Thesis

Advancing Model-Based Reinforcement Learning with Applications in Nuclear Fusion

Ian Char

April 11, 2024
CMU-ML-24-102

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was funded by: National Science Foundation Graduate Research Fellowship Program award DGE1745016; Department of Energy award DESC0021414; and US Army awards W911NF1820218, W911NF20F0017 and W519TC23C0030.

Keywords: reinforcement learning, bayesian optimization, uncertainty quantification, plasma control, nuclear fusion, machine learning

Abstract

Reinforcement learning (RL) may be the key to overcoming previous insurmountable obstacles, leading to technological and scientific innovations. One such example where RL could have a sizable impact is in tokamak control. Tokamaks are one of the most promising devices for making nuclear fusion into a viable energy source. They operate by magnetically confining a plasma; however, sustaining the plasma for long periods of time and at high pressures remains a challenge for the tokamak control community. RL may be able to learn how to sustain the plasma, but like many exciting applications of RL, it is infeasible to collect data on the real device in order to learn a policy.

In this thesis, we explore learning policies using surrogate models of the environment, and especially using surrogate models that are learned from an offline data source. To start in Part I, we investigate the scenario in which one has access to a simulator that can be used to generate data, but the simulator is too computationally taxing to use data-hungry deep RL algorithms. We instead suggest a Bayesian optimization algorithm to learn such a policy. Following this, we pivot to the setting in which surrogate models of the environment can be learned with offline data. While these models are much more computationally cheap, their predictions inevitably contain errors. As such, both robust policy learning procedures and good uncertainty quantification of model errors are crucial for success. To address the former, in Part II we propose a trajectory stitching algorithm that accounts for these modeling errors and a policy network architecture that is adaptive, yet robust. Part III shifts focus onto uncertainty quantification, where we propose a more intelligent uncertainty sampling procedure and a neural process architecture for learning uncertainties efficiently. In the final part, we detail how we learned models to predict plasma evolution, how we used these models to train a neutral beam controller, and the results of deploying this controller on the DIII-D tokamak.

Acknowledgments

Coming to Carnegie Mellon was a very uncertain time for me. I had been a late accept to the program, had very little previous machine learning experience, and was unsure what I wanted to focus my research on. However, I did know that I wanted to use machine learning as a tool for discovery, preferably in the sciences. I therefore feel extremely fortunate that my advisor, Jeff Schneider, took a chance on me to work on nuclear fusion with him. I am grateful not only to have gotten the opportunity to work on a once-in-a-lifetime project, but also to have been able to do it with such an amazing advisor. Under Jeff, I have learned what are the important research questions to ask. Perhaps even more important, I have learned a great deal about how to interact with collaborators by watching Jeff lead the lab with respect and patience towards his students and collaborators.

I would also like to thank Ruslan Salakhutdinov, Zico Kolter, Martin Riedmiller, and Egemen Kolemen for serving on committee and giving feedback on my work. I would especially like to thank Egemen Kolemen for guiding me through the world of nuclear fusion. His expertise was invaluable when it came time to run experiments on DIII-D.

I have an enormous amount of gratitude for the collaborators I had on the nuclear fusion project, who I now also consider to be close friends. I was lucky to work with Youngseog Chung from the very start. Not only is he one of my closest collaborators, but also one of the best friends I have made at CMU. I would like to thank him for being such a kind and hard-working collaborator over these years. I would also like to thank him for letting me beat him in Super Smash Bros so many times. My other close collaborator that I feel very fortunate to have worked with is Viraj Mehta. Viraj is perhaps one of the most brilliant, yet also warm, people that I have met, and I am excited to see what the future holds for him. Although my PhD only overlapped with his for a year, I am very grateful to have worked with Willie Neiswanger. I have come to view Willie as a second advisor, and through working with him, I have learned a great deal about how to conduct research and the power of optimism. I know that he will be a fantastic professor.

There are many others from Jeff's lab that I would like to thank. I have shared an office with Conor Igoe almost from the beginning of

PhD, and during this time, he became another one of my closest friends at CMU. I would like to thank him for all of our amazing conversations, on research or otherwise. I am also thankful to Adam Villaflor for teaching me so much about machine learning and for helping me out of many research ruts. I would also like to thank Kirthevasan Kandasamy for serving as a mentor and teaching me about Bayesian optimization when I first got to CMU. Finally, I would like to thank Ben Freed, Swapnil Pande, Brian Yang, Tejus Gupta, Yeeho Song, Arundhati Banerjee, and Biswajit Paria, and Rohan Shah for their friendship and wisdom.

Much of this thesis would not have been possible without the help of our collaborators in Egemen Kolemen's lab. I am enormously thankful for Joe Abbate's wisdom, kindness, and patience. I owe him a great deal for his help, and it was such a joy to work with him. I would also like to thank Rory Conlin and Andy Rothstein for all of the assistance over the years. There are also many people to thank within the broader fusion community including Keith Erickson, Jayson Barr, Nikolas Logan, Laszlo Bardoczi, Nathan Richner, Al Hyatt, and Heather Shen.

I also feel very lucky to have been a part of a department with such a warm, supportive environment. For this, I am thankful to Sharon Cavlovich, Dorothy Holland-Minkley, Roni Rosenfeld, and especially Diane Stidle. Within the department I made many close friends that I am thankful for including (but no limited to) Charvi Rastogi, Audrey Huang, Jeffrey Li, Terrance Liu, Jacob Tyo, Elan Rosenfeld, Jeremy Cohen, Juyong Kim, Ezra Winston, Robin Schmucker, Chenghui Zhou, Chirag Gupta, Helen Zhou, Ben Eysenbach, Euxhen Hasanaj, Ratana Pukdee, and Shantanu Gupta. I am also grateful that I got the opportunity to be a member of the Auton Lab, and I would like to thank Artur Dubrawski and Predrag Punosevac for all they do for the lab.

I have had many amazing educators that prepared me for my PhD experience. I would especially like to thank Manuel Lladser and Tom Yeh, who advised me during my time at the University of Colorado Boulder. I am very grateful for the opportunities they gave me, the skills that they taught me, and for fighting for me to get into a PhD program. I would also like to thank my 5th and 6th grade teacher Dave Dunham, who sparked a passion for learning in me at a young age. Last, but not certainly not least, I would like to thank my sister Erin Char, my mother Connie Char, my father William Char, and my partner Simone Zhang for their continual love and support no matter where life

takes me.

Throughout my PhD I was fortunate enough to be supported by the National Science Foundation Graduate Research Fellowship Program under Grant Numbers DGE1252522, DGE2140739, and DGE1745016. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. In addition, I was fortunate to receive funding through the Department of Energy award DESC0021414.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent progress has evolved machine learning from a useful tool into something that is able to create magic. Large language models assist us in finding information, planning tasks, and writing code [Brown et al., 2020, Jiang et al., 2024, Team et al., 2023, Touvron et al., 2023]; diffusion models allow us to generate images and videos through textual prompts [Ramesh et al., 2021]; and neural networks can predict complex structures like protein structure with unprecedented accuracy [Jumper et al., 2021]. These achievements point to a future in which machine learning will enable us to overcome previously insurmountable obstacles that are necessary for transformative technological and scientific discoveries.

Reinforcement learning (RL) will be a crucial tool in realizing this future. The goal of RL is to learn a policy which, given the current state of the system, gives an action (or distribution of actions) to be played in the system. Lately, RL has proven to be an invaluable tool for reinforcement learning with human feedback (RLHF) [Ouyang et al., 2022], which is a key step in the training pipeline to make intelligent AI agents. However, RL (and especially deep RL) also has immense potential for difficult control tasks. These difficulties may arise from a high-dimensional observation space, a large number actuators that need to be coordinated, or because the dynamics of the system cannot be fully modeled. When this is the case, it may be impossible for an engineer to create a controller, and a data-driven approach may be the only answer. Indeed, deep RL has already shown remarkable achievements in video game playing [Vinyals et al., 2019], legged locomotion [Agarwal et al., 2023], and dexterous manipulation [Zhu et al., 2019].

However, applications of RL with the most impact often have the most difficulties. One of these difficulties comes from the fact that deep RL is data hungry and collecting data in the real system is bottlenecked due to economic costs, the speed at which samples can be collected, or safety concerns. Successful applications of

RL avoid this problem by using surrogate models, often in the form of simulators [Agarwal et al., 2023, Kaufmann et al., 2023, Degrave et al., 2022]. However, such simulators need to be simultaneously computationally cheap and accurate. If the simulator is accurate but expensive, alternative methods for policy learning must be employed. Using reinforcement learning is even more infeasible when the simulator is inaccurate. This is especially unfortunate in cases where the dynamics of the system are unknown, and one of the cases where a data-driven approach like RL would be most useful.

Alternatively, one could use a database of historical interactions with the system to learn a policy. This is known as the "offline" RL setting [Levine et al., 2020] and has garnered much interest from the research community. Crucially, we are still unable to gather additional information on the real system; all of the data is collected from previously deployed policies that may be unrelated to the task at hand. Of particular interest to this thesis is "model-based" offline RL, in which a surrogate model of the system is learned using historical data [1]. The learned model can be used to generate additional experience leveraged in the policy learning. Furthermore, depending on the system and the amount of data, this learned model may be more accurate than a simulator and is almost certainly less computationally expensive. Learning such a surrogate model has been shown to be valuable [Yu et al., 2020, Janner et al., 2019b] especially in cases where the collected data is "undirected," i.e. the logged data is agnostic to the task we wish to train the policy for.

Of course, the main challenge in using a learned surrogate model is that there are bound to be errors between its predictions and the real system. In the online setting, this issue can be mitigated by limiting the number of steps that can be made with the model and increasing this amount as more data is collected [Janner et al., 2019b]. In the offline setting, previous approaches turn to adding penalizers based on model uncertainty to ensure that the policy stays within the support of the training data [Yu et al., 2020, Kidambi et al., 2020a]. However, there are several factors that make these approaches difficult to employ in real world offline RL setups. For one thing, these algorithms rely on tuning the amount of pessimism. This becomes difficult if one is truly in an offline setting where the policy can only be deployed in the true environment at test time. Additionally, the benchmark environments considered in these works are usually deterministic, and it is unclear how uncertainty based penalizers are affected when aleatoric noise is present in the system. That being said, there are also aspects of these benchmark tasks that may be harder than some real world environments. In particular, the dynamics models learned for most benchmark

[1] note that one could argue the setting in which a simulator is used is also "model-based"; however, this term is usually reserved when the surrogate model is learned from data

tasks in the literature are only accurate for a handful of steps. In this thesis, however, we find that there are real world tasks of interest for which dynamics models can generate full trajectories accurately.

With these factors in mind, a better offline model-based RL procedure may be one in which we rely less on pessimism and more on two different factors: sophisticated uncertainty quantification in the dynamics modeling and policy learning algorithms that produce robust, adaptable policies. To expand on this, if one can learn a realistic distribution for possible transition functions, one can then learn a policy that is able to quickly adapt its actions to cater towards any given sampled transition function. Indeed, there has already been initial work showing that learning an adaptable policy allows one to reduce the amount of pessimism needed [Chen et al., 2021b, Ghosh et al., 2022]. However, substantial improvements in both uncertainty quantification and policy learning procedures are necessary in order to fully realize this vision.

Towards this end, this thesis explores the algorithmic developments needed to learn a policy with surrogate models for challenging, realistic decision making tasks. In particular, we focus on three areas:

1. How do we learn a policy when given access to a simulator which is prohibitively expensive to run deep RL algorithms with?
2. How do we learn robust, adaptable policies for the offline model-based RL setting?
3. How do we learn sophisticated, well-calibrated uncertainties and how should we draw samples from these distributions?

The main motivating application for this thesis is tokamak control for nuclear fusion. In the following section, we give a brief overview of tokamak control before giving a more thorough overview of this thesis in Section 1.2. For a more thorough review of tokamaks and their control systems, please refer to Walker et al. [2020].

1.1 A Brief Overview of Nuclear Fusion and Tokamaks

The climate crisis has put an urgency on turning to alternative, green energy sources to power our society, and nuclear fusion has the potential to make a substantial impact in this effort. Nuclear fusion is the process in which two light elements (here we consider Deuterium and Tritium, two isotopes of Hydrogen) are fused together into a single heavier element (in our case Helium). This reaction results in some of the mass in the system being converted into energy. Indeed our solar system is powered by this process, where the pressure due to the immense mass of the sun causes millions of tons of Hydrogen atoms to fuse together every second.

Forming a reactor around controlled nuclear fusion reactions would have several substantial advantages over pre-existing energy sources, such as nuclear fission. The byproducts of the reaction would not contribute to global warming nor would they be highly radioactive; the fuel source is readily abundant and can be extracted from sea water; and there is virtually no risk of a catastrophic meltdown like there is in nuclear fission. However, creating such a reactor remains a challenge.

While there are several possible approaches to such a reactor, perhaps the most promising involves a device called a tokamak. This is a toroidal device which heats the gas inside until is becomes a plasma, and at these extreme temperatures the atoms are energetic enough to fuse together. This plasma is then magnetically confined using toroidal field coils. Although there are currently no tokamak reactors, there are a number of tokamaks that exist (or are being built) which scientists can run experiments on in order to further develop understanding of nuclear fusion. Perhaps the most ambitious of these tokamaks currently being built is the ITER tokamak. This is an international effort in order to prove that tokamaks can produce net positive energy. The main tokamak of interest for this thesis is the DIII-D tokamak managed by General Atomics in San Diego, CA.

While it is unclear how many actuators a viable tokamak reactor will have, there are several actuators on the DIII-D tokamak that can be used to affect the plasma. On top of the toroidal field coils that shape and confine the plasma, there is an ohmic coil that acts as a transformer and induces current into the plasma, gas valves which can increase the density of the plasma, radio frequency actuators can heat and drive current into the plasma, and neutral beams that inject power and torque into the plasma by shooting neutrally charged particles into the plasma. Throughout this thesis we will use different characterizations for the state of the plasma; however, the most expansive set of information includes both scalar and *profile* information. Profile measurements are taken from the core of the plasma out to the wall, and are a good representation of the plasma assuming ideal MHD conditions.

Creating a system which fully controls a tokamak is inherently a high dimensional problem in which many different actuators need to be coordinated. While the fusion control community has made significant progress on individual control loops, coordination amongst these loops remains an open problem of interest [Humphreys et al., 2015]. On top of this, there are several types of instabilities which may develop, and these instabilities must be avoided in a successful tokamak reactor. Deep reinforcement learning could therefore be a valuable tool for tokamak control since it would allow us to learn neural network policies that can make intelligent, coordinated decisions quickly.

At the same time, the challenges outlined at the beginning of this chapter apply to this problem. There are some aspects of the plasma are fairly well modeled, such as the shape of the plasma. In this case, good feedforward actuator plans can be made and paired with several PID controllers for feedback control [Anand et al., 2017]. In addition, simulators for the plasma's shape have been successfully paired with deep reinforcement learning to create controllers for the TCV tokamak [Degrave et al., 2022, Tracey et al., 2023]. However, other aspects of the plasma, such as the plasma's profiles, can only be approximately predicted. This, paired with their computational expense, makes using these simulators for reinforcement learning a tough sell.

1.2 Thesis Overview

Part I

The first part of this thesis considers the scenario in which we have access to a simulator which is computationally expensive, but is accurate enough to be informative for control. Because of this expense, we must look to alternative policy learning algorithms other than the ones found in the deep reinforcement learning literature. In Chapter 2, we therefore take an approach that relies on Bayesian optimization. The trade off here is that the policy is learned under a myopic objective. However, this restriction allows us to use an expensive simulator to learn a mapping from a low dimensional representation of the plasma to a neutral beam setting.

Part II

The second part of this thesis moves past expensive simulators and explores how to learn a policy assuming that a surrogate model of the system has been learned from data. While many offline model-based RL algorithms use actor-critic policy learning algorithms that are similar to those used in the standard RL setup, in Chapter 3 we explore how the offline dataset and the learned model can be used to construct a tabular Markov Decision Process (MDP). Given such an MDP, policy learning becomes easy since one can use policy iteration, which comes with theoretical guarantees.

In Chapter 4, we move back to using actor-critic algorithms for policy learning; however, we re-examine the architectural assumptions made by previous algorithms. In particular, most real world systems are partially observable and require aggregating information across the history of observations in order to make good, adaptive decisions. The majority of previous works use recurrent architectures to perform this aggregation; however, we find that using these architectures makes the policy's performance at test-time brittle to errors in the surrogate model. Inspired by the PID controller, we propose an alternative architecture that combines history using integration and differencing. We find that this inductive bias strikes a good balance between flexibility and robustness to modeling errors for many control tasks of interest.

Part III

The third part of this thesis shifts from policy learning to uncertainty quantification (UQ) of the learned dynamics model predictions. UQ plays an essential role for imagining the possible states the policy's actions may lead to, estimating which new data points would be the most valuable to collect, and, of course, for generating experience that can be used with a policy. Chapter 5 focuses on the last of these points. In particular, we point out that the majority of sampling schemes used in model-based RL works results in a non-smooth dynamics function. Doing so, however, leads to miscalibration over time which can result in undesired behavior in the learned policies. We propose a simple fix to this sampling scheme that can be applied on top of pre-existing dynamics models.

Following this, in Chapter 6, we focus on how we can learn more complex predictive uncertainties that can be conditioned on previous observations. In particular, we focus on neural processes: a class of neural networks that can meta-learn uncertainty given a distribution of different functions at training time. However, these neural processes are data hungry and are often brittle to distribution shift. In this chapter, we operate under the assumption that often times the only information needed to form good uncertainty is the distances between data points. Under this assumption, we form a graph of the data and propose two neural process architectures to operate over such a graph. We empirically show that doing so leads to more robust and sample efficient neural processes.

Part IV

Lastly, in part 4 of this thesis, we investigate using model-based reinforcement learning for control on the DIII-D tokamak. In Chapter 7, we summarize the experiment run on the DIII-D tokamak in which we aimed to learn a controller that could track the β_N and differential rotation quantities in the plasma. In particular, we outline the process of training a dynamics model from historical data, using said model to generate experience for a reinforcement learning algorithm, and deploying the learned controller on the DIII-D tokamak. Following this experiment, we continued to refine our dynamics models for the plasma, and in Chapter 8, we outline our improved dynamics models and evaluate how well it can predict full shots.

Open Source Code Contributions

To assist in much of the work presented in this thesis, we created a library referred to as the "Dynamics Toolbox" which contains tools to train dynamics models, evaluate these dynamics models, and use these models to train policies. In particular, this library was heavily used for the experiments in Chapters 5, 7, and 8. The inspiration for creating such a library came from the fact that the majority of previous model-based RL works (and code bases) use the exact same dynamics model architecture. In contrast, the Dynamics Toolbox has the flexibility to try many different types of neural network dynamics models. This library is publicly available and can be found at `https://github.com/IanChar/dynamics-toolbox`.

Part I

Myopic Policy Learning with Expensive Simulators

Chapter 2

Offline Contextual Bayesian Optimization

This chapter is based on Char et al. [2019]:
Char, I., Chung, Y., Neiswanger, W., Kandasamy, K., Nelson, A. O., Boyer, M., Kolemen, E., & Schneider, J. (2019). Offline contextual bayesian optimization. Advances in Neural Information Processing Systems, 32.

This chapter explores how to use a computationally expensive simulator to learn a controller via Bayesian optimization. To make this problem feasible, we make a simplification to the problem: rather than attempt to learn the optimal policy, we instead aim to learn a policy that is myopically optimal (i.e. in the context of a Markov Decision Process, a policy that plays actions that maximizes the reward function at every state). Given this simplification, the approach is to do an optimization for each state of the plasma of interest ahead of time (hence the term "offline"). The optima are then stored in a look-up table so that actuator responses can be made rapidly at test-time. Since the plasma state space is continuous, some interpolation scheme must be done in order to do this method in practice; however, this was not explored in this work.

While tokamak control was the motivation for this work, the problem statement of doing several different optimizations simultaneously in itself is very general and can be applied to a large class of different problems (see Section 2.1). Therefore the framing, nomenclature, and notation differs slightly from the rest of this thesis. What we refer to as a "state" in the rest of the thesis is referred to as a *context* or a *task* in this chapter and is denoted by x, the policy (usually denoted by π) is simply referred to as a mapping denoted by h, and we aim to maximize a general function

f instead of a reward function r.

2.1 Introduction

Black-box optimization is the problem in which one tries to find the maximum of an unknown function using only evaluations for specified inputs. In many interesting scenarios, there is a collection of unknown, possibly correlated functions (or *tasks*) that need to be simultaneously optimized. This problem set up often occurs in applications where one wants to design an agent that makes an action based on some contextual information from the environment. During its online execution we prefer not to run potentially costly or poor performing experimental actions. Also, because the agent may have to make these decisions at a rapid pace, we do not have time to compute an expensive experimentation policy. We consider applications that provide the ability to run offline experiments, either on a surrogate system or on a simulation. These experiments are used to discover a good action policy which is then encoded into a fast cache, such as a look-up table. Even though the experiments are done offline, they are still expensive and we must search the design space efficiently. The following are examples of this problem:

- **Nuclear Fusion** A current obstacle in realizing sustained nuclear fusion in tokamaks is the difficulty in maintaining the plasma's stability at the required temperatures and pressures for a prolonged period of time. We consider the stability of the plasma as an output to optimize, where the input is the controls for the tokamak. The optimal action depends on the current state of the plasma, so each plasma state can be regarded as its own task to optimize. We cannot search for a good control policy during live experiments on the device because of cost and limited time available on the device as well as the need to provide a real-time controller that operates in a millisecond scale control loop. However, we do have a simulation (forward model) that may be used with Bayesian optimization offline to discover a good controller. Importantly, the simulator allows one to manually set the current state of the plasma, and thus prudently selecting states to optimize over becomes an important part of the problem.
- **Database Tuning** Consider the problem of tuning the configuration of a database so as to minimize the latency, the CPU/memory footprint or any other desired criteria. The performance of a configuration depends critically on the underlying hardware and the workload [Van Aken et al., 2017]. Since these variables can change when databases are deployed in production, we need to simultaneously optimize for these different tasks.

In each of the above settings, difficulty of the tasks may vary drastically. For example, in the nuclear fusion application, if the current state of the plasma is already stable, the stability may be less sensitive to controls, leading to an easy optimization landscape. On the other hand, when the plasma is in an unstable state, it may be that only a small set of controls will lead to improved stabilization and finding them may require many more experiments.

Because experiments are usually expensive, one must be prudent with how resources are distributed across tasks. In this chapter, we propose a Thompson sampling approach for adaptively picking the next task and input for evaluation. This algorithm comes with theoretic guarantees, and we show that it often enjoys a significant boost in performance when compared to uniformly distributing resources across tasks. Another important contribution of this chapter is showing the significance of model choice in this setting. We argue that when using a single Gaussian process (GP) to jointly model correlated tasks, the choice of kernel is crucial for estimating the difficulty of each task. Without reasonable estimates, it is impossible to optimally distribute resources among tasks. As such, we propose a kernel that is expressive enough to capture variation in difficulty.

We end this chapter by showing an application of our method to the nuclear fusion problem. In particular, we optimize tokamak controls for a set of different plasma states using a tokamak simulator. We observe that our method is able to identify where best to devote resources, leading to efficient optimization.

2.2 Related Work

As is common in Bayesian optimization, we use a GP prior to guide us in selecting next evaluations to make. Previously, in the context of active learning and active sensing, techniques have been made that use GPs to select the most informative points for evaluation [Pasolli and Melgani, 2011, Seo et al., 2000, Guestrin et al., 2005]. In contrast, our goal is optimization which is more in line with bandit methods. Under the bandits setting, Srinivas et al. [2009] use an upper confidence bound approach with GPs and show that such a strategy results in sublinear cumulative regret. As an alternative to the upper confidence bound approach, Russo and Van Roy [2014] show that one can achieve sublinear cumulative regret using a posterior sampling (or Thompson sampling) approach. The method we present here is also a posterior sampling method, and it falls into the general framework of myopic posterior sampling described by Kandasamy et al. [2019a].

Our setting is related to online *contextual* bandits [Krause and Ong, 2011, Agrawal and Goyal, 2013, Auer, 2002], where each task can be viewed as a different context. In these earlier works, the agent chooses an action online for a context that is chosen by the environment. In our setting, we wish to find the optimal action offline in advance and can choose the contexts we invest our experimentation effort on. The model in the work of Krause and Ong [2011] is particularly of interest. They use a GP to jointly model correlated contexts and propose a general structure for the joint GP's kernel. We adopt a similar strategy, however, we observe that their model fails to capture differences in length-scale between contexts, which we argue is crucial for the offline contextual Bayesian optimization problem.

A similar contextual optimization problem shows up in reinforcement learning (RL). While the common RL setup has contexts delivered solely by the environment, there is some work on actively choosing contexts [Fabisch and Metzen, 2014, Fabisch et al., 2015]. This work proposes methods for approximating the expected improvement (EI) in the overall objective. Similarly, the objective can be written in terms of entropy and experiments may be chosen in terms of its expected improvement [Metzen, 2015, Swersky et al., 2013]. In our empirical study, we compare to expected improvement for task and action selection.

The works that are most similar to ours is that of Ginsbourger et al. [2014], Pearce and Branke [2017], and Pearce and Branke [2018]. In particular, the algorithms of Ginsbourger et al. [2014] and Pearce and Branke [2017] are based on expected improvement (EI), and we show that our posterior sampling algorithm provides a competitive, theoretically-sound alternative to EI-based methods. Crucially, in all of these works, experiments are done using a stationary kernel. We claim that this only works well if the reward structure of each task is similar; otherwise, methods that try to pick tasks intelligently may end up doing more harm than good.

2.3 Thompson Sampling for Multi-Task Optimization

2.3.1 Preliminaries

For the following, let $\mathcal{X}$ be the finite collection of tasks and let $\mathcal{A}$ be the compact set of possible actions. For simplicity, we will assume that the same set of actions is available for each task, although this assumption is not necessary. Let $f : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ be the reward function, where $f(x, a)$ is the reward for performing action a in task x. It is assumed that this reward function is always bounded. Let $\hat{h} : \mathcal{X} \to \mathcal{A}$ be our estimated mapping from task to action. Our goal is then to find such an $\hat{h}$ which maximizes the following objective:

$$\sum_{x \in \mathcal{X}} f\left(x, \hat{h}(x)\right) \omega(x) \tag{2.1}$$

where $\omega(x) \geq 0$ is some weighting on x that may depend on the probability of seeing x at evaluation time or the importance of x. At round t of optimization, we pick a task x_t and an action a_t to perform a query (x_t, a_t) and observe a noisy estimate of the function $y_t = f(x_t, a_t) + \epsilon_t$, where $\epsilon_t \sim N(0, \sigma_\epsilon^2)$ and is iid. Let D_t be the sequence of queried tasks, actions, and rewards up to time t, i.e. $D_t = \{(x_1, a_1, y_1,), \ldots, (x_t, a_t, y_t)\}$. Additionally, define $\hat{y}_t(x)$ to be the best reward observed for task x up to time t, $\hat{a}_t(x)$ to be the action made to see this corresponding reward, and $\mathcal{A}_t(x)$ to be the set of all actions made for task x up to time t.

In this work, we assume that f is drawn from a *Gaussian process* prior. A GP is characterized by its mean function, $\mu(\cdot)$ and kernel (or covariance) function $\sigma(\cdot, \cdot)$. Then for any finite set of variables, $z_1, \ldots, z_n \in \mathcal{X} \times \mathcal{A}$, $[f(z_1), \ldots, f(z_n)]^T \sim N(m, \Sigma)$, where $m \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n \times n}$, $m_i = \mu(z_i)$, and $\Sigma_{i,j} = \sigma(z_i, z_j)$. It is important to note that by selecting different kernel functions we make implicit assumptions about the smoothness of f. A valuable property of the GP is that its posterior is simple to compute. We denote μ_t and σ_t to be the posterior mean and posterior kernel functions after seeing t evaluations. For more information about GPs see Rasmussen and Williams [2005].

2.3.2 Multi-Task Thompson Sampling

We now describe our proposed algorithm called Multi-Task Thompson Sampling (MTS), which is presented in Algorithm 1 for the case in which tasks are correlated. The algorithm, simply put, is to act optimally with respect to samples drawn from the posterior. That is, at every round a sample for the reward function is drawn, and this sample is used as if it was ground truth to identify the task in which the most improvement can be made. After doing this for T iterations, we return the estimated mapping $\hat{h}$ such that $\hat{h}(x) = \hat{a}_T(x)$ if an evaluation was made for task x; otherwise, $\hat{h}(x)$ maps to an $a \in \mathcal{A}$ drawn uniformly at random. Note that when tasks are assumed to be independent, Algorithm 1 can be modified by instead using a separate GP prior for each task and drawing samples for each one at every iteration. Furthermore, for the noiseless case, note that one can simplify the algorithm by considering improvement over $\hat{y}_t(x)$ instead of $\max_{a \in \mathcal{A}_t(x)} \widetilde{f}(x, a)$ (we make this simplification in our experiments).

One benefit of this algorithm is that it comes with theoretic guarantees. For the following, define $a_t^*(x)$ to be the past action played for task x that yields the largest expected reward. That is,

$$a_t^*(x) := \begin{cases} \underset{a \in \mathcal{A}_t(x)}{\text{argmax}} f(x, a) & \mathcal{A}_t(x) \neq \emptyset \\ \underset{a \in \mathcal{A}}{\text{argmin}} f(x, a) & \text{else} \end{cases}$$

Algorithm 1 Multi-Task Thompson Sampling (MTS)

```
**Input:** capital $T$, initial capital $t_{init}$, mean function $\mu$, kernel function $\sigma$.
Do random search on tasks in round-robin fashion until $t_{init}$ evaluations are expended.
**for** $t = t_{init} + 1$ **to** $T$ **do**
    Draw $\widetilde{f} \sim GP(\mu, \sigma) | D_{t-1}$.
    Set $x_t = \underset{x \in \mathcal{X}}{\text{argmax}} \left[ \left( \max_{a \in \mathcal{A}} \widetilde{f}(x, a) - \max_{a \in \mathcal{A}_t(x)} \widetilde{f}(x, a) \right) \omega(x) \right]$.
    Set $a_t = \underset{a \in \mathcal{A}}{\text{argmax}} \widetilde{f}(x_t, a)$.
    Observe $y_t = f(x_t, a_t)$.
    Update $D_t = D_{t-1} \cup \{(x_t, a_t, y_t)\}$.
**end for**
**Output:** $\hat{h}$
```

Theorem 1. *Define the maximum information gain to be $\gamma_T := \max_{D_T} I(D_T; f)$, where $I(\cdot;\cdot)$ is the Shannon mutual information. Assume that $\mathcal{X}$ and $\mathcal{A}$ are finite. Then if Algorithm 1 is played for T rounds where $t_{init} = 0$,*

$$\mathbb{E}\left[\frac{\sum_{x \in \mathcal{X}} \omega(x) \left(\max_{a \in \mathcal{A}} f(x, a) - f(x, a_T^*(x))\right)}{\sum_{x \in \mathcal{X}} \omega(x) \left(\max_{a \in \mathcal{A}} f(x, a) - \min_{a \in \mathcal{A}} f(x, a)\right)}\right] \leq |\mathcal{X}| \left(\frac{1}{T} + \sqrt{\frac{|\mathcal{X}||\mathcal{A}|\gamma_T}{2T}}\right)$$

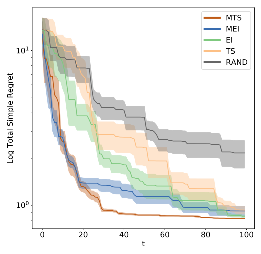
where the expectation is with respect to the data sequence collected and f.

The proof of this theorem (see Section A.1) uses ideas from Kandasamy et al. [2019a]. This result gives a bound on the expected normalized *total simple regret*. Here, *simple regret* is the difference between the best reward and the best reward for a played action (i.e. $\max_{a \in \mathcal{A}} f(x, a) - f(x, a_T^*(x))$), and *total simple regret* refers to the simple regret summed across all tasks. The $\sqrt{|\mathcal{X}||\mathcal{A}|}$ factor in the theorem accounts for the number of actions that can be taken at every step, and the $\sqrt{\gamma_T}$ factor characterizes the complexity of the prior over the tasks. We suspect that our proof technique may have lead to a somewhat loose bound because there is an extra dependence of $|\mathcal{X}|$; that being said, we still get that the rate of decrease is dominated by $\sqrt{\frac{\gamma_T}{T}}$, which is the same as the single-task regret rate [Russo and Van Roy, 2014].

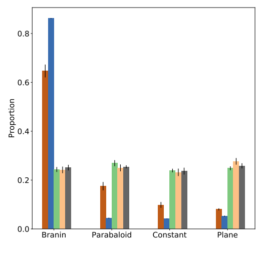
An important implication of this result is that there is no task in which we will have especially bad results, and when $\gamma_T = o(T)$, the normalized simple regret converges to 0 in expectation for every task. We note that Srinivas et al. [2009] give bounds on the maximum information gain for a single GP in a few standard cases. For example, when dealing with a GP over a d-dimensional compact set using an

RBF kernel, $\gamma_T^{(RBF)} = \mathcal{O}(\log(t)^{d+1})$. When tasks are independent and multiple GPs are used, we note that if each GP has an RBF kernel $\gamma_T \leq |\mathcal{X}|\gamma_T^{(RBF)} = o(T)$. Also note that this result can be generalized to infinite action spaces via known techniques [Russo and Van Roy, 2016, Bubeck et al., 2011].

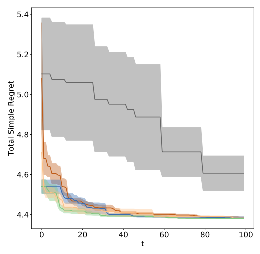
2.3.3 Synthetic Experiments



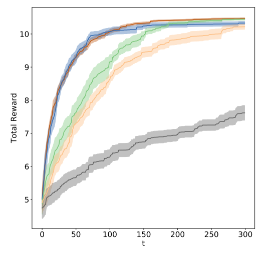
(a)



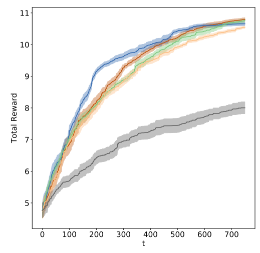
(b)



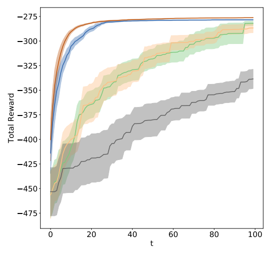
(c)



(d)



(e)



(f)

Figure 2.1: **Synthetic experiments for MTS**. Each of the above show average values and standard error from 10 trials. Where *total reward* refers to (2.1), the plots are as follows: **(a)** total simple regret using a log scale where tasks are Branin-Hoo, parabaloid, plane, and constant; **(b)** proportion of queries made to each task in (a); **(c)** total simple regret for five copies of the Branin-Hoo function; **(d)** total performance for 2D random functions; **(e)** total performance for 4D random functions; **(f)** total performance for 30 correlated functions.

We now present synthetic experiments to demonstrate how MTS can select both tasks and actions intelligently. We compare our algorithm against querying task-action pairs at random, the standard Thompson sampling (TS) and expected improvement (EI) acquisitions where tasks are selected uniformly at random, and an EI variant of MTS that we call *Multi-Task Expected Improvement* (MEI). In MEI, the greatest expected improvement over the best seen reward is found for each task, and the task with the greatest increase is picked. We include this baseline since most approaches we have seen take an EI approach to selecting tasks and are quite similar to MEI [Ginsbourger et al., 2014, Pearce and Branke, 2017, Swersky et al., 2013]. The following experiments are averaged over 10 trials. We start by evaluating each task with 5 points drawn uniformly at random. Each task is modeled by a GP with an RBF kernel, and hyperparameters are tuned for a GP every time an observation is seen for its corresponding task. For two-dimensional functions, hyperparameters are tuned according to marginal likelihood, but for greater dimensions, tuning is done using a blend of marginal likelihood and posterior sampling. This method was shown to be more robust by Kandasamy et al. [2019b]. Lastly, in every experiment we assume that $\omega(x) = 1$ for all $x \in \mathcal{X}$ and that observations are noiseless.

