidates [BHT+21; WB20]. Recent work in optimization-based MPC in an non-RL setting show the ability to reduce a controller to a policy from optimal control labels [CSA+18; ZBB20]. On the other hand, simply re-training a dynamics model after every episode and potentially simulating it to collect on-policy data for a value function estimate adds substantial background processing in addition to the decision-time bottleneck.

7.5 Solving Objective Mismatch

Objective mismatch occurs when the optimization of a model is separate from the optimization of the final policy in MBRL [LAYC20]. Dynamics models can be used to generate policies in different ways, such as the training of a model inducing a policy in a decision-time planner akin to model predictive control or via a model being used to collect data for an actor-critic policy optimization. Generally, there is no use of observed reward nor gradients from the controller in this process, which leaves the model as an object solely optimized for

accuracy over an automatically collected dataset [1]. Automatically collected datasets are often biased by the exploration mechanism or environment when compared to hand-engineered demonstration-based datasets that are focused on specific areas of coverage. Objective mismatch emerges in many ways, but poses difficulty in achieving clear understanding on the limits of MBRL algorithms in challenging tasks.

There are lines of work both directly positing to mitigate objective mismatch and studying its effects from related perspectives. Many methods attempt to use information about the task in the model training or rollouts, such as a value function gradient for model training [VLGF22], goal-aware predictions [NSF20], or implicit gradients through the model [NAAB21]. Alternatively, software is being developed that allows direct differentiation of the model through controllers [AJSBK18] or simulators [LSWP21].

Mismatch is also deeply related to systems where the dynamics models are only trained implicitly for accuracy, with the gradients being focused on rewards [SHM+21]. Comparing the predictions and decision of models trained in different manners would illuminate how mismatch manifests in an optimization, and if the posited solutions operate as intended.

Open questions in addressing objective mismatch involve investigating the underlying cause of this phenomena. Some potential causes include:

- *Uneven model accuracy*: if the downstream reward is well correlated because the common deep models do not cover the training data in proportion to their importance for control (or if the training data is too non-uniform). While batch gradient descent is efficient and flexible, it may be that for dynamics models practitioners must prioritize accuracy at a few crucial states rather than on average.

- *Policy exploitation*: if the optimizer is too strong given a certain class of model, it will always be able to exploit non-grounded phenomena. As dynamics models get more accurate, comparing action selections at different model snapshots can illuminate the causes of policy exploitation.

7.6 Bridging Curiosity-based Motivation, Pixel- and State-based MBRL, and Exploration

Three closely related fields have been developing methods to learn and use forward dynamics models for control. State-based MBRL has continued to develop algorithms for continuous control from states, intrinsic curiosity has been using learned dynamics models for unsupervised RL from states and pixels, and pixel-based MBRL has been setting state-of-the-art scores on sample-efficiency game-playing tasks. Bridging the insights from these closely related fields represents opportunity for aligning progress and understanding of the community.

[1] Models trained solely on existing data are also not suited for some application domains with high-stakes, such as autonomous vehicles. In these cases generalization is needed to prevent some failures.

Model-based RL methods, especially those utilizing sample-based control, explore in a related, passive manner where the model error contributes indirectly to exploration. The sample-based MPC algorithms propose action sequences that may not have been previously executed by the agent. The optimizer selects the action with the highest perceived reward, but when the model is very inaccurate it is not understood if this action selection reduces to random sampling or some biased sampling related to the coverage of dynamics model error. The exploration mechanism is often described as one variety model exploitation. As the dynamics model gets more accurate these MBRL algorithms explore less and focus on the pre-described task.

Intrinsic curiosity is a growing subfield of work optimizing intelligent exploration of unknown environments. Curiosity-based methods utilize the error signal of a trained dynamics models to select actions with different RL agents [BEP+18]. This error-based reward signal can be viewed as running an MBRL agent with a time-varying reward function because the model error changes each time it is updated. The idea behind curioisty-based exploration is to reward an agent to re-visit states where a learned dynamics model is high-error to gather data that is "interesting". The Intrinsic Model Predictive Control exploration agent bridges the gap between model exploitation and intrinsic rewards by using a curiosity model in parallel to a learned dynamics model [LWW+22], but how it relates to off-policy curiosity and task-aware MBRL is left to future work.

State-of-the-art pixel-based algorithms create a latent state that is used for control [HLNB21; HWS22], yet there are now comparisons to how the behavior of this state-space mirrors or differs from existing methods in state-based continuous control. As in state-based algorithms, model exploitation plays a heavy role in exploration, but it is not rigorously characterized.

7.7 Additional Benefits of Learning a Model

Learned dynamics models convey increased potential for designing agents capable in more nuanced manners than scalar reward maximization. For example, a model can expand an existing agent's abilities to avoid regions of a state-space (safety) or understand why and how decisions are made (interpretability). These example model uses can be expanded into many categories, such as explainability, fairness, or causality, without solely combining them into the reward function of the RL agent.

Safe control is a starting point to extract additional information from models. Current methods integrate safety techniques from optimal control, such as system stability [BTSK17] or safe set constraints [KBTK18]. Future work in safety for MBRL will involve integrating this well-studied techniques from optimal control with advanced methods that can integrate with value functions or supervision [TBR+20]. A less clearly defined problem is that of interpretability – where we hope to be able to pinpoint why a certain action was selected by an agent. Prior work in explainable AI (XAI) for RL [PV20] shows that is a problem well beyond translating existing solutions for supervised learning, with little existing work studying the most popular methods of deep RL. Initial work investigating Deep Q-Networks

revealed hierarchical state abstractions in the learned value function and studied activation regions of learned policies [ZBM16]. That paper focuses on evaluating current capabilities, and accepts that there is little understanding in how the agent converges to this state. A learned model is a substantially compressed snapshot of the agent's knowledge, requiring less complex methodology for building tools for more broadly intelligent agents.

Chapter 8

Conclusion

In the near future we will have agents capable of exploring new environments, rapidly learning new skills, and working with humans. These factors will lead to technologies transforming society with new solutions to at-scale control problems. Reinforcement learning is an open-ended framework touted as the most likely tool to achieve this dream in the near term. The RL method is powerful and appealing due to its open-ended framing and self-supervised deployment, but these apparent strengths can result in agents prone to exploitation.

Designing algorithms that capture the upside of RL while mitigating the risks is crucial for translation of RL from a proof of concept to a widespread engineering tool. Model-based reinforcement learning represents a promising candidate for the efficient learning of effective controllers that humans can understand and control. The promise of models as a structured form of learning conveys a series of benefits to an intelligent agent itself and to an engineer looking to safely deploy this technology. This thesis presents a substantial overview of the challenges of integrating a learned dynamics model with dynamic decision making systems. Creating new modular, autonomous systems with synergistic components is an exciting engineering challenge in the pursuit of building AI systems that do good. The goal is to lay the groundwork for a next generation of algorithms that integrate seamlessly with societal goals. By deeply understanding the challenges of compounding error and objective mismatch, the future generations of model-based reinforcement learning algorithms can unlock not only high-performance agents, but intelligent systems that are safe, interpretable, and transferable. The powerful solutions of RL systems can only be deployed for the good of all when the gap between understanding and practice is closed. In the future, I want to build these empirically grounded reinforcement learning systems.

Bibliography

[AJSBK18] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. "Differentiable mpc for end-to-end planning and control". In: *Advances in neural information processing systems* 31 (2018).

[AMKL19] Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michel L Littman. "Combating the compounding-error problem with a multi-step model". In: *arXiv preprint arXiv:1905.13320* (2019).

[AN04] Pieter Abbeel and Andrew Ng. "Learning first-order Markov models for control". In: *Advances in neural information processing systems* 17 (2004).

[ASYW21] Brandon Amos, Samuel Stanton, Denis Yarats, and Andrew Gordon Wilson. "On the model-based stochastic value gradient for continuous reinforcement learning". In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 6–20.

[AWR+17] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017).

[AY20] Brandon Amos and Denis Yarats. "The differentiable cross-entropy method". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 291–302.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[BCXLT17a] S. Bansal, Roberto Calandra, T. Xiao, S. Levine, and C. J. Tomlin. "Goal-Driven Dynamics Learning via Bayesian Optimization". In: *IEEE Conference on Decision and Control (CDC)*. 2017, pp. 5168–5173. DOI: 10.1109/CDC.2017.8264425.