To start, we consider four two-dimensional functions for tasks: the negative Branin-Hoo function [Branin, 1972], a parabaloid, a plane, and a constant. Not only does the Branin-Hoo function have a greater scale than the other functions, but it is also much more complex. Thus, one might imagine that most resources should be invested in optimizing this function, which is the behavior displayed by MEI. However, we see that MTS does best by distributing resources more liberally. Even when all tasks are replaced with five copies of Branin-Hoo, meaning there is no advantage to distributing resources amongst tasks intelligently, MTS still performs competitively. The total simple regret for these experiments, is plotted in Figure 2.1.

We also test these methods on 30 randomly generated functions in two and four dimensions (see Appendix A.2 for details) and on 30 correlated tasks. The correlated tasks are formed by taking 30 equispaced slices of the Branin function, resulting in one-dimensional functions. In these experiments, MTS and MEI are competitive and both enjoy a boost in performance compared to standard EI and TS. That being said, this performance increase is much more substantial for lower dimensional tasks. We hypothesize that, as dimension increases, the gains one gets from choosing tasks in a smart manner diminishes because hyperparameters become harder to estimate, which in turn makes it hard to gauge the difficulty of a task.

2.4 Modeling Variation in Difficulty

The selection of hyperparameters for the kernel function of a GP is often key to whether the landscape can be modeled well. Usually these hyperparameters include *length-scale*, which determines how correlated points are based on their distance to each other, and *scale*, which determines the magnitude of correlation. Intuitively, these values provide some indication of the optimization landscape's difficulty. For example, larger length-scales imply more smooth functions, which are often easier to optimize for. From a more theoretical standpoint, the hyperparameters have a direct effect on the maximum information gain and therefore impact regret bounds shown by Theorem 1 and Russo and Van Roy [2014].

Therefore, it stands to reason that hyperparameters should vary between tasks in order to adequately model any difference in difficulty between them. As such, we propose using a specific form of the Gibbs kernel [Gibbs, 1998]. The Gibbs kernel is a non-stationary variant of the RBF kernel that allows the length-scale and scale to vary over the space. Where $z, z' \in \mathcal{X} \times \mathcal{A}$, $P_X = \dim(\mathcal{X})$ and $P_A = \dim(\mathcal{A})$,

$$\sigma(z, z') = \prod_{p=1}^{P_X+P_A} \left[\sqrt{\frac{2\ell_p(z)\ell_p(z')}{\ell_p^2(z) + \ell_p^2(z')}} \exp\left(\frac{-(z_p - z_p')^2}{\ell_p^2(z) + \ell_p^2(z')} \right) \right] \tag{2.2}$$

Here, ℓ_p is the non-negative *length-scale function* that characterizes the hyperparameters for the p^{th} dimension. The above can be separated into the product of a kernel over the task space and a kernel over the action space, i.e. $\sigma(z, z') = \sigma_X(x, x')\sigma_A(z, z')$ where $z = (x, a)$ and $z' = (x', a')$. To suit our needs, we make all length-scale functions for σ_X constant functions so that $\ell_i(z) = \ell_i$ where $\ell_i \in \mathbb{R}$ and $\ell_i > 0$ for all $i = 1, \ldots, P_X$. As for σ_A, we limit the length-scale functions to only depend on the task-component of z. Altogether,

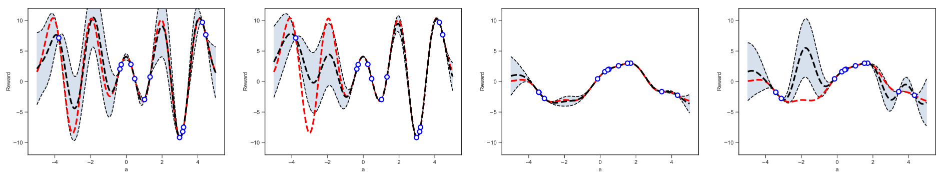
$$\sigma(z, z') = \sigma_X(x, x')\sigma_A(z, z') \tag{2.3}$$

$$= \left(\prod_{i=1}^{P_X} \exp\left(\frac{-(x_i - x_i')^2}{2\ell_i^2} \right) \right) \left(\prod_{j=1}^{P_A} \sqrt{\frac{2\ell_j(x)\ell_j(x')}{\ell_j^2(x) + \ell_j^2(x')}} \exp\left(\frac{-(a_j - a_j')^2}{\ell_j^2(x) + \ell_j^2(x')} \right) \right) \tag{2.4}$$

Note that with this modification σ_X reduces to the RBF kernel. Furthermore, for any fixed task $x \in \mathcal{X}$ (i.e. we only consider $z = (x, a), z' = (x, a')$) the entire kernel reduces to the RBF kernel. As such, we are left with a locally stationary kernel, where the hyperparameters only vary as the task varies. In the proceeding section, we suggest a form for the length-scale functions in σ_A and pair this model with our posterior sampling methods.

Synthetic Example

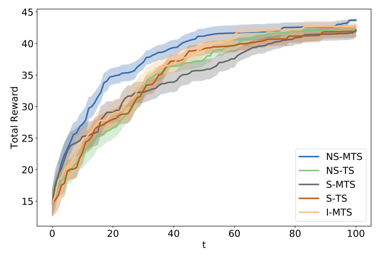
For the correlated task experiment in Section 2.3.3, tasks are generally quite similar, so MTS and MEI can do well when the GP uses an RBF kernel. However, we now wish to optimize 10 correlated tasks of variable difficulty. To create the tasks, we take slices from the function visualized in Figure 2.3 (see Section A.3 in Appendix for details). Like many real-world tasks, this function has areas that make for an interesting optimization problem and others that are quite boring. In order to optimize well, we use the kernel presented in (2.3), where the length-scale function of each action dimension is the soft plus of a quadratic polynomial, and the coefficients of each polynomial are treated as hyperparameters. We form a hierarchical probabilistic model by placing Normal priors over each hyperparameter. Then, for every iteration of our algorithm, we now make decisions according to a posterior sample drawn from this hierarchical model. In practice, this does a superior job at modeling each task. To show this, each of the ten tasks were evaluated at five points. Then, both our suggested model and a stationary model using an RBF kernel was fit to the data. Figure 2.2 visualizes the difference between the models. The difference becomes especially clear when looking at tasks that are relatively flat functions, since the stationary GP estimates there being large peaks. This can be especially damaging in our case where we select tasks based on possible performance improvements.



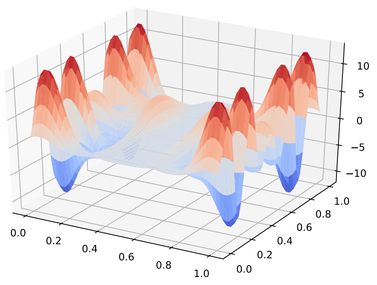
(a) (b) (c) (d)

Figure 2.2: **Comparison between stationary and non-stationary GP fit**. **(a)** shows our proposed model on a hard task, **(b)** shows a stationary model on the same hard task, **(c)** shows our proposed model on an easy task, and **(d)** shows a stationary model on the same easy task. Here, the red line shows the true function, the black line shows the posterior mean, the blue points show evaluations made for the corresponding task, and the shaded area shows high confidence regions.

In the following, we run optimization using MTS and standard Thompson sampling where tasks are picked uniformly at random. Moreover, we run these algorithms using the model described above, using a single GP that jointly models tasks using an RBF kernel, and using several GPs, each corresponding to a task and using an RBF kernel (i.e. assume that tasks have no correlation). In all cases, we use posterior sampling to select hyperparameters. For simplicity, we will append prefixes to these methods where "I" stands for independent GPs, "S" stands for stationary GP, and "NS" stands for non-stationary GP. The results in Figure 2.3 show that one can be negatively affected by picking tasks given an ill-suited model. That is, S-MTS performs worse than both S-TS and I-MTS, showing it is better to either forego shared information to better model the function or distribute resources to tasks uniformly. That being said, disregarding both shared information and picking tasks intelligently, as in I-TS, results in the worst performance (not pictured here). Notice that when tasks can be modeled appropriately, distributing resources according to our algorithm is again beneficial as shown by NS-MTS.



(a)



(b)

Figure 2.3: **Performance with taska of variable difficulty.** **(a)**: Average best seen rewards summed across all tasks of varying difficulty. Each curve is averaged over 12 trials, and the shaded region shows the standard error. **(b)**: Surface of the function used to generate correlated tasks.

2.5 Application to Nuclear Fusion

In this section, we demonstrate how our methods can be used in a novel nuclear fusion application. Most recent methods for nuclear fusion heat up isotopes of hydrogen to temperatures of hundreds of millions of degrees. At this point, the nuclei of two nearby atoms may overcome electrostatic repulsion force between them to form a single nucleus and release energy in the process. At such high temperatures, the atoms are in a plasma state, and any instability in the plasma can give rise to events called ”disruptions”. If this happens, the plasma is lost rapidly and the nuclear reaction is halted. It is remains an open problem for how the tokamak controls should be modified to address the varying state of the plasma in order to sustain the fusion reaction. We tackle this problem by attempting to learn optimal controls offline via a simulator. In particular, we apply our algorithm to determine a mapping from plasma state to tokamak neutral beam controls that could be used as a look-up table during run time. A version of this work that fleshes this application out in more detail can be found in Chung et al. [2020].

2.5.1 Tokamak Control Optimization

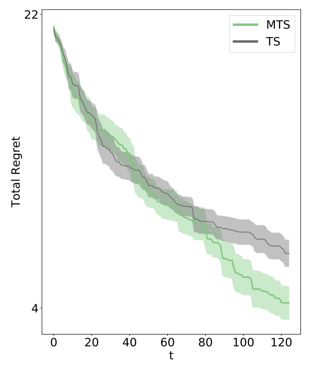
We consider a collection of 8 independent tasks that represent different plasma states. An evaluation of an action on a task corresponds to setting the tokamak beam controls and conducting a simulation on the selected state of the plasma. These simulations are run on TRANSP, which is a predictive simulator of tokamaks. The output is a reward measure, which we designate as a weighted sum of plasma stability and fusion reaction efficiency. We limit the control space to two dimensions: power coefficient of co-current beams and counter-current beams, each with domain [0.001, 1.0]. Section A.4 in the Appendix details the physics background and tools of this experiment.

The performance of MTS was compared against standard Thompson sampling under the same experimental settings as the two-dimensional synthetic experiments in Section 2.3.3, except for 2 differences: each trial consists of 125 evaluations, and we allow up to 20 evaluations to be run in parallel. We rely on parallel optimization here since each query has high simulation overhead ($>$ 1 hour per simulation experiment). For more details regarding the setup for the fusion simulation experiments, see Section A.5 in the Appendix.

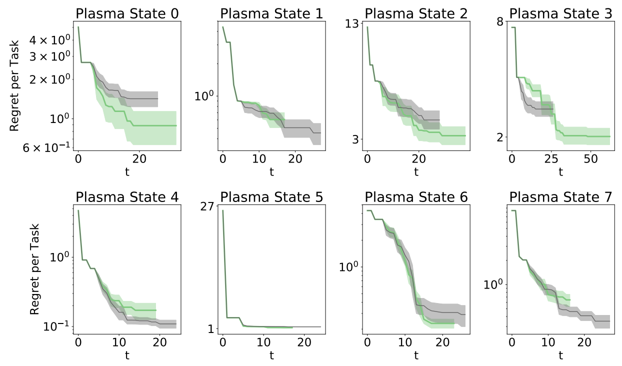
Over 125 evaluations, we observe that MTS yields better results than TS. Because the optimization allows up to 20 parallel evaluations to be run, the first few batches of 20 queries are made with little information. Hence, MTS appears to make uninformed task choices initially and lags behind TS. However, once MTS accumulates sufficient informed queries, it enjoys a boost in performance, as it concentrates on the tasks that are predicted to provide most improvement. This is evident in the increase in performance slope around $t = 20$ in Figure 2.4 (a). This behavior can also be seen in the performance and query plots per task in Figure 2.4 (b). Once the reward has levelled off in a certain task (e.g. Task 4, 5), MTS stops querying the task and queries other tasks that are predicted to provide improvement, while TS will still query the task as it chooses tasks randomly. These results are profound as well as promising, not only from an algorithmic perspective, but also from a physics perspective. While there have been applications of machine learning techniques in nuclear fusion, they primarily focus on detecting disruptions and plasma instabilities [Cannas et al., 2013, Tang et al., 2016, Montes et al., 2019, Kates-Harbeck et al., 2019]. To the best of our knowledge, our application is one of the first attempts in conducting offline optimization of plasma stability and discharge.

2.6 Discussion

In this chapter, we have proposed methods for dealing with many optimization problems that need to be solved simultaneously. We introduced a posterior sampling



(a)



(b)

Figure 2.4: **Fusion Simulation Experiments.** Each of the above show average values and standard error from 10 trials. **(a)** shows the total reward summed across all tasks and **(b)** reward achieved in each task. Note that curves differ in length for (b) since different amounts of resources were allocated for each task.

approach that has theoretic guarantees and often has dominant performance when compared to methods which do not distributed resources intelligently. This Thompson sampling method pairs nicely with our proposed locally stationary model, and we demonstrated that more sophisticated models are key when functions vary in difficulty. Finally, we used our algorithm to derive real results for nuclear fusion. For the future, we hope to broaden the scope of this application by increasing the plasma states we optimize over and using results to create a closed loop controller. Additionally, we note that in many applications, one may wish to perform MTS with parallel evaluations. From Section 2.5.1, it seems that naively making parallel evaluations without any adjustments may lead to decreased performance. That being said, it may be straightforward to create a more sophisticated parallel variant of the algorithm following the work of Groves et al. [2018] and Kandasamy et al. [2018].

Since this work's publication in 2019, there have been several significant extensions. Of particular note is Li et al. [2022a], which proposes an upper-confidence bound based algorithm that is able to learn the optimal policy (not just myopically optimal) and comes with strong theoretical guarantees. Mehta et al. [2023c] then extended this work to the setting in which comparisons are made rather than labels and applied this algorithm to reinforcement learning with human feedback Mehta et al. [2023b]. The majority of these works still rely on standard Gaussian processes and are therefore restricted to low dimensional spaces. Alternative modeling approaches, such as neural processes (see Chapter 6), will be crucial to making these algorithms applicable to higher dimensional problems.

Part II

Learning Policies with Imperfect Surrogate Models

Chapter 3

BATS: Best Action Trajectory Stitching

This chapter is based on Char et al. [2022]:
Char, I., Mehta, V., Villaflor, A., Dolan, J. M., & Schneider, J. (2022). Bats: Best action trajectory stitching. arXiv preprint arXiv:2204.12026.
This work was equal contribution between myself and Viraj Mehta.

We now move on from using expensive simulators to generate data and instead assume that we have access to a cheaper surrogate model that can assist in learning a policy. While this work was developed with a model learned from an offline data source in mind (e.g. a neural network that predicts next states given current states and actions), this need not be the case. The work in this part aims to address issues in policy learning when the surrogate model has errors, which may be a factor when using cheap (non-learned) simulators as well. Indeed, Chapter 4 does not even mention learning a model.

Assuming a learned dynamics model is used, this work falls into the so-called offline model-based reinforcement learning setting. Most works in this setting account for modeling errors by introducing some notion of pessimism [Yu et al., 2020, Kidambi et al., 2020a, Yu et al., 2021]. These mechanisms penalize the policy for going into regions or playing actions which the dynamics models have high uncertainty about. We did not think this mechanism was appropriate for the fusion application, however, since the dynamics models learned are much more accurate for relatively long periods of time, and tuning the penalty coefficient is difficult without access to the real device. Since these methods were developed for the fusion application in mind, they do not continue the pessimism line of research.

We explore two different strategies for learning a policy. The first strategy, which is the subject of this chapter, foregoes deep actor-critic methods and leans more on an offline dataset. The main idea is to view this offline dataset as a tabular Markov Decision Process (MDP), add additional transitions in the MDP via planning with a learned dynamics model, and learn a policy using the policy iteration algorithm. This sidesteps optimism bias problems that often occur in offline RL by assuming a good policy can be made by "stitching" together good sub-trajectories.

3.1 Introduction

The goal of Reinforcement Learning (RL) is to learn a policy which makes optimal actions for a decision making problem or control task. The field of *deep* RL, in which one learns neural network models to represent key quantities for decision making, has recently made great strides [Silver et al., 2016, Mnih et al., 2013, Kober et al., 2013]. In many deep RL algorithms, this involves learning a neural network for both the policy and the value function, which estimates the value of states or state-action-pairs with respect to the current policy. Many promising model-based methods [Chua et al., 2018a, Janner et al., 2019b] also learn a deep dynamics function that estimates next states given current states and actions.

In the standard, *online* setting, the policy is repeatedly deployed in the environment during training time, which provides a continual stream of on-policy data that stabilizes the learning procedure. However, the *online* setting is unreasonable for applications, since it requires a way to cheaply and safely gather a large number of on-policy samples. As such, there has been increasing interest in the so-called *offline* setting [Levine et al., 2020] in which a policy is learned solely from logged off-policy data.

However, the offline setting comes with its own problems. Simply applying deep reinforcement learning algorithms designed for the online setting will often cause exploding value estimates because of distribution mismatch and recursive updates [Kumar et al., 2019]. In model-based methods, the combination of small initial errors and test-time distribution shift often leads to rapidly accumulating model error. While distribution shift and model exploitation are potential issues in online RL, these problems are more severe in the offline setting, as the agent cannot collect additional experience to rectify compounding errors in estimation or planning. To address these problems, offline RL algorithms add constraints to encourage the agent to only operate in the support of the data by either constraining the policy Wu et al. [2019], Kumar et al. [2019] or penalizing uncertain state-actions Yu et al. [2020], Kidambi et al. [2020b], Kumar et al. [2020], Yu et al. [2021].

Rather than trying to implicitly constrain the agent to stay in the support of the data, in this work we explore what happens if we plan over the logged data directly. In particular, we create a tabular MDP by planning short trajectories between states in the dataset, and then we do exact value iteration on this MDP. Unlike other model-based methods which are limited to short imagined trajectories, trajectories from our MDP are mostly comprised of real transitions from the dataset and can therefore be rolled out for much longer with confidence. As such, we argue that our algorithm is able to better reason about the dataset as a whole. In this work, we show that re-imagining the offline dataset in this way allows for the following:

- By coupling together long trajectories with exact value iteration, our algorithm is able to better estimate the resulting policy's value. We prove that under the correct distance metrics our algorithm can be used to form upper and lower bounds for the value function. We demonstrate empirically that this aligns well with the value of a policy behavior cloned on these trajectories.
- By performing full rollouts in our tabular MDP, we are able to approximate our optimal policy's occupancy distribution. We show how many algorithms that uniformly constrain the learned policy to actions on the dataset struggle with "undirected" datasets (i.e., data collected without a specific reward function in mind) and demonstrate that our algorithm avoids this problem by filtering out data unrelated to the task.

3.2 Preliminaries

In this work, we assume the environment can be represented as a *deterministic*, infinite horizon MDP $M = \langle \mathcal{S}, \mathcal{A}, \gamma, T, r, \rho \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\gamma \in (0, 1)$ is the discount factor, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and ρ is the initial state distribution. We refer to an MDP as *tabular* if it has a finite state and action space. While we assume that the true environment in question has every action available to be played at every state (i.e. $T(s, a)$ is well-defined $\forall s \in \mathcal{S}, a \in \mathcal{A}$), later in this work we also consider MDPs that have only a subset of actions for each state. When this is the case, we denote $\mathcal{A}^s \subset \mathcal{A}$ to be the actions available to be played at state $s \in \mathcal{S}$. Such MDPs are defined as $M = \langle \mathcal{S}, \{\mathcal{A}^s\}_{s \in \mathcal{S}}, \gamma, T, r, \rho \rangle$.

In reinforcement learning, we attempt to learn a stochastic policy $\pi(a|s) : \mathcal{S} \to P(\mathcal{A})$, where $P(\mathcal{A})$ is the set of all distributions over $\mathcal{A}$. We desire our learned policy to maximize the expected discounted sum of rewards, $\mathbb{E}_{\pi,\rho}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$, where $s_t = T(s_{t-1}, a_{t-1})$, $a_t \sim \pi(\cdot|s_t)$, and $s_0 \sim \rho$. To facilitate the optimization of this quantity, we can define an optimal state-action value function $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that satisfies the following recurrence relation known as the

Bellman Equation:

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'} Q^*(T(s,a),a') \tag{3.1}$$

Then, we can reformulate policy optimization as trying to solve $\pi(s) = \arg\max_a Q^*(s,a)$, $\forall s \in \mathcal{S}$, where we can estimate Q^* by iteratively performing the bellman update to $Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \max_{a'} Q_k(T(s,a),a')$. In tabular MDPs, the Q-function and its updates can be written and performed exactly. Thus, this procedure – known as *value iteration* – will eventually converge, i.e. $Q_k(s,a) \to Q^*(s,a)$, $\forall (s,a) \in \mathcal{S} \times \mathcal{A}$ as $k \to \infty$. However, in general MDPs where there is a possibly infinite number of states or actions, we must rely on function approximation and finite samples to instead perform *approximate value iteration*, which is not guaranteed to converge. For notational convenience, we denote the value of a state as $V^*(s) = \max_a Q^*(s,a)$. Policy π's *occupancy distribution* is defined as $\mu_\pi(s) \propto \sum_{i=0}^{\infty} \gamma^i p(s_i = s)$, where $s_i = T(s_{i-1}, a_{i-1})$, $a_{i-1} \sim \pi(\cdot|s_{i-1})$, and $p(s_0) \equiv \rho$. We denote the value function for π as V^π; that is, $V^\pi(s)$ is the expected, cumulative discounted sum of rewards from playing policy π starting from state s. When it is not clear from context, we denote V_M^π as the value for the function π, specifically over MDP M.

In offline reinforcement learning, one assumes access to a fixed set of environment interactions. In this work, we assume that we have access to a dataset, $D = \bigcup_{j \in [N]} \{(s_{ji}, a_{ji}, s'_{ji}, r_{ji})\}_{i=1}^{t_j}$, which is comprised of N trajectories of possibly varying lengths, t_j. For the remainder of Section 3.2, we use $s_{ji}, s'_{ji}, a_{ji}, r_{ji}$ to represent the current state, next state, action played, and reward received for the i^{th} timestep of the j^{th} trajectory. Also note that, if $i < t_j$, then $s_{j(i+1)} = s'_{ji}$.

Structures over the Offline Data. Given a dataset, D, collected in MDP $M = \langle \mathcal{S}, \mathcal{A}, \gamma, T, r, \rho \rangle$, one can construct a tabular MDP that incorporates only the states and actions observed in the dataset. We denote this MDP as $M_0 = \langle \mathcal{S}_0, \{\mathcal{A}_0^s\}_{s \in \mathcal{S}_0}, \gamma, T_0, r_0, \rho_0 \rangle$, where $\mathcal{S} = \cup_{j \in [N], i \in [t_j]} \left(\{s_{ji}\} \cup \{s'_{ji}\} \right)$, $\mathcal{A}_0^s = \{a_{ji} | \forall j \in [N], \forall i \in [t_j] \text{ s.t. } s_{ji} = s\}$, $T_0(s,a) = T(s,a)$, $r_0(s,a) = r(s,a)$, and ρ_0 is a discrete uniform distribution over $\{s_{j0}\}_{j=1}^{M}$.

It will often be beneficial to describe the offline dataset from a graphical perspective. A graph, $G := (V, E)$, is fully characterized by its vertex set, V, and its edge set, E. We note that the notation for the vertex set is overloaded with the value function, but the difference is often clear from context. For any MDP, we can define a corresponding graph that has a vertex set which is the same as the MDP's state space and an edge set which matches the MDP's transition function. For example, the graph corresponding to M_0, $G_0 = (V_0, E_0)$, has vertex set, $V_0 = \mathcal{S}_0$, and edge set, $E_0 = \{(s, T_0(s,a)) | s \in \mathcal{S}_0, a \in \mathcal{A}_0^s\}$. Specific to this paper, we also consider the undirected, *neighbor graph*, G_ϵ, which has the same vertex set, but has edge set such that $\{s, s'\}$ is an edge iff $\|s - s'\| \leq \epsilon$ for a specified norm and $\epsilon > 0$.

Bisimulation Metric. In this work, we use the *on-policy bisimulation distance* from Castro [2019], which also gives a sampling-based algorithm for approximating such a metric. We denote this as $d^\pi_\sim(\cdot, \cdot)$. A key result about this metric is the following:

Theorem 2 (Theorem 3 from Castro [2019])**.** *Given states $s, t \in \mathcal{S}$ in an MDP, M, and a policy, π,*

$$|V^\pi(s) - V^\pi(t)| \leq d^\pi_\sim(s, t).$$

In other words, $d^\pi_\sim(\cdot, \cdot)$ is a metric over states for which the value function is 1-Lipschitz continuous. We discuss bisimulation further in Appendix B.2.

3.3 Method

The MDP, M_0, as described in Section 3.2, has several desirable properties. First, it is tabular, so one can easily apply value iteration to find the optimal policy. Second, policies defined over M_0 will be conservative since policies can only choose actions that appear in the dataset. Unfortunately, M_0 is so conservative that it is uninteresting since there is little to no choice in what actions can be made at each state. We must make additions in order to create a more interesting MDP to optimize. Our solution is to create transitions via planning: an operation we call *stitching*.

3.3.1 The Stitching Operation

Broadly speaking, the so-called stitching operation simply adds a transition (or sequence of transitions) from one pre-existing state in an MDP to another pre-existing state via planning in a learned model. To flesh this operation out, suppose that, for a tabular MDP, $\hat{M} = \langle \hat{S}, \{\hat{\mathcal{A}}^s\}_{s \in \hat{\mathcal{S}}}, \gamma, \hat{T}, \hat{r}, \rho_0 \rangle$, we would like to add the ability to transition from $s \in \hat{\mathcal{S}}$ to $s' \in \hat{\mathcal{S}}$. Here, $\hat{M}$ is either M_0, or M_0 with some additional states, actions, and transitions included. Using a learned dynamics model, $\tilde{T}$, as a proxy to the the true dynamics, T, we can find actions that transition from s to s' via *planning*, i.e. we can solve the following optimization problem:

$$\underset{a_0, \ldots, a_{k-1} \in \mathcal{A}}{\text{argmin}} \left\| s' - s_k \right\| \quad \text{where } s_j = \tilde{T}(s_{j-1}, a_{j-1}), \forall j = 1, \ldots, k \quad \text{and } s_0 = s \tag{3.2}$$

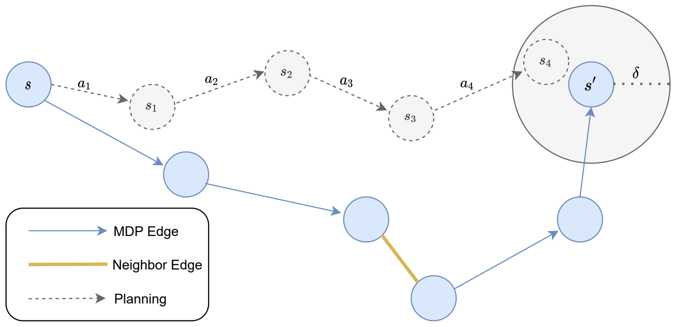


Figure 3.1: **A diagram of the stitching procedure in BATS.** The blue edges come from the directed graph, G_i, the yellow edge comes from the neighbor graph, G_ϵ, and dashed gray edges are the planned actions. Here, the (s, s') state pair is a viable candidate to try to stitch with $k = 4$ actions because there are 4 blue edges and one yellow edge forming a path from s to s'. This would be considered a successful stitch since s_4 is within δ of s'.

where k is the number of actions allowed and $\mathcal{A}$ is the set of actions available in the environment. We choose to optimize this objective with the Cross Entropy Method (CEM), as used in Chua et al. [2018a]. For a specified tolerance, $\delta \in \mathbb{R}$, we consider it possible to transition from s to s' if there exists a solution such that $\|s' - s_k\| < \delta$. If the tolerance cannot be achieved, we leave $\hat{M}$ unchanged. Otherwise, we set $\hat{\mathcal{S}} = \hat{\mathcal{S}} \cup \{s_i\}$, $\hat{\mathcal{A}}^{s_i} = \hat{\mathcal{A}}^{s_i} \cup \{a_i\}$ for $i = 0, \ldots, k-1$, where $\hat{\mathcal{A}}^{s_i} = \emptyset$ if $s_{i-1} \notin \hat{\mathcal{S}}$. If $i < k-1$, we set , $\hat{T}(s_i, a_i) = s_{i+1}$, and otherwise set $\hat{T}(s_{k-1}, a_{k-1}) = s'$. Lastly, $\hat{r}(s_i, a_i) = \tilde{r}(s_i, a_i) - cd(s_k, s')$ for $i = 0, \ldots, k-1$, where $\tilde{r}$ is a learned estimate of the reward function, c is a penalty coefficient, and d is an appropriate distance metric. The addition of the penalty term encourages policies to choose transitions that occur in the dataset over the possibly erroneous ones that are added via stitching. Choosing d to be a bisimulation distance has theoretical ramifications which we will discuss in Section 3.4.

3.3.2 The BATS Algorithm

Given unlimited compute, the ideal algorithm would be to attempt to stitch all pairs of states in the graph and then perform value iteration on the resulting tabular MDP. However this is not often feasible, which is where our algorithm, *Best Action Trajectory Stitching (BATS)*, comes into play. BATS is an iterative algorithm where, for each $i = 0, 1, \ldots, n-1$, we perform value iteration for MDP, M_i, to find optimal policy π_i, we narrow the pool of candidate stitches to those that are both feasible and impactful, and lastly we run the stitching procedure over M_i and set the results as M_{i+1}. We will now discuss the heuristics for identifying feasible and impactful stitches. The full algorithm is written out concretely in Appendix B.1.

Identifying Feasible Stitches. We first establish a notion of *feasible* stitches. We operate on the following heuristic: if there exists a sequence of actions that lead to s' from s, there is likely a similar sequence of actions that lead to s' starting at a state neighboring s. Concretely, for iteration i, we only consider stitching a state, s, to another state, s', if there exists a path from s to s' that uses at most K edges from graph G_i (i.e. the graph corresponding to M_i) and exactly one edge from the nearest neighbor graph G_ϵ (this is visualized in Figure 3.1). If we find that k edges from G_i are used in the path from s to s', we limit the planning procedure in the stitching operation to optimize for k actions. To introduce more conservatism, we also only consider $s' \in \mathcal{S}_0$; that is, we do not consider stitching to any "imagined" states that may be the result of previous stitching. This constraint enforces the agent to stay in distribution.

Identifying Impactful Stitches. To help identify impactful stitches during iteration i, we focus on making stitches that maximizes $\mathbb{E}_{s\sim\mu_{\pi_{i+1}}}\left[V^{\pi_{i+1}}_{M_{i+1}}(s)\right]$. To do this heuristically, we first sample $s_1, \ldots, s_m$ from μ_{π_i} and find all feasible destinations each sample could stitch to. Let s be one such sample, and let s' be a feasible destination state. Suppose there is a path connecting these states that uses exactly k edges from G_i and one edge from G_ϵ. Let s'_k be the state that π_i transitions to after acting k times in M_i. Then, we consider (s, s') to be a candidate stitch if $V^{\pi_i}(s') > V^{\pi_i}(s'_k)$. In other words, if it is believed that s' can be reached from s in k transitions, then the stitch between s and s' only deserves our attention if s' is more valuable than the state that π_i currently transitions to in k steps.

After running the BATS algorithm for n iterations, we are left with an optimal policy, π_n for the stitched, tabular MDP, M_n; however, this policy cannot be deployed on the true, continuous MDP, M, since the domain of π_n is only a subset of $\mathcal{S}$. To remedy this, we collect a large number of trajectories using π_n in M_n to generate a dataset of state-action tuples to train a parametric policy with behavioral cloning. However, we note that alternative policy learning algorithms could be used to make a policy well-defined over $\mathcal{S}$.

Hyperparameters. Our algorithm has a number of hyperparameters of interest. Dynamics model training, metric learning, behavior cloning, and CEM all have parameters which trade off computation and performance. However, these are well-known methods which operate independently and for which hyperparameter selection is a relatively well-understood problem. The BATS algorithm itself requires a tolerance for value iteration, ϵ for the neighbors graph, δ for planning tolerance, m for the number of samples from occupancy distribution per iteration, K for the max number of actions in a stitch, and n for the number of iterations. We further discuss how we determined hyperparameters for our experiments in the Appendix B.4.1.

3.4 Analyzing BATS with Bisimulation Distance

Assumptions Let M_0 be a tabular MDP formed from an offline dataset collected in M, as previously formulated. We can extend π to the domain $\mathcal{S}$ by behavior cloning; that is, by finding the member of a parameterized policy class which has minimum training error regularized with weight norm. For a slight simplification of our analysis, we assume that the hypothesis class is rich enough that it can interpolate the training set; that is, it can reach zero behavior cloning error for all $s \in \hat{\mathcal{S}}$. We often refer to both a policy and its extension as π.

We also assume that on a finite MDP an optimal policy can be found efficiently, and that the learned dynamics model, $\tilde{T}$, is accurate for short transitions in the support of the data. Although in practice, we will learn the reward function, in this analysis we also assume the reward function, r, is known. Lastly, we assume that we are able to learn an embedding, $\phi^\pi : \mathcal{S} \to Z$, such that the $L2$ norm in the latent space, Z, is the on-policy bisimulation metric. That is, we can learn a ϕ^π such that if $||\phi^\pi(s) - \phi^\pi(s')|| < \epsilon$ then $|V^\pi(s) - V^\pi(s')| < \epsilon$.

Sandwich Bound on Value Consider the collection of ℓ tuples $\{(b_j, c_j, a_j)\}_{j=1}^{\ell}$, where $b_j, c_j \in \mathcal{S}_0$, $a_j \in \mathcal{A}$, and $a_j \notin \mathcal{A}_0^{b_j}$ for all $j \in [\ell]$. Then define M^- as the MDP derived by starting from M_0 and, for each $j \in [\ell]$, setting $\mathcal{A}_0^{b_j} = \{a_j\} \cup \mathcal{A}_0^{b_j}$, $T_0(b_j, a_j) = c_j$, and $r_0(b_j, a_j) = r(b_j, a_j) - \gamma\epsilon_j$, where $\epsilon_j > 0$ is some notion of penalty. In other words, M^- is the result of making ℓ stitches in M_0 where $K = 1$. There exists a policy π^- which is the optimal policy on M^- and extends by behavior cloning to $\mathcal{S}$. Similarly, we can construct MDP M^+ in the exact same way as M^-, but by setting reward to $\hat{r}(b_j, a_j) = r(b_j, a_j) + \gamma\epsilon_j$ for each $j \in [\ell]$. In this setting we can bound the value of π^- in the true MDP on either side by its value attained in the finite MDPs just defined. We formalize this notion in the following Theorem.

Theorem 3. *For $j \in [\ell]$, let each penalty term, ϵ_j, be such that $||\phi^{\pi^-}(T(b_i, a_i)) - \phi^{\pi^-}(c_i)|| < \epsilon_j$.*

Then

$$\forall s \in \mathcal{S}_0, V_{M^-}^{\pi^-}(s) \le V_M^{\pi^-}(s) \le V_{M^+}^{\pi^-}(s).$$

In other words, under the correct assumptions, we can construct a pessimistic and optimistic MDP. The value of policy π^- in the pessimistic and optimistic MDP will bound the true value of this policy from below and above, respectively. We give the proof in Appendix B.3 and a short sketch here: The value function can be

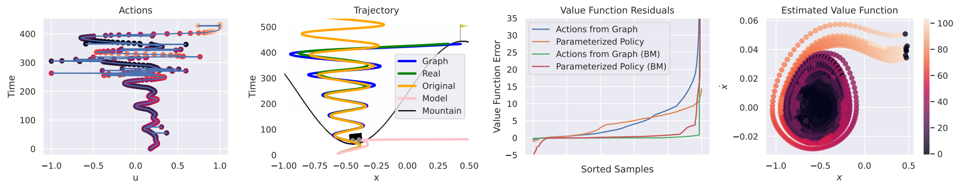


Figure 3.2: **Mountain Car Example**. The plots show example actions, associated trajectories, the value function lower bound, and residuals for the standard dynamics model and the bisimulation metric.

written as $V_M^{\pi^-}(s) = \sum_{s_i} \gamma^i r(s_i, \pi(s_i))$, which can be lower and upper-bounded using Theorem 2 for every transition in the expansion which does not exist in the dataset. This is accomplished by taking into account a pessimistic planning error of the dynamics model. Although Theorem 3 is for the case where we limit stitches to have at most one action, it is likely easy to extend this result to more actions.