[BCXLT17b] Somil Bansal, Roberto Calandra, Ted Xiao, Sergey Levine, and Claire J Tomiin. "Goal-driven dynamics learning via Bayesian optimization". In: *IEEE Conference on Decision and Control (CDC)*. 2017, pp. 5168–5173.

[Bel57] Richard Bellman. "A Markovian decision process". In: *Journal of mathematics and mechanics* (1957), pp. 679–684.

[BEP+18] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. "Large-scale study of curiosity-driven learning". In: *arXiv preprint arXiv:1808.04355* (2018).

[Ber95] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA, 1995.

[BHT+21] Arunkumar Byravan, Leonard Hasenclever, Piotr Trochim, Mehdi Mirza, Alessandro Davide Ialongo, Yuval Tassa, Jost Tobias Springenberg, Abbas Abdolmaleki, Nicolas Heess, Josh Merel, et al. "Evaluating model-based planning and planner amortization for continuous control". In: *arXiv preprint arXiv:2110.03363* (2021).

[Bit16] AB Bitcraze. *Crazyflie 2.0*. 2016.

[Boo97] Gary Boone. "Efficient reinforcement learning: Model-based acrobot control". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 1. IEEE. 1997, pp. 229–234.

[Bry18] Arthur Earl Bryson. *Applied optimal control: optimization, estimation and control*. Routledge, 2018.

[BSH+20] Mohammad Babaeizadeh, Mohammad Taghi Saffar, Danijar Hafner, Harini Kannan, Chelsea Finn, Sergey Levine, and Dumitru Erhan. "Models, pixels, and rewards: Evaluating design trade-offs in visual model-based reinforcement learning". In: *arXiv preprint arXiv:2012.04603* (2020).

[BSWR14] Joschka Boedecker, Jost Tobias Springenberg, Jan Wülfing, and Martin Riedmiller. "Approximate real-time optimal control based on sparse gaussian process models". In: *2014 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE. 2014, pp. 1–8.

[BTSK17] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. "Safe model-based reinforcement learning with stability guarantees". In: *Advances in neural information processing systems* 30 (2017).

[CCML18] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Neural Information Processing Systems*. 2018.

[CD12] Frank M Callier and Charles A Desoer. *Linear system theory*. Springer Science & Business Media, 2012.

[CGH+20] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. "Lagrangian neural networks". In: *arXiv preprint arXiv:2003.04630* (2020).

[CNF+18] Ignasi Clavera, Anusha Nagabandi, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Learning to Adapt: Meta-Learning for Model-Based Control". In: *CoRR* abs/1803.11347 (2018). arXiv: 1803.11347. URL: http://arxiv.org/abs/1803.11347.

[CRBD18] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural ordinary differential equations". In: *Neural Information Processing Systems*. 2018, pp. 6571–6583.

[CSA+18] Steven Chen, Kelsey Saulnier, Nikolay Atanasov, Daniel D Lee, Vijay Kumar, George J Pappas, and Manfred Morari. "Approximating explicit model predictive control using constrained neural networks". In: *2018 Annual American control conference (ACC)*. IEEE. 2018, pp. 1520–1527.

[CSM+16] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. "Transfer from simulation to real world through learning deep inverse dynamics model". In: *arXiv preprint arXiv:1610.03518* (2016).

[CSPD16] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. "Bayesian optimization for learning gaits under uncertainty". In: *Annals of Mathematics and Artificial Intelligence* 76.1-2 (2016), pp. 5–23.

[DDN+17] Andreas Doerr, Christian Daniel, Duy Nguyen-Tuong, Alonso Marco, Stefan Schaal, Toussaint Marc, and Sebastian Trimpe. "Optimizing long-term predictions for model-based policy search". In: *Conference on Robot Learning*. 2017, pp. 227–238.

[DFB+22] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. "Magnetic control of tokamak plasmas through deep reinforcement learning". In: *Nature* 602.7897 (2022), pp. 414–419.

[DGLZ21] Sarah Dean, Thomas Krendl Gilbert, Nathan Lambert, and Tom Zick. "Axes for sociotechnical inquiry in AI research". In: *IEEE Transactions on Technology and Society* 2.2 (2021), pp. 62–70.