There are 3 major implications of the theorem. First, if the behavior cloning, dynamics model, and bisimulation assumptions hold, the value function estimates must be accurate. Second, reasoning by contraposition gives that if the value function estimates are bad, it is due to errors in these components. As such, one should recompute the edge penalties as the policy changes by fine-tuning the bisimulation metric. Third, if the current lower bound is higher than a previous upper bound on the value, the policy is guaranteed to have improved.

We formalize the third fact in the following corollary. Starting with the setup from before, let M'^- and M'^+ be tabular MDPs constructed using the alternative sequence of tuples $\{(b'_j, c'_j, a'_j)\}_{j=1}^{\ell'}$. Let ϵ'_j be the penalty term used in formulating these MDPs, and let π'^- be the optimal policy for M'^-.

Corollary 1. *Let ϵ_j and ϵ'_j satisfy the assumptions of Theorem 3 for mappings ϕ^{π^-} and $\phi^{\pi'^-}$, respectively. If for some $s \in \mathcal{S}_0$, $V_{M^+}^{\pi^-}(s) < V_{M'^-}^{\pi'^-}(s)$, then $V_M^{\pi^-}(s) < V_M^{\pi'^-}(s)$.*

This corollary is a natural consequence of Theorem 3 and implies the policy $\hat{\pi}^-$ is better at state s than π^-. If this holds on average for $s \sim \rho$, then we can conclude that $\hat{\pi}^-$ is the better policy.

3.5 Illustrative Example: Mountain Car

As an initial test of the algorithm and analysis, we trained BATS on a small dataset consisting mostly of random actions along with 5 expert trajectories on a continuous mountain car environment. Behavior cloning fails on this dataset, but running BATS and subsequently behavior cloning trajectories from the stitched MDP solves the problem, reliably getting returns of 95 (90 is considered solved). In Figure 3.2, we show how this happens. Starting at the left, the actions are in general concomitant with a good mountain car policy but with occasional spikes. The spikes result from the planning step in BATS, which has the objective solely to reach the next intended state as accurately as possible. Although the large control inputs are costly in this problem, they are only intermittently added and in general result in a policy that solves the problem.

The second panel makes clear how BATS leverages real transitions to produce realistic, trustworthy trajectories. To show this, we show a stitched trajectory (blue) that was originally (gold) unable to reach the goal. Replaying the actions from this trajectory in the real environment, we find that the trajectory closely matches what happens in the environment (green). However, replaying the same actions in our learned dynamics model results in horrendous error before 100 timesteps (pink). This demonstrates how our method can produce more effective novel long-term trajectories than purely relying on rolling out a learned model.

The third panel shows the error in the graph value function estimates over a sampling of states from the graph from the true returns experienced both by executing the actions associated with the edges taken in the graph and by executing a policy cloned from the graph. We also train a bisimulation model following work from Zhang et al. [2020], Castro [2019] and execute BATS according to this metric (see Appendix B.4.2). We find that doing this results in the value function estimates that are quite accurate. One interesting feature is that on the left side, there are actually states where the cloned policy does *better* than the graph policy. We believe this is likely due to the cloned policy smoothing out some of the control spikes output by the planning process and using less control input. This panel admits a natural decomposition of the errors in BATS. The small errors in executing the bisimulation graph policy (green) show that the dynamics model training and bisimulation metric is likely working here, while the additional errors induced by the corresponding cloned policy (red) show that here, the behavior cloning process induces a slight additional error. We also note that the value function errors are much smaller when the bisimulation metric is used (red / green) than when the Euclidean metric is used (blue / orange), providing empirical evidence for its theoretical benefits. Finally, on the right we see a very sensible looking value function plot, where values are higher as the policy winds its way out of the valley.

3.6 Related Work

Offline Reinforcement Learning In the past two years, there has been substantial interest in offline reinforcement learning and many algorithms have been developed to address the problem. In order to mitigate the issues of distribution shift and model exploitation, recent prior work in offline RL have explored incorporating many different types of conservatism in the optimization process. These approaches can be broadly grouped based on the type of conservatism they incorporate.

The first set of approaches use actor-critic methods [Wu et al., 2019, Kumar et al., 2019, Wang et al., 2020, Nair et al., 2020a], but incorporate policy constraints to limit the difference between the learned policy and the behavioral policy that collected the data in order to mitigate distribution shift during the Bellman updates and at test time. The second set of approaches use model-based RL [Yu et al., 2020, Kidambi et al., 2020b], but leverage uncertainty-aware dynamics models to perform Model-Based Policy Optimization (MBPO) [Janner et al., 2019b] while deterring the policy from taking action with high model uncertainty. The third set of approaches add conservatism directly to the Q-function [Kumar et al., 2020, Yu et al., 2021] in order to optimize a lower-bound on the true value function and avoid extrapolating optimistically. Finally, an alternate approach attempts to filter out the best available transitions in the data for behavior cloning by learning an upper bound on the value function Chen et al. [2020].

Graphical Methods in Reinforcement Learning There have been recent prior works which leverage finite MDPs and their graphical representations in order to estimate value functions. The simplest take the highest-returning actions from a particular state [Blundell et al., 2016] or leverage differentiable memory to give a weighted combination of recent experiences [Pritzel et al., 2017]. Marklund et al. [2020] gives a method of Exact then Approximate Dynamic Programming. The method quantizes the state space into variable-sized bins and treats the quantized MDP as finite. They solve this MDP via Value Iteration and then use these values to warm-start DDQN [Van Hasselt et al., 2016]. This method is close to ours, but assumes discrete action space, quantizes the state space, and does not leverage a dynamics model.

Another method, DeepAveragers [Shrestha et al., 2021], constructs a finite MDP from the dataset and extends the optimal value function via kNN. Their theoretical analysis relies on assumptions on the Lipschitzness of Bellman backups that directly affects the value of a cost hyperparameter, while we use the properties of the bisimulation metric to guarantee our bounds. It also only works on problems with discrete actions and doesn't add to the dataset in any way.

Other methods, like Zhu et al. [2020], Hu et al. [2021], use episodic memory techniques to organize the replay buffer, using averaging techniques and implicit planning to generalize from the replay buffer. However, they cannot plan novel actions to explicitly connect the dataset and are not designed with the offline setting in mind.

3.7 Experiments

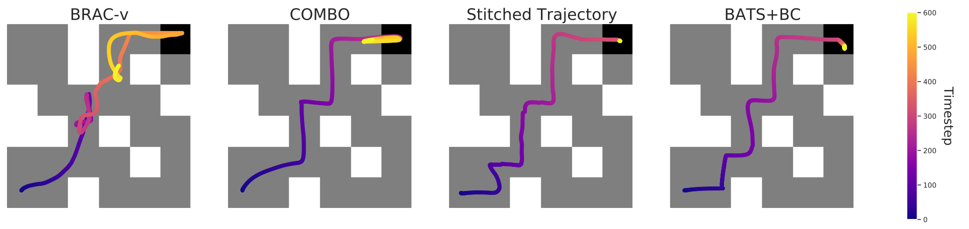


Figure 3.3: **Trajectories in maze2d-medium.** The two leftmost plots show the top baselines on this task. While BRAC-v is able to make it to the goal, it is apparent that the constraints imposed make it sub-optimal as it makes several loops throughout the trajectory. While COMBO is able to efficiently navigate to the goal, it too is unable to remain in the goal region.

Task	BATS+BC (Ours)	BC	SAC	BRAC-v	BEAR	CQL	COMBO
umaze	**141.8** ± **4.4**	3.8	88.2	-16.0	3.4	5.7	76.2 ± 0.5
medium	**133.6** ± **11.6**	30.3	26.1	33.8	29.0	5.0	74.8 ± 35.2
large	107.7 ± 22.0	5.0	-1.9	40.6	4.6	12.5	**150.3** ± **22.0**

Table 3.1: **D4RL Maze 2D.** The above shows undiscounted returns for each algorithm with the higest average bolded. For algorithms that we ran, we include the standard error. Results are averaged over three seeds, except for BATS+BC which was averaged over three seeds on each of the three graphs.

In this section we explore BATS experimentally. By planning in the approximate MDP, BATS can identify high-value sequences of actions. Unlike other state-of-the-art offline RL algorithms, BATS can also reason about which regions of the state space are important for the task at hand. We find that this is crucial for “undirected” datasets (i.e., datasets in which the data were collected without a specific reward function in mind).

In the following experiments we first use BATS to find a good policy in the approximate, finite MDP, and then learn a policy for the true, continuous MDP by behavior cloning on optimal trajectories from the approximate MDP. We also assume that we have access to the start state distribution, and we use this to go through the data and label additional states as start states, since many datasets have few starts. Once BATS is complete, data is collected by unrolling trajectories with the optimal policy. Because we find that not all logged trajectories were stitched to good regions of state space, we filter out any trajectory that does not meet a required value threshold. Then, a neural network policy that outputs the mean and variance of a normal distribution is learned on the collected data.

To implement the algorithm, we rely on the *graph-tool* library. We used compute provided by XSEDE [Towns et al., 2014] to run BATS three times for each task. In order to save on compute, we execute the BATS algorithm with no penalty and with relaxed hyperparameters. After, we perform simple grid search of hyperparameters by relabelling and deleting stitched transitions and re-running value iteration on the resulting MDP. For the penalty term, we use $L2$ distance scaled by a constant hyperparameter. For more details on this procedure see Appendix B.4.1.

For baselines, we compare to methods reported in D4RL: off-policy SAC Haarnoja et al. [2018a], CQL Kumar et al. [2020], BRAC Wu et al. [2019], BEAR Kumar et al. [2019]. Since D4RL does not report any model-based methods, we compare against the COMBO algorithm, which alters the CQL algorithm to include short model rollouts. We used our own implementation of COMBO, which we find gets comparable results to the results reported (see Appendix B.4.3 for details). Final results can be seen in Table 3.1.

2D Maze Environments. We evaluate our procedure on D4RL's *maze2d* environments. While the baseline algorithms we compare against produce amazing results on the Mujoco control tasks, they struggle to match the performance of an expert policy on these relatively simple maze environments. We assert that a key reason for this is that the dataset is undirected. For each of the maze tasks, the dataset was collected by a PD controller that randomly wandered through the mazes. This presents a problem for algorithms which try to constrain the learned policy to be close to the behavioral policy, since there may be many instances where the behavioral policy performs an action that is in direct opposition to a specified goal. We see this concretely in the maze case, where most of the baseline policies are able to find their way to the goal, but start moving away from the goal once it is reached (see Figure 3.3). This happens because the policies are trained on a significant amount of data where the behavioral policy leaves the goal, but there are few to no data where the behavioral policy remains stationary in the goal. Even COMBO, which leverages a learned dynamics model, is unable to stay in the goal cell for the umaze and medium maze tasks.

BATS addresses this problem directly by filtering out any data that are unimportant for the task at hand. Training solely on the data seen by the optimal agent in the approximate MDP ensures that the policy for the continuous MDP is never trained on any disastrous actions or any data that are too far out of the policy's state distribution. At the same time, BATS can use the learned dynamics model to reason about how to stay stationary within the goal cell, as shown by Figure 3.3.

3.8 Conclusion

In this work, we presented an algorithm which eschews learning a deep value function by instead maintaining a finite MDP of transitions from the dataset and augmenting it with transitions planned using a learned model. The algorithm's performance is promising on low-dimensional tasks with theoretical support for its value function estimates. While stitching is hard on higher-dimensional tasks, we believe this work can be improved by incorporating low dimensional representation learning. Ideally this representation should be related to the bisimulation metric. Although we achieved promising results using the model architecture as described in Zhang et al. [2020] on mountain car, we were unable to leverage the same model in our other experiments. We hope that new developments in learning bisimulation metrics will unlock additional potential in BATS. We also hope to formalize the equivalent algorithm for when transitions are stochastic, and we hope that this extension will help the algorithm generalize to more complex problems.

Another challenge of this approach is the computational and memory footprint incurred by constructing MDP. This is especially true as the dataset becomes large and diverse, which is the situation we would hope to do best in with a learning based method for control. In hindsight, although the core ideas behind BATS are good, I do not think that the implementation of these ideas as presented in this chapter will lead to breakthroughs in offline RL. Instead, future work should instead rely upon function approximators that thrive with high amounts of data. For example, generative models have recently shown great progress in offline RL [Janner et al., 2022, 2021, Chen et al., 2021a], and interesting research into how trajectory stitching can better these approaches has already begun [Wu et al., 2024] As mentioned in Wu et al. [2024], the ideas at the core of BATS may be of use in these future research directions.

Chapter 4

PID-Inspired Inductive Biases for Deep Reinforcement Learning in Partially Observable Control Tasks

This chapter is based on Char and Schneider [2023]:
Char, I., & Schneider, J. (2024). PID-Inspired Inductive Biases for Deep Reinforcement Learning in Partially Observable Control Tasks. Advances in Neural Information Processing Systems, 36.

In contrast to the previous chapter, in this chapter we will embrace deep actor-critic methods but instead call into question choices in neural network architectures when faced with a Partially Observable MDP (POMDP). In particular, we find that common architectures that accumulate information over the history of observations are brittle to errors in the surrogate model.

This can be catastrophic for an application such as tokamak control where we know that there are likely key features of the plasma that are not observed. Even in the unlikely case that the true environment is fully-observable, the surrogate model used to train the model offline is usually a POMDP. This is because a good offline training scheme will randomize system parameters if using a simulator (i.e. domain randomization) or have a distribution of possible dynamics functions if the surrogate model is learned (i.e. a learned predictive distribution). This is discussed further in Section 4.4. A policy trained with this perspective in mind should be able to leverage the variability seen during offline training to adapt to what it observes online (see Chen et al. [2021b], Ghosh et al. [2022]). It is therefore vital to explore which neural network architectures can combine information over observations yet remain robust.

4.1 Introduction

Deep reinforcement learning (RL) holds great potential for solving complex tasks through data alone, and there have already been exciting applications of RL in video game playing [Vinyals et al., 2019], language model tuning [OpenAI, 2023], and robotic control [Agarwal et al., 2023]. Despite these successes, there still remain significant challenges in controlling real-world systems that stand in the way of realizing RL's full potential [Dulac-Arnold et al., 2021]. One major hurdle is the issue of partial observability, resulting in a Partially Observable Markov Decision Process (POMDP). In this case, the true state of the system is unknown and the policy must leverage its history of observations. Another hurdle stems from the fact that policies are often trained in an imperfect simulator, which is likely different from the true environment. Combining these two challenges necessitates striking a balance between extracting useful information from the history and avoiding overfitting to modelling error. Therefore, introducing the right inductive biases to the training procedure is crucial.

The use of recurrent network architectures in deep RL for POMDPs was one of the initial proposed solutions [Heess et al., 2015] and remains a prominent approach for control tasks [Meng et al., 2021, Yang and Nguyen, 2021, Ni et al., 2022]. Theses architectures are certainly flexible; however, it is unclear whether they are the best choice for control tasks, especially since they were originally designed with other applications in mind such as natural language processing.

In contrast with deep RL methods, the Proportional-Integral-Derivative (PID) controller remains a cornerstone of modern control systems despite its simplicity and the fact it is over 100 years old [Alpi, 2019, Minorsky, 1922]. PID controllers are single-input single-output (SISO) feedback controllers designed for tracking problems, where the goal is to maintain a signal at a given reference value. The controller adjusts a single actuator based on the weighted sum of three terms: the current error between the signal and its reference, the integral of this error over time, and the temporal derivative of this error. PID controllers are far simpler than recurrent architectures and yet are still able to perform well in SISO tracking problems despite having no model for the system's dynamics. We assert that PID's success teaches us that in many cases only two operations are needed for successful control: summing and differencing.

To investigate this assertion, we conduct experiments on a variety of SISO and multi-input multi-output (MIMO) tracking problems using the same featurizations as a PID controller to encode history. We find that this encoding often achieves superior performance and is significantly more resilient to changes in the dynamics during test time. The biggest shortcoming with this method, however, is that it can only be used for tracking problems. As such, we propose an architecture that is built on the same principles as the PID controller, but is general enough to be applied to arbitrary control problems. Not only does this architecture exhibit similar robustness benefits, but policies trained with it achieve an average of *1.7x better performance* than previous state-of-the-art methods on a suite of locomotion control tasks.

4.2 Preliminaries

The MDP and POMDP We define the discrete time, infinite horizon Markov Decision Process (MDP) to be the tuple $(\mathcal{S}, \mathcal{A}, r, T, T_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, $T_0 \subset \Delta(\mathcal{S})$ is the initial state distribution, and γ is the discount factor. We use $\Delta(\mathcal{S})$ to denote the space of distributions over $\mathcal{S}$. Importantly, the Markov property holds for the transition function, i.e. the distribution over a next state s' depends only on the current state, s, and current action, a. Knowing previous states and actions does not provide any more information. The objective is to learn a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the objective $J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1})\right]$, where $s_0 \sim T_0$, $a_t \sim \pi(s_t)$, and $s_{t+1} \sim T(s_t, a_t)$. When learning a policy, it is often key to learn a corresponding value function, $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which outputs the expected discounted returns after playing action a at state s and then following π afterwards.

In a Partially Observable Markov Decision Process (POMDP), the observations that the policy receives are not the true states of the process. In control this may happen for a variety of reasons such as noisy observations made by sensors, but in this work we specifically focus on the case where aspects of the state space remain unmeasured. In any case, the POMDP is defined as the tuple $(\mathcal{S}, \mathcal{A}, r, T, T_0, \Omega, \mathcal{O}, \gamma)$, where Ω is the space of possible observations, $\mathcal{O} : \mathcal{S} \times \mathcal{A} \to \Delta(\Omega)$ is the conditional distribution of seeing an observation, and the rest of the elements of the tuple remain the same as before. The objective remains the same as the MDP, but now the policy and value functions are not allowed access to the state.

Crucially, the Markov property does not hold for observations in the POMDP. That is, where $o_{1:t+1} := o_1, o_2, \ldots, o_{t+1}$ are observations seen at times 1 through $t+1$, $o_{1:t-1} \not\perp o_{t+1} | o_t, a_t$. A naive solution to this problem is to instead have the policy take in the history of the episode so far. Of course, it is usually infeasible to learn a policy that takes in the entire history for long episodes since the space of possible histories grows exponentially with the length of the episode. Instead, one can encode the information into a more compact representation. In particular, one can use an encoder ϕ which outputs an encoding $z_t = \phi(o_{1:t}, a_{1:t-1}, r_{1:t-1})$ (note that encoders need not always take in the actions and rewards). Then, the policy and Q-value functions are augmented to take in (o_t, z_t) and (o_t, a_t, z_t), respectively.

Tracking Problems and PID Controllers. We first focus on the tracking problem, in which there are a set of signals that we wish to maintain at given reference values. For example, in espresso machines the temperature of the boiler (i.e. the signal) must be maintained at a constant reference temperature, and a controller is used to vary the boiler's on-off time so the temperature is maintained at that value [Marzocco, 2015]. Casting tracking problems as discrete time POMDPs, we let $o_t = \left(x_t^{(1)}, \dots, x_t^{(M)}, \sigma_t^{(1)}, \dots, \sigma_t^{(M)}\right)$ be the observation at time t, where $x_t^{(i)}$ and $\sigma_t^{(i)}$ are the i^{th} signal and corresponding reference value, respectively. The reward at time t is simply the negative error summed across dimensions, i.e. $-\sum_{m=1}^{M} \left| x_t^{(m)} - \sigma_t^{(m)} \right|$.

When dealing with a single-input single-output (SISO) system (with one signal and one actuator that influences the signal), one often uses a Proportional-Integral-Derivative (PID) controller: a feedback controller that is often paired with feedforward control. This controller requires no knowledge of the dynamics, and simply sets the action via a linear combination of three terms: the error (P), the integral of the error (I), and the derivative of the error (D). When comparing other architectures to the PID controller, we will use orange colored text and blue colored text to highlight similarities between the I and D terms, respectively. Concretely, the policy corresponding to a discrete-time PID controller is defined as

$$\pi^{\text{PID}}(o_t) = K_P(x_t^{(1)} - \sigma_t^{(1)}) + K_I \sum_{i=1}^{t} (x_i^{(1)} - \sigma_i^{(1)})dt + K_D \frac{\left(x_t^{(1)} - \sigma_t^{(1)}\right) - \left(x_{t-1}^{(1)} - \sigma_{t-1}^{(1)}\right)}{dt} \tag{4.1}$$

where K_P, K_I, and K_D are scalar values known as gains that must be tuned. PID controllers are designed for SISO control problems, but many real-world systems are multi-input multi-output (MIMO). In the case of MIMO tracking problems, where there are M signals with M corresponding actuators, one can control the system with M separate PID controllers. However, this assumes there is a clear breakdown of which actuator influences which signal. Additionally, there are often interactions between the different signals, which the PID controllers do not account for. Beyond tracking problems, it is less clear how to use PID controllers without substantial engineering efforts.

4.3 Methodology

To motivate the following, consider the task of controlling a tokamak: a toroidal device that magnetically confines plasma and is used for nuclear fusion. Nuclear fusion holds the promise of providing an energy source with few drawbacks and an abundant fuel source. As such, there has recently been a surge of interest in applying machine learning [Abbate et al., 2021, Char et al., 2019], and especially RL [Char et al., 2023a, Mehta et al., 2022, Degrave et al., 2022, Wakatsuki et al., 2021, Seo et al., 2021, 2022, Mehta et al., 2021b], for tokamak control. However, applying deep RL has the same problems as mentioned earlier; the state is partially observable since there are aspects of the plasma's state that cannot be measured in real time, and the policy must be trained before-hand on an imperfect simulator since operation of the actual device is extremely expensive.

How should one choose a historical encoder with these challenges in mind? Previous works [Ni et al., 2022, Melo, 2022] suggest using Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], Gated Recurrent Units [Cho et al., 2014b], or transformers [Vaswani et al., 2017]. These architectures have been shown to be powerful tools in natural language processing, where there exist complicated relationships between words and how they are positioned with respect to each other. However, do the same complex temporal relationships exist in something like tokamak control? The fact that PID controllers have been successfully applied for feedback control on tokamaks suggests this may not be the case [Walker et al., 2000, Hahn et al., 2016]. In reality, the extra flexibility of these architectures may become a hindrance when deployed on the physical device if they overfit to quirks in the simulator.

In this section, we present two historical encoders that we believe have good inductive biases for control. They are inspired by the PID controller in that they only sum and difference in order to combine information throughout time. Following this, in Section 4.5, we empirically show the benefits of these encoders on a number of control tasks including tokamak control.

The PID Encoder. Under the framework of a policy that uses a history encoder, the standard PID controller (4.1) is simply a linear policy with an encoder that outputs the tracking error, the integral of the tracking error, and the derivative of the tracking error. This notion can be extended to MIMO problems and arbitrary policy classes, resulting in the *PID-Encoder* (PIDE). Given input $o_{1:t}$, this encoder outputs a $3M$ dimensional vector consisting of $(x_t^{(m)} - \sigma_t^{(m)})$, $\sum_{i=1}^{t}(x_i^{(m)} - \sigma_i^{(m)})dt$, and $\frac{\left(x_t^{(m)} - \sigma_t^{(m)}\right) - \left(x_{t-1}^{(m)} - \sigma_{t-1}^{(m)}\right)}{dt}$ $\forall m = 1, \ldots, M$. For SISO problems, policies with this encoder have access to the same information as a PID controller. However, for MIMO problems the policy has access to all the information that each PID controller, acting in isolation, would have. Ideally a sophisticated policy would coordinate each actuator setting well.

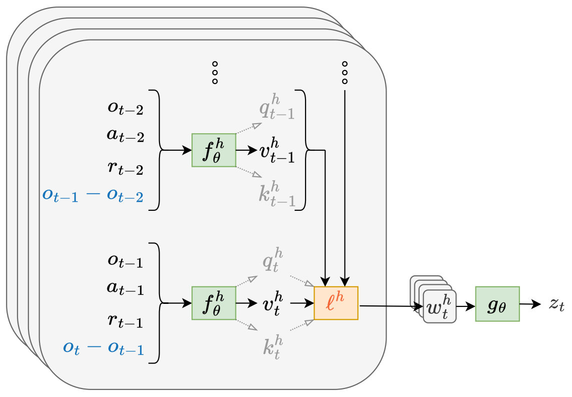


Figure 4.1: **Architecture for GPIDE.** The diagram shows how one encoding, z_t, is formed. Each of the gray, rounded boxes corresponds to one of the heads that makes up GPIDE. Each green box shows a function to be learned from data, and the orange box shows the weighted summation of all previous vectors, $v_{1:t}^h$. We write the difference in observations in blue text to highlight the part of GPIDE that relates to a PID controller's D term. Note that $q_{1:t}^h$ and $k_{1:t}^h$ only play a role in this process if head h uses attention; as such, we write these terms in gray text.

The Generalized PID Encoder. A shortcoming of PIDE is that it is only applicable to tracking problems since it operates over tracking error explicitly. A more general encoder should instead accumulate information over arbitrary features of each observation. With this in mind, we introduce the *Generalized-PID-Encoder* (GPIDE).

GPIDE consists of a number of "heads", each accumulating information about the history in a different manner. When there are H heads, GPIDE forms history encoding, z_t, through the following:

$$\begin{aligned}
v_i^h &= f_\theta^h(\text{concatenate}(o_{i-1}, a_{i-1}, r_{i-1}, o_i - o_{i-1})) && \forall i \in \{1, \dots, t\}, h \in \{1 \dots, H\} \\
w_t^h &= \ell^h(v_{1:t}^h) && \forall h \in \{1 \dots, H\} \\
z_t &= g_\theta(\text{concatenate}(w_t^1, w_t^2, \dots, w_t^h))
\end{aligned}$$

Here, GPIDE is parameterized by θ. For head h, f_θ^h is a linear projection of the previous observation, action, reward, and difference between the current and previous observation to $\mathbb{R}^D$, and ℓ^h is a weighted summation of these projections. g_θ is a decoder which combines all of the information from the heads. A diagram of this process is shown in Figure 4.1. Note that θ is trained along with the policy and Q networks end-to-end with a gradient based optimizer.

Notice that the key aspects of the PID controller are present here. The difference in observations is explicitly taken before the linear projection f_θ^h. We found that this simple method works best for representing differences when the observations are scalar descriptions of the state (e.g. joint positions). Although we do not consider image observations in this work, we imagine a similar technique could be done by taking the differences in image encodings. Like the integral term of the PID, ℓ^h also accumulates information over time. In the following, we consider several possibilities for ℓ^h, and we will refer to these different choices as "head types" throughout this work. We omit the superscript h below for notational convenience.

Summation. Most in line with PID, the projections can be summed, i.e. $\ell(v_{1:t}) = \sum_{i=1}^t v_i$.

Exponential Smoothing. In order to weight recent observations more heavily, exponential smoothing can be used. That is, $\ell(v_{1:t}) = (1-\alpha)^{t-1}v_1 + \sum_{i=2}^t \alpha(1-\alpha)^{t-i}v_i$, where $0 \leq \alpha \leq 1$ is the smoothing parameter. Unlike summation, this head type cannot accumulate information in the same way because it is a convex combination.

Attention. Instead of hard-coding a weighted summation of the projections, this weighting can be learned through attention [Vaswani et al., 2017]. Attention is one of the key components of transformers because of its ability to learn relationships between tokens. To implement this, two additional linear functions should be learned that project concatenate$(o_{i-1}, a_{i-1}, r_{i-1}, o_i - o_{i-1})$ to $\mathbb{R}^D$. These new projections are referred to as they key and query vectors, denoted as k_i and q_i respectively. The softmax between their inner products is then used to form the weighting scheme for $v_{1:t}$. We can rewrite the first two steps of GPIDE for a head that uses attention as

$$v_i, k_i, q_i = f_\theta(\text{concatenate}(o_{i-1}, a_{i-1}, r_{i-1}, o_i - o_{i-1})) \quad \forall i \in \{1, \dots, t\}$$
$$w_{1:t} = \ell(q_{1:t}, k_{1:t}, v_{1:t}) = \text{softmax}\left(\frac{q_{1:t}k_{1:t}^T}{\sqrt{D}}\right) v_{1:t}$$

Here, $q_{1:t}$, $k_{1:t}$, and $v_{1:t}$ are treated as $t \times D$ dimensional matrices. Since it results in a convex combination, attention has the capacity to reproduce exponential smoothing but not summation.

To anchor the GPIDE architecture back to the PID controller, we note that the P, I, and D terms can be formed exactly. At a high level, this is achieved when f_θ^h simply subtracts the target from the state measurement and when using summation and exponential smoothing heads with $\alpha = 1$. We write down the specific instance of GPIDE that results in these terms in Appendix C.1.1. While it is trivial for GPIDE to reconstruct the P, I, and D terms, it is less clear how an LSTM or GRU would achieve this, especially because of the I term. At the same time, GPIDE is much more flexible than the PID-representation since altering f_θ^h results in different representations at each time step and altering the type of head results in different temporal relationships.

4.4 Related Work

A control task may be partially observable for a myriad of reasons including unmeasured state variables [Han et al., 2019, Yang and Nguyen, 2021, Heess et al., 2015], sensor noise[Meng et al., 2021], and unmeasured system parameters [Yu et al., 2017, Packer et al., 2018]. When there are unmeasured system parameters, this is usually framed as a meta-reinforcement learning (MetaRL) [Wang et al., 2016] problem. This is a specific subclass of POMDPs where there is a collection of MDPs, and each episode, an MDP is sampled from this collection. Although these works do consider system parameters varying between episodes, the primary focus of the experiments usually tends to be on the multi-task setting (i.e. different reward functions instead of transition functions) [Zintgraf et al., 2019, Dorfman et al., 2020, Rakelly et al., 2019]. We consider not only differing system parameters but also the presence of unmeasured state variables; therefore, the class of POMDPs considered in this paper is broader than the one studied in MetaRL.

Using recurrent networks has long been an approach for tackling POMDPs [Heess et al., 2015], and is still a common way to do so in a wide variety of settings [Duan et al., 2016, Wang et al., 2016, Vinyals et al., 2019, Ni et al., 2022, Meng et al., 2021, Yang and Nguyen, 2021, Team et al., 2021, Caccia et al., 2022, Agarwal et al., 2023]. Moreover implementations are publicly available both for on-policy [Liang et al., 2017, Hill et al., 2018] and off-policy [Ni et al., 2022, Yang and Nguyen, 2021, Caccia et al., 2022] algorithms, making it an easy pick for those wanting a quick solution. Some works [Igl et al., 2018, Zintgraf et al., 2019, Han et al., 2019, Dorfman et al., 2020, Akuzawa et al., 2021] use recurrent networks to estimate the belief state [Kaelbling et al., 1998], which is a distribution over the agent's true state. However, Ni et al. [2022] recently showed that well-implemented, recurrent versions of SAC [Haarnoja et al., 2018b] and TD3 [Fujimoto et al., 2018] perform competitively with many of these specialized algorithms. In either case, we believe works that estimate the belief state are not in conflict with our own since their architectures can be modified to use GPIDE instead of a recurrent unit.

Beyond recurrent networks, there has been a surge of interest in applying transformers to reinforcement learning [Li et al., 2023]. However, we were unable to find many instances of transformers being used as history encoders in the online setting, perhaps because of their difficulty to train. Parisotto et al. [2020] introduced a new architecture to remedy these difficulties; however, Melo [2022] applied transformers to MetaRL and asserted that careful weight initialization is the only thing needed for stability in training. We note that GPIDE with only attention heads is similar to a single multi-headed self-attention block that appears in many transformer architectures; however, we show that attention is the least important type of head in GPIDE and often hurts performance (see Section 4.5.3).

Perhaps closest to our proposed architecture is PEARL [Rakelly et al., 2019], which does a multiplicative combination of Gaussian distributions corresponding to each state-action-reward tuple. However, their algorithm is designed for the MetaRL setting specifically. Additionally, we note that the idea of summations and averaging has been shown to be powerful in prior works. Specifically, Oliva et al. [2017] introduced the Statistical Recurrent Unit, an alternative architecture to LSTMs and GRUs that leverages moving averages and performs competitively across several supervised learning tasks.

There are many facets of RL where improvements can be made to robustness, and many works focus on altering the training procedure. They use techniques such as optimizing the policy's worst-case performance [Rajeswaran et al., 2016, Jiang et al., 2021] or using variational information bottlenecking (VIB) [Alemi et al., 2016] to limit the information used by the policy [Lu et al., 2020, Igl et al., 2019, Eysenbach et al., 2021]. In contrast, our work specifically focuses on how architecture choices of history encoders affect robustness, but we note our developments can be used in conjunctions with these other directions, possibly resulting in improved robustness. We perform additional experiments that consider VIB in Appendix C.6.1.

Lastly, we note that there is a plethora of work interested in the intersection of reinforcement learning and PID control [Jesawada et al., 2022, Guan and Yamamoto, 2021, Lawrence et al., 2020, Fujii et al., 2021, Wang et al., 2021a, Carlucho et al., 2020]. These works focus on using reinforcement learning to tune the coefficients of PID controllers (often in MIMO settings). We view these as important works on how to improve PID control using reinforcement learning; however, we view our own work as how to improve deep reinforcement learning by leveraging ideas from PID control.

4.5 Experiments

In this section, we experimentally compare PIDE and GPIDE against recurrent and transformer encoders. In particular, we explore the following questions:

- How does the performance of a policy using PIDE or GPIDE do on tracking problems? In addition, how well can policies adapt to different system parameters and how robust to modelling error are they on these problems? (Section 4.5.1)
- Going beyond tracking problems, how well does GPIDE perform on higher dimensional locomotion control tasks (Section 4.5.2)

- How important is each type of head in GPIDE? (Section 4.5.3)

For the following tracking problems we use the Soft Actor Critic (SAC) [Haarnoja et al., 2018b] algorithm with each of the different methods for encoding observation history. Following Ni et al. [2022], we make two separate instantiations of the encoders for the policy and value networks, respectively. Since the tracking problems are relatively simple, we use a small policy network consisting of 1 hidden layer with 24 units; however, we found that we still needed to use a relatively large Q network consisting of 2 hidden layers with 256 units each to solve the problems. All hyperparameters remain fixed across baselines and tracking tasks; only the history encoders change.

For the recurrent encoder, we use a GRU and follow the implementation of Ni et al. [2022] closely. Our transformer encoder closely resembles the GPT2 architecture [Radford et al., 2019], and it also includes positional encodings for the observation history. For GPIDE, we use $H = 6$ heads: one summation head, two attention heads, and three exponential smoothing heads (with $\alpha = 0.25, 0.5, 1.0$). This choice was not optimized, but rather was picked so that all types of heads were included and so that GPIDE has roughly the same amount of parameters as our GRU baseline. As a reference point for these RL methods, we also evaluate the performance of a tuned PID controller. Not only do PID controllers have an incredibly small number of parameters compared to the other RL-based controllers, but the training procedure is also much more straightforward since it can be posed as a black-box optimization over the returns. While there exists many sophisticated extensions of the PID controller (especially in MIMO systems [Boyd et al., 2016]), we only consider the vanilla PID controller since we believe it serves as a good reference point. All methods are built on top of the rlkit library [Pong and Nair, 2018–]. More details about implementations, hyperparameters, and computation can be found in Appendices C.2, C.3, and C.4, respectively. We also include additional experiments regarding variational information bottlenecking (VIB) and lookback size ablations in Appendices C.6.1 and C.6.2. Implementations can be found at `https://github.com/IanChar/GPIDE`.

4.5.1 Tracking Problems

In this subsection we consider a number of tracking problems. For each environment, the observation consists of the current signals, the reference values, and additional information about the last action made. Unless stated otherwise, the reward is as described in Section 4.2. More information about environments can be found in Appendix C.5. To make a fair comparison against PID controls, we choose to only encode the history of observations. For evaluation, we use 100 fixed settings of the environment (each setting consists of targets and system parameters). To avoid overfitting to these 100 settings, we used a separate set of 100 settings and averaged over 3 seeds when developing our methods. We evaluate policies throughout training, but report the average over the last 10% of evaluations as the final returns. We allow each policy to collect one million environment transitions, and all scores are averaged over 5 seeds. Lastly, each table shows scores formed by scaling the returns by the best and worst average returns across all methods in a particular variant of the environment, where scores of 0 and 100 correspond to the worst and best returns respectively.

Mass Spring Damper Tracking The first tracking task is the control of a classic 1D toy physics system in which there is a mass attached to a wall by a spring and damper. The goal is then to apply a force to the mass in order to move it to a given reference location. There are three system parameters to consider here: the mass, spring constant, and damping factor. We also consider the substantially more difficult problem in which there are two masses sandwiched between two walls, and the masses are connected to the walls and each other by springs and dampers (see Appendix C.5.1 for a diagram of this). Overall there are eight system parameters (three spring constants, three damping factors, and two masses) and two actuators (a force applied to each mass). We refer to the first problem as Mass-Spring-Damper (MSD) and the second problem as Double-Mass-Spring-Damper (DMSD).

Additionally, we test how adaptive these policies are by changing system parameters in a MetaRL-type fashion (i.e. for each episode we randomly select system parameters and then fix them for the rest of the episode). Similar to Packer et al. [2018], we train the policies on three versions of the environment: one with no variation in system parameters, one with a small amount of variation, and one with a large amount of variation. We evaluate all policies on the version of the environment with large system parameter variation to test generalization capabilities.

Table 4.1 shows the scores achieved for each of the settings. While GRU and transformers seem to do a good job at encoding history for the MSD environment, both are significantly worse on the more complex DMSD task when compared to our proposed encoders. This is true especially for GRU, which performs worse than two independent PID controllers for every configuration. Additionally, while it seems that GRU can generalize to large amounts of variation in system parameters when a small amount is present, it fails horribly when trained on fixed system parameters. On the other hand, transformers are able to generalize surprisingly well when trained on both fixed system parameters and with small variation. We hypothesize the autoregressive nature of GRU may make it particularly susceptible to overfitting. Comparing PIDE and GPIDE, we see that PIDE tends to shine in the straightforward cases where there is little change in system parameters, whereas GPIDE is able to adapt when there is a large variation in parameters since it has additional capacity.

Environment (Train/Test)	PID Controller	GRU	Transformer	PIDE	GPIDE
MSD Fixed/Fixed	0.00 ± 3.96	83.73 ± 3.48	85.79 ± 1.98	**100.00 ± 0.66**	83.72 ± 2.86
MSD Small/Small	0.00 ± 5.58	**100.00 ± 1.59**	73.27 ± 4.98	75.51 ± 1.31	80.21 ± 8.59
MSD Fixed/Large	36.58 ± 2.86	0.00 ± 3.42	**53.70 ± 1.71**	34.92 ± 0.93	29.55 ± 2.32
MSD Small/Large	43.52 ± 2.82	**87.63 ± 2.28**	81.44 ± 0.82	53.21 ± 1.31	68.03 ± 4.43
MSD Large/Large	45.60 ± 1.71	**100.00 ± 0.61**	92.60 ± 1.49	69.88 ± 0.69	93.03 ± 1.27
Average	25.14	74.27	**77.36**	66.70	70.91
DMSD Fixed/Fixed	24.33 ± 3.97	0.00 ± 8.69	22.05 ± 3.58	**100.00 ± 1.08**	76.23 ± 6.26
DMSD Small/Small	16.17 ± 3.09	0.00 ± 7.79	43.74 ± 3.70	**100.00 ± 0.94**	86.74 ± 3.94
DMSD Fixed/Large	63.59 ± 2.91	0.00 ± 2.28	59.84 ± 1.13	**78.77 ± 1.16**	63.89 ± 2.16
DMSD Small/Large	70.35 ± 1.44	39.26 ± 2.37	73.81 ± 1.60	88.52 ± 0.83	**89.66 ± 1.33**
DMSD Large/Large	78.77 ± 1.97	52.01 ± 2.01	84.45 ± 1.41	86.90 ± 0.18	**100.00 ± 0.91**
Average	50.64	18.25	56.78	**90.84**	83.30
Total Average	37.89	46.26	67.07	**78.77**	77.11

Table 4.1: **Mass Spring Damper Task Results**. The scores presented are averaged over five seeds and we show the standard error for each score.

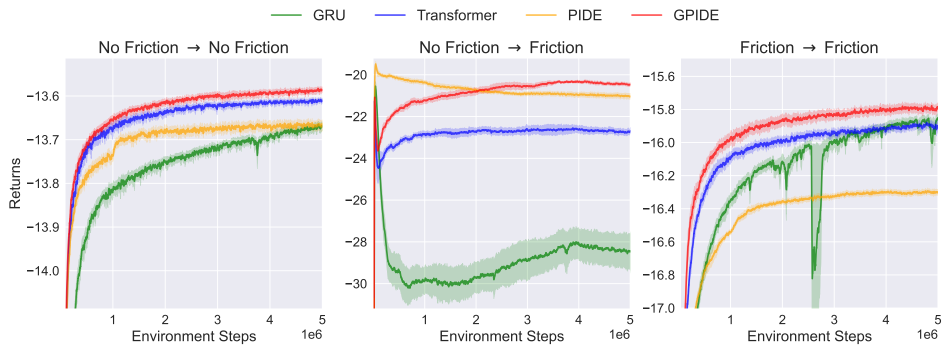


Figure 4.2: **Average Returns for Navigation Environments.** The curves show the average over five seeds and the shaded region shows the standard error. For this plot, we allowed for 5x the normal amount budget to allow all methods to converge. We omit the PID controllers from this plot since it gets substantially worse returns.

Navigation Environment To emulate the setting where the policy is trained on an imperfect simulator, we consider an environment in which the agent is tasked with moving itself across a surface to a specified 2D target as quickly and efficiently as possible. At every point in time, the agent can apply some force to move itself, but a penalty term proportional to the magnitude of the force is subtracted from the reward. Suppose that we have access to a simulator of the environment that is perfect except for the fact that it does not model friction between the agent and the surface. We refer to this simulator and the real environment as the “No Friction“ and “Friction” environment, respectively. In both environments, the mass of the agent is treated as a system parameter that is sampled for each episode; however, the Friction environment has a larger range of masses and also randomly samples the coefficient of friction each episode.

Figure 4.2 shows the average returns recorded during training for both navigation environments and when the policies trained in No Friction are evaluated in Friction. A table of final scores can be found in Appendix C.7.3. One can see that GPIDE not only achieves the best returns in the environments it was trained in, but is also robust when going from the frictionless environment to the one with friction. On the other hand, PIDE has less capacity and therefore cannot achieve the same results; however, it is immediately more robust than the other methods, although it begins to overfit over time. It is also clear that using GRU is less sample efficient and less robust to changes in the test environment.

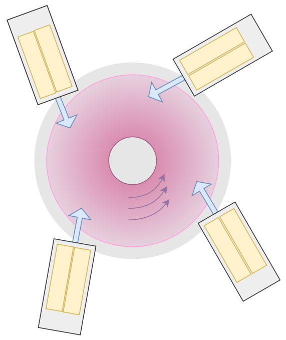


Figure 4.3: **Illustration of DIII-D from Above.** [Char et al., 2023a] Each beamline in the figure contains two independent beams (yellow boxes). The plasma is rotating counter-clockwise and the two beams in the bottom left of the figure are oriented in the counter-current direction, allowing power and torque to be decoupled. This figure gives a rough idea of beam positioning but is not physically accurate.

Tokamak Control For our last tracking experiment we return to tokamak control. In particular, we focus on the DIII-D tokamak, a device operated by General Atomics in San Diego, California. We aim to control two quantities: β_N,

Environment (Train/Test)	PID Controller	GRU	Transformer	PIDE	GPIDE
β_N-Track Sim/Sim	40.69 ± 0.32	$\mathbf{100.00 \pm 0.20}$	97.56 ± 0.19	0.00 ± 1.05	98.33 ± 0.41
β_N-Track Sim/Real	$\mathbf{89.15 \pm 0.99}$	40.96 ± 5.45	40.05 ± 11.91	0.00 ± 21.04	55.21 ± 4.44
β_N-Track Real/Real	98.45 ± 0.77	98.24 ± 0.38	98.74 ± 0.29	$\mathbf{100.00 \pm 0.23}$	99.30 ± 0.64
Average	76.10	79.73	78.79	33.33	**84.28**
β_N-Rot-Track Sim/Sim	0.00 ± 0.83	99.06 ± 0.22	96.22 ± 0.94	67.98 ± 0.50	$\mathbf{100.00 \pm 0.29}$
β_N-Rot-Track Sim/Real	$\mathbf{83.71 \pm 2.64}$	39.76 ± 5.84	33.31 ± 0.69	0.00 ± 8.89	51.00 ± 1.92
β_N-Rot-Track Real/Real	92.02 ± 0.84	98.34 ± 0.52	96.32 ± 0.31	98.21 ± 0.23	$\mathbf{100.00 \pm 0.46}$
Average	58.58	79.05	75.28	55.40	**83.67**
Total Average	67.34	79.39	77.03	44.36	**83.97**

Table 4.2: **Tokamak Control Task Results**. The scores presented are averaged over five seeds and we show the standard error for each score.

the normalized ratio between plasma and mag-
netic pressure, and rotation, i.e. how fast the plasma is spinning around the toroid. These are important quantities to track because β_N serves as an approximate economic indicator and rotation control of the plasma has been suggested to be key for stability [Bardoczi et al., 2021, Tobias et al., 2016, Buttery et al., 2008, Reimerdes et al., 2007, Politzer et al., 2008]. The policy has control over the eight neutral beams [Grierson et al., 2021], which are able to inject power and torque by blasting neutrally charged particles into the plasma. Importantly, two of the eight beams can be oriented in the opposite direction from the others, which decouples the total combined power and torque to some extent (see Figure 4.3).

To emulate the sim-to-real training experience, we create a simulator based on the equations described in Boyer et al. [2019] and Scoville et al. [2007]. This simulator has two major shortcomings: it assumes that certain states of the plasma (e.g. its shape) are fixed for entire episodes, and it assumes that there are no events that cause loss of confinement of the plasma. We make up for part of the former by randomly sampling plasma states each episode. The approximate "real" environment addresses these shortcomings by using a data-driven simulator. This approach to simulating has been shown to be relatively accurate [Char et al., 2023a, Seo et al., 2021, 2022, Abbate et al., 2021], and we use an adapted version of the simulator appearing in Char et al. [2023a] for our work. This simulator accounts for a greater set of the plasma's state, and the additional information is rich enough that loss of confinement events play a role in the dynamics.

We consider two versions of this task: the first is a SISO task where total power is controlled to achieve a β_N target, and the second is a MIMO task where total power and torque is controlled to achieve β_N and rotation targets. The results for both of these tasks are shown in Table 4.2. Most of the RL techniques are able to do well if tested in the same environment they were trained in; the exception of this is PIDE, which curiously is unable to perform well in the simulator environment. While no reinforcement learning method matches the robustness of a PID controller, policies trained with GPIDE fare significantly better.

4.5.2 PyBullet Locomotion Tasks

Moving past tracking problems, we evaluate GPIDE on the PyBullet [Coumans and Bai, 2016–2021] benchmark proposed by Han et al. [2019] and adapted in Ni et al. [2022]. The benchmark has four robots: halfcheetah, hopper, walker, and ant. For each of these, either the current position information or velocity information is hidden from the agent. Except for GPIDE and transformer encoder, we use all of the performance traces given by Ni et al. [2022]. In addition to SAC, they also train using PPO [Schulman et al., 2017], A2C [Mnih et al., 2016], TD3 Fujimoto et al. [2018], and VRM [Han et al., 2019], a variational method that uses

Environment	PPO-GRU	TD3-GRU	VRM	SAC-Transformer	SAC-GPIDE
HalfCheetah-P	27.09 ± 7.85	$\mathbf{85.80 \pm 5.15}$	-107.00 ± 1.39	37.00 ± 9.97	82.63 ± 3.46
Hopper-P	49.00 ± 5.22	84.63 ± 8.33	3.53 ± 1.63	59.54 ± 19.64	$\mathbf{93.27 \pm 13.56}$
Walker-P	1.67 ± 4.39	29.08 ± 9.67	-3.89 ± 1.25	24.89 ± 14.80	$\mathbf{96.61 \pm 1.60}$
Ant-P	39.48 ± 3.74	-36.36 ± 3.35	-36.39 ± 0.17	-10.57 ± 2.34	$\mathbf{66.66 \pm 2.94}$
HalfCheetah-V	19.68 ± 11.71	$\mathbf{59.03 \pm 2.88}$	-80.49 ± 2.97	-41.31 ± 26.15	20.39 ± 29.60
Hopper-V	13.86 ± 4.80	57.43 ± 8.63	10.08 ± 3.51	0.28 ± 8.49	$\mathbf{90.98 \pm 4.28}$
Walker-V	8.12 ± 5.43	-4.63 ± 1.30	-1.80 ± 0.70	-8.21 ± 1.31	$\mathbf{36.90 \pm 16.59}$
Ant-V	1.43 ± 3.26	17.03 ± 6.55	-13.41 ± 0.12	0.81 ± 1.31	$\mathbf{18.03 \pm 5.10}$
Average	20.04	36.50	-28.67	7.80	**63.18**

Table 4.3: **PyBullet Task Results.** Each score is averaged over four seeds and we report the standard errors. Unlike before, we scale the returns by the returns of an oracle policy (i.e. one which sees position and velocity) and a policy which does not encode any history. For the environment names, "P" and "V" denote only position or only velocity in the observation, resepctively.

recurrent units to estimate the belief state. We reproduce as much of the training and evaluation procedure as possible, including using the same hyperparameters in the SAC algorithm and giving the history encoders access to actions and rewards. For more information see Appendix C.3.2. Table 4.3 shows the performance of GPIDE along with a subset of best performing methods (more results can be found in Appendix C.7.5). These results make it clear that GPIDE is powerful in arbitrary control tasks besides tracking since the average score achieved across all tasks is a 73% improvement over TD3-GRU, which we believe is the previous state-of-the-art for this benchmark at the time of this work.

Moreover, GPIDE dominates performance for every robot except HalfCheetah. The only setting where GPIDE achieves significantly worse performance is HalfCheetah-V. This setting seemed to cause stability issues in some seeds, lowering the average score. We believe that these stability issues stem from the attention heads, and we found that removing these heads fixed stability issues and resulted in a competitive average score (see Appendix C.7.5).

	MSD	DMSD	Navigation	β_N Track	β_N-Rot Track	PyBullet
ES	+2.69%	-11.14%	-0.11%	+2.57%	+0.29%	+5.81%
ES + Sum	-8.33%	+5.49%	-1.65%	+4.22%	+0.76%	+11.00%
Attention	-0.36%	-54.95%	-3.91%	-8.85%	-7.55%	-39.44%

Table 4.4: **GPIDE Ablation Percent Difference for Average Scores.** All final scores can be found in Appendix C.7.

4.5.3 GPIDE Ablations

To investigate the role of each type of head, we reran all experiments with three variants of GPIDE: one with six exponential smoothing heads (ES), one with five exponential smoothing heads and one summation head (ES + Sum), and one with six attention heads (see Appendix C.3.3 for details). We choose these three configurations specifically to better understand the roles that attention and summation play.

Table 4.4 shows the differences in the average scores for each environment. The first notable takeaway is that having summation is often important in some of the more complex environments. The other takeaway is that much of the heavy lifting is being done by the exponential smoothing. GPIDE fares far worse when only having attention heads, especially in DMSD and the PyBullet environments.

We visualize some of the attention schemes learned by GPIDE for MSD with small variation and HalfCheetah (Figure 4.4). While the attention scheme learned for MSD could potentially be useful since it recalls information from near the beginning of the episode when the most movement is happening, it appears that the attention scheme for HalfCheetah is simply a poor reproduction of exponential smoothing, making it redundant and suboptimal. In fact, we found this phenomenon to be true across all attention heads and PyBullet tasks. We believe that the periodicity that appears here is due to the oscillatory nature of the problem and lack of positional encoding (although we found including positional encoding degrades performance).

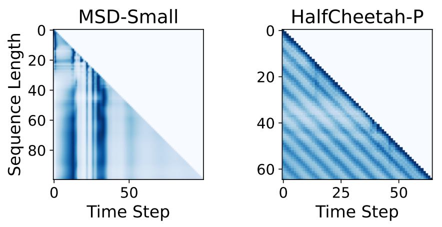


Figure 4.4: **Averaged Attention Schemes for MSD-Small and HalfCheetah-P.** Each y-position on the grid corresponds to an amount of history being recorded, and each x-position corresponds to a time point in that history. As such, each of the left-most points are the oldest observation in the history, and the diagonals correspond to the most recent observation. The darker the blue, the greater the weight that is assigned to that time point.

4.6 Discussion

In this work, we introduced the PIDE and GPIDE history encoders to be used for reinforcement learning in partially observable control tasks. Although both are far simpler than prior methods of encoding, they often result in powerful yet robust controllers. We hope that this work inspires the research community to think about how pre-existing control methods can inform architecture choices.

Limitations There are many different ways a control task may be partially observable, and we do not believe that our proposed methods are solutions to all of them. For example, we do not think GPIDE is necessarily suited for tasks where the agent needs to remember events (e.g. picking up a key to unlock a door).

As with any bias, the PID-inspired biases that we propose in this work come at the cost of flexibility. For the experiments considered in this work, this trade off is beneficial and results in better policies. However, it is unclear whether this trade off is always worth making. It is possible that in higher dimensional environments or environments with more complex dynamics that having more flexibility is preferable to our proposed architecture.

Lastly, some tasks may require the policy to act on images as observations. We are optimistic that PIDE and GPIDE are still useful architectures in this setting, but we speculate that this is contingent on training an image encoder that is well-suited for these architectures, and we leave this research direction for future work.

Part III

Uncertainty Quantification in Dynamics Modeling

Chapter 5

Correlated Trajectory Uncertainty for Adaptive Sequential Decision Making

This chapter is based on Char et al. [2023b]:
Char, I., Chung, Y., Shah, R., Neiswanger, W., & Schneider, J. (2023, December). Correlated Trajectory Uncertainty for Adaptive Sequential Decision Making. In NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World.
This work was equal contribution between myself and Youngseog Chung.

A large focus in Chapter 4, was training a policy over a distribution of different transition functions. This often took the form of having a distribution over possible system parameters and sampling parameters for each episode. However, in the offline model-based RL case, this could also take the form of sampling a possible transition function from a predictive distribution. Such a distribution would ideally reflect the epistemic uncertainty of the model (i.e. the uncertainty stemming from a lack of data) and cover the true dynamics function. Thus, ensuring that a good uncertainty distribution has been learned and properly sampling from the distribution is of utmost importance.

In this chapter, we start with the observation that, while the uncertainty learned in prior model-based reinforcement learning works is reasonably calibrated, the sampling methods used result in non-smooth transition functions. This is a problem not only because we expect the true transition function to be smooth, but we also find that this can cause miscalibration over time. To remedy this, this chapter discusses a simple post-hoc method that can be used to sample smooth functions from the uncertainty distribution.

5.1 Introduction

One of the great challenges with decision making tasks on real world systems is the fact that data is sparse and acquiring additional data is expensive. In these cases, it is often crucial to make a model of the environment to assist in making decisions. At the same time, limited data means that learned models are erroneous, making it just as important to equip the model with good predictive uncertainties. In the context of learning sequential decision making policies, these uncertainties can prove useful for informing which data to collect for the greatest improvement in policy performance [Mehta et al., 2021b, 2022] or helping the policy identify and avoid regions of state and action space that are uncertain during test time [Yu et al., 2020]. Additionally, assuming that realistic samples of the environment can be drawn, an adaptable policy can be trained that attempts to make optimal decisions for any given possible instance of the environment [Ghosh et al., 2022, Chen et al., 2021b].

In this work, we examine the so-called "probabilistic neural network" (PNN) model that is ubiquitous for learning a transition function in model-based reinforcement learning (MBRL) works. We argue that while PNN models may have good marginal uncertainties, they form a distribution of non-smooth transition functions. Not only are these function samples unrealistic and may hamper adaptability, but we also assert that this leads to poor uncertainty estimates when predicting multiple step trajectories. To address this, we propose a simple sampling method that can be implemented on top of pre-existing models. We evaluate our sampling technique on a number of control environments, including a realistic nuclear fusion task. Not only do smooth transition function samples produce more calibrated uncertainties, but they also lead to better downstream performance for an adaptive policy.

5.2 Method

Preliminaries. In this work, we focus on finding optimal policies for infinite-horizon Markov Decision Processes (MDPs). We define the following MDP, $\mathcal{M} := (\mathcal{S}, \mathcal{A}, r, T, T_0, \gamma)$. $\mathcal{S}$ is the set of states; $\mathcal{A}$ is the set of actions that can be played at any state; $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function over current state, action, and next state; $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function; $T_0 \subset \Delta(\mathcal{S})$ is the initial state distribution; and γ is the discount factor. $\Delta(\mathcal{S})$ and $\Delta(\mathcal{A})$ denotes the class of probability distributions over the state and action space, respectively, and we assume that $\mathcal{S} \subset \mathbb{R}^D$. Also, in this work we constrain the transition function to be deterministic. Our goal is to learn a policy function, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the objective $J(\pi) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t, s_{t+1})\right]$, where $a_t \sim \pi(s_t)$, $s_{t+1} = T(s_t, a_t)$, and the expectation is over randomness in initial state and policy actions. We focus on deep reinforcement learning algorithms in which π is a neural network.

One difficulty that arises with deep reinforcement learning methods is the large number of samples needed to learn a good policy. Model-based reinforcement learning (MBRL) methods alleviates this by also learning a model of the environment, $\hat{T}$. One can then reduce the number of samples needed from the true MDP by supplementing the data with fictitious samples generated using $\hat{T}$ and π. For notational convenience, let $\mathcal{X} := \mathcal{S} \times \mathcal{A}$ be the space of concatenated state-action pairs. A trajectory in the MDP of length H can then be written as $(x_1, x_2, \dots, x_H)$, where $x_t := (s_t, a_t) \in \mathcal{X}$, $s_t = T(x_{t-1})$, $a_t \sim \pi(s_t)$, and $s_1 \sim T_0$. Instead of learning the transition to the next state $x_t \to s_{t+1}$, in practice one usually learns the state delta: $x_t \to s_{t+1} - s_t$, such that the learned model $f_\theta : \mathcal{X} \to \Delta(\mathbb{R}^D)$ can predict the next state as $\hat{T}(x_t) = s_t + f_\theta(x_t)$, where θ is the parameters of the model. One can then use the learned model to "rollout" a fictitious trajectory $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_H)$, where $\hat{x}_t = (\hat{s}_t, \hat{a}_t)$, $\hat{s}_t \sim \hat{T}(\hat{x}_{t-1})$, $\hat{a}_t \sim \pi(\hat{s}_t)$, and $\hat{s}_1 \sim T_0$. As a last piece of notational convenience, we use superscripts to denote particular dimensions of vectors. For example, $s_t^{(d)}$ is the d^{th} dimension of s_t and $f_\theta^{(d)}(x)$ is the d^{th} dimension of the model's output.

5.2.1 The "Probabilistic Neural Network"

One of the first works to emphasize the importance of uncertainty in MBRL was Chua et al. [2018b]. To model the transition function, this work proposes using an ensemble of so-called "probabilistic neural networks" (PNN), which are models that output predictive distributions instead of point predictions. In particular, they propose learning a PNN that outputs a mean vector and diagonal covariance matrix which parameterize a multivariate Gaussian distribution. In the context of our work, when f_θ is such a PNN, we write $f_\theta(x) = (\mu_\theta(x), \sigma_\theta(x))$, where $\mu_\theta : \mathcal{X} \to \mathbb{R}^D$ and $\sigma_\theta : \mathcal{X} \to \mathbb{R}^D$ are the mean and standard deviation functions respectively. We will always assume this form of predictive

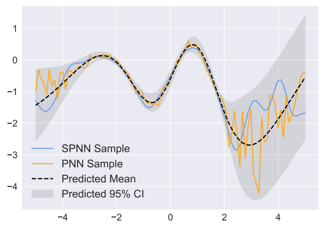


Figure 5.1: **A visual example of function samples**. The vanilla PNN sample can be seen in orange and a sample from our proposed SPNN method can be seen in blue. Crucially both samples stem from the same mean and standard deviation predictions.

distribution when referring to a PNN. Since this
work, PNN's have been used in many MBRL works, including works from both the online [Janner et al., 2019a] and offline [Yu et al., 2020, Chen et al., 2021b, Yu et al., 2021] settings.

In their paper, Chua et al. [2018b] motivate the ensemble of PNNs by claiming that the predicted variance of a PNN, σ_θ, captures aleatoric uncertainty (i.e. the randomness inherent in the true dynamics of the MDP) and ensembling accounts for epistemic uncertainty (i.e. estimation uncertainty stemming from a finite number of datapoints from the system). However, we claim that in reality this breakdown is not so clear. For one thing, the main experiments presented in Chua et al. [2018b] as well as subsequent works that leverage this architecture [Janner et al., 2019a, Yu et al., 2020, Chen et al., 2021b] test their methods in environments with no aleatoric uncertainty. Despite this absence of aleatoric uncertainty, σ_θ seems to play an important role and is even used for penalization in Yu et al. [2020] and Chen et al. [2021b] to indicate that the policy is leaving the support of the data. It seems that PNNs can play an important role in helping capture epistemic uncertainty, and we demonstrate this empirically in Appendix D.2 with a toy example.

That being said, a PNN can only capture *marginal* uncertainty (i.e. the uncertainty for any single input $x \in \mathcal{X}$), it has no notion of *joint* uncertainty over the input space $\mathcal{X}$. As a result, it is unable to draw smooth samples of the transition function (see Figure 5.1). Why does this matter? Consider predicting a full trajectory and assume that the true dynamics, T, and the predicted mean function, μ_θ, are Lipschitz smooth. Then, if consecutive inputs x_{t-1} and x_t are close (i.e. $\|x_{t-1} - x_t\|$ is small), we expect the residuals $T(x_{t-1}) - \mu_\theta(x_{t-1})$ and $T(x_t) - \mu_\theta(x_t)$ to also be close. In other words, we often expect there to be temporal correlations in the residuals that stem from the smoothness in the dynamics and policy (we empirically show these correlations exist in Appendix D.8). This may be problematic for two reasons: first, these correlations in residuals can lead to miscalibrated uncertainty predictions (see Appendix D.7) and over-confident behavior in downstream policy learning. Second, assuming that an adaptive policy is being trained, these temporal correlations may be key in distinguishing and adapting to different environments.

Learning Residual Correlation An ideal model should therefore model uncertainty jointly over $\mathcal{X}$ space, and as a result, be able to sample smooth transition functions that can be used to generate trajectory predictions. For the following, we fix an output dimension d and a time step in the dynamics rollout t. To motivate our method, we first note that the predictive distribution of a PNN can be rewritten as $\mu_\theta^{(d)}(x_t) + \sigma_\theta^{(d)}(x_t)Z_t$ where all of the stochasticity comes from $Z_t \sim \mathcal{N}(0, 1)$. In the vanilla PNN, $Z_1, \ldots, Z_t$ are i.i.d random variables; however, one can instead learn a joint distribution over these random variables. Intuitively, this joint distribution should be such that, when $\|x_i - x_j\|$ is small, the correlation between Z_i and Z_j is high. To achieve this, for each dimenions d, we leverage a kernel function $\kappa_d : \mathcal{X} \times \mathcal{X} \to [0, 1]$. This function indicates the similarity between two points, and we assume that $\kappa_d(x, x) = 1$ in our work. Then, $Z_1, \ldots, Z_t$ are distributed as a multivariate Gaussian distribution with mean 0 and covariance matrix Σ, where $\Sigma_{i,j} = \kappa_d(x_i, x_j)$. The parameters of the kernel function can then be tuned by optimizing the likelihood of the data via a gradient-based optimizer. Because this technique results in smooth transition function samples, we refer to it as the Smooth Probabilistic Neural Network (SPNN).

There are several important details to note about this method. First, because of our assumption on the kernel function, the marginal distribution remains the same as the vanilla PNN; only the smoothness of the function samples will change. Second, since every collection of $Z_1, \ldots, Z_t$ is a multivariate Gaussian random variable, this implies that we are modelling the function of standardized residuals (i.e. $\frac{T(x) - \mu_\theta(x)}{\sigma_\theta(x)}$) as a Gaussian Process (GP) parameterized by a constant mean function of 0 and kernel function κ_d. Although GPs are often used as a prior over functions, our method is purely frequentist as the GP is fit to the residuals in the dataset by optimizing kernel parameters via maximum likelihood, and a posterior is never computed. That being said, we can take advantage of GP computational efficiencies in our method. As discussed in Rahimi and Recht [2007] and Wilson et al. [2020], function samples from a GP can be approximated as Fourier series. Following this, for every trajectory, we can sample a function $g : \mathcal{X} \to \mathbb{R}^D$ where $g^{(d)}(x) = \sqrt{\frac{2}{B}} \sum_{b=1}^{B} \cos(\phi_{b,d}^T x + \tau_{b,d})$. Here, B is the number of bases in the approximation, $\tau_{b,d} \sim \mathcal{U}(0, 2\pi)$, and $\phi_{b,d} \sim p_{\kappa_d}$ where p_{κ_d} is the spectral density corresponding to kernel κ_d. One can then simply use $g^{(d)}(x_t)$ in place of Z_t. See Appendix D.3 for more details.

5.3 Experiments

We empirically evaluate the impact of drawing smooth transition function samples. We measure these effects in two ways: first, by assessing how smooth samples affect modeling metrics such as likelihood and calibration under a test set, and secondly, by assessing how smooth samples affect the downstream training of policies.

Environments. We test our method on a number of different environments where there exists some safety critical limit that the agent must avoid. We give brief descriptions of each environment here, but more details can be found in Appendix D.4. **(Fusion)** We first consider controlling a tokamak for nuclear fusion, an application that has gained interest in the RL community [Degrave et al., 2022, Char et al., 2023a, Seo et al., 2021, 2022]. In our environment, adapted from Char and Schneider [2023], the goal is to push β_N (the normalized ratio between plasma and magnetic pressure) to be as close to a fixed limit as possible by adjusting the amount of power injected; however, if the limit is exceeded, the episode ends and the agent is given a large negative reward. The static dataset is comprised of five trajectories in which a PID controller was used to hit a static β_N target well below the limit. **(Mountain)** Next, we consider an environment where an agent is tasked with traversing a mountain ridge in a two-dimensional space. Not only can the agent fall off either side of the ridge, but there is also a cliff at the end of the ridge that the agent must get as close as possible to without falling over. To move, the agent has a thruster which can be angled to accelerate the agent in different directions. To form the dataset for this environment, we train an agent on the true environment and collect data at different stages of performance. In total, we consider three variants of datasets which we name Random, Medium, and Expert. Note that Medium is a supser set of Random, and Expert is a super set of Medium. **(Cart Pole)** Finally, we consider the standard environment in which an agent controls a cart with a pole attached and is tasked with balancing the pole while ensuring the cart does not go out of bounds. For this environment, we collect 250 data points with a poor performing policy.

Training For each of the environments listed above, we train an ensemble of five PNNs. Although forming an ensemble by itself does not give the ability to sample smooth transition functions, it can introduce correlation in residuals assuming one chooses and fixes an ensemble member for each trajectory prediction. We found that for all environments (except the mountain ridge environment), each trained PNN was miscalibrated. Thus, we use the Uncertainty Toolbox [Chung et al., 2021a] to recalibrate each member by finding a constant scaling for the standard deviation of each output dimension. To learn a policy, we use the Soft Actor Critic (SAC) [Haarnoja et al., 2018b] algorithm. Following Ni et al. [2021], Chen et al. [2021b], we use a recurrent network for the policy to enable the policy to adapt to different instances of the dynamics. See Appendix D.5 for more details.

Modelling Metric Results We first use uncertainty metrics to evaluate the quality of multi-step trajectory samples generated using our sampling technique. Specifically, we measure the *average calibration of centered prediction intervals at each timestep*. Average calibration (also referred to as probabilistic or quantile calibration) [Gneiting et al., 2007, Kuleshov et al., 2018, Song et al., 2019, Chung et al., 2021b] is a standard metric in uncertainty quantification which measures the average discrepancy between the expected and observed proportion of data covered by a

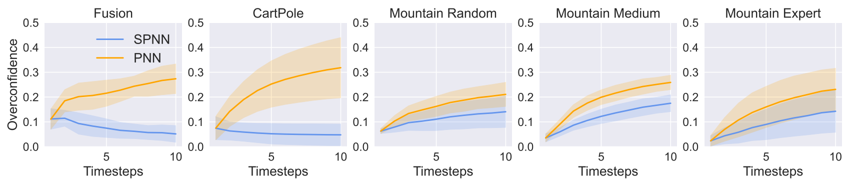


Figure 5.2: **Overconfidence vs. Timestep**. The plots above show the computed overconfidence by timestep in the rollouts when a single PNN or SPNN is used. The solid line shows the mean overconfidence computed over five random seeds, and the shaded region shows the standard error. We use the in-distribution dataset to compute the metrics for the plots shown here.

predictive interval. We also report the component of miscalibration (i.e. error in average calibration) stemming from overconfidence - where prediction intervals are too narrow and the observed proportion within the interval is thus less than the expected proportion. We call this metric the *overconfidence*, and we focus on it since it is often preferable to be underconfident, especially in safety-critical tasks.

To compute these metrics, we start by splitting a test set of data into sub-trajectories of length 10 (a sub-trajectory is a contiguous segment of a full trajectory in the dataset that need not include the start of the trajectory). Following this our evaluation procedure is as follows: first, a number of "replay" samples are drawn for each sub-trajectory (i.e. the model is used to predict the sub-trajectory using the same action sequence), these samples are then used to form centered prediction intervals for each sub-trajectory, and finally miscalibration and overconfidence metrics can be computed for each step of the replays. We compute these metrics for both an in-distribution (ID) dataset, collected using the same policy as the train set, and an out-of-distribution (OOD) dataset, collected using an expert policy. Concrete definitions of metrics and more details on evaluation procedure can be found in Appendix D.6.

Table 5.1 shows the miscalibrations and overconfidences averaged across time steps for each environment. Especially in the single PNN case, it is clear that adding smoothness dramatically decreases the overconfidence in the PNN. This is apparent in Figure 5.2, where it is clear that without smoothness, overconfidence grows much faster over time. We also observe that ensembling often helps with uncertainty, especially when evaluating on OOD datasets. While adding smoothness to the samples can cause miscalibration due to underconfidence, we see that the least overconfident models are the ones which couple ensembling and smoothness.