[Die+98] Thomas G Dietterich et al. "The MAXQ Method for Hierarchical Reinforcement Learning." In: *ICML*. Vol. 98. Citeseer. 1998, pp. 118–126.

[DLSP18] Daniel S Drew, Nathan O Lambert, Craig B Schindler, and Kristofer SJ Pister. "Toward controlled flight of the ionocraft: a flying microrobot using electrohydrodynamic thrust with onboard sensing and no moving parts". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 2807–2813.

[DR11] Marc P. Deisenroth and Carl E. Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *International Conference on Machine Learning*. 2011, pp. 465–472.

[DSD12] Bradley B Doll, Dylan A Simon, and Nathaniel D Daw. "The ubiquity of model-based reinforcement learning". In: *Current opinion in neurobiology* 22.6 (2012), pp. 1075–1081.

[EFD+18] 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". In: *arXiv preprint arXiv:1812.00568* (2018).

[Ell21] Ashok Elluswamy. "Planning and Control". Tesla AI Day. 2021. URL: `https://www.youtube.com/watch?v=j0z4FweCy4M&t=4802s`.

[ESL19] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. "Search on the replay buffer: Bridging planning and reinforcement learning". In: *Advances in Neural Information Processing Systems* 32 (2019).

[ET12] Tom Erez and Emanuel Todorov. "Trajectory optimization for domains with contacts using inverse dynamics". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 4914–4919.

[Far18] Amir-massoud Farahmand. "Iterative value-aware model learning". In: *Advances in Neural Information Processing Systems*. 2018, pp. 9072–9083.

[FBN17] Amir-massoud Farahmand, Andre Barreto, and Daniel Nikovski. "Value-aware loss function for model-based reinforcement learning". In: *Artificial Intelligence and Statistics*. 2017, pp. 1486–1494.

[FLA16] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4019–4026.

[GDZL22] Thomas Krendl Gilbert, Sarah Dean, Tom Zick, and Nathan Lambert. "Choices, Risks, and Reward Reports: Charting Public Policy for Reinforcement Learning Systems". In: *arXiv preprint arXiv:2202.05716* (2022).

[GGL18] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. "Learning Actionable Representations with Goal-Conditioned Policies". In: *arXiv preprint arXiv:1811.07819* (2018).

[Gil20] William Gilpin. "Deep learning of dynamical attractors from time series measurements". In: *arXiv preprint arXiv:2002.05909* (2020).

[GJK+19] Omer Gottesman, Fredrik Johansson, Matthieu Komorowski, Aldo Faisal, David Sontag, Finale Doshi-Velez, and Leo Anthony Celi. "Guidelines for reinforcement learning in healthcare". In: *Nature medicine* 25.1 (2019), pp. 16–18.

[GKBNB19] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. "Deepmdp: Learning continuous latent space models for representation learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2170–2179.

[GMMK20] Jayesh K Gupta, Kunal Menda, Zachary Manchester, and Mykel Kochenderfer. "Structured mechanical models for robot learning and control". In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 328–337.

[GMR16] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. "Improving PILCO with Bayesian neural network dynamics models". In: *Data-Efficient Machine Learning workshop, ICML*. Vol. 4. 34. 2016, p. 25.

[GPM89] Carlos E Garcia, David M Prett, and Manfred Morari. "Model predictive control: theory and practice—a survey". In: *Automatica* 25.3 (1989).

[GSWWK17] Wojciech Giernacki, Mateusz Skwierczyński, Wojciech Witwicki, Paweł Wroński, and Piotr Kozierski. "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering". In: *IEEE International Conference on Methods and Models in Automation and Robotics (MMAR)*. 2017, pp. 37–42.

[GWH14] Robert Grande, Thomas Walsh, and Jonathan How. "Sample efficient reinforcement learning with gaussian processes". In: *International Conference on Machine Learning*. PMLR. 2014, pp. 1332–1340.

[HA17] Wolfgang Hönig and Nora Ayanian. "Flying multiple uavs using ros". In: *Robot Operating System (ROS)*. Springer, 2017, pp. 83–118.