Policy Performance We now turn to examining the performance of an adaptive policy trained with and without smooth samples. For this section we include an additional baseline: an ensemble of neural networks outputting a point prediction

	Method	Fusion	Cart Pole	Mountain Random	Mountain Medium	Mountain Expert
ID	PNN	0.22 $\pm$ 0.01 (0.22)	0.27 $\pm$ 0.01 (0.24)	0.16 $\pm$ 0.01 (0.16)	0.19 $\pm$ 0.01 (0.18)	0.16 $\pm$ 0.02 (0.15)
	SPNN	**0.10 $\pm$ 0.01** (0.08)	**0.14 $\pm$ 0.02** (0.05)	0.11 $\pm$ 0.01 (0.11)	0.12 $\pm$ 0.01 (0.12)	0.12 $\pm$ 0.01 (0.09)
	PNN Ensemble	0.13 $\pm$ 0.01 (0.03)	0.23 $\pm$ 0.01 (0.02)	0.06 $\pm$ 0.00 (0.04)	0.10 $\pm$ 0.01 (0.08)	**0.10 $\pm$ 0.01** (0.05)
	SPNN Ensemble	0.17 $\pm$ 0.01 (**0.01**)	0.29 $\pm$ 0.01 (**0.00**)	**0.05 $\pm$ 0.00** (**0.02**)	**0.09 $\pm$ 0.00** (**0.05**)	0.11 $\pm$ 0.01 (**0.03**)
OOD	PNN	0.35 $\pm$ 0.02 (0.35)	0.32 $\pm$ 0.01 (0.29)	**0.15 $\pm$ 0.02** (0.08)	0.37 $\pm$ 0.01 (0.37)	0.23 $\pm$ 0.02 (0.23)
	SPNN	0.27 $\pm$ 0.02 (0.26)	0.18 $\pm$ 0.02 (0.13)	0.17 $\pm$ 0.01 (0.05)	0.33 $\pm$ 0.01 (0.33)	0.16 $\pm$ 0.02 (0.15)
	PNN Ensemble	0.18 $\pm$ 0.01 (0.16)	**0.16 $\pm$ 0.01** (0.03)	0.18 $\pm$ 0.01 (0.06)	0.28 $\pm$ 0.00 (0.28)	0.14 $\pm$ 0.02 (0.11)
	SPNN Ensemble	**0.17 $\pm$ 0.01** (**0.14**)	0.19 $\pm$ 0.01 (**0.01**)	0.20 $\pm$ 0.01 (**0.04**)	**0.26 $\pm$ 0.00** (**0.26**)	**0.12 $\pm$ 0.01** (**0.07**)

Table 5.1: **Miscalibrations and Overconfidences.** The table shows the miscalibration averaged over time steps for each method of sampling and environment. In addition we show the standard error over the five seeds and the proportion of the miscalibration due to overconfidence (in parentheses). All of these quantities are rounded to two digits. We bold the lowest mean miscalibration and overconfidence for each of the environments. The top and bottom blocks show the metrics computed over in-distribution (ID) and out-of-distribution (OOD) datasets, respectively.

Method	Fusion	Cart Pole	Mountain Random	Mountain Medium	Mountain Expert	Average
NN	65.95 $\pm$ 1.63	**100 $\pm$ 0.00**	63.57 $\pm$ 9.80	26.79 $\pm$ 0.62	49.73 $\pm$ 2.50	61.21
PNN	65.34 $\pm$ 12.94	99.98 $\pm$ 0.02	64.29 $\pm$ 4.01	23.39 $\pm$ 1.34	44.34 $\pm$ 13.45	59.47
SPNN	**91.46 $\pm$ 2.50**	98.78 $\pm$ 1.22	**65.93 $\pm$ 1.72**	**39.08 $\pm$ 8.24**	**84.72 $\pm$ 4.73**	**75.99**

Table 5.2: **Normalized Policy Returns.** Each of the reported numbers is averaged over the last 20% of recorded evaluation episodes during training and five random seeds. We also report the standard errors from the five seeds, and we bold the result with the highest mean. All numbers are normalized using the performance of a poor and expert policy (i.e. a normalized score of 100 is the same performance as the expert policy).

(we refer to this as NN). Table 5.2 shows the returns achieved by the policy averaged over the last 20% of training steps. For the fusion environment and all configurations of the Mountain Ridge environment, we see that it is beneficial to have smooth samples and that this better sampling prevents the final policy from crossing the β_N limit or falling off a cliff for each respective environment. Interestingly, we see that the medium version of the mountain ridge environment is the most difficult to optimize. We hypothesize this may be because the medium version of the dataset has a greater spread of data so the dynamics are more confident yet do not have enough data to fully model the dynamics. This is therefore a case where it is especially important that there are smooth samples of the dynamics, and we visually show the difference in policy performance in Figure D.19 in the Appendix. Lastly, while there seems to be little difference between the final scores in Cart Pole, in Figure D.15, we show that the returns while training are much more reliable when using smooth dynamics samples. We hypothesize this may be due to the smooth dynamics being more controllable and easier to adapt to.

Discussion. To summarize, in this work we highlighted the existence of correlated errors in dynamics models and how sampling smooth trajectory functions is key to capture this phenomenon in uncertainty estimates. Experimentally, we show that with more intelligent sampling we can achieve less overconfident uncertainty predictions and better performing policies. In future works, we hope to extend our results to more complex and higher dimensional environments.

5.4 Conclusion

After presenting this work at the Workshop on Adaptive Experimental Design and Active Learning in the Real World, we attempted to scale up experiments to higher dimensional problems but ran into difficulties. Ensuring smooth functions were drawn had little impact on the performance and negatively impacted policy learning stability. There may be a variety of reasons for this outcome, e.g. only the MuJoCo D4RL tasks were attempted and perhaps different hyperparameters are needed for our method.

In my opinion, however, I believe that there are two phenomena that affect calibration over time and both were not correctly accounted for. The first, which we explored in this chapter, is the phenomenon of over-confidence over time if transition samples are drawn independently. While over-confidence may not always occur, it is often observed and we theoretical motivated why in Appendix D.7. The second phenomenon, which was not accounted for, is the rapid growth in errors that occurs from performing auto-regression. Intuitively, this would likely cause under-confidence since predictions should spread out rapidly because of increasing errors. Moreover, in high-dimensional environments I would expect this phenomenon to be exacerbated since it is easier for the agent to be pushed out of distribution, which causes the model to make more extreme predictions, which then causes the model to be pushed even further out of distribution. These two phenomena are likely at odds with each other, and my hypothesis is that the second phenomenon is less important in the experiments performed in this chapter but is dominant in high-dimensional environments.

Even if this is the case, one would expect sampling smooth transition functions would be beneficial for training an adaptable policy that uses the history of observations since a distribution of smooth transition functions is a more realistic distribution. However, this did not seem to be the case for high-dimensional tasks. It is possible that the jumps in state space from time step to time step were big enough that correlation in residuals was destroyed (although preliminary analysis showed this correlation did exist). Another possibility is that the procedure laid out by Chen et al. [2021b] does not actually create adaptable policies; perhaps it is just the case that the recurrent architecture leads to a better representation for the policy network. Although I do believe that the framework laid out by Chen et al. [2021b] is worthy of future research pursuits, I do think that further investigation needs to be done into whether these policies are truly adapting to their environment at test-time.

This work assumes that the model predicts a Gaussian distribution and that this Gaussian distribution captures epistemic uncertainty. This is more of a byproduct of the model's training procedure rather than an intentional design choice. Ideally future works would remove this modeling assumption, especially since the original purpose of predicting a Gaussian distribution is to capture aleatoric uncertainty. Perhaps an alternative to our proposed method is to train a very large, diverse ensemble of neural networks. Although the number of possible transition functions remains finite, if the ensemble size is big enough this may not matter. The downside of this approach is both computational expense and that it may require significant hyperparameter tuning to ensure the spread in ensemble members is appropriate. Yet another alternative is to use neural processes, which we explore in the next chapter.

Chapter 6

Graph Transformer Neural Processes

This chapter is based on:
Char, I., Igoe, C. & Schneider J. Graph Transformer Neural Processes *In Submission*, 2024.
Conor Igoe was an equally contributing author for this work.

One challenge with the previous chapter was that the uncertainty estimates relied on several assumptions, the main one being that the Gaussian distribution predicted by the network captured epistemic uncertainty. Aside from this assumption, another assumption that is ubiquitous in the literature regards ensembles. In particular, by training an ensemble one is finding a distribution of neural network parameters that achieve good loss. The assumption is that doing so also means that the distribution of predictions made by the ensemble members is in some sense good (e.g. well-calibrated).

In this chapter, we focus on neural processes. Unlike with ensembles, neural processes can directly learn a distribution in the output space that captures both aleatoric and epistemic uncertainty. The assumption made is instead about the data. In particular, it is assumed that the dataset contains a distribution of different functions to learn from (thus this setting can be thought of as meta-learning uncertainty). In what follows, we explore improvements needed to make neural processes more viable for real-world cases, including how to make neural processes more sample-efficient and more robust to test-time distribution shift.

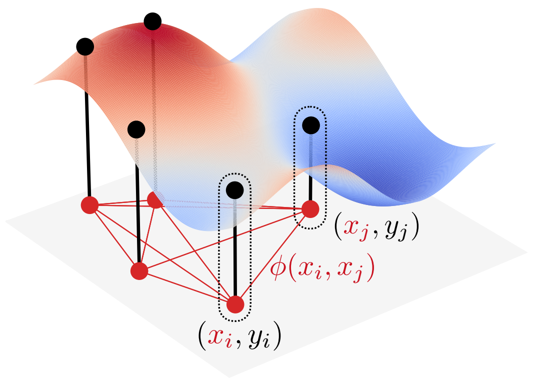


Figure 6.1: **A Visualization of the Data as a Graph**. The diagram depicts how one can construct a graph (in red) from observations from an unknown function (points in black) using a distance or similarity metric ϕ.

6.1 Introduction

When tasked with a decision making problem in the real world, one common approach is to use collected data to learn a surrogate model that can inform action selection. This strategy is not without its challenges, however, since there is uncertainty in the model's predictions due to inherent noise in the real-world system or a lack of data. Forming accurate predictive uncertainties is therefore essential, and indeed, intelligent decision making algorithms take advantage of such uncertainties [Snoek et al., 2012, Chua et al., 2018b, Shyam et al., 2019, Mehta et al., 2022, Li et al., 2022a].

In this work, we learn these uncertainties using a type of model known as a Neural Process (NP) [Garnelo et al., 2018b]. These models are neural networks which take a context set of previous observations as input and predict a distribution for a specified target set. Rather than assuming a prior over possible functions—as is done with a Gaussian Process (GP)—we instead use collected data to meta-learn [Hospedales et al., 2021] these uncertainties. This model class is powerful and has been used for many real-world applications such as neuroscience [Pakman et al., 2020, Cotton et al., 2020], astronomy [Čvorović-Hajdinjak et al., 2022, Park and Choi, 2021, Pondaven et al., 2022], and robotics [Chen et al., 2022, Li et al., 2022b, Yildirim and Ugur, 2022].

Most NP models operate directly on the raw data representations; however, this does not take advantage of invariances that may be in the data, resulting in the sample inefficiency. An important exception to this are Convolutional NPs [Gordon et al., 2019]. These models enforce translational equivariance through so-called "convolutional deep sets". However, these models require forming a grid over the support of the data, which becomes infeasible with high dimensional input spaces.

To remedy this, we advocate for representing the condition and target set as a graph. Depending on the choice of edge labels, this representation can be invariant to translations or rotations, and at the same time, it can be extended to higher dimensional settings. We start this work by first describing how to formulate the context and target set into a graph. We then suggest two architectures that can operate over such a graph: the first is a graph transformer architecture [Shi et al., 2020] and the second is a novel architecture inspired by Gaussian elimination. Through synthetic experiments we show that our proposed models not only achieve strong results, but are dramatically more sample efficient. We then show how these advantages translate to real data experiments on a nuclear fusion application and on a cheminformatics application.

6.2 Method

6.2.1 Preliminaries

Let $\mathcal{X} \subset \mathbb{R}^{D_X}$ and $\mathcal{Y} \subset \mathbb{R}$ be input and output spaces, and let $f : \mathcal{X} \to \Delta(\mathcal{Y})$ be a random function, where $f \sim \mathcal{P}$ and $\Delta(\mathcal{Y})$ is the set of distributions over $\mathcal{Y}$. Sometimes $\mathcal{P}$ is known or assumed (e.g one could use a Gaussian Process prior [Williams and Rasmussen, 2005]); however, it may be the case that one only has access to data produced by the random function. Let $\mathcal{D}$ be such a dataset where

$$\mathcal{D} = \{\{(x_j, y_j \sim f_i(x_j)\}_{j=1}^{N_i}\}_{i=1}^{M}$$

and where $f_i \sim \mathcal{P}$. In other words, $\mathcal{D}$ is a dataset composed of M different function samples, and for the i^{th} sample, there are N_i function evaluations.

This work focuses on learning a deep network that approximates the random function f using the dataset $\mathcal{D}$. Let p_θ be such a network where θ is the set of parameters for the network. Moreover, let p_θ be a so-called *Neural Process* (NP) [Garnelo et al., 2018b]. The hallmark of these models is that they can produce a predictive distribution given an input x and a context set of previous observations.

Concretely, let $\mathbf{x} \in \mathcal{X}^N$ be a collection of N "points" in $\mathcal{X}$. Further, let the first $C \in \mathbb{Z}^+$ of these points, $\mathbf{x}_{1:C}$, belong to the context set, and let the remainder of the points, $\mathbf{x}_{C+1:N}$, be the target set. Along with this, the model also has access to the observations corresponding to the context set, $\mathbf{y}_{1:C} \in \mathcal{Y}^C$. The goal is for the model to predict a distribution for the target observations, $\mathbf{y}_{C+1:N}$, which usually takes the form of a multivariate normal distribution where target points are independent from each other. In summary, the model produces $\mu, \sigma = p_\theta(\mathbf{x}, \mathbf{y}_{1:C})$, where $\mu, \sigma \in \mathbb{R}^{N-C}$ are the mean and standard deviations of the predicted normal distribution. Lastly, for notational convenience, we let $p(\cdot|\mathbf{x}, \mathbf{y}_{1:C})$ be the pdf of the predicted distribution.

For most NPs (including the ones we will propose), the training objective is to maximize the log likelihood. That is, the objective function is

$$J(\theta) = \mathbb{E}_{C,\mathbf{x},\mathbf{y}}\left[\log p_\theta(\mathbf{y}_{C+1:N}|\mathbf{x}, \mathbf{y}_{1:C})\right] \tag{6.1}$$

where context size, **x**, and **y** all have underlying distribution. In the rest of this subsection, we give a brief overview of different members of the NP family that are most salient for our work. For a more thorough review, see Jha et al. [2022].

Latent, Conditional, and Attentive Neural Processes Latent and conditional NPs (LNP[1] [Garnelo et al., 2018b] and CNP [Garnelo et al., 2018a], respectively) were the first proposed NP models. Both of these architectures use a `DeepSet` [Zaheer et al., 2017] over the context set to create an encoding which captures the relevant information of the context set. This encoding is then used, along with $\mathbf{x}_{C+1:N}$, to predict independent normal distributions for the targets. The key difference between the two is that the LNP assumes a latent distribution over the encoding and therefore uses a variational approximation. Kim et al. [2019] improves on these NP architectures by introducing attention [Vaswani et al., 2017] into the encoding scheme. In particular, they replace the `DeepSet` with self-attention and add cross-attention between the context and target sets.

Convolutional Neural Processes All of the NPs discussed up until now work directly on x representations. This is not the case, however, with the Convolutional Conditional NP (ConvCNP) [Gordon et al., 2019]. Instead of using a `DeepSet` architecture, the ConvCNP uses a `ConvDeepSet`, which operates over a grid covering **x**. By operating over this grid rather than x representations, the ConvCNP is translation equivariant and more sample efficient than other NPs. While operating over a grid is natural for image based domains, it has challenges when operating in the regression setting, especially when dealing with high dimensional $\mathcal{X}$. While there have been follow up works introducing new convolutional neural processes [Wang et al., 2021b, Kawano et al., 2021], these methods are still bound to a grid.

Autoregressive Neural Processes and Transformers Most NPs assume conditional independence over the target set given the context set. That is, the output of these networks are the mean and standard deviation parameterizing an independent normal distribution for each $\mathbf{y}_{C+1:N}$. While there are works which predict multivariate normal distributions over the target set [Bruinsma et al., 2021, Markou

[1]In the original paper, this model is simply called a "Neural Process". Following other works, we instead call it a Latent Neural Process to differentiate it from the broad class of models.

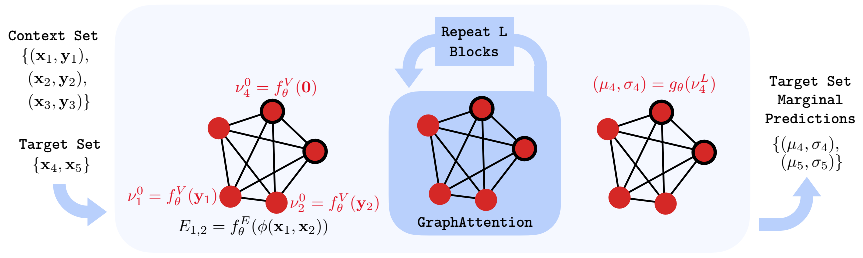

Context Set
$\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), (\mathbf{x}_3, \mathbf{y}_3)\}$
Target Set
$\{\mathbf{x}_4, \mathbf{x}_5\}$
Repeat L Blocks
$\nu_4^0 = f_\theta^V(\mathbf{0})$
$\nu_1^0 = f_\theta^V(\mathbf{y}_1)$
$\nu_2^0 = f_\theta^V(\mathbf{y}_2)$
$E_{1,2} = f_\theta^E(\phi(\mathbf{x}_1, \mathbf{x}_2))$
GraphAttention
$(\mu_4, \sigma_4) = g_\theta(\nu_4^L)$
Target Set Marginal Predictions
$\{(\mu_4, \sigma_4), (\mu_5, \sigma_5)\}$

Figure 6.2: **An Illustration of the GTNP Architecture.** Data in the context and target set are first formed into a graph. The vertex features of the graph are then transformed via a graph transformer network [Shi et al., 2020]. Lastly, mean and standard deviations are predicted using the final vertex features.

et al., 2022, Nguyen and Grover, 2022], recent work has shown that models that are trained assuming conditional independence can predict the joint distribution in an auto-regressive fashion to produce competitive results [Bruinsma et al., 2023, Nguyen and Grover, 2022]. Concretely, one can form the joint distribution using the chain rule:

$$p_\theta(\mathbf{y}_{C+1:N}|\mathbf{x}, \mathbf{y}_{1:C}) = \prod_{i=C}^{N-1} p_\theta(\mathbf{y}_{i+1}|\mathbf{x}_{1:i+1}, \mathbf{y}_{1:i})$$

where $\mathbf{y}_i$ is the i^{th} observation in $\mathbf{y}$. As noted by Bruinsma et al. [2023], using these NPs in "autoregressive mode" results in highly expressive model at the cost of coherence. In this work, we will default to the autoregressive mode for evaluation.

An architecture of particular interest for this work is the transformer [Vaswani et al., 2017]. Previous works have shown that transformers are excellent at meta-learning for reinforcement learning [Melo, 2022], drug discovery [Chen and Bajorath, 2023], and—most related to this work—for approximating Bayesian inference [Müller et al., 2021]. Nguyen and Grover [2022] use the transformer architecture to create the Transformer NP (TNP). This model frames the context and target sets as sequences, and then uses a transformer architecture to directly predict normal distributions for each target. Because our proposed architectures are variants of the transformer, we briefly review self-attention.

Consider M embeddings $z_1, z_2, \ldots, z_N \in \mathbb{R}^d$. We use d to denote the size of embeddings throughout this work. Fix an $i \in [1, N]$. To start, three linear projections of z_i are made. They are known as the key, query, and value vectors and are denoted by $k_i, q_i, v_i \in \mathbb{R}^d$, respectively. We denote the attention operator for the ith embedding as $\texttt{Attention}_i$ and define it as the following:

$$\texttt{Attention}_i(z_1, \dots, z_N) = \sum_{n=1}^{N} \alpha_{i,n} v_n$$

$$\text{where} \quad \alpha_{i,j} = \frac{\exp\left(\frac{1}{\sqrt{d}}\langle q_i, k_j\rangle\right)}{\sum_{n=1}^{N} \exp\left(\frac{1}{\sqrt{d}}\langle q_i, k_n\rangle\right)}$$

In practice, transformers use multi-headed self attention, where the self-attention mechanism is repeated H times, the results are concatenated together, and an additional linear projection is performed.

6.2.2 Framing the Data as a Graph

In this subsection, we argue for representing the context and target sets as a graph. We define a graph as $\mathcal{G} := (V, E)$, where V is the vertex set and E is the set of edges between the vertices. Here, vertex $\nu_i \in V$, where $i \in \{1, \dots, N\}$, corresponds to $\mathbf{x}_i$. We will often refer to $\{\nu_i\}_{i=1}^{C}$ as context vertices and $\{\nu_i\}_{i=C+1}^{N}$ as target vertices. In a slight abuse of notation, we consider consider E to be a tensor in $\mathbb{R}^{N \times N \times d}$, where $E_{i,j} \in \mathbb{R}^d$ is the label for the edge between ν_i and ν_j, i is the row of the tensor, and j is the column of the tensor. A diagram of such a graph can be seen in Figure 6.1. Although the exact formation of the graph is architecture dependent, the key idea is that $E_{i,j}$ should capture information from $\phi(\mathbf{x}_i, \mathbf{x}_j)$, where ϕ can be a distance metric or a kernel function. This representation is powerful because one can encode different properties depending on the choice of ϕ. For example, when $\phi(x, x') = \|x - x'\|_2$, the representation is both translation and rotation invariant.

In the following, we suggest two NP architectures which operate over such graphs. The first architecture is a graph transformer [Shi et al., 2020] and focuses on evolving the vertex features in the graph. The second is a novel architecture which is highly expressive and focuses on evolving the edge features of the graph.

Graph Transformer Neural Process To start, we describe a model that takes advantage of the graph transformer architecture [Shi et al., 2020]. Accordingly, we refer to this model as the Graph Transformer Neural Process (GTNP). To start, the graph is initialized by setting $E_{i,j} = f_\theta^E(\phi(\mathbf{x}_i, \mathbf{x}_j))$ and initializing vertex features to $\nu_i^0 = f_\theta^V(\mathbf{y}_i)$ when $i \leq C$ and $\nu_i^0 = f_\theta^V(\mathbf{0})$ otherwise; here, the super-script on ν_i indicates how many graph transformer block transformations the vertex encoding has undergone. Both f_θ^V and f_θ^E are fully-connected neural networks.

After this intialization, the vertices of the graph are updated via a number of graph transformer blocks. These blocks closely match the structure of those in

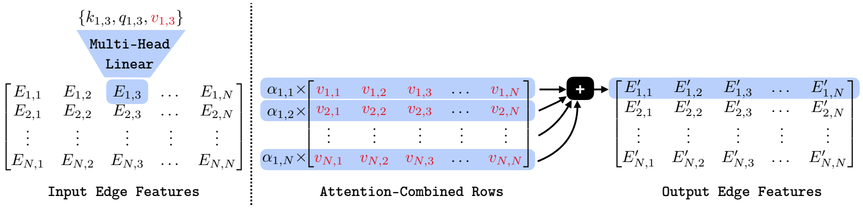


Figure 6.3: **An Illustration of the Key Idea behind `GEAttention`**. Each edge embedding outputs a key, query, and value (left of the divider). However, we combine entire rows of values together inspired by the row operations of Gaussian Elimination (right of the divider).

the GPT-2 model Radford et al. [2019] (where each instead of token embeddings we have vertex embeddings); however, attention is replaced with graph attention in order to fold in edge information. For each edge, $E_{i,j}$, key and value linear projections are made, which we denote as $\lambda_{i,j}, \beta_{i,j} \in \mathbb{R}^d$, respectively. The graph attention operator for vertex i is defined as

$$\texttt{GraphAttention}_i(V, E) = \sum_{n=1}^{N} \alpha_{i,n} \left(v_n + \beta_{i,n} \right)$$

$$\text{where} \quad \alpha_{i,j} = \frac{\exp\left(\frac{1}{\sqrt{d}} \langle q_i, k_j + \lambda_{i,j} \rangle\right)}{\sum_{n=1}^{N} \exp\left(\frac{1}{\sqrt{d}} \langle q_i, k_n + \lambda_{i,n} \rangle\right)}$$

In practice we use a multi-headed version of graph attention. Additionally, we also use a masking scheme that prevents all vertices from attending to a target vertex. After L blocks, the vertices are used for predicting the mean and standard deviation of a normal, i.e. $\mu_i, \sigma_i = g_\theta \left(\nu_i^L \right)$, where g_θ is another full-connected network. See Figure 6.2 for a diagram of this architecture. In practice, we use the masking trick in Nguyen and Grover [2022] to train the GTNP autoregressively (i.e. akin to TNP-A in Nguyen and Grover [2022]). We detail this as well as a thorough description of the architecture in Appendix E.1.1.

Graph Edge Evolution Neural Process In GTNP, the embeddings for the vertices are updated after each block; however, a more powerful model would iteratively update the edge embeddings. Towards this, we propose the Graph Edge Evolution NP (GEENP) which acts exclusively on the edge embeddings of the graph. As such, we must include both information from **x** and **y** in the edges. The edge encoder, f_θ^E

now takes in five inputs:

$$E^0_{i,j} = f^E_\theta\big(\phi(\mathbf{x}_i, \mathbf{x}_j), \mathbf{y}_i \mathbb{1}\{i \le C\}, \mathbf{y}_j \mathbb{1}\{j \le C\}, \\ \mathbb{1}\{i \le C\}, \mathbb{1}\{j \le C\}\big)$$

This additional information is also used to indicate which vertices are context vertices and which are target vertices.

How should one update these edge embeddings? One naive approach would be to use a transformer on the N^2 edge embeddings. Not only would the computational costs of this scale with $\mathcal{O}(N^4)$, but this model would likely have too much flexibility and not enough structure. To strike the correct balance, we look to the Gaussian Process (GP) for inspiration.

The GP is a stochastic process where the distribution of any finite collection of points is distributed as a multivariate normal. The GP is fully characterized by a mean function and a kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let $K \in \mathbb{R}^{N \times N}$ be the matrix such that $K_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. After conditioning on a context set of observations, the posterior is again a multivariate normal distribution. The bulk of the computational effort for computing the posterior is dedicated to inverting $K_{1:C,1:C}$, and one straight-forward way of doing this is via Gaussian elimination. Thus, perhaps the only operation necessary for sophisticated behavior is updating each row of E with a linear combination of the other rows' features.

Leveraging this idea, we replace the regular attention scheme over the N^2 edge embeddings with "Gaussian Elimination Attention" (GEAttention). Thinking about the set of edges as a matrix of embeddings, the key idea is to constrain the attention mechanism to only make convex combinations of values in the same column. Moreover, the weighting scheme for these values must be the same for each column in E. This key idea is depicted in Figure 6.3. The `GEAttention` operator for the edge embedding at row i and column j is

$$\texttt{GEAttention}_{i,j}(E) = \sum_{r=1}^{N} \alpha_{i,r} v_{r,j}$$

$$\text{where} \quad \alpha_{i,r} = \frac{1}{N} \sum_{n=1}^{N} \frac{\exp\left(\frac{1}{\sqrt{d}} \langle q_{i,n}, k_{r,n} \rangle\right)}{\sum_{m=1}^{N} \exp\left(\frac{1}{\sqrt{d}} \langle q_{i,n}, k_{m,n} \rangle\right)}$$

Because it is somewhat unconventional, we will now further explain $\alpha_{i,j}$ for `GEAttention`. First note that $\alpha_{i,r}$ prescribes the weighting on values from row r when updating embeddings in row i. The term inside the summation is standard attention across the n^{th} column. The sum is then used for averaging this information across columns. This scheme reduces the computational complexity from $\mathcal{O}(N^4)$ to $\mathcal{O}(N^3)$. Again, we use a multi-headed version of this scheme in practice. After L blocks, the mean and standard deviation of a normal are predicted via $\mu_i, \sigma_i = g_\theta\left(E^L_{i,i}\right)$. The full architecture description can be found in Appendix E.1.2.

6.3 Experiments

In this section, we experimentally answer the following questions:

- How expressive are GTNP and GEENP compared to other NPs? How robust are each of these NPs to translational shifts at test time? (Section 6.3.1)
- How sample efficient are GTNP and GEENP compared to other NPs? (Section 6.3.2)
- How do GTNP and GEENP perform on real world datasets? (Sections 6.3.3 and 6.3.4)

For comparison, we use the attentive versions of LNP and CNP (we refer to these as AttnLNP and AttnCNP, respectively), ConvCNP, and the autoregressive version of TNP (i.e. TNP-A). The implementation for AttnLNP, AttnCNP, and TNP were taken from the official TNP GitHub repository[2], and we use the same hyperparameters as in Nguyen and Grover [2022] for each of these models. For ConvCNP, we use a repository made by one of the authors of Gordon et al. [2019] [3]. We use the same parameters as their regression experiments and choose the "XL" version of their architecture, which leverages a U-Net [Ronneberger et al., 2015]. Unfortunately, we were unable to find any implementations extending the "off-the-grid" version of ConvCNP for any dimensions higher than 1D. While higher dimensional versions of the algorithm are theoretically possible, forming a grid with high enough fidelity quickly becomes computationally taxing as the dimensionality increases.

For our proposed graph NPs, we try to match the hyperparameters of TNP as closely as possible. In particular, we use 6 blocks, $H = 4$ attention heads, and an embedding size of $d = 64$ for each architecture. We use the following for ϕ,

$$\phi(x, x') = W \bigoplus_{h=1}^{H} \left\| x \otimes w_h - x' \otimes w_h \right\|_2$$

where $\oplus$ representation concatenation, $\otimes$ represents the Hadamard product, and $W \in \mathbb{R}^{d \times H}$ and $w_h \in \mathbb{R}^{D_x}$ are learnable parameters. Note that this ϕ results in a translation invariant representation of the data.

[2]https://github.com/tung-nd/TNP-pytorch/tree/master

[3]https://github.com/cambridge-mlg/convcnp

Table 6.1: **Infinite Dataset Results**. The reported metric average joint log likelihood over the test set. We use five seeds to report the average and the standard error.

Dimension	AttnLNP	AttnCNP	ConvCNP	TNP	GTNP	GEENP
1D	12.92 ± 0.03	12.62 ± 0.03	20.64 ± 0.08	21.34 ± 0.02	21.80 ± 0.01	$\mathbf{22.48 \pm 0.05}$
2D	-4.43 ± 0.02	-4.69 ± 0.02	—	-4.16 ± 0.55	-2.99 ± 0.01	$\mathbf{-2.54 \pm 0.03}$
4D	-9.37 ± 0.01	-9.40 ± 0.00	—	-9.86 ± 0.00	$\mathbf{-9.25 \pm 0.00}$	$\mathbf{-9.25 \pm 0.00}$

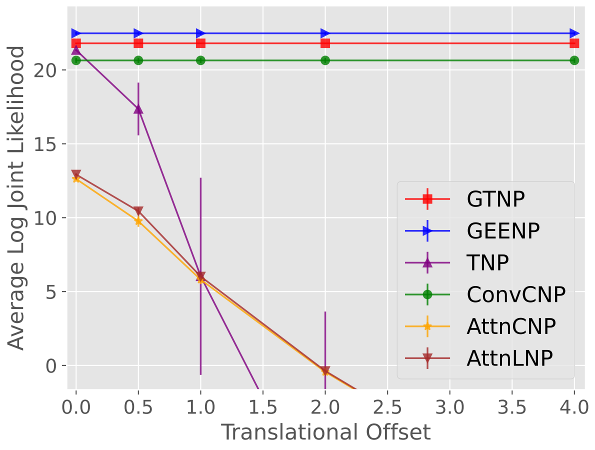


Figure 6.4: **Log Likelihood vs. Translational Offset in the 1D Infinite Data Case.** The y-axis shows the average log joint likelihood over the test set and the x-axis shows the amount of translational shift made on the x inputs.

6.3.1 Synthetic Experiments in the Infinite Data Regime

We start by repeating the experiment first done in Garnelo et al. [2018a] in which the NP is trained from data generated from a hierarchical, 1D GP. The GP has an RBF kernel parameterized by lengthscale $\ell \sim \mathcal{U}(0.1, 0.6)$ and scale $\sigma_\kappa \sim \mathcal{U}(0.1, 1.0)$. Each point in **x** is drawn independently from $\mathcal{U}(-2.0, 2.0)$, and we draw N and C uniformly from $[6, 50]$ and $[3, 47]$, respectively. We refer to this regime as the "infinite data regime" because there is no fixed dataset $\mathcal{D}$. Instead, because $\mathcal{P}$ is known exactly, new **x** and **y** are sampled every batch. This is a luxury that is usually absent from real-world cases with fixed datasets.

In addition to this 1 dimensional GP, we also consider 2 and 4 dimensional GPs. We keep $\mathcal{X} = [-2.0, 2.0]^{D_x}$, but we adjust the range of lengthscales to be $(0.1\sqrt{D_x}, 0.6, \sqrt{D_x})$. We use anisotropic kernels and sample lengthscales independently for each dimension. As an evaluation metric, we compute the joint log likelihood using the NPs in "autoregressive mode", see Section 6.2.1 and Bruinsma et al. [2023]).

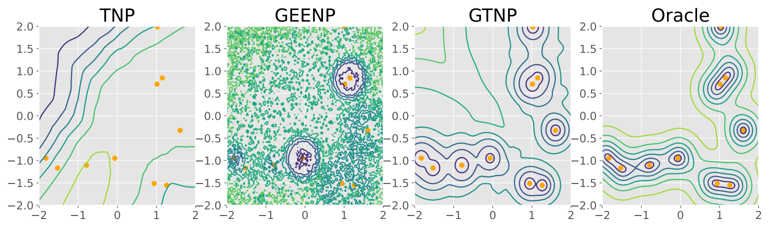


Figure 6.5: **Contour of the Predicted Standard Deviation for the 2D 1K Task**. The far right plot titled "Oracle" shows the standard deviation of the posterior GP that generated the data. The orange points show points in the context set.

Table 6.1 shows the results averaged over five random seeds. Both of our proposed architectures achieve better results than the other baselines, likely because they are translation invariant while not being bound to a grid. We also show how the test metric changes as a translational shift is applied to $\mathbf{x}$ in Figure 6.4. This figure helps demonstrate that our architectures and ConvCNP are robust to translational shifts while the other baselines are highly susceptible.

6.3.2 Synthetic Experiments in the Finite Data Regime

We now consider the "finite" data regime in which there is a fixed dataset $\mathcal{D}$ to learn from. To better understand the performance of the NPs, we simplify the problem by making the data generating process a GP with fixed parameters. In particular, we set the lengthscale for each dimension to the midpoint of the ranges in Section 6.3.1, we set σ_k to 1.0, and we fix $N = 50$. At test time, we compute the joint likelihood for every possible context size and average the results together. We also standardize this metric so that 0.0 corresponds to predicting a standard normal for each point, and 100.0 corresponds to matching the performance of the true underlying GP. More details can be found in Appendix E.2.4.

Table 6.2 shows the results for datasets generated from 1D, 2D, and 4D GPs. In addition, we consider dataset sizes of 1K, 10K, and 100K points (i.e. the 1K dataset has $1,000 \div 50 = 20$ GP function samples). In these experiments, it is clear that our methods (especially GTNP) shine in low data regimes. Figure 6.5 shows contour plots for the standard deviation predictions in 2D for a dataset size of 1K. Under the limited data regime, it is clear that TNP cannot learn any useful uncertainty. While the predictions for GEENP are more reasonable, GTNP produces far better uncertainties than any other NP in this regime. Overall, these results suggest that GTNP is the best choice when there is a limited amount of data, and GEENP is preferential given a large dataset.

Table 6.2: **Finite Dataset Results**. The metric is a standardized score where a score of 100 means that the NP has exactly reproduced the GP, and a score of 0 means that the NP produces a worse log likelihood than simply predicting the prior (see Appendix E.2.4 for the exact definition of this metric). We use five seeds to report the average and the standard error.