[HAFZ20] Lukas Hewing, Elena Arcari, Lukas P Fröhlich, and Melanie N Zeilinger. "On simulation and trajectory prediction with gaussian process dynamics". In: *Learning for Dynamics and Control*. 2020, pp. 424–434.

[HGD96] Håkan Hjalmarsson, Michel Gevers, and Franky De Bruyne. "For model-based control design, closed-loop identification gives better performance". In: *Automatica* 32.12 (1996), pp. 1659–1673.

[HHA19] Hado P van Hasselt, Matteo Hessel, and John Aslanides. "When to use parametric models in reinforcement learning?" In: *Advances in Neural Information Processing Systems* 32 (2019).

[HK11] Elnaz Jahani Heravi and Sohrab Khanmohammadi. "Long term trajectory prediction of moving objects using gaussian process". In: *IEEE International Conference on Robot, Vision and Signal Processing*. 2011.

[HLF+19] 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*. PMLR. 2019, pp. 2555–2565.

[HLNB21] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. "Mastering Atari with Discrete World Models". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=0oabwyZbOu.

[HS18a] David Ha and Jürgen Schmidhuber. "Recurrent world models facilitate policy evolution". In: *Advances in neural information processing systems* 31 (2018).

[HS18b] David Ha and Jürgen Schmidhuber. "World models". In: *arXiv preprint arXiv:1803.10122* (2018).

[HWS22] Nicklas Hansen, Xiaolong Wang, and Hao Su. "Temporal Difference Learning for Model Predictive Control". In: *arXiv preprint arXiv:2203.04955* (2022).

[JFZL19] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. "When to trust your model: Model-based policy optimization". In: *Neural Information Processing Systems*. 2019, pp. 12498–12509.

[JHF21] Tom Z Jiahao, M Ani Hsieh, and Eric Forgoston. "Knowledge-based learning of nonlinear dynamics and chaos". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.11 (2021), p. 111101.

[JLL21] Michael Janner, Qiyang Li, and Sergey Levine. "Offline Reinforcement Learning as One Big Sequence Modeling Problem". In: *Advances in neural information processing systems* 34 (2021).

[JML20] Michael Janner, Igor Mordatch, and Sergey Levine. "gamma-models: Generative temporal difference learning for infinite-horizon prediction". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1724–1735.

[JS08] Nicholas K Jong and Peter Stone. "Hierarchical model-based reinforcement learning: R-max+ MAXQ". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 432–439.

[Kaw99] Mitsuo Kawato. "Internal models for motor control and trajectory planning". In: *Current opinion in neurobiology* 9.6 (1999), pp. 718–727.

[KB07] George Dimitri Konidaris and Andrew G Barto. "Building Portable Options: Skill Transfer in Reinforcement Learning." In: *IJCAI*. Vol. 7. 2007, pp. 895–900.

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[KBM+20] Łukasz Kaiser et al. "Model Based Reinforcement Learning for Atari". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=S1xCPJHtDB.

[KBTK18] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. "Learning-based model predictive control for safe exploration". In: *2018 IEEE conference on decision and control (CDC)*. IEEE. 2018, pp. 6059–6066.

[KGZKM21] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. "Characterizing possible failure modes in physics-informed neural networks". In: *Advances in Neural Information Processing Systems* 34 (2021).

[KHT21] Balázs Kégl, Gabriel Hurtado, and Albert Thomas. "Model-based micro-data reinforcement learning: what are the crucial model properties and which model to choose?" In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=p5uylG94S68.

[Kir12] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2012.

[KNST16] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems* 29 (2016).

[KST+19] Nan Rosemary Ke, Amanpreet Singh, Ahmed Touati, Anirudh Goyal, Yoshua Bengio, Devi Parikh, and Dhruv Batra. "Learning Dynamics Model in Reinforcement Learning by Incorporating the Long Term Future". In: *arXiv preprint arXiv:1903.01599* (2019).

[KVC+21] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. "Mt-opt: Continuous multi-task robotic reinforcement learning at scale". In: *arXiv preprint arXiv:2104.08212* (2021).

[KZGR21] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. "A survey of generalisation in deep reinforcement learning". In: *arXiv preprint arXiv:2111.09794* (2021).

[LA14] Sergey Levine and Pieter Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in neural information processing systems* 27 (2014).