Dataset	AttnLNP	AttnCNP	ConvCNP	TNP	GTNP	GEENP
1D 1K	4.88 ± 1.74	-2.72 ± 4.20	34.26 ± 5.38	-2.62 ± 1.86	$\mathbf{61.90 \pm 1.92}$	44.81 ± 0.95
1D 10K	66.83 ± 0.75	63.94 ± 1.06	65.48 ± 4.52	82.01 ± 0.58	$\mathbf{83.73 \pm 0.59}$	74.94 ± 2.00
1D 100K	71.76 ± 0.74	71.92 ± 0.68	93.04 ± 0.35	92.95 ± 0.38	$\mathbf{93.86 \pm 0.23}$	93.45 ± 0.66
2D 1K	-2.56 ± 1.48	-2.80 ± 1.79	—	-8.78 ± 7.38	$\mathbf{56.94 \pm 0.64}$	11.20 ± 7.68
2D 10K	61.18 ± 0.96	57.81 ± 2.24	—	63.92 ± 0.42	$\mathbf{73.49 \pm 1.06}$	62.58 ± 2.74
2D 100K	66.04 ± 2.10	68.33 ± 1.07	—	82.33 ± 0.35	82.92 ± 0.34	$\mathbf{88.67 \pm 1.08}$
4D 1K	-31.04 ± 15.97	-41.94 ± 19.01	—	-9.24 ± 1.22	$\mathbf{-6.10 \pm 2.91}$	-9.52 ± 3.07
4D 10K	24.74 ± 7.62	-43.79 ± 29.08	—	-2.59 ± 2.31	$\mathbf{78.90 \pm 1.40}$	70.04 ± 3.34
4D 100K	59.83 ± 2.40	51.47 ± 4.58	—	71.30 ± 1.54	$\mathbf{91.08 \pm 0.49}$	90.31 ± 0.60
Average	35.74	24.69	—	41.03	**68.53**	58.50

6.3.3 Nuclear Fusion Application

We now shift our attention to real world datasets. In this subsection, we apply our models to an application for tokamaks: one of the most promising devices for making nuclear fusion into an energy source. A tokamak is a toroidal device that achieves fusion by magnetically confining plasma. An important yet challenging problem fusion scientists face in this area is predicting the evolution of the plasma within the device. Machine learning can play a valuable role in this task since one can use historical data in order to learn a dynamics model [Abbate et al., 2021, Char et al., 2023a, Jalalvand et al., 2021, Seo et al., 2024, Wang et al., 2023].

However, predicting the evolution of the plasma remains a challenging task. This is not only due to the complicated nature of the dynamics, but also because the conditions of the tokamak may change from day to day and the nature of the plasma may change based on the experiment (here an experiment usually consists of a few runs of the tokamak known as "shots"). Assuming that the dynamics function is similar for an experiment, we wish to predict uncertainties for the dynamics models by meta-learning over the history of previous experiments.

For this task, we wish to predict two properties of the plasma: its rotation and a measurement called β_N, which is the normalized ratio of plasma pressure to magnetic pressure. We let $\mathcal{X} \subset \mathbb{R}^{10}$ be a description of the plasma's state and the actuator settings of the tokamak at a given time step (more details can be found in Appendix E.2.5). Let $T : \mathcal{X} \to \mathbb{R}^2$ be the transition function which outputs β_N and the rotation of the plasma 25ms into the future. Furthermore, let $\hat{T} : \mathcal{X} \to \mathbb{R}^2$ be a learned model of the transition function. The random function f of interest is the residuals between the estimated and real transition function, i.e.

$f(x) = T(x) - \hat{T}(x)$ for $x \in \mathcal{X}$. Note that $Y \subset \mathbb{R}^2$ and as such, we adjust all NPs to output a multivariate normal with a diagonal covariance matrix for each target point.

We use 3,226 experimental groupings consisting of 736,900 specific transitions from the DIII-D tokamak. We use an 80/10/10 split for training $\hat{T}$, training the NPs to predict uncertainties over the residuals, and computing test results, respectively. We set $N = 50$ and use the most recent data to the target set as the context set. The context may be from the current shot in progress, a previous shot, or a mix. For $\hat{T}$ we use the same "Probabilistic Neural Network" (PNN) architecture described in Chua et al. [2018b]. This network outputs a mean and standard deviation that parameterize a normal distribution. We only use the outputted mean for the point prediction, but we use the standard deviation predictions from this network as an additional baseline. On top of this, we compare against predicting a fixed normal distribution with zero mean and standard deviation estimated from the data.

Table 6.3 shows the results of the experiments. The reported metric is the joint log likelihood averaged over all context sizes. We additionally divide by the size of the target set for each context size in order to make sure all log likelihood are on the same scale. Concretely,

$$\frac{1}{(N-1)M} \sum_{m=1}^{M} \sum_{c=1}^{N-1} \frac{1}{N-c} \log p_\theta(\mathbf{y}^{(m)}_{c+1:N} | \mathbf{x}^{(m)}, \mathbf{y}^{(m)}_{1:c}) \tag{6.2}$$

where $\mathbf{x}^{(m)}$ and $\mathbf{y}^{(m)}$ are the m^{th} collection of points in the test set. Many of the NPs are unable to form better uncertainties than the PNN, which does not condition on previous information from the experiment. However, both the TNP and GTNP are able to achieve better scores, with the GTNP getting the highest score out of all methods. Interestingly, our choice of ϕ assumes that f is translation invariant, but this is unlikely to be true. We speculate that the sample efficiency gained by this assumption outweighs the bias incurred.

Table 6.3: **Average Joint Log Likelihood for Nuclear Fusion Task**. We use five seeds to report the average and the standard error. The method "Fixed Normal" is the result of predicting a single normal distribution over the test set using statistics from the training dataset and as such does not have an associate standard error.

Fixed Normal	PNN	AttnLNP	AttnCNP	TNP	GTNP	GEENP
-2.65	-2.26 ± 0.00	-2.37 ± 0.03	-3.56 ± 0.06	-2.21 ± 0.01	$\mathbf{-2.17 \pm 0.01}$	-2.27 ± 0.01

6.3.4 Lipophilicity Application

Lipophilicity is a critical physicochemical property that plays a significant role in the pharmacokinetics and pharmacodynamics of drugs Miller et al. [2020], Constantinescu et al. [2019]. The ability to make accurate predictions with uncertainty quantification is therefore valuable in cheminformatics and drug discovery processes Isert et al. [2023]. In this section, we use GTNP and GEENP to meta-learn posteriors over unseen molecule lipophilicty given a context set of the most similar molecules available from a training dataset.

Specifically, we leverage the Lipophilicity dataset from the Gaussian Process Chemistry Library, GAUCHE Griffiths et al. [2023]. This is a dataset of 4,200 compounds represented as SMILES strings curated from the ChEMBL database Zdrazil et al. [2024]. The label for each compound is the octanol/water distribution coefficient (log D at pH 7.4). For more details on this dataset please refer to the original publication in Griffiths et al. [2023].

For this experiment we train our proposed NPs by meta-learning over many different local regression problems. To motivate this, we highlight that in many real-world tasks, local Gaussian Process models significantly outperform global models [Eriksson et al., 2019, Krityakierne and Ginsbourger, 2015]. This can be understood by noting the rigidity of many kernels used in practice and the challenge involved in finding a single set of kernel hyperparameters that fit all the training data. For example, if different kernel lengthscales are appropriate for different regions of $\mathcal{X}$, then forcing a GP model to commit to a single lengthscale to explain the entire training dataset can lead to poorer quality predictions. This is especially true when compared to training and tuning GPs on a local dataset relevant to a particular test-time task [Eriksson et al., 2019]. Indeed, in Appendix-E.2.6, we provide experimental results validating that for our Lipophilicity task, when using a state-of-the-art molecule representation and kernel, there is a clear multi-modality of optimal lengthscales as a function of the region of molecule space.

We follow Griffiths et al. [2023] and choose to represent each molecule using Mordred descriptors [Moriwaki et al., 2018] followed by dimensionality reduction with PCA to ultimately represent each molecule as a feature vector in $\mathbb{R}^{51}$. We use two local GP models for baselines. The first GP conditions on the context set and uses a fixed set of kernel hyperparameters that performs well on the training set (see Appendix-E.2.6 for more details on how these hyperparameters were selected). The second GP conditions on the context set and tunes the kernel hyperparameters locally by maximising the marginal log-likelihood of the context set using L-BFGS Liu and Nocedal [1989]. Both GPs use use the Rational Quadratic (RQ) kernel, and we replace the norm in ϕ with the RQ kernel as well.

To generate the training data for our experiments, we first split the data into equal train/validate/test splits, each containing 1,400 molecules. For each split and each molecule, we find the 24 most similar molecules in the same split according to the RQ kernel. These 24 molecules are treated as a context set, and the single molecule they are related to is used for the target set (i.e. $N = 25$ and $C = 24$).

Note that we do not compare against other Neural Process models in this task as the dimension of the molecule representation is too large and the training data is too small to render other NP models appropriate. We additionally compare against an uninformed baseline that uses a fixed standard normal to predict the target point, which can be considered a prior given that the labels have been whitened.

Table 6.4 shows both the mean and median log likelihood over the test set for the single target point. Overall, both GTNP and GEENP are competitive with GP models on this task, demonstrating that meta-learning can be a successful alternative to an assumed prior. This performance is especially notable given the comparative paucity of training data by Neural Process standards. Furthermore, although the tuned GP gets approximately the same median log likelihood as GTNP, it achieves a significantly worse mean score than any other method. This is due to some context sets resulting in poor parameters in the kernel after tuning; however, both GTNP and GEENP are robust to this phenomenon.

Table 6.4: **Target Log Likelihood for Lipophilicity Task** We use five seeds to report the mean and standard deviation of the statistics.

	Mean	Median
Standard Normal	-1.53	-1.21
GP (Fixed)	-1.40 ± 0.00	-1.10 ± 0.00
GP (With Tuning)	-13.31 ± 0.00	$\mathbf{-0.97 \pm 0.00}$
GTNP	$\mathbf{-1.21 \pm 0.00}$	-0.98 ± 0.01
GEENP	-1.26 ± 0.01	-1.02 ± 0.02

6.4 Related Work

There are several other NPs that incorporate graphs into their architecture. Nassar et al. [2018] introduce the "Conditional Graph NP" (CGNP), which leverages bipartite graph convolutions [Nassar, 2018] to group samples in the context set that are in some neighborhood of each other. Importantly, they do not label the edges of their graph, a key aspect of our architectures. In addition, the "Function Neural Process" (FNP) [Louizos et al., 2019] first projects $\mathbf{x}$ into a latent space and then constructs two directed acyclic graphs in this space: one between the context points and one which is a bipartite graph between context and target sets. This avoids the need to create a global latent encoding that summarizes the context set. Our work differs from theirs since our graph encodes relationships in the original $\mathcal{X}$ space via edge labels. Lastly, in contrast to our work which frames regression data as a graph, there are also works on neural processes that operate on graph datasets [Carr and Wingate, 2019, Day et al., 2020, Liang and Gao, 2022].

6.5 Discussion

In this work, we advocate for representing the context and target sets for an NP as a graph. While both of our proposed models often had stronger performance over previous baselines, we found that GEENP shines when there is a plethora of data whereas GTNP has stronger performance when the amount of data is limited. There are several limitations to our architectures, however. One of these limitations is that these architectures incur higher computational costs (Appendix E.2.7). This cost is unavoidable when viewing the data as a fully-connected graph since the architecture must account for the N^2 edges. Additionally, in this work we only use ϕ that is based on the norm between two points in $\mathcal{X}$. Depending on the application, this may be a drawback if the underlying process is non-stationary in $\mathcal{X}$ or if the data lies on a lower dimensional manifold.

In terms of nuclear fusion applications, in this chapter we explored augmenting the dynamics model with a neural process for better uncertainty predictions. However, there are additional applications that this model would be well suited for in nuclear fusion. In particular, this model could be used rather than a Gaussian process or other probabilistic model in a Bayesian optimization set up. Mehta et al. [2024] showed that a simple ensemble of models paired with an active learning algorithm could have significant improvement for shot rampdown. Using a more sophisticated model, such as GTNP or GEENP, has the opportunity to further push this impact.

Part IV

Applications to Nuclear Fusion

Chapter 7

Offline Model-Based Reinforcement Learning for Tokamak Control

This chapter is based on Char et al. [2023a]:
Char, I., Abbate, J., Bardóczi, L., Boyer, M., Chung, Y., Conlin, R., Erickson K., Mehta V., Richner, N., Kolemen, E., & Schneider, J. (2023, June). Offline model-based reinforcement learning for tokamak control. In Learning for Dynamics and Control Conference (pp. 1357-1372). PMLR.

We now discuss applying model-based reinforcement learning for tokamak control on the DIII-D device. Unfortunately, because of the timing of experiments, we were not able to employ any of the algorithmic improvements detailed in Parts II and III of this thesis. Additionally, after the experiments outlined in this chapter, improvements were made to the dynamics model, and this improved dynamics model is outlined in Chapter 8.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, using the DIII-D National Fusion Facility, a DOE Office of Science user facility, under Awards DE-AC02-09CH1146 and DE-FC02-04ER54698. This work was also supported by DE-SC0021414 and DE- SC0021275 (Machine Learning for Real-time Fusion Plasma Behavior Prediction and Manipulation).

7.1 Introduction

Unlocking the potential of nuclear fusion as an energy source would have profound impacts on the world. Nuclear fusion is an attractive energy source since the fuel is abundant, there is no risk of meltdown, and there are no high-level radioactive byproducts [Walker et al., 2020]. Perhaps the most promising technology for harnessing nuclear fusion as a power source is the tokamak: a device that relies on magnetic fields to confine a toroidal plasma. While strides are being made to prove that net energy output is possible with tokamaks [Meade, 2009], there are still crucial control challenges that exist with these devices [Humphreys et al., 2015].

At the same time, exciting developments in reinforcement learning (RL) have provided the possibilities for learning complex controls. While there have been some astounding results that leverage RL, they depend either on a cheap, accurate simulator or an expensive set up where many samples can be collected on the actual device. In our setting, unfortunately, it is infeasible to collect enough samples on the real device, and simulators are both expensive and do not reflect the true dynamics for many aspects of the plasma. Thus, in this work we focused learning controls entirely from historical data. In particular we learned controls for DIII-D, a device operated by General Atomics in San Diego, California. This device has been in operation since 1986, during which there have been over one hundred thousand "shots" (runs of the device). We use approximately 15k of these shots to learn dynamics models that predict the evolution of the plasma, subject to different actuator settings. These surrogate models can then be used as a simulator that generates experience for the RL algorithm to train with. We applied this method to train a controller that uses DIII-D's eight neutral beams to achieve desired β_N (the normalized ratio between plasma pressure and magnetic pressure) and differential rotation targets.

In the following, we first give an overview of the state and actuator variables considered for this control task and the training procedure for learning the controller. We then review our validation experiments conducted on DIII-D for both feedforward and feedback control. The results show the effectiveness of using the learned dynamics models for feedforward control, and while we found feedback control to be more challenging, our controller showed clear promise for β_N tracking. We believe this is the first work for doing offline RL for feedback control on a tokamak, and one of the first works to apply offline reinforcement learning to an expensive device. Thus, we end this paper with a discussion of perspectives on the offline RL problem gleaned from this application.

7.2 Related Work

Reinforcement Learning. Several advancements in the field of deep RL have made the prospect of doing continuous control within reach. Strides in both on-policy algorithms [Schulman et al., 2015, 2017, Mnih et al., 2016] and off-policy algorithms [Lillicrap et al., 2015, Fujimoto et al., 2018, Haarnoja et al., 2018b] have resulted in relatively stable optimization procedures that can produce controls for complex, high-dimensional problems. However, these "model-free" methods are data hungry and usually require millions of samples from the environment. To address this, "model-based" reinforcement learning (MBRL) algorithms can often learn to control with fewer samples by simultaneously learning a model of the dynamics. These models can either be used for better estimates of the value function [Feinberg et al., 2018, Amos et al., 2021] or can be used to generate additional, fictitious data for the agent to train on [Kurutach et al., 2018, Janner et al., 2019a]. We use the latter MBRL approach in this work.

The aforementioned developments in RL target the standard *online* setting, in which agents gather experience through interactions with the environment. In contrast, *offline* RL [Levine et al., 2020] attempts to learn a policy only through logged, historical interactions from possibly many different policies. This is an attractive setting since many real-world problems will have logged interactions to leverage; however, the added restriction usually causes deep RL algorithms designed for the online setting to fail because they pick actions that are out of distribution. To combat this problem, offline RL algorithms add in extra penalization to ensure that the optimization procedure chooses actions close to the support of the dataset [Kumar et al., 2020, Wu et al., 2019, An et al., 2021]. There have also been a number of offline MBRL algorithms which rely on the uncertainty in the dynamics models for penalization [Yu et al., 2020, Kidambi et al., 2020a, Yu et al., 2021]. In our work we decided against using a penalization scheme for a couple of reasons. First, the amount of penalization needs to be tuned by evaluating the controller on a real device, something that we do not have the luxury of. Second, the dynamics models in these works are only accurate for a few time steps, a problem that usually plagues autoregressive models due to the multiplicative accumulation of error [Asadi et al., 2018]. For example, in Yu et al. [2020] the model is only used for 1 or 5 time steps (see Appendix G). We believe our setting is unique since we are able to learn a dynamics model that is often accurate for entire shots.

Learning Controls for Tokamaks There has recently been a surge of interest in applying machine learning for controls of tokamaks. Many of these works focus on predicting disruptions for avoidance or safe shutdowns [Fu et al., 2020b, Parsons, 2017, Rea et al., 2019, Boyer et al., 2021b]; however, in this work, we focus on control during stable operation. Under these conditions, Char et al. [2019] used contextual Bayesian optimization to find controls that balance increasing β_N and keeping the plasma stable. While this technique is fully automated, Baltz et al. [2017] present an algorithm that performed human-in-the-loop optimization to increase plasma confinement.

In terms of modeling dynamics, Abbate et al. [2021] used a convolutional neural network to model the evolution of the plasma's profiles, and they later used this model for control via MPC [Abbate et al., 2023]. Many of the choices for our dynamics modeling, such as the signals to use and the data preprocessing, were directly inspired by this work. Additionally, Seo et al. [2021, 2022] learned a dynamics model for the KSTAR tokamak. They then used RL for tracking several scalar values including β_N; however, they used this policy to generate feedforward controls only. While Wakatsuki et al. [2021] trained a feedback controller to do ion temperature gradient control for the JT-60 tokmak, this controller was both trained and tested in the same TOPICS simulator. To the best of our knowledge, the only RL feedback controller deployed on a real device up to this point was done by Degrave et al. [2022]. They leveraged a simulator to learn a controller for the plasma's shape on Tokamak a Configuration Variable [Coda et al., 2019]. Our work differs not only in the goal and actuators used, but also from the fact that we leveraged historical data exclusively. The dynamics for the plasma's shape is more well-understood than other aspects of the plasma, and as such, can be modeled and controlled relatively precisely [Walker et al., 2020, 1997]. While the potential impact of learning controls for other aspects of the plasma is great, the corresponding simulations are expensive and less precise, which prompted us to leverage logged data.

7.3 Method

Problem Description We cast the problem of control of the tokamak as a discrete-time, infinite-horizon Markov decision process (MDP). In particular, let $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, \gamma, T, r, \rho \rangle$ be the MDP, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space,

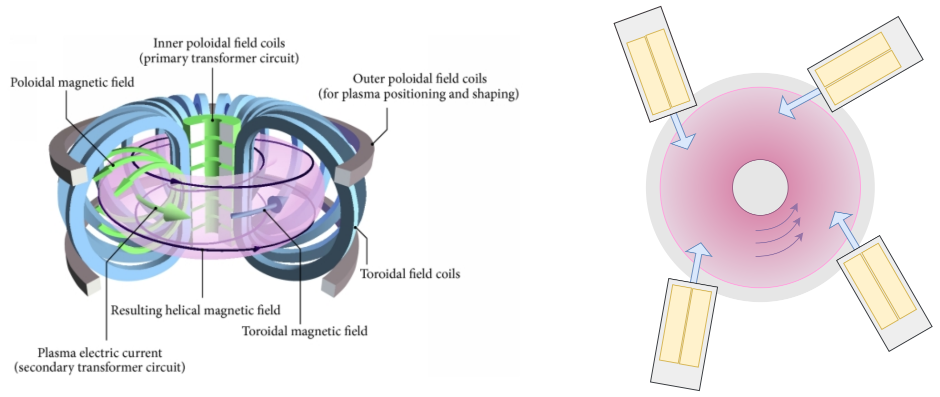

Inner poloidal field coils
(primary transformer circuit)
Poloidal magnetic field
Outer poloidal field coils
(for plasma positioning and shaping)
Toroidal field coils
Resulting helical magnetic field
Toroidal magnetic field
Plasma electric current
(secondary transformer circuit)

Figure 7.1: **Diagram of Tokamak (Left) and Top-Down View of DIII-D (Right)**. Looking at the right figure, one can see that each beam line contains two independent neutral beam sources (yellow boxes). Here, the plasma is rotating in the counter-clockwise direction, and the two beams in the bottom left of the figure are oriented to be counter-current. Because of this, the total power and torque are decouple. The left image is from Li et al. [2014]. The right image gives a rough idea of the beam positioning and is not drawn to proportion.

$\gamma \in (0,1)$ is the discount factor, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and ρ is the initial state distribution. Lastly, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the transition function, where $\mathcal{P}(\mathcal{S})$ denotes the space of distributions over $\mathcal{S}$. Each transition corresponds to a 100ms time step in real time. In practice, it is difficult to observe all state variables in real-time for feedback control. As such, we learn a policy for the partially observable MDP (POMDP). Let $\mathcal{O}$ be the observation space and let $h : \mathcal{S} \to \mathcal{O}$ be the mapping from states to observations. The overall goal is then to learn a policy $\pi : \mathcal{O} \to \mathcal{P}(\mathcal{A})$ that maximizes the expected discounted sum of rewards $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$, where $s_0 \sim \rho$, $a_t \sim \pi(h(s_t))$, and $s_{t+1} \sim T(s_t, a_t)$.

State and Action Spaces DIII-D has a number of actuators for controlling the plasma. Of key interest to our work are the eight neutral beams that inject particles into the core of the plasma (see Figure 7.1) [Grierson et al., 2021]. These are used to inject both power and torque into the plasma, and because two of these beams can be oriented in the opposite toroidal direction, the amount of total power and torque injected can be decoupled. We also consider the the ohmic coil (for controlling current), gas valves (for controlling plasma density), and toroidal field coils for our modelling [Luxon, 2002]. Lastly, one can also control the shape of the plasma via the field coils. We assume that these controls are sophisticated enough to the point that we can control the elongation, top triangularity, bottom triangularity, and the minor radius of the plasma exactly. While there are many other ways of affecting

			MDP Spaces		
Signal Group	**Signals**	**Actuator**	**State ($\mathcal{S}$)**	**Action ($\mathcal{A}$)**	**Observation ($\mathcal{O}$)**
Scalar States	β_N	✗	✓	✗	✓
	li (Internal Inductance)	✗	✓	✗	✗
	Line Averaged Density	✗	✓	✗	✗
Profile States	Ion Rotation, Electron Density, Electron Temperature, Pressure, q	✗	✓	✗	✗
Neutral Beam Variables	Power Injected and Torque Injected	✓	✓	✓	✓
Shape Variables	Elongation, Top Triangularity, Bottom Triangularity, a_{minor}	✓	✓	✗	✗
Other Actuators	Current Target, Density Target, and Toroidal Field	✓	✓	✗	✗
Observations	DR, DR Target, and β_N Target	✗	✗	✗	✓
Total Dimensions		9D	47D	2D	10D

Table 7.1: **Overview of Signals**. Note that DR and both targets are not in the state space. DR is calculated from rotation and q (but is not modeled explicitly), and the targets do not influence the transition function. The total dimension of the state space factors in the number of PCA components used to represent the profiles, and the total dimensions of both state and observation spaces accounts for measurements 100ms in the past. While the state dimension is relatively high, the dynamics model only predicts future scalar and profile state variables (19D).

the plasma, this subset encapsulates most standard runs on the device. All of these actuators could potentially be part of the action space; however, we focused on only the neutral beams for this work. As such, the action space is simply the total power and torque injected from the neutral beams.

For the state space, we assume that the plasma can be fully characterized by the current settings of the above actuators, three scalar values, and five "profiles" which consist of discretized measurements of physical quantities along the minor radius of the toroid. The scalar states consist of the line-averaged electron density, the internal inductance, and β_N, which is the normalized ratio between plasma pressure and magnetic pressure. β_N is an important quantity as it can be used as a rough economic indicator of efficiency. Radial profiles of the ion rotation, pressure, electron temperature, electron density, and the safety factor, known as "q", are also used. The q profile is the number of toroidal transits per poloidal transit of a magnetic fieldline, and is an important indicator of stability of the plasma (the higher the q fac-

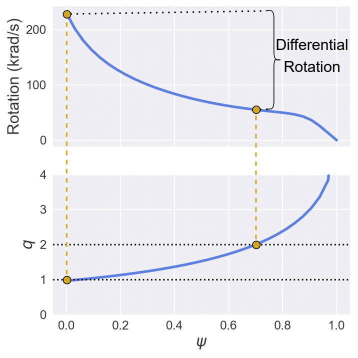


Figure 7.2: **Visual Representation of Differential Rotation (DR)**. The top and bottom plots show examples of rotation and q profiles, respectively. The q profile dictates the ψ (flux surface) locations to measure on the rotation profile.

tor the better). Following Boyer and Chadwick [2021], we reduced the dimensionality of these profile states (originally 64 dimensions) by using principal component analysis (PCA). We found that we can explain 99% of the variance in the data by using two principal components for the q and pressure profiles, and by using four components for the rest of the profiles. For these scalar and profile descriptions of the plasma, we assume the state can be represented by the current measurements as well as the measurements 100ms in the past. This assumption stems decisions made about the learned dynamics model. In particular, including this history increased the predictive performance of the dynamics model; however, including history for the actuators variables made the model susceptible to overfitting).

Objective and Reward Function The control objective is to do target tracking for two quantities: β_N and *differential rotation* (DR). Specifically in this work, DR refers to the difference in the rotation profile at the locations where $q = 1$ and $q = 2$ (see Figure 7.2). This is an important quantity of interest as it is hypothesized that higher DR results in a more stable plasma [Bardoczi et al., 2021, Tobias et al., 2016, Buttery et al., 2008, Reimerdes et al., 2007, Politzer et al., 2008]. While control for β_N is relatively straight forward, DR relies on the correct measurements of two profiles and is therefore harder to predict and control. For every episode in the MDP, new targets β'_N and DR' are drawn from target distributions. In particular, $\beta'_N \sim \mathcal{U}(1.25, 2.5)$ and $DR' \sim \mathcal{U}(10, 80)$. The reward at time step t, $r(s_t)$, is then

$$\frac{-1}{C_1}\left(\beta_N^{(t)} - \beta'_N\right)^2 + \frac{-1}{C_2}\left(DR^{(t)} - DR'\right)^2,$$

where $\beta_N^{(t)}$ and $DR^{(t)}$ are the current measurements at time t, and C_1 and C_2 are positive normalizing constants that puts each term onto the same scale.

7.3.1 The Dynamics Model and Controller

Dynamics Model We chose to approximate $\mathcal{T}$ using a fully connected neural network which takes in the current state of the system and the actuators settings planned for 100ms in the future, and then outputs predictions for the (non-actuator) state variables 100ms into the future. We trained this network on a dataset consisting of 15,534 shots (or 268,702 time steps), which was pre-processed in the same manner as Abbate et al. [2021]. In particular, at each 100ms increment we formed the observation for each signal by averaging over every measurement 25ms previous to that point.

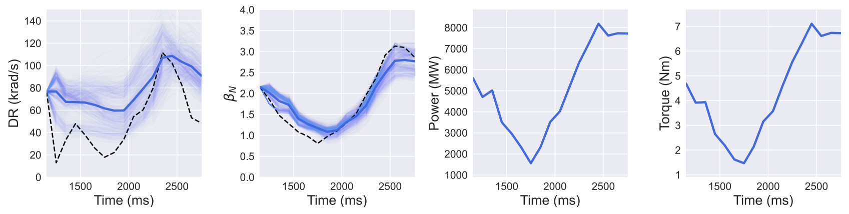


Figure 7.3: **Replay of Shot 187076.** Here, the model receives the first observations of β_N and DR, but then autoregressively predicts these values into the future. The faded lines are different samples of the neural network parameterization, and the solid lines are the average over the different predictions. Note that there are no faded lines for the power and torque plots since actuators are given to the model and are not predicted. The black dashed lines are the real observations.

To ensure that a controller learned on this model will still perform on the real device, it is important to learn many different possibilities of what the dynamics could be. This has been shown to be essential both in the context of MBRL [Chua et al., 2018b], and in the context of doing "sim2real" (e.g. domain randomization [Tobin et al., 2017]). We incorporated uncertainty by learning a subspace of network parameters [Wortsman et al., 2021, Benton et al., 2021], which has been shown to better calibrated models over standard ensembling. In particular, we used an ensemble of five networks to learn a simplex of network parameters following the procedure described in Wortsman et al. [2021]. We repeated this training procedure five times to learn five different simplices. By making a uniform draw from this collection of network parameters, we can sample a new possibility for the dynamics.

To do hyperparameter tuning and evaluation, we took the most recent 10% of shots as our test set. Our tuning procedure targets high explained variance (EV) for one-step predictions. After performing grid search, we settled on a model with 4 hidden layers of 512 units, and a learning rate of 3e-4. When learning the simplex, we encouraged diversity by adding a cosine similarity penalizer to the loss function (see Wortsman et al. [2021]), and we found that a coefficient of 5 to this penalty gets the best results. Averaged across five seeds and one hundred samples from the simplex for each seed, these mean predictions achieve an EV score of 0.46 for β_N, 0.43 for the first rotation PCA component, and 0.33 averaged across all output signals. We use the Uncertainty Toolbox library Chung et al. [2021a] to evaluate our ensemble's predictive uncertainty. We find that our model tends to be overconfident and achieves a miscalibration area of 0.26 for β_N, 0.25 for the first rotation PCA component, and 0.297 averaged across all predictions. We believe that part of the reason these scores are poor is that the future shots in the test set are meaningfully different. However, qualitatively the model often captures the trends of the state quite well. Figure 7.3 shows the replay for shot 187076, which was not seen during training. This shot is significantly unique from other shots in the dataset in that there is a drastic drop then increase of both power and torque. Lastly, although we do find that our predictions into the future are generally stable, there are rare cases where the prediction error explodes. To mitigate against this, we bound the state of the plasma and the amount that it can change to be between the 2.5% and 97.5% quantiles of the dataset.

Controller This learned dynamics model can be used to generate data for learning a controller. For the start state distribution, ρ, we used a uniform distribution over over the first 500ms of flat top (i.e. where current stops ramping up and becomes stable) for all shots in the dataset. Since the controller is only allowed to counterfactually set the total power and torque of the neutral beams, all other actions are the same as what happened in the historical shot corresponding to the start state. We trained a controller on these generated shots using the Proximal Policy Optimization (PPO) algorithm [Schulman et al., 2017]. The controller is able to observe the targets as well as the current and past values of β_N, DR, power injected, and torque injected. We use a policy and value network with 2 hidden layers consisting of 500 units each, a gradient clipping parameter of $\epsilon = 0.25$, and a learning rate of $3e - 4$ for both the policy and value networks. We decided on these hyperparameters by using an off-policy evaluation procedure in which we have two sets of dynamics models: one used for training and one used for evaluation. The only difference between the two sets is that the model used for evaluation was trained using both the training and testing set. Additionally, for the start state selection and actuator replays, we reserved some historical shots for the evaluation period. The final set of policies were trained on the test set of models (with some historical shots still held out), and the model that was ultimately selected for deployment was the one with the best returns on the held out shots.

7.4 Experiment

To test the controller on the device, we implemented our trained policy in DIII-D's plasma control system (PCS) [Margo et al., 2020]. We used the Keras2C library [Conlin et al., 2021] in order to transfer our policy network (originally implemented in PyTorch [Paszke et al., 2019a]) to a deterministic subset of C, meaning no dynamic memory allocation, system calls, or use of external libraries. For beam control, we used the algorithm presented in Boyer et al. [2019] to decide the duty cycles of each of the eight beams to hit the requested power and torque targets. For inputs to the policy, we relied on the profile fitting algorithm [Shousha et al., 2023] and the charge exchange recombination (CER) diagnostic system [Gohil et al., 1991]. The policy sends requests for updates to the beams roughly every $10\ \mathrm{ms}$. During our testing session, we were able to test the β_N and DR tracking separately. We used shot 164987 (shots are labeled) as a reference shot, and the actuators besides the beams were mostly used from this shot.

Feedforward Control To disentangle the predictive power of the dynamics model from the policy learning procedure, we used the dynamics model only to prepare feedforward control for the neutral beams. In particular, we used the model to evaluate how closely target values are achieved given fixed controls where the power and torque are ramped up to constant values. A two-dimensional grid search was performed to find the constant values corresponding to the highest cumulative rewards averaged over the sampled shots. We used an even larger ensemble of models for this procedure where the additional models have slightly altered inputs and outputs. In particular, we included five additional network simplexes that consider the actuators from the previous time steps as input and five network simplexes that only consider quantities relevant for this task as inputs and outputs, i.e. β_N, rotation, q, and beam information. This results in a total of 15 different network simplexes, each composed of 5 networks. We found the performance of each type of model is dependent on the shot, so we used all models in the hope for the most robust solution.

We pick a target $\beta_N = 1.75$ and try to push differential rotation to be high by setting the target to $DR = 40\ \mathrm{krad/s}$, which is relatively high for this reference shot. Our optimization procedure found that setting the total power to $3.6\ \mathrm{MW}$ and the total torque to $2.1\ \mathrm{Nm}$ was best. As shown in Figure 7.4, the choice of these actuators resulted in hitting the β_N target remarkably well. Although the DR achieved was lower than the target, one can see that it does achieve higher DR. For reference, DR has a standard deviation of 35.2 and an interquartile range of 47.2 amongst all shots in the dataset, so the error between the target and the value achieved is not as bad as it may appear.

Feedback Control Next, we tested the learned policy's ability to do feedback control. We started by having the controller track increasing β_N targets. Because β_N primarily relies on the power injected, we used the policy to control the injected power only and set the total torque to be $2\ \mathrm{Nm}$ throughout the shot. For shot 191611 in Figure 7.4, one can see that the controller increases the power in order to hit the target values. The last target is overshot slightly and some oscillatory behavior occurs. After the experiment, we identified a bug in our set up where the magnitude of change in power is greater than what was requested by the controller when there

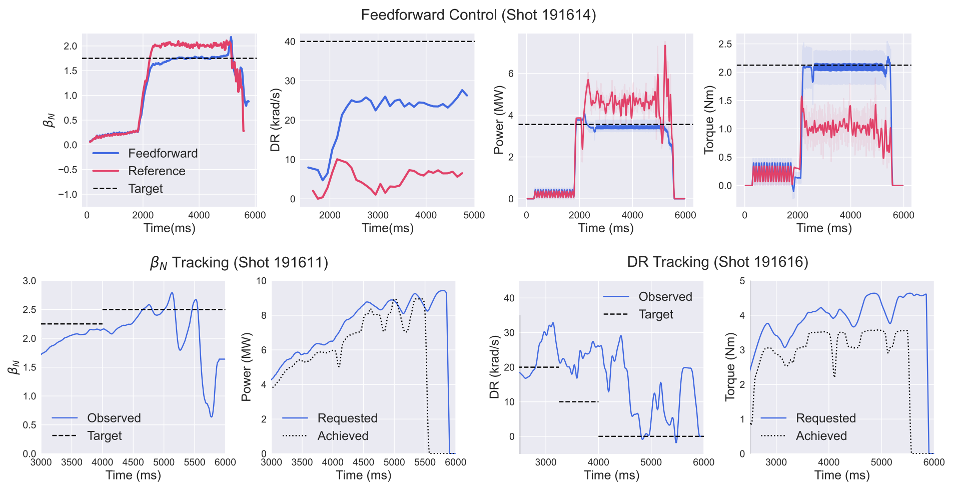


Figure 7.4: **Experiment Shots.** The top four plots show the β_N, DR, total power, and total torque during both shot 191614 (feedforward control) and the reference shot 164987 (red). These values are smoothed when needed and the original, unsmoothed values are shown as faded lines. The DR values are taken after doing preprocessing and dimensionality reduction via PCA. For the bottom plots, the left pair of plots shows the experiment controlling the power to hit β_N targets, while the right pair of plots shows controlling the torque injected to hit DR targets. In each pair, we show the requested amount of power or torque requested by our controller vs the actual value achieved.

is high beam usage. This occurs at the 4500 ms point onward, and this phenomenon could perhaps be the reason behind the oscillatory behavior. The fluctuation in β_N is then further exacerbated by a disruption in the plasma, and all control is lost. Because of limited time, we were unable to compare against pre-existing controllers on our set up [Boyer et al., 2019, Scoville et al., 2007]. While they are likely to track β_N more reliably, we still believe that this is a step in the right direction for showing MBRL's value in learning controls.