[LAYC20] Nathan Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. "Objective Mismatch in Model-based Reinforcement Learning". In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 761–770.

[LDY+19] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. "Low-level control of a quadrotor with deep model-based reinforcement learning". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4224–4230.

[LHB+21] Michael Lutter, Leonard Hasenclever, Arunkumar Byravan, Gabriel Dulac-Arnold, Piotr Trochim, Nicolas Heess, Josh Merel, and Yuval Tassa. "Learning dynamics models for model predictive agents". In: *arXiv preprint arXiv:2109.14311* (2021).

[LLSA21] Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. "Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 6131–6141.

[Lor63] Edward N Lorenz. "Deterministic nonperiodic flow". In: *Journal of atmospheric sciences* 20.2 (1963), pp. 130–141.

[LPB17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in neural information processing systems* 30 (2017).

[LPC22] Nathan Lambert, Kristofer Pister, and Roberto Calndra. "Investigating Compounding Prediction Errors in One-step Dynamics Models". In: *arXiv preprint* (2022).

[LR85] Richard Lewis and Gregory C Reinsel. "Prediction of multivariate time series by autoregressive model fitting". In: *Journal of multivariate analysis* 16.3 (1985), pp. 393–411.

[LSDP20] Nathan O Lambert, Craig B Schindler, Daniel S Drew, and Kristofer SJ Pister. "Nonholonomic yaw control of an underactuated flying robot with model-based reinforcement learning". In: *IEEE Robotics and Automation Letters* 6.2 (2020), pp. 455–461.

[LSWP21] Michael Lutter, Johannes Silberbauer, Joe Watson, and Jan Peters. "Differentiable physics models for real-world offline model-based reinforcement learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 4163–4170.

[LTL+20] Nathan O Lambert, Farhan Toddywala, Brian Liao, Eric Zhu, Lydia Lee, and Kristofer SJ Pister. "Learning for Microrobot Exploration: Model-based Locomotion, Sparse-robust Navigation, and Low-power Deep Classification". In: *2020 International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS)*. IEEE. 2020, pp. 1–6.

[LWW+22] Nathan Lambert, Markus Wulfmeier, William Whitney, Arunkumar Byravan, Michael Bloesch, Vibhavari Dasagi, Tim Hertweck, and Martin Riedmiller. "The Challenges of Exploration for Offline Reinforcement Learning". In: *arXiv preprint arXiv:2201.11861* (2022).

[LWZPC21] Nathan Lambert, Albert Wilcox, Howard Zhang, Kristofer SJ Pister, and Roberto Calandra. "Learning accurate long-term dynamics for model-based reinforcement learning". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE. 2021, pp. 2880–2887.

[MHBST16] Alonso Marco, Philipp Hennig, Jeannette Bohg, Stefan Schaal, and Sebastian Trimpe. "Automatic LQR tuning based on Gaussian process global optimization". In: *International Conference on Robotics and Automation*. 2016, pp. 270–277.

[MKC12] Robert Mahony, Vijay Kumar, and Peter Corke. "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor". In: *IEEE Robotics and Automation magazine* 19.3 (2012), pp. 20–32.

[MLJ+19] Daniel J Mankowitz, Nir Levine, Rae Jeong, Abbas Abdolmaleki, Jost Tobias Springenberg, Timothy Mann, Todd Hester, and Martin Riedmiller. "Robust Reinforcement Learning for Continuous Control with Model Misspecification". In: *arXiv preprint arXiv:1906.07516* (2019).

[MS17] Christopher D McKinnon and Angela P Schoellig. "Learning multimodal models for robot dynamics online with a mixture of Gaussian process experts". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 322–328.

[MSA+21] John McLeod, Hrvoje Stojic, Vincent Adam, Dongho Kim, Jordi Grau-Moya, Peter Vrancx, and Felix Leibfried. "Bellman: A Toolbox for Model-based Reinforcement Learning in TensorFlow". In: *arXiv:2103.14407* (2021). URL: `https://arxiv.org/abs/2103.14407`.

[MZR+22] Amol Mandhane, Anton Zhernov, Maribeth Rauh, Chenjie Gu, Miaosen Wang, Flora Xue, Wendy Shang, Derek Pang, Rene Claus, Ching-Han Chiang, et al. "MuZero with Self-competition for Rate Control in VP9 Video Compression". In: *arXiv preprint arXiv:2202.06626* (2022).

[NAAB21] Evgenii Nikishin, Romina Abachi, Rishabh Agarwal, and Pierre-Luc Bacon. "Control-Oriented Model-Based Reinforcement Learning with Implicit Differentiation". In: *arXiv preprint arXiv:2106.03273* (2021).

[NKLK20] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. "Deep dynamics models for learning dexterous manipulation". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1101–1112.

[NP11] Duy Nguyen-Tuong and Jan Peters. "Model learning for robot control: a survey". In: *Cognitive processing* 12.4 (2011), pp. 319–340.

[NSF20] Suraj Nair, Silvio Savarese, and Chelsea Finn. "Goal-aware prediction: Learning to model what matters". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7207–7219.

[NXD20] Kamil Nar, Yuan Xue, and Andrew M Dai. "Learning Unstable Dynamical Systems with Time-Weighted Logarithmic Loss". In: *arXiv preprint arXiv:2007.05189* (2020).

[NYA+18] Anusha Nagabandi, Guangzhao Yang, Thomas Asmar, Ravi Pandya, Gregory Kahn, Sergey Levine, and Ronald S Fearing. "Learning image-conditioned dynamics models for control of underactuated legged millirobots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4606–4613.

[OAC18] Ian Osband, John Aslanides, and Albin Cassirer. "Randomized prior functions for deep reinforcement learning". In: *Advances in Neural Information Processing Systems* 31 (2018).

[ORA17] Masashi Okada, Luca Rigazio, and Takenobu Aoshima. "Path integral networks: End-to-end differentiable optimal control". In: *arXiv preprint arXiv:1706.09597* (2017).

[PAZLC21] Luis Pineda, Brandon Amos, Amy Zhang, Nathan O. Lambert, and Roberto Calandra. "MBRL-Lib: A Modular Library for Model-based Reinforcement Learning". In: *Arxiv* (2021). URL: `https://arxiv.org/abs/2104.10159`.

[Pre00] Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[PV20] Erika Puiutta and Eric Veith. "Explainable reinforcement learning: A survey". In: *International cross-domain conference for machine learning and knowledge extraction*. Springer. 2020, pp. 77–95.

[Ras03] Carl Edward Rasmussen. "Gaussian processes in machine learning". In: *Summer school on machine learning*. Springer. 2003, pp. 63–71.

[RG99] Sam Roweis and Zoubin Ghahramani. "A unifying review of linear Gaussian models". In: *Neural computation* 11.2 (1999), pp. 305–345.

[RYRF21] Rafael Rafailov, Tianhe Yu, Aravind Rajeswaran, and Chelsea Finn. "Offline reinforcement learning from images with latent space models". In: *Learning for Dynamics and Control*. PMLR. 2021, pp. 1154–1168.

[RZN+21] Oleh Rybkin, Chuning Zhu, Anusha Nagabandi, Kostas Daniilidis, Igor Mordatch, and Sergey Levine. "Model-based reinforcement learning via latent-space collocation". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9190–9201.

[SAH+20] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839 (2020), pp. 604–609.

[ŞB04] Özgür Şimşek and Andrew G Barto. "Using relative novelty to identify useful temporal abstractions in reinforcement learning". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 95.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SDDS88] William D Stanley, Gary R Dougherty, Ray Dougherty, and H Saunders. "Digital signal processing". In: (1988).

[SHD18] Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. "Meta reinforcement learning with latent variable gaussian processes". In: *arXiv preprint arXiv:1803.07551* (2018).

[SHM+21] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. "Online and offline reinforcement learning by planning with a learned model". In: *Advances in Neural Information Processing Systems* 34 (2021).

[SHS+18] 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*. PMLR. 2018, pp. 4470–4479.

[SJALF18] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Universal planning networks: Learning generalizable representations for visuomotor control". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4732–4741.

[SKS03] David H Shim, H Jin Kim, and Shankar Sastry. "Decentralized nonlinear model predictive control of multiple flying robots". In: *IEEE International Conference on Decision and Control*. Vol. 4. 2003, pp. 3621–3626.