Unlike β_N tracking, there is no other controller in the DIII-D PCS that specifically tracks the difference in rotation between the $q = 1$ and $q = 2$ surfaces. To test our controller's ability to do so, we set a series of decreasing DR targets for the controller to achieve using only total torque. We set the power injected to a constant value of 5 MW, and although torque could vary since the 210 beams were in the counter-current orientation, this still restricts the total torques that can be achieved. The controller is unable to track the DR targets nearly as well as the β_N targets. While there are some instances of the policy doing the right thing (e.g. torque is decreased at time 4000 ms time to drop DR to the target), the policy shortly after observes DR dropping too quickly and raises torque back up again (shot 191616 in Figure 7.4).

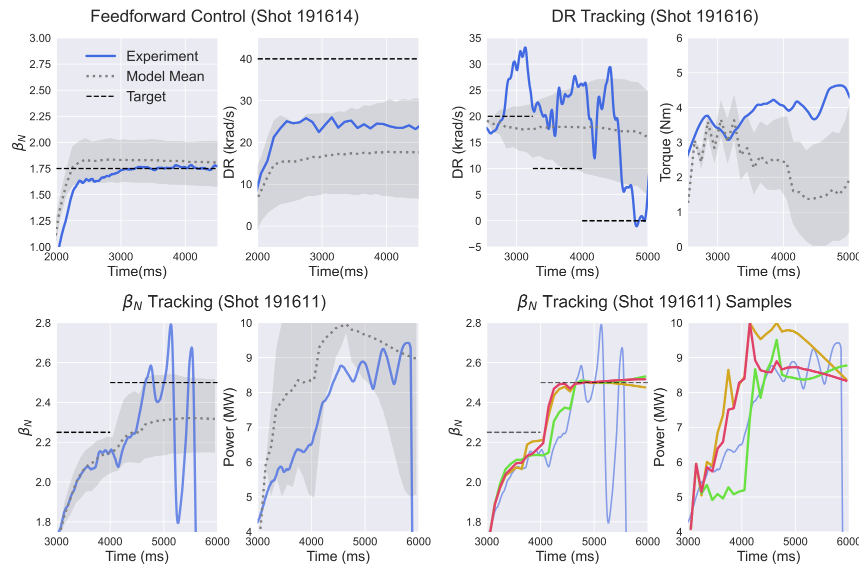


Figure 7.5: **Replay of Experiments.** The top and bottom left pair of plots show the experiment observations (blue) and the mean prediction from the model (gray dotted line). We used 20 sampled shots per model in the ensemble, i.e. we used 300 samples for shot 191614 and 100 samples for the other shots. The gray region is the area spanned by the 5^{th} and 95^{th} percentile sample. The bottom right pair of plots show the three highest return samples (shown in red, yellow, and green) for the β_N feedback control shot.

7.4.1 Post Experiment Analysis

To aid in the analysis of these experiments, we can see how predictions in our dynamics model line up with what actually happened in the experiment. Starting with the feedforward shot (191614), predictions were made using the reference shot; however instead of the original power and torque controls, the planned controls are used instead. In Figure 7.5, one can see that the that the true shot is indeed contained within the predicted distribution. Despite the model predicting that the target DR would not be achieved, this was the optimal configuration landed on by the model because increasing the torque injected causes β_N to overshoot the target in the simulated environment. Moreover, because the spread in DR is much higher than β_N, the optimizer implicitly favors tracking β_N.

Next, for the β_N tracking shot (191611), we replayed the shot in our simulated environment using the learned controller. Many of the sampled trajectories cannot achieve the second target of $\beta_N = 2.5$. This may be due to the fact that this is a relatively high β_N value for this reference shot, and the dynamics model may have learned that a loss of confinement usually happens at this range of β_N. When this happens, the controller keeps increasing the power in order to try to achieve the target value. For the samples of the dynamics model that are able to achieve higher β_N, the controller is able to hit the target well, and the schedules in power injected used to achieve the targets are comparable to what was seen in the actual experiment (Figure 7.5). For these samples, there is no overshoot of the target or oscillation of β_N, and it is possible that without the problem with the beams that the controller would have been able to hit the target more reliably during the experiment.

Lastly we replayed the shot 191616, and use the controller to try to achieve the DR targets. Unlike control of β_N, it does not seem that DR is controllable to the same extent for this shot. However, looking at the values of the torque injected there are clear changes in the torques as the target values change. We also find that the controller does not make any drastic changes to torque if it cannot hit the DR target. We hypothesize that this is because β_N is affected by the torque injected, and it is the more reliable quantity to track. The controller will therefore not drastically change the torque if it compromises the tracking of β_N. Furthermore, we find that the uncertainty in the model increases with lower settings of torque, which may further deter the controller from decreasing torque.

7.5 Discussion

In this work, we show the first steps towards doing feedback control on a tokamak by learning through logged data alone. Furthermore, through our feedforward controls, we have demonstrated the predictive ability of our dynamics models for control. While we faced challenges throughout the course of this work that are common to every application of sim2real [Ibarz et al., 2021] and offline RL [Levine et al., 2020], we believe that our work provides takeaways for the RL community.

MBRL on Undirected Data. Many offline RL benchmark tasks assume that much of the collected data has the test-time task or reward function in mind. While the D4RL benchmarks [Fu et al., 2020a] do have tasks with undirected datasets (e.g. the maze tasks and FrankaKitchen), these datasets contain sub-trajectories of good behavior that simply need to be "stitched" together. We believe that our application falls into another interesting setting that these baselines do not cover. This setting is one in which there are not necessarily sub-trajectories of good behavior, but reasonable dynamics models can be learned either because the dataset is sufficiently expansive or through injecting prior information into the models (e.g. Mehta et al. [2021a], Yin et al. [2021]). Unique challenges and opportunities would likely come from further studying this setting.

Model Diversity. As seen in Section 7.4.1, it was important to have diversity in the dynamics models to ensure that the true dynamics are covered and that the controller can handle different possibilities. While we used PPO in conjunction with these models, it is possible better results could be achieved by using recent developments that leverage the diverse model predictions for test-time adaptation [Ghosh et al., 2022, Chen et al., 2021b]. This also raises the question: how should one evaluate uncertainty estimates in this setting? While we chose to use miscalibration area as our metric in this work, it is unclear which metric is indicative of good policies being learned downstream.

Policy Evaluation. While it is known that models are a useful tool for off-policy evaluation [Thomas and Brunskill, 2016, Jiang and Li, 2016], we believe that it is important that these models are learned in such a way that they can test the generalization capabilities of the policy. Our method of doing this was to set aside some shots that only these testing models would train on; however, there are possibly more sophisticated procedures for doing this. This was useful for making key decisions for our learning pipeline (e.g. we found policies trained with SAC [Haarnoja et al., 2018b] had worse generalization compared to those trained with PPO).

Chapter 8

Full Shot Predictions for the DIII-D Tokamak via Deep Recurrent Networks

This chapter is based on Char et al. [2024]:
Char, I., Chung, Y., Abbate, J., Kolemen, E. & Schneider, J. Full Shot Predictions for the DIII-D Tokamak via Deep Recurrent Networks. *arXiv preprint arXiv:2404.12416*, 2024.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, using the DIII-D National Fusion Facility, a DOE Office of Science user facility, under Awards DE-AC02-09CH1146 and DE-FC02-04ER54698. This work was also supported by DE-SC0021414 and DE- SC0021275 (Machine Learning for Real-time Fusion Plasma Behavior Prediction and Manipulation).

or reflect those of the United States Government or any agency thereof.

8.1 Introduction

In a wide range of fields, dynamics modeling is a fundamental tool that can be used to gain better understanding of a given system. Dynamics models are especially useful in the context of control, as they allow for prediction of responses to system perturbations over time, which can then be used to design and implement control sequences that optimize for desirable behaviors.

Such benefits are especially apparent in tokamak systems. Tokamaks are toroidal devices which magnetically confine plasma at high temperatures and pressures for prolonged periods, during which nuclear fusion reactions occur within the plasma. The tokamak system is one of the most promising approach to realizing nuclear fusion as an energy source. While strides are being made to improve the efficiency, stability, and reliability of the system, there are crucial control challenges which remain [Humphreys et al., 2015].

Since running these devices is extremely expensive, domain experts rely on virtual representations of the system dynamics, such as simulators and dynamics models. Simulators typically rely on first principles and simulate the dynamics via known equations which describe the theoretical behavior of the plasma. However, simulators are prohibitively expensive in terms of time and computation, and despite these costs, they are often still unable to accurately describe the plasma's dynamics.

Concurrently, massive strides have been made in machine learning (ML), where advances in algorithms and modeling architectures paired with data and compute have allowed for a completely data-driven approach to learning highly accurate models. This approach is promising for the nuclear fusion setting, and indeed, numerous recent works have applied ML methods to tokamak modeling [Abbate et al., 2021, Boyer et al., 2021a, Seo et al., 2021, 2022, Char et al., 2023a].

In this work, we focus on learning a dynamics model for the DIII-D tokamak, a tokamak in San Diego, California operated by General Atomics. Since the device has been in operation since 1986, we are able to draw from a wealth of previous plasma discharges (or "shots") from the device to train a deep recurrent network. A typical shot on DIII-D lasts around 6-8 seconds, with a 1 second ramp up phase, several second flat top phase, and one second ramp down phase. DIII-D also has several real-time and post-shot diagnostics that measure the magnetic equilibrium and plasma parameters with high temporal resolution. We find that learned models are able to predict these measurements for entire shots remarkably well.

We further investigate the impacts of our modelling choices. Along with architecture and training choices, we highlight the importance of uncertainty quantification and explore which methods of forming predictive distributions results in the most calibrated models. As more interest accumulates in control of tokamaks via data-driven models [Seo et al., 2021, 2022, Char et al., 2023a, Seo et al., 2024], we hope that this work provides valuable insights that accelerates prediction and control.

8.2 Related Work

8.2.1 Simulators for Tokamaks

Predictive modelling of the plasma through first principle equations is difficult since different aspects of the plasma evolve at different time scales. State of the art simulators solve this problem by evolving these aspects independently [Felici et al., 2011]. While these simulators have been useful for exploring different regimes for the plasma [Rodriguez-Fernandez et al., 2022] and making new controls [Felici and Sauter, 2012], they are nevertheless limited in that they require additional information such as an estimate for the density at the edge of the plasma. Our learned models are unique from these in that the only information they require are the settings for the difference actuators throughout the shot.

8.2.2 System Identification and Machine Learning for Dynamics

There is a long lineage of methods for inferring behavior of a dynamic systems from data. "System identification" is one broad categorization of such methods and provide a classification along a spectrum of "white" to "black" based on how much prior domain knowledge is incorporated. Whereas white-box models rely strictly on prior knowledge of the relationships between system variables to infer the system parameters, black-box methods rely on purely observed data to model their plausible relationships. We refer the reader to Ljung [2010] and Schoukens and Ljung [2019] for a more in-depth, comprehensive survey of existing methods in system identification.

Recently, neural networks (NN) have been widely used for modeling dynamics and have shown substantial success [Wang and Yu, 2021]. Many of these methods require at least some prior knowledge of the system (i.e. they are grey box models). For example, one may be able to describe the system with a set of ODEs that capture some (but not necessarily all) of the dynamics [Mehta et al., 2021a, Yin et al., 2021], or one may have a set of PDEs and is unable to compute a solution easily [Raissi and Karniadakis, 2018, Raissi et al., 2019]. Unlike these works, however, we explore black box models in which there is no prior knowledge available.

Group	Representation	Type	Signal	Dimension
States	Scalar	Shape	κ, a_{minor}, Triangularity Top, Triangularity Bottom, R and Z Coordinates of Magnetic Axis	6
		Other	β_N, Line Averaged Density, Internal Inductance (li), q_0, q_{95}, n1rms, n2rms, n3rms, vloop, wmhd, Differential Rotation [Char et al., 2023a]	11
	Profile		Electron Temperature, Ion Temperature, Density, Rotation, Pressure, q	20
Actuators	Scalar	Beam	Power Injected, Torque Injected	2
		Gas	gasA, gasB, gasC, gasD	4
		Shape	12 Shape Controls	12
		Other	Current Target, Density Target, Toroidal Field	3
Total Dimension: 58				

Table 8.1: **List of all state and actuator variables**

8.2.3 Machine Learning in Nuclear Fusion

There has been a recent surge of interest in applying machine learning to predict the state of the plasma within tokamaks. Perhaps one of the areas with the greatest interests is predicting whether the plasma is in (or is about to be in) a disruptive state. Fu et al. [2020b], Parsons [2017], Rea et al. [2019], Boyer et al. [2021b] learn predictive models of whether the plasma will disrupt and use these models to take preventative actions to stabilize the plasma. Char et al. [2019] also use Bayesian Optimization with a similar goal in mind; however, they learn the actions to apply directly rather than a prediction of whether the plasma is disruptive or not.

In terms of learning the evolution of the plasma state, Abbate et al. [2021] learn a deep neural network in order to predict the profiles of the plasma; however, they focus on one step predictions for their model. In contrast, Seo et al. [2021, 2022] learn a recurrent neural network to predict scalar states of the plasma. They can use this model to autoregressively predict these states into the future, and they leverage this to plan shots on the KSTAR tokamak. Recently, Char et al. [2023a] used a (non-recurrent) learned model that predicted both scalar and profile states in an autoregressive manner. They used this model as a simulator to train a reinforcement learning agent, which was then deployed on DIII-D. Whereas most of these works focus on the control aspect of dynamics modelling, we do a deeper investigation on the modelling itself. We hope our evaluation techniques and insights will benefit future model-based control works.

8.3 Method

8.3.1 Data

We begin this section by describing the data used to train our dynamics model. In total, we use 7,884 historical shots from the DIII-D tokamak. We include both the ramp up and flat top phases of each shot, and each shot is subdivided into a number of ”time steps” 25ms apart from each other. For each time step, we average the measurements collected 25ms previous to that point in time.

We partition the input signals into two groups: state signals and actuators signals. All of these signals can be found in Table 8.1. For the state signals, we use 17 different *scalar* states and 6 so-called *profile* states. While scalar states provide a summary statistic of one aspect of the current plasma state with a single scalar, profile states are 1D measurements of the plasma, and in our dataset, they consist of 33 discrete measurements along the minor radius of the cross-section of the tokamak. Following previous works in profile modeling [Char et al., 2023a, Boyer et al., 2021a], we choose to lower the dimensionality of the (originally 33-dimensional) profile states via Principle Component Analysis (PCA). In particular, we use four principal components to represent all profiles states except for the pressure and q profile. For these two signals, we use the first two principal components to represent the profile. For actuator signals, we use 21 different scalar values that summarize neutral beam settings, current and density targets, gas settings, and plasma shape control.

We choose to separate these signals into two groups (states and actuators) since we assume that all of the actuators are known a priori (from the perspective of the experiment operator). As such, the model takes as input the current state measurements, the current actuators settings, and the actuator settings 25ms into the future. The model then predicts the change in the state variables for the next 25ms. Once trained, the model can predict many more steps into the future by autoregressively feeding in predicted next states back into the model as inputs.

8.3.2 Model Architecture and Training

In designing our model architecture, we use a recurrent neural network (RNN) with a gated recurrent unit (GRU) [Cho et al., 2014a]. We use 6 hidden layers (including encoder and decoder), each with 512 hidden units and residual connections [He et al., 2016]. A visual diagram of the model architecture can be seen in Figure 8.1. We train our recurrent model with full length shots, the longest of which is 225 time steps. We use a learning rate of $3e-4$ and a weight decay of 0.001.

For the model output, rather than making point predictions, we have two output

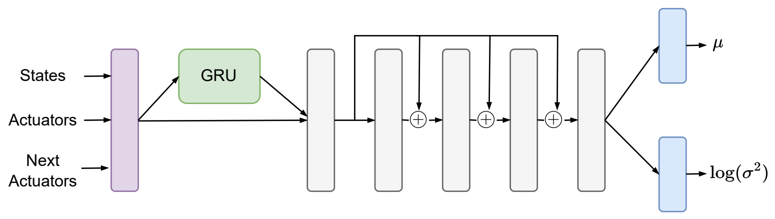


Figure 8.1: **Architecture for the Recurrent Model.** The encoder is a single layer MLP which embeds the states, actuators, and next actuators into a 512 dimensional space. This is fed to the GRU unit which outputs a 128 embedding which is concatenated with the original embedding before being fed to the decoder. The double headed outputs are single linear layers outputting the mean and log variance of a Gaussian. Note the pluses with circles denote a residual connection.

heads, where each head predicts the mean and log variance of a Gaussian distribution, respectively. The negative log likelihood (NLL) is computed with this Gaussian prediction, and the model is trained to optimize the NLL loss. This method of predicting the parameters of a Gaussian distribution via the outputs of a neural network is also known as a mean-variance network or a probabilistic neural network (PNN) [Nix and Weigend, 1994, Lakshminarayanan et al., 2017], and is one of the most widely used methods of modeling predictive uncertainty. We extend our discussion on modeling uncertainty in Section 8.4.1.

Following Chua et al. [2018b], we found it essential to place a soft bound on the log variance using a learned lower and upper bound to ensure stability during training. The difference between the upper and lower bound is then added as an additional penalty term to the loss function, encouraging the width of the bounds to be as small as possible.

With the full dataset of shots available, we dedicate 90% of shots for training, 5% for validation, and 5% for testing. The shots are sorted chronologically before the splits are made, and we ensure that the testing shots consist of the 5% most recent shots. This is essential for testing since experiments (shots) run on the same day tend to be similar. Further, the tokamak is upgraded over time, which alter the dynamics of the plasma. Hence, enforcing the chronological order not only allows us to test the generalization of the learned dynamics model, but also reflects the realistic test setting that a practitioner will be faced with.

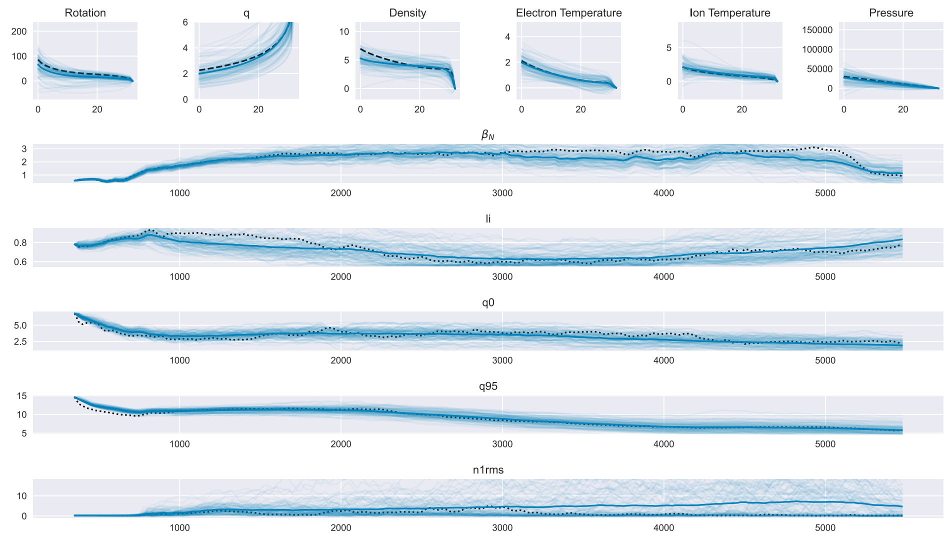


Figure 8.2: **Replay of a Test Shot** The replay was generated with an ensemble of models which sample from their respective Gaussian distribution at each step. While the model has access to the true actuators throughout the entire shot, it only takes in the first true state and autoregressively predicts the rest. The faded blue lines show one sampled trajectory, while the darker blue line shows the average over the trajectories. The black lines show the true values for the experiment. The top row shows the reconstructed profiles at the last time step. Here, the x-axis is over the minor radius of the tokamak, where 0 is the closest to the magnetic axis and 33 is closest to the wall. The other plots show the scalar values over time. The x-axis shows the time into the shot in ms. This shot is part of the test set, and we observe that the model of all signals well excluding n1rms.

8.3.3 Evaluation

To start, we visualize a “replay” of a shot from the test set. That is, we predict the full shot using only the initial state of the plasma and the sequence of actuators. We show the results of this in Figure 8.2, where we find that the model is able to predict the trend across time for the majority of the plasma’s states remarkably well.

To quantitatively evaluate the model’s accuracy, we use *Explained Variance* (EV) as an interpretable metric. This metric is used for the 1D regression setting where, given ground-truth label $y \in \mathbb{R}$ and prediction $\hat{y} \in \mathbb{R}$, the metric is defined

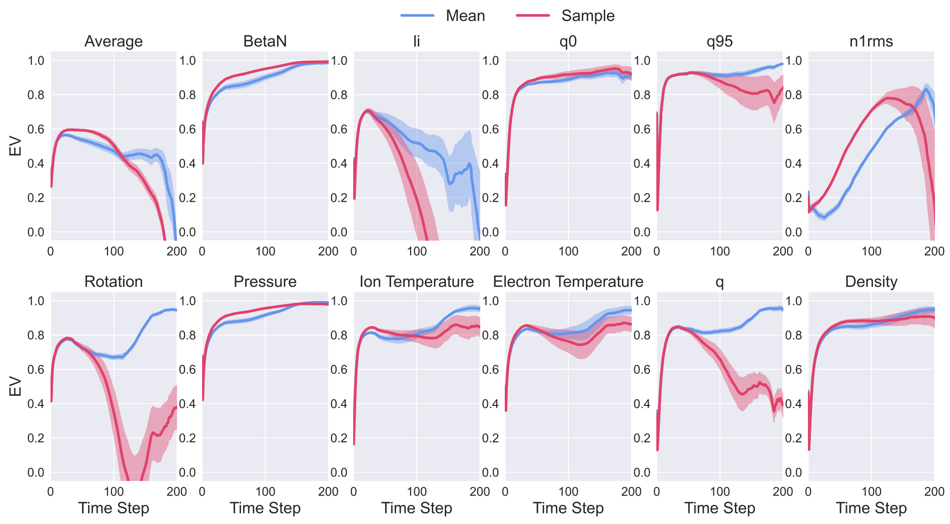


Figure 8.3: **Explained Variance per Time Step.** Each of the colored lines show a different way of generating trajectories with the same models. The blue lines simply take the mean of the Gaussian distribution while the red line samples from the Gaussian distribution at every step. Each curve shows the mean over four different models with different random seeds. The shaded area shows the standard error. In the bottom row, we show the EV for the first principle component of the corresponding profiles.

as

$$\text{EV} := 1 - \frac{\text{Var}\,(y - \hat{y})}{\text{Var}\,(y)} \tag{8.1}$$

Here, $\text{Var}\,(y - \hat{y})$ and $\text{Var}\,(y)$ are the empirical variance of the residuals and labels, respectively. Intuitively, this metric shows how much of the variability in the dataset the model can explain, with the maximum (best) score being 1. Since the plasma's state is multi-dimensional, we compute EV for each of the dimensions and for each time step into the future.

We choose to compute the EV for the difference in the plasma's state at some time step t and the plasma's initial state. With respect to Equation 8.1, we set $y = s_t - s_0$ and $\hat{y} = \hat{s}_t - s_0$, where s_t is a single dimension of interest in the plasma state at time step t, $\hat{s}_t$ is the model's prediction at time step t, and s_0 is the initial state. We believe that this choice in label better aligns with the task of predicting the evolution of the plasma. Indeed, we found if we measure EV of the plasma state (i.e. we do not subtract s_0), EV is nearly perfect for small t since the plasma does not evolve drastically from time step to time step.

For each of the test shots, we evaluate the model by starting prediction at different time steps during the shot (but always seeing the history up to that point), and then autoregressively predicting the remainder of the shot. We assemble the dataset to compute EV using every possible starting time step in all testing shots. We also compare the EV when sampling from the learned distribution versus using only the mean of the Gaussian. For this sampling method, we sample 30 different trajectories and take the mean over them before computing the EV.

Figure 8.3 displays the EV over time for a select set of scalar quantities, the first component of the profiles, and averaged across all input dimensions. We chose these scalar signals because of their significance. In particular, β_N is the normalized ratio between plasma and magnetic pressure and can be used as an indicator of economic performance; q_0 and q_{95} are two points along the q (or safety factor) profile, which is an important indicator of stability of the plasma; and n1rms is the root mean squared of magnetic fluctuations for toroidal mode number $n = 1$, which can signify an event such as a tearing mode in the plasma. We note that we expect n1rms in particular to be hard to predict.

For each of these plots, EV starts low and grows over time, which can be explained by a number of factors. First, there is noise in the system which makes it difficult to predict on the 25ms time scale. However, we hypothesize that the signal starts dominating over the noise after a handful of time steps. In addition, as the number of time steps starts to grow the variance of $s_t - s_0$ continues to grow (i.e. the denominator of Equation 8.1), making it easier to achieve higher EV. In terms of using the mean of the Gaussian versus sampling from it, it seems sampling helps with short term predictions; however, for long term predictions, it appears that using the mean can be more reliable.

8.4 Ablations

In this section, we examine a number of choices that we have made for modeling and inference procedures. Our objective is to provide valuable insights that can assist other practitioners in the field when developing dynamic models of plasma evolution in tokamak devices.

8.4.1 Uncertainty Quantification

Uncertainty quantification is a crucial aspect in any modeling or prediction task, especially in the face of system stochasticity or insufficient data. Adequately

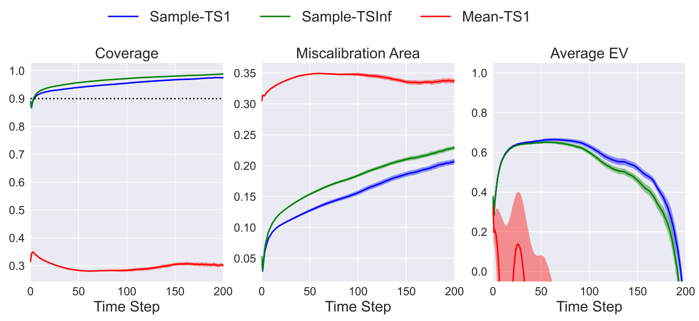


Figure 8.4: **Uncertainty Metrics over Time.** The leftmost plot shows coverage of the 90% prediction interval. Models with good predictive uncertainties should therefore match this 90% (shown as dotted black line). For the miscalibration area plot, the lower the score, the better calibrated the model is. Each of the metrics is averaged over all output dimensions. Moreover the curves show the mean over four models, and the shaded region shows the standard error.

modeling uncertainty has been shown to be especially critical for dynamics models when they are leveraged for control [Chua et al., 2018b].

In our modeling efforts, we account for uncertainty by producing predictive distributions instead of point predictions, and we do so by relying on two different methods: by predicting a Gaussian distribution and by ensembling predictions. These two methods were utilized by Chua et al. [2018b] to capture the aleatoric uncertainty (the uncertainty inherent in the system) and the epistemic uncertainty (the uncertainty stemming from insufficient data), respectively. The ensemble consists of 4 models, each of which have the same model architecture as described in Section 8.3.2, but each model was randomly initialized and trained independently [Lakshminarayanan et al., 2017] on the same dataset.

Inspired by Chua et al. [2018b], we test three methods of generating predictive distributions from our ensemble of networks, each of which predict Gaussian distributions. In the first method, which we denote as "Mean-TS1", we take the mean prediction of the Gaussian distribution, but sample a new model from the ensemble to generate the next state every step. In the other two methods, we sample from the Gaussian distribution, and we either choose to sample a model from the ensemble every step or every shot. We denote these two methods as "Sample-TS1" and "Sample-TSInf", respectively. Because of ensembling and the auto-regressive nature of the model, we do not have a closed form predicted distribution for the plasma's state. Instead, we approximate this distribution with independent Gaussians for each dimension of the plasma's state. The mean and standard deviations are estimated from 30 samples from the dynamics model.

To evaluate the predictive uncertainties, we measure the *Coverage* of a 90% prediction interval (PI) and the so-called *Miscalibration Area*. At a high level, Coverage is the empirical frequency of observations that fall within a constructed PI. Ideally, if the PI is constructed to capture $1-\alpha$ probability mass, $(1-\alpha)\times 100\%$ of the data should fall within this interval. Concretely, given data points $y_n \in \mathbb{R}$ and $(1-\alpha)$ intervals $PI_{n,(1-\alpha)}$ for $n=1,\dots,N$, the $(1-\alpha)$ coverage is defined by

$$\text{Coverage}_{(1-\alpha)} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}\{y_n \in \text{PI}_{n,(1-\alpha)}\}.$$

Building on this, the deviation from the expected probability is regarded as miscalibration, and *Miscalibration Area* takes the average deviation over a set of expected probabilities. Given a set of M expected probabilities drawn uniformly from $[0,1]$: $p_i \sim U[0,1], i \in [M]$, and the observed Coverage_{p_i}, Miscalibration Area is calculated as

$$\frac{1}{M}\sum_{i\in[M]} \mid p_i - \text{Coverage}_{p_i} \mid .$$

Much like EV, these metrics are for single dimensional spaces. As such, we compute these metrics for each of the dimensions of the state space and average the results together to produce a single metric. We also compute both metrics for each time step into the future.

Figure 8.4 shows the Coverage, Miscalibration Area (computed with the "Uncertainty Toolbox" [Chung et al., 2021a]), and EV for the three methods of generating predictive distributions. We see that in the ensemble regime, purely taking the mean of the distribution is detrimental. Not only does the EV suffer, but we also observe extreme overconfidence as shown by the low Coverage ("Mean-TS1" method in Figure 8.4).

Looking at the other two methods, we see that both methods are very well-calibrated at the beginning in their short-term predictions. Moreover, there is an improvement in average EV over the single model case (displayed in the top left plot of Figure 8.3), indicating the significance of utilizing an ensemble approach beyond simply quantifying uncertainty. We find that in all aspects, Sample-TS1 dominates over every other method. This aligns with the suggestion given by Chua et al. [2018b].

As the prediction horizon increases, Miscalibration Area steadily increases, and as evidenced by the Coverage plot, the predictive distributions become increasingly under-confident (i.e. higher than expected coverage). In many cases, one can apply recalibration [Kuleshov et al., 2018] methods to adjust the calibration; however, we are unaware of any such method for the autoregressive setting.

Modelling Choice	**Scaled One-Step MSE**
MLP + Gaussian	1.20
LSTM + Gaussian	1.05
GRU + Point Prediction	1.06
GRU + Gaussian	**1.0**

Table 8.2: **One-Step MSE for Model Choices** We scale the MSE by dividing each score by the best MSE that appearing in the table. That is, 1.0 is the best score, while a score of 1.20 means that the MSE achieved was 20% worse than the best MSE. Each score is the mean over four different seeds.

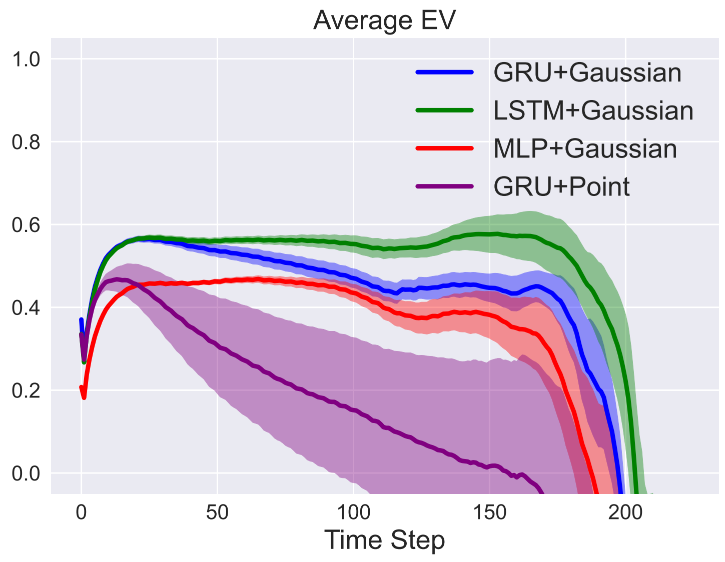


Figure 8.5: **Explained Variance Averaged over All Output Dimensions.** Each curve was generated by taking the average over four different trained models. The shaded area shows the standard error. All curves were generated by taking the mean output of the Gaussian predicted Gaussian distributions (where applicable).

8.4.2 Recurrent Unit

Next, we consider the impact of the recurrent unit chosen. We consider two alternatives besides the GRU component discussed in Section 8.3: a model with no recurrent unit at all (e.g. an MLP) and a model with an LSTM [Hochreiter and Schmidhuber, 1997] unit. From Table 8.2, one can see that the GRU is superior in terms of single step MSE. Indeed, overall we see that GRU seems to be a good choice when looking at shorter horizon predictions. However, from Figure 8.5 it appears that LSTMs are better when considering a longer time horizon. Therefore, one may want to decide on the best recurrent unit based on the downstream application of the dynamics model. In either case, we see that recurrent units are essential since a standard MLP struggles both with one-step MSE and EV.

8.4.3 Point vs Distributional Estimate

Lastly, we look at the effect of having the model learn a distribution, as proposed in Section 8.3, as opposed to having the model output a point prediction and training with MSE loss. Interestingly, even though the models that output a point prediction are trained on MSE, they achieve worse MSE on the test set according to Table 8.2. We hypothesize this is because learning a Gaussian distribution prevents the network from overfitting on the training data. Indeed, we observe that on the training set, models with point predictions achieve roughly 20% lower MSE when compared with those that output Gaussian distributions. On top of this, Figure 8.5 shows that while models with point predictions have decent EV at first, as one predicts further into the future they achieve worse performance than even models with no recurrent units.

8.5 Discussion

In this work, we show that deep recurrent models are a powerful tool that can be used for full shot predictions in tokamak devices. We emphasize that these models were simply given the initial state and the actuators to be applied during the duration of the shot. We encourage the fusion community to leverage data driven models when designing controllers and exploring actuator choices, and we hope that insights shown in this work will prove useful in those pursuits.

Part V

Conclusion

Chapter 9

Conclusion

In this thesis, we explored how reinforcement learning can be used to learn policies for challenging systems in which we can only use surrogate models that are often learned from data. There are many challenges within this broad problem statement, but in this thesis we focused on how to efficiently make use of computationally expensive simulators (Part I), how best to learn policies given erroneous surrogate models (Part II), and how to improve uncertainty quantification in dynamics models (Part III). Along with these algorithmic developments, we also detailed the steps needed in order to learn and run a policy on a real tokamak. I will now give concluding thoughts about the future of offline model-based RL, uncertainty quantification, and how AI can transform tokamak control and nuclear fusion.

Concluding Thoughts on Offline Model-Based Reinforcement Learning

One of the settings that reinforcement has shown the most recent progress in is the setting in which one has a large, diverse dataset at its disposal, possibly where the agent is doing many different tasks. In such a setting, trajectory stitching of the best sub-trajectories (as discussed in Chapter 3), would likely lead to good behavior since the dataset covers much of the state-action space. Rather than explicitly learn a dynamics model and making "stitches" within the MDP, however, the most recent methods instead learn generative models that sample both viable state and actions [Chen et al., 2021a, Janner et al., 2021, Wu et al., 2024]. In particular, these are conditional generative models, and at test time the model can be conditioned on a specific task or "returns-to-go" value to sample an action plan. In doing so, the planning procedure and dynamics modeling are folded into one. While the BATS algorithm itself may not be competitive with these generative modeling methods, the ideas presented in Chapter 3 about how to stitch trajectories together may be useful for improving these generative models as noted in Wu et al. [2024].

This line of research has had some exciting achievements [Reed et al., 2022]; however, I do not think that this methodology will be the single golden bullet for applying reinforcement learning to offline data. For example, the offline dataset we used for tokamak control has a relatively small dataset in comparison, and we would potentially need to make novel sequences of actions rather than stitching together pre-existing behavior from the dataset. Applying generative models may provide reasonable actuator plans for relatively simple shots; however, to get truly sophisticated, real-time control I believe we need to use a model-based reinforcement learning setup like described in Part IV.

Another factor that makes control for nuclear fusion challenging is that it is difficult or impossible to "close the loop" without a vast amount of resources. That is, it is unlikely that one can collect a meaningful amount of additional training data during test-time deployment, and it is also unlikely that repeated train-test cycles will be granted on the device. Many times this is not the case, and indeed, training on additional real data from the system can greatly boost performance of policies [Kaufmann et al., 2023, Nair et al., 2020b].

For these cases in which extremely limited access to the real device is given, I am excited about augmenting offline model-based reinforcement learning to learn adaptive policies. The key idea here (and explored by Chen et al. [2021b]) is to learn a distribution of different models of the environment (which in itself can be viewed as a POMDP) and learn a policy that can adapt and perform well given any model of the environment sampled from this learned distribution. The successful execution of this idea relies on two things: a policy architecture that can quickly adapt yet be robust to modeling errors (Chapter 4) and good epistemic uncertainty that successfully covers the true dynamics function (Chapter 5). While I believe that the works in this thesis are valuable steps towards this goal, I suspect that significant additional advancement is needed in order to realize this on complicated, real-world systems successfully.

Concluding Thoughts on Uncertainty Quantification

As previously mentioned, good epistemic uncertainty will be crucial for sophisticated model-based RL. To make epistemic uncertainty estimates, however, there must be some assumption made. Gaussian processes assume that points in $\mathcal{Y}$ space (i.e. label space) are distributed as a multi-variate Gaussian with a covariance matrix prescribed by the kernel function. In deep learning, a popular choice is to use the distribution of functions formed by an ensemble of neural networks. This implicitly assumes that the set of learnable functions contains the true function and that the distribution of network parameters found translates to a good predictive distribution in $\mathcal{Y}$ space. In practice, this method does well, especially in cases where the training set distribution is the same as the test-time distribution [Lakshminarayanan et al., 2017]; however, in decision making problems this is rarely the case.

Given that queries made at test time will be out of the training distribution, an assumption about how data relates to each other in $\mathcal{Y}$ space may be a more appropriate assumption. This inspired me to turn to neural processes, where the key assumption is that we have access to a distribution of function samples that we can learn from. The proposed models in Chapter 6 strive to make neural processes well-equipped for real-world problems (i.e. sample efficient and robust to test-time distribution shifts). For the future, I am excited to see what alternative assumptions can be made to generate epistemic uncertainty.

9.1 Thoughts on the Future of AI for Tokamaks

The Future of Reinforcement Learning for Tokamak Control

Part IV of this thesis detailed steps for learning a tokamak controller via model-based reinforcement learning. While these were exciting initial steps, there are still many improvements that need to be made in order for reinforcement learning to make an impact on tokamak control. In my opinion, the performance of the controller is currently bottlenecked by the accuracy and sophistication of the surrogate model used for training the policy. In particular, I believe the surrogate model could be improved in the following ways:

- **Addition of Stability Predictors.** Our current version of the surrogate model has no notion of plasma stability. As such, learned neutral beam controllers may push power as high as possible or change the beams rapidly with no consideration of stability. One obvious next step for improvement is to include a tearing mode or disruption predictor [Seo et al., 2024, Fu et al., 2020b, Rea et al., 2019, Keith et al., 2024] that can be leveraged for feedback to the policy.
- **Incorporating Prior Knowledge.** While learning neural network models to predict the evolution of the plasma was key to making reinforcement learning feasible, a fully data-driven surrogate model has weaknesses. Primarily, we do not expect to predict the evolution of the plasma accurately out of distribution if we only rely on data. However, on top of this, the model may learn spurious correlations due to feedback controllers operating on the device. These problems may be ameliorated by including prior information similar to what was done in [Mehta et al., 2021a]. Promising work towards this end has already been started by [Wang et al., 2023, 2024].
- **More Advanced Machine Learning Models.** The model detailed in Chap-

ter 8 is a relatively straight-forward recurrent network. While standard transformer architectures were tried for this problem (but suffered from overfitting to the relatively small dataset), there have been developments in transformers for time series [Wen et al., 2022] that may lead to more accurate predictions.

- **Accounting for Heterogeneity of the Data.** The dynamics models described in Part IV were learned with the assumption that the data is homogeneous. In reality, the DIII-D tokamak changes over time. For example, the neutral beam orientations may change from experiment to experiment (which is information that could be provided to future models). There are also changes that are unobservable from a data perspective, such as upgrades to the device itself. Accounting for this would likely require a more sophisticated architecture that accounts for and infers latent variables within the data. This may be especially challenging because of the relatively limited size of the dataset.
- **Richer Representations of the Plasma.** The dynamics models were trained on states of the plasma as informed by post-processing tools as discussed in Lao et al. [1985]. An alternative route may be to train the model on the raw diagnostic measurements themselves. This would vastly increase the number of features in the data, although it is not clear whether the amount of information would necessarily increase. It would also sidestep any bias introduced by these post-processing tools.
- **Making Actuators More Realistic.** Besides predicting the evolution of the plasma, more work needs to be done to better replicate how the actuators act on the device. A significant amount of work was already done for this with respect to viable power-torque requests for the neutral beams (see Appendix C.5.3). Even with the neutral beams, however, more effort needs to be made to make this as close to the DIII-D device as possible (e.g. coding in the beam dead-zone issues that caused oscillations described in Section 7.4). This will require close collaborations with scientists that are intimately familiar with DIII-D.

Building AI Assistants for Scientists

Besides using reinforcement learning for control, I believe there is great potential for designing an AI agent that can collaborate with scientists for discovery in nuclear fusion. Recently, Mehta et al. [2023a] made a retrieval-augmented-generation (RAG) system that can assist scientists using information from the log information from the DIII-D and Alcator C-Mod tokamaks. These are exciting first steps, but there are two next steps that could make an AI fusion system much more powerful. The first is allowing the language model to search over literature in order to access more information and possibly do hypothesis generation. The second improvement is to allow the language model to be able to use a surrogate model (such as the one proposed in this thesis) as a tool. Doing so would allow the language model to aid in shot predictions, reason about how different actuator settings would affect the plasma, and perform shot planning automatically for the scientist. Beyond nuclear fusion, I believe such agents will revolutionize scientific discovery, and preliminary steps to create such agents have already begun [Bran et al., 2023].

Appendices

Appendix A

Appendix for Chapter 2

A.1 Proof of Theorem 1

We start by defining notation for the following analysis. Let $\mathcal{F}$ be the collection of all possible reward functions (i.e. the support of our prior). Recall that $D_T = \{(x_t, a_t, y_t)\}_{t=1}^T$ is the data sequence collected up to time T. Let $\mathcal{D}_T$ be the collection of all such T-length sequences, and note that the observations seen in this sequence depend on the reward function in question. We also denote $\mathcal{D} = \cup_{t=1}^{\infty} \mathcal{D}_t$. For $f \in \mathcal{F}$ and $D_T \in \mathcal{D}_T$, we define $\lambda(f, D_T)$ as

$$\lambda(f, D_T) := 1 - \frac{\sum_{x \in \mathcal{X}} \omega(x) \left(\max_{a \in \mathcal{A}} f(x,a) - f(x, a_T^*(x))\right)}{\sum_{x \in \mathcal{X}} \omega(x) \left(\max_{a \in \mathcal{A}} f(x,a) - \min_{a \in \mathcal{A}} f(x,a)\right)} \tag{A.1}$$

Note that $\lambda(f, D_T) \geq 0$ for all $f \in \mathcal{F}, D_T \in \mathcal{D}_T$ and that $\max_{D \in \mathcal{D}} \lambda(f, D) = 1$ for all $f \in \mathcal{F}$. We define $D_{T,f}^*$ to be the optimal T-length data sequence with respect to f; that is,

$$D_{T,f}^* := \underset{D_T \in \mathcal{D}_T}{\text{argmax}} \lambda(f, D_T) \tag{A.2}$$

We will often write this sequence as D_T^* when it is clear from context. Note that the optimal data sequence is also the one which makes greedy decisions at every time step. To better understand D_T^*, note that if $T = 1$, the evaluation selected will be the task and its corresponding best action that yields the greatest reward out of any task. If this optimal strategy is continued for more evaluations, each optimal task-action pair will be evaluated in order of descending reward, and after making $|\mathcal{X}|$ such evaluations, $\lambda(f, D_T^*) = 1$. After this, it does not matter which task-action pairs are evaluated.

Because the strategy of Algorithm 1 is to play myopically optimal with respect to a posterior sample, it falls into the broad class of algorithms known as *Myopic*

Posterior Sampling (MPS) [Kandasamy et al., 2019a]. We first restate known properties of these algorithms in the context of our problem.

Condition 1 ([Kandasamy et al., 2019a])**.** *Let $H \in \mathcal{D}$ be any arbitrary set of starting task-action evaluations. For reward functions $f, f' \in \mathcal{F}$ and corresponding optimal data sequences $D^*_{T,f}, D^*_{T,f'}$, there exists sequences $\{\epsilon_T\}_{T\geq 1}$ and $\{\tau_T\}_{T\geq 1}$ such that*

1. *The optimal data sequences achieve asymptotically similar performance:*

$$\sup_{f,f'\in\mathcal{F}} \sup_{H\in\mathcal{D}} \left\{\mathbb{E}\left[\lambda(f, H \cup D^*_{T,f})\right] - \mathbb{E}\left[\lambda(f', H \cup D^*_{T,f'})\right]\right\} \leq \epsilon_T$$

   where the expectations are over the observed rewards.

2. *The rate of convergence is better than $\mathcal{O}(1/\sqrt{T})$. That is, where $\sqrt{\tau_T} = 1 + \sum_{t=1}^{T} \epsilon_t$, we have that $\tau_T = o(T)$.*

Condition 2 ([Kandasamy et al., 2019a])**.** *Let $\mathbb{E}_{Y_{x,a}}$ denote the expectation over the likelihood $Y \sim \mathbb{P}(\cdot|x, a, f)$. Where $j < k$, let $D_j, D_k \in \mathcal{D}$ be such that D_j is a prefix of D_k (i.e. the first j members of D_k make up D_j). For all such $D_j, D_k \in D$, $x \in \mathcal{X}$, $a \in \mathcal{A}$, and $f \in \mathcal{F}$ the following holds:*

1. *λ is monotone, meaning that $\mathbb{E}_{Y_{x,a}}[\lambda(f, D_j \cup \{(x, a, Y_{x,a})\}] \geq \lambda(f, D_j)$.*
2. *λ is adaptive submodular, meaning that,*

$$\begin{aligned}&\mathbb{E}_{Y_{x,a}}[\lambda(f, D_j \cup \{(x, a, Y_{x,a})\})] - \lambda(f, D_j)\\ &\geq \mathbb{E}_{Y_{x,a}}[\lambda(f, D_k \cup \{(x, a, Y_{x,a})\})] - \lambda(f, D_k)\end{aligned}$$

Note that λ as defined in (A.1) satisfies Condition 1. As mentioned before, the optimal strategy $\lambda(f, D^*_{T,f}) = 1$ for $T \geq |\mathcal{X}|$ and $\forall f \in \mathcal{F}$. This is true regardless of the initial data sequence H. Therefore, we see that $\epsilon_T = 0$ for $T \geq |\mathcal{X}|$. Whenever $T < |\mathcal{X}|$, the largest $E\left[\lambda(f, H \cup D^*_{T,f})\right] - \mathbb{E}\left[\lambda(f', H \cup D^*_{T,f'})\right]$ can be is 1 since λ is bounded. Therefore, we can set $\epsilon_T = 1$ for $T < |\mathcal{X}|$. Putting this together,

$$\begin{aligned}\tau_T &= \left(1 + \sum_{t=1}^{T} \epsilon_t\right)^2\\ &\leq \left(1 + \sum_{t=1}^{|\mathcal{X}|-1} 1\right)^2\\ &= (1 + |\mathcal{X}| - 1)^2 = |\mathcal{X}|^2\end{aligned}$$

Thus Condition 1 holds with $\tau_T = |\mathcal{X}|^2$.

Our definition of λ also meets the requirements of Condition 2. First of all, λ is monotonically increasing because it relies on the maximum reward seen, and therefore λ can only increase after seeing new data. λ is also adaptive submodular since any improvement from seeing a new evaluation can only be more impactful when less data has been seen. For intuition, consider any evaluation $\{x, a, y\}$. There are two cases:

1. $\{x, a, y\}$ is better than any other evaluation for task x in D_k. Therefore D_j and D_k will both have the same maximum played reward for task x; however, the previous maximum may have been greater for D_k since it is a superset of D_j. Thus, the increase in λ must be greater or equal for D_j.
2. $\{x, a, y\}$ is not best evaluation for task x in D_k. In this case,

$$\lambda(f, D_k \cup \{(x, a, y\}) - \lambda(f, D_k) = 0$$

and the condition holds.

Theorem 4 ([Kandasamy et al., 2019a])**.** *Assume that λ satisfies conditions 1 and 2, and let τ_T be as defined in Condition 1. Let D_T be data collected by playing myopically optimal according to posterior samples. Then, for all $0 < \rho < 1$,*

$$\mathbb{E}\left[\lambda(f, D_T)\right] \geq (1-\rho)\mathbb{E}\left[\lambda(f, D^*_{\rho T})\right] - \sqrt{\frac{|\mathcal{X}||\mathcal{A}|\tau_T\gamma_T}{2T}}$$

Using this theorem, the proof for Theorem 1 is relatively straightforward.

Proof *(Theorem 1).* Note that Theorem 4 can be used because Algorithm 1 optimizes λ as defined in (A.1) with respect to posterior samples,

$$\begin{aligned}\mathbb{E}\left[\lambda(f, D^*_{\rho T})\right] - \mathbb{E}\left[\lambda(f, D_T)\right] &\leq \sqrt{\frac{|\mathcal{X}||\mathcal{A}|\tau_T\gamma_T}{2T}} + \rho\mathbb{E}\left[\lambda(f, D^*_{\rho T})\right] \\ &\leq |\mathcal{X}|\sqrt{\frac{|\mathcal{X}||\mathcal{A}|\gamma_T}{2T}} + \rho\end{aligned}$$

In the case where $T > |\mathcal{X}|$, we can set $\rho = \frac{|\mathcal{X}|}{T}$. In this case,

$$\begin{aligned}\mathbb{E}\left[\lambda(f, D^*_{|\mathcal{X}|})\right] - \mathbb{E}\left[\lambda(f, D_T)\right] &= 1 - \mathbb{E}\left[\lambda(f, D_T)\right] \\ &= \mathbb{E}\left[\frac{\sum_{x\in\mathcal{X}}\omega(x)\left(\max_{a\in\mathcal{A}} f(x,a) - f(x, a^*_T(x))\right)}{\sum_{x\in\mathcal{X}}\omega(x)\left(\max_{a\in\mathcal{A}} f(x,a) - \min_{a\in\mathcal{A}} f(x,a)\right)}\right] \\ &\leq |\mathcal{X}|\sqrt{\frac{|\mathcal{A}||\mathcal{X}|\gamma_T}{2T}} + \frac{|\mathcal{X}|}{T}\end{aligned}$$

If $T \leq |\mathcal{X}|$,

$$\mathbb{E}\left[\frac{\sum_{x\in\mathcal{X}}\omega(x)\left(\max_{a\in\mathcal{A}} f(x,a) - f(x, a^*_T(x))\right)}{\sum_{x\in\mathcal{X}}\omega(x)\left(\max_{a\in\mathcal{A}} f(x,a) - \min_{a\in\mathcal{A}} f(x,a)\right)}\right] \leq 1 < |\mathcal{X}|\left(\frac{1}{T} + \sqrt{\frac{|\mathcal{A}||\mathcal{X}|\gamma_T}{2T}}\right)$$

Therefore, the theorem holds. □

A.2 Random Function Generation

In order to run experiments in which there are many different optimization problems, we consider drawing problems at random from a particular class of functions. These functions are over the domain $[0, 1]^d$ and have the following form:

$$f_{m,s,c,b}(x) = \sum_{k=1}^{m} \sum_{i=1}^{d} s_i \exp\left[-\frac{(x_i - c_{k,i})^2}{b_{k,i}} \right]$$

where m is the number of point masses, s_i are the scales, $c_{k,i}$ control the centers of the masses, and $b_{k,i}$ are the bandwidths. These quantities are generated at random in order to construct a suite of different functions. To ensure varying difficulty in the 30 randomly drawn functions, we drew 15 "easy" functions, 10 "average" functions, and 5 "hard" functions. These classes of difficulty are characterized by the possible values the parameters of f can take (except for the centers of the mass, which can be anywhere in $[0, 1]^d$). To sample a function, each parameter is drawn uniformly at random over the support given in Table A.1.

Difficulty	Number of Masses	Scale Range	Bandwidth Range
Easy	$\{0, 1, 2\}$	$[0, 1]$	$[0.7, 0.9]$
Average	$\{3, 4\}$	$[0.25, 1]$	$[0.4, 0.6]$
Hard	$\{5, 6, 7\}$	$[0.5, 1]$	$[0.25, 0.4]$

Table A.1: Ranges of parameters for randomly generated function by difficulty class.

A.3 Correlated Tasks of Varying Difficulty

For correlated tasks of varying difficulty we use the function $g : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$, where $\mathcal{X} = [0, 1]$ and $\mathcal{A} = [0, 1]$. From this, we discretize the problem by choosing ten equispaced tasks from $\mathcal{X}$. The function g is defined as follows:

$$g(x, a) = \sum_{m=1}^{M} s_m \exp\left[\frac{-(a - c_{a,m})^2}{b_{a,m}} + \frac{-((|x - 0.5| - c_{x,m})^2}{b_{x,m}} \right] \qquad \text{(A.3)}$$

where the specific quantities are shown in the table below.

m	s_m	$c_{a,m}$	$b_{a,m}$	$c_{x,m}$	$b_{x,m}$
1	15	0.1	0.005	0.45	0.01
2	-15	0.2	0.005	0.45	0.01
3	16	0.3	0.005	0.45	0.01
4	-6	0.4	0.005	0.45	0.01
5	7	0.5	0.005	0.45	0.01
6	-7	0.6	0.005	0.45	0.01
7	16	0.7	0.005	0.45	0.01
8	-16	0.8	0.005	0.45	0.01
9	15	0.9	0.005	0.45	0.01
10	4	0.1	0.05	0.25	0.01
10	-4	0.3	0.075	0.25	0.01
11	8	0.6	0.05	0.25	0.01
12	-4	0.8	1	0.25	0.01
13	4	0.1	0.05	0.15	0.0025

Table A.2: Parameters for (A.3)

A.4 Nuclear Fusion Experiments Preliminaries

For our nuclear fusion experiments in Section 2.5, we use the TRANSP program to simulate runs on the DIII-D tokamak. TRANSP is a time-dependent transport code used for interpretive analysis and predictive simulations of tokamaks, and DIII-D is a tokamak in San Diego that is operated by General Atomics. Access to TRANSP and running TRANSP experiments were possible thanks to our collaborators at Princeton Plasma Physics Lab. We run the predictive module of TRANSP (PTRANSP) where each experiment on TRANSP (referred to as ”run”) simulates a fusion experiment already conducted on DIII-D (referred to as ”shot”) while predicting how changes in the input parameters would affect the plasma state. When conducting a run on a given shot, at each time step of the run, we can identify task variables that correspond to the state of the plasma at the time during the experiment. These variables can include β_n, initial total energy eigenvalues, total pressure, plasma current, etc.

β_n is a ratio of the pressure of the plasma to the magnetic energy density, and thus represents how effectively the plasma is confined during the shot. It is essentially a proxy for the economic output of the fusion reaction. Total energy eigenvalues represent the amount of change in energy within and outside the plasma due to certain perturbations, and the minimum value of the total energy eigenvalues (which we will refer to as ”minimum eigenvalue”) indicates the stability of the plasma.

When conducting a run (i.e. experiment on TRANSP), we can apply controls that specify parameters of the neutral beams, which include power, energy, full energy fraction and half energy fraction. DIII-D has a total of 8 physical neutral beams, 6 of which are co-current beams (which inject in the same direction as the plasma current) and 2 of which are counter-current beams (inject in the opposite direction of the plasma current). In our experiments, we confine the action space to 2 dimensions: power coefficient of co-current beams and power coefficient of counter-current beams, each with domains [0.001, 1.0]. These power coefficients are applied by multiplying the maximum power of the set of beams by the coefficient. By ranging the power coefficient from 0.001 to 1.0, we essentially scale the beam powers from the minimum to the maximum power level possible.

After each TRANSP run, we can extract a measure of stability of the plasma at each time step of the run, which is the minimum eigenvalue. The shots that we are concerned with in our experiments are those in which a common instability called ”tearing” occurred. This occurs when the layers of magnetic fields that confine the plasma rip apart and reform to create magnetic field islands, which causes disturbances in the plasma. Therefore, in our experiments, we apply our designated controls before tearing occurs and try to maximize the stability of the plasma after the time of tearing.

Preliminary experiments have shown that purely optimizing the stability (minimum eigenvalue) will always decrease the power of all beams, which is not desirable because this will also dissipate the plasma. Therefore, we add β_n to our objective in order to steer the optimization towards maintaining high reaction output while stabilizing the plasma. Hence, our optimization objective is $C_1\epsilon_m + C_2\beta_n$, where C_1, C_2 are constants and ϵ_m is the minimum eigenvalue.

A.5 Nuclear Fusion Experiment Setup

The 8 shots selected to comprise the independent tasks in the experiments in Section 2.5 are shots 145699, 149205, 149689, 153145, 155215, 162939, 170473, 171315 on the DIII-D tokamak. All of these shots had a tearing instability that terminated the fusion reaction soon after the occurrence of the instability. The plasma state that is of interest then is the state of the plasma shortly before tearing occurred. Ideally, we would like to identify these states that will quickly turn unstable and apply controls to prevent the instability.

Therefore, in our optimization experiments, the tasks are identified by the state of the plasma 50 milliseconds (ms) before tearing time. Each query of a set of controls corresponds to a run (simulation experiment on TRANSP) from 50ms before tearing to 50ms after tearing time. After the run completes, we extract the minimum eigenvalues ϵ_m and β_n values at 5ms increments throughout the duration of the run and average them to produce $\bar{\epsilon}_m$ and $\bar{\beta}_n$. Because the scales of $\bar{\epsilon}_m$ and $\bar{\beta}_n$ are very different, we scale them to be roughly the same scale. The sum of these two scaled terms is the optimization objective, which signifies maximizing the stability of the plasma $(\max \bar{\epsilon}_m)$ while maximizing the reaction efficiency, or economic output $(\max \bar{\beta}_n)$. The objective used in our experiments was $10\bar{\beta}_n + 100\bar{\epsilon}_m$.

The optimization experiment results presented in Section 2.5.1 are averaged over 10 trials, each with 125 query capital. In each trial, for each task, 5 initial points are drawn uniformly at random for evaluation. Each task is modeled by a GP with an RBF kernel, and hyperparameters are tuned for a GP every time an observation is seen for its corresponding task by marginal likelihood.

Optimization was asynchronously parallelized with 20 workers. Kandasamy et al. [2018] proposed parallelized versions of standard Thompson sampling, and the algorithms used for the fusion experiment were the asynchronous Thompson sampling from Kandasamy et al. [2018] and the analougous parallel version of MTS.

Appendix B

Appendix for Chapter 3

B.1 The BATS Algorithm

Below we state the steps of the BATS algorithm. Algorithm 2 shows the main loop, while Algorithm 3 shows the subroutines.

Algorithm 2 The BATS Algorithm

```
**BATS Input:** Offline Dataset $D = \bigcup_{j\in[M]}\{(s_{ji}, a_{ji}, s'_{ji}, r_{ji})\}_{i=1}^{t_j}$, Max Stitching Length $K$, Number of Iteration $n$, Number of Samples per Iteration $m$, Neighborhood Radius $\epsilon$, Planning Tolerance $\delta$, Discount Factor $\gamma$, Penalty Coefficient $c$, and Distance Metric $d$
$M_0 = \langle \mathcal{S}_0, \{\mathcal{A}_0^s\}_{s\in\mathcal{S}_0}, \gamma, T_0, r_0, \rho_0 \rangle$
Learn dynamics estimate, $\tilde{T}$, and reward estimate, $\tilde{r}$ from $D$
for $i = 0, 1, \ldots, (n-1)$ do
  $\hat{V}_i^*(\cdot), \pi_i(\cdot) \leftarrow$ valueIteration$(M_i)$
  neighbors $\leftarrow \mathrm{getNeighbors}(\mathcal{S}, \pi, M_i, \epsilon)$
  $M_{i+1} \leftarrow M_i$
  for $j = 1, 2, \ldots, m$ do
    $s \sim \mu_{M_i}(s \mid \pi_i)$
    $E \leftarrow \mathrm{getCandidateEdges}(s, M_i,$ neighbors$, j)$
    for $(s, s') \in E$ do
      actions = $\mathrm{stitch}((s, s'), K, \delta)$
      $s'' = s$
      for $a \in$ actions do
        $\mathcal{A}_{i+1}^{s''} \geq \{a\} \cup \mathcal{A}_{i+1}^{s''}$
        if $a$ is last action in actions then
          $T_{i+1}(s'', a) \leftarrow \tilde{T}(s'', a)$
          $r_{i+1}(s'', a) \leftarrow \tilde{r}(s'', a) - cd\left(\tilde{T}(s'', a), T(s'', a)\right)$
          $s'' \leftarrow \tilde{T}(s'', a)$
          $\mathcal{S}_{i+1} \leftarrow \{s''\} \cup \mathcal{S}_{i+1}$
        else
          $T_{i+1}(s'', a) \leftarrow s'$
          $r_{i+1}(s'', a) \leftarrow \tilde{r}(s'', a) - cd\left(\tilde{T}(s'', a), s'\right)$
        end if
      end for
    end for
  end for
end for
$V(\cdot), \pi(\cdot) \leftarrow$ valueIteration$(M_n)$
return $M_n$
```

Algorithm 3 Subroutines for BATS

```
getCandidateEdges Input:$s, t, \hat{M}$, neighbors, $j$
if k == 0 then
    return []
end if
edges = []
vertexNeighbors $\leftarrow$ neighbors$[t]$
MDPNeighbors $\leftarrow \hat{M}[s, \hat{\mathcal{A}}_s]$
for $n \in$ vertexNeighbors do
    edges += getFutureEdges($s$, $n$, $\hat{M}$, $j$)
end for
for $n \in$ MDPNeighbors do
    successorNeighbors $\leftarrow$ neighbors$[t]$
    for $n' \in$ successorNeighbors do
        neighbors += [($s$, $n'$)]
    end for
    neighbors += getCandidateEdges$(s, n, \hat{M}$, neighbors, $j - 1)$
end for
return edges
getFutureEdges Input:$s, n, \hat{M}, j$
MDPNeighbors $\leftarrow \hat{M}[s, \hat{A}_s]$
edges $\leftarrow$ []
for $n' \in$ MDPNeighbors do
    edges += $[(s, n')]$
    edges += getFutureEdges$(s, n', \hat{M}, j - 1)$
end for
return edges
stitch Input:e, k, $\epsilon$
minDistance $\leftarrow \infty$
bestActions $\leftarrow$ []
for i = 1:j do
    distance, actions $\leftarrow$ CEM$(e, k)$
    if distance ¡ minDistance then
        minDistance $\leftarrow$ distance
        bestActions $\leftarrow$ actions
    end if
end for
if minDistance ¡ $\epsilon$ then
    return []
end if
return bestActions
```

B.2 Additional Explanation of Bisimulation

In this work, we make use of *bisimulation metrics* as introduced by Givan et al. [2003] in order to guarantee accurate value function estimates in our stitched graph. Informally, bisimulation metrics compare states based solely on their expected future reward distribution and thus ignore information that does not affect the reward. Specifically, they are constructed in Ferns et al. [2012] as a generalization of the following equivalence relation:

Definition 1. *$\sim$ is a bisimulation relation if $s \sim s'$ implies that $\forall a \in \mathcal{A}, r(s,a) = r(s',a)$ and $T(s,a) \sim T(s',a)$.*

This means that two states are *bisimilar* (related via $\sim$) if they attain the same one-step rewards and their future state distributions also return the same under the same actions. Intuitively, this relation serves to ignore attributes of the state space which are not related to the returns attained executing actions. This relation and its derived equivalence classes group together the states in which the agent receives the same rewards under the same actions for an arbitrary choice of actions. However, the restriction that states are only similar if they give the same future rewards under any sequence of actions is quite strong–if an obviously bad action executed in a pair of otherwise bisimilar states gives different rewards which are both bad, those states will not be bisimilar. Any competent reinforcement learning algorithm will not take an obviously bad action, so the states will be indistinguishable for RL purposes.

To rectify this, Castro [2019] give a coarser bisimulation relation and associated metric which is based on the actions a particular policy would take:

Definition 2. *The on-policy bisimulation relation $\sim_\pi$ for a stochastic policy π is the strongest relation such that $s \sim_\pi s'$ iff $\sum_a \pi(a \mid s) r(s,a) = \sum_a \pi(a \mid s') r(s',a)$ and $\forall C \in S/\sim_\pi$, $P^\pi(C \mid s) = P^\pi(C \mid s')$ where $S/\sim_\pi$ is the partition induced by $\sim_\pi$ and $P^\pi(C \mid s) = \sum_{a \in \mathcal{A}} (\pi(a \mid s) \mathbf{1}[T(s,a) \in C])$.*

These works also give metrics which relax these equivalence classes, preserving a notion of approximate bisimilarity. In our work, we use the *on-policy bisimulation distance* from Castro [2019], which also gives a sampling-based algorithm for approximating such a metric, which we'll denote $d^\pi_\sim(\cdot,\cdot)$. As one might imagine, it turns out that this distance is closely related to the value function for π by the following theorem from the paper:

Theorem 5 (Theorem 3 from Castro [2019])**.** *Given any two states $s, t \in \mathcal{S}$ in an MDP M and a policy π, $|V^\pi(s) - V^\pi(t)| \leq d^\pi_\sim(s,t)$.*

This result gives us a metric over states for which the value function is 1-Lipschitz continuous. This property allows us to unify the graphical perspective of stitching we take with the more traditional value function approach to RL.

There is an extensive literature on bisimulation, and further discussion, theory, and empirical investigation can be found in Ferns et al. [2011, 2012], Zhang et al. [2020], Castro [2019].

B.3 Proof of Theorem 3

We prove theorem 3 in this section. The proof proceeds by taking the infinite expansion of a value function and correcting for planning errors using Theorem 2. We can do this for each transition which is added by BATS instead of being take from the dataset, allowing us to compare the upper and lower bounds with their true values.

Proof. WLOG, suppose that for a fixed $z \leq \ell$, $\pi^-(b_i) = a_i$ if $i \leq z$ and $\pi^-(b_i) \neq a_i$ if $z < i \leq \ell$. In other words, the π^- chooses to take advantage of z of the stitches made. Note that z can possibly be 0, but in this case the theorem holds trivially.

Let T^- be the transition function in M^-, and let $s_0, s_1, \ldots$ be the infinite sequence of states that π^- visits in M^- starting from $s_0 = s$ and where $\hat{s}_i = T^-(\hat{s}_{i-1}, \pi^-(\hat{s}_{i-1}))$. Let τ be the ordered set hitting times of the states where π^- uses a stitched transition plus 0 and ∞, i.e. $\tau = \{0\} \cup \{t | t \in \mathbb{N}^+ \text{ s.t. } s_t \in \{b_i\}_{i=1}^z\} \cup \{\infty\}$, and let t_i be the i^{th} sorted element of τ.

We can expand the value function $V(s)$ as follows:

$$\begin{aligned} V_M^{\pi^-}(s) &= \sum_{i=0, s_0=s, s_i=T(s_{i-1}, \pi^-(s_{i-1}))} \gamma^i r(s_i, \pi^-(s_i)) \\ &= \sum_{t_i \in \tau} \left(\sum_{j=t_i, s_j=T(s_{j-1}, \pi^-(s_{j-1}))}^{j=t_{i+1}-1} \gamma^j r(s_j, \pi^-(s_j)) \right). \end{aligned}$$

Leveraging our mapping, ϕ, and Theorem 2, note that

$$\begin{aligned} V_M^{\pi^-}(s_{t_i}) &= r(s_{t_i}, \pi(s_{t_i})) + \gamma V_M^{\pi^-}(s_{t_i+1}) \\ &= r(s_{t_i}, \pi(s_{t_i})) + \gamma V(s_{t_i+1}) + \gamma V_M^{\pi^-}(c_{t_i+1}) - \gamma V_M^{\pi^-}(c_{t_i+1}) \\ &\leq r(s_{t_i}, \pi(s_{t_i})) - \gamma \left\| \phi(s_{t_i+1}) - \phi(c_{t_i}) \right\| + \gamma V_M^{\pi^-}(c_{t_i}) \\ &\leq r(s_{t_i}, \pi(s_{t_i})) - \gamma \epsilon_{t_i} + \gamma V_M^{\pi^-}(c_{t_i}) \end{aligned}$$

We can apply this inequality at each hitting time to get the below:

$$V_M^{\pi^-}(s) \geq \sum_{t_i \in \tau} \left(\gamma^{t_i} r(s_{t_i}, \pi^-(s_{t_i})) - \gamma^{t_i+1} \epsilon_i + \sum_{j=t_i+1, s_{t_i}=c_{t_i}, s_j=T^-(s_{j-1}, \pi^-(s_{j-1}))}^{j=t_{i+1}-1} \gamma^j r(s_j, \pi^-(s_j)) \right)$$

$$V_M^{\pi^-}(s) \leq \sum_{t_i \in \tau} \left(\gamma^{t_i} r(s_{t_i}, \pi^-(s_{t_i})) + \gamma^{t_i+1}\epsilon_i + \sum_{j=t_i+1, s_{t_i}=c_{t_i}, s_j = T^-(s_{j-1}, \pi^-(s_{j-1}))}^{j=t_{i+1}-1} \gamma^j r(s_j, \pi^-(s_j)) \right).$$

Note that, by construction, these lower and upper bounds equal the value functions in the MDPs M^- and M^+, respectively.

$$V_{M^-}^{\pi^-}(s) = \sum_{t_i \in \tau} \left(\gamma^{t_i} r(b_{t_i}, \pi^-(b_{t_i})) - \gamma^{t_i+1}\epsilon_i + \sum_{j=t_i+1, s_{t_i}=c_{t_i}, s_j = T^-(s_{j-1}, \pi^-(s_{j-1}))}^{j=t_{i+1}-1} \gamma^j r(s_j, \pi^-(s_j)) \right) \tag{B.1}$$

and

$$V_{M^+}^{\pi^-}(s) = \sum_{t_i \in \tau} \left(\gamma^{t_i} r(b_{t_i}, \pi^-(b_{t_i})) + \gamma^{t_i+1}\epsilon_i + \sum_{j=t_i+1, s_{t_i}=c_{t_i}, s_j = T^-(s_{j-1}, \pi^-(s_{j-1}))}^{j=t_{i+1}-1} \gamma^j r(s_j, \pi^-(s_j)) \right). \tag{B.2}$$

Combining the above gives the desired result for our arbitrary s:

$$V_{M^-}^{\pi^-}(s) \leq V_M^{\pi^-}(s) \leq V_{M^-}^{\pi^-}(s).$$

□

B.4 Experiment Details

B.4.1 Hyperparameters and Training Procedure

Dynamics Models. To learn dynamics models, we use the architecture introduced by Chua et al. [2018a], and follow the procedure described in Yu et al. [2020], making a few minor changes. Like Yu et al. [2020], we train seven different dynamics models and take the best five based on a validation set of 1,000 points. Each model is a neural network with 4 hidden layers of size 200 and 2 heads at the end: one predicting mean and one predicting log-variance. These models are trained using batches of size 256 and using negative log likelihood as the loss. We use ReLU for our hidden activation function, and unlike Yu et al. [2020], we do not use spectral normalization. Following their procedure, we use a validation set of one thousand points to select the best model to use after training.

BATS. For each of the experiments in Section 3.7, we perform