[SPS99] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[SRSSP21] Sumeet Singh, Spencer M Richards, Vikas Sindhwani, Jean-Jacques E Slotine, and Marco Pavone. "Learning stabilizable nonlinear dynamics with contraction-based regularization". In: *The International Journal of Robotics Research* 40.10-11 (2021), pp. 1123–1150.

[Sut91] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM Sigart Bulletin* 2.4 (1991), pp. 160–163.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Neural Information Processing Systems*. 2014, pp. 3104–3112.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).

[Tal17] Erik Talvitie. "Self-correcting models for model-based reinforcement learning". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[TBR+20] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. "Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3612–3619.

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *IEEE International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[Tou97] Claude F Touzet. "Neural reinforcement learning for behaviour synthesis". In: *Robotics and Autonomous Systems* 22.3-4 (1997), pp. 251–281.

[VHB15] Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. "Improving multi-step prediction of learned time series models". In: *AAAI Conference on Artificial Intelligence*. 2015.

[VLGF22] Claas A Voelcker, Victor Liao, Animesh Garg, and Amir-massoud Farahmand. "Value Gradient weighted Model-Based Reinforcement Learning". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=4-D6CZkRXxI.

[VS15] Harm Vanseijen and Rich Sutton. "A deeper look at planning as learning from replay". In: *International conference on machine learning*. PMLR. 2015, pp. 2314–2322.

[WB19] Tingwu Wang and Jimmy Ba. "Exploring model-based planning with policy networks". In: (2019). URL: https://openreview.net/forum?id=0xDIv1GlJYx.

[WB20] Tingwu Wang and Jimmy Ba. "Exploring Model-based Planning with Policy Networks". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=H1exf64KwH.

[WBC+19] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. "Benchmarking model-based reinforcement learning". In: *arXiv preprint arXiv:1907.02057* (2019).

[WFT14] Aaron Wilson, Alan Fern, and Prasad Tadepalli. "Using trajectory data to improve bayesian optimization for reinforcement learning". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 253–282.

[WHF06] Jack Wang, Aaron Hertzmann, and David J Fleet. "Gaussian process dynamical models". In: *Neural Information Processing Systems*. 2006, pp. 1441–1448.

[Wie06] Pierre-Brice Wieber. "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations". In: *IEEE-RAS International Conference on Humanoid Robots*. 2006, pp. 137–142.

[WSBR15] 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* 28 (2015).

[WWG+17] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. "Information theoretic MPC for model-based reinforcement learning". In: *International Conference on Robotics and Automation*. 2017, pp. 1714–1721.

[WYLFT15] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning". In: *Advances in neural information processing systems* 28 (2015).

[YBL+22] Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. "Don't Change the Algorithm, Change the Data: Exploratory Data for Offline Reinforcement Learning". In: *arXiv preprint arXiv:2201.13425* (2022).

[YC14] Michael C Yip and David B Camarillo. "Model-less feedback control of continuum manipulators in constrained environments". In: *IEEE Transactions on Robotics* 30.4 (2014), pp. 880–889.

[ZBB20] Xiaojing Zhang, Monimoy Bujarbaruah, and Francesco Borrelli. "Near-optimal rapid MPC using neural networks: A primal-dual policy learning framework". In: *IEEE Transactions on Control Systems Technology* 29.5 (2020), pp. 2102–2114.

[ZBM16] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. "Graying the black box: Understanding dqns". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1899–1908.

[ZDG+96] Kemin Zhou, John Comstock Doyle, Keith Glover, et al. *Robust and optimal control*. Vol. 40. Prentice hall New Jersey, 1996.

[ZPN+21] Michael R Zhang, Tom Le Paine, Ofir Nachum, Cosmin Paduraru, George Tucker, Ziyu Wang, and Mohammad Norouzi. "Autoregressive dynamics models for offline policy evaluation and optimization". In: *arXiv preprint arXiv:2104.13877* (2021).

[ZRP+21] Baohe Zhang, Raghu Rajan, Luis Pineda, Nathan Lambert, André Biedenkapp, Kurtland Chua, Frank Hutter, and Roberto Calandra. "On the importance of hyperparameter optimization for model-based reinforcement learning". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 4015–4023.

[ZVS+19] 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*. PMLR. 2019, pp. 7444–7453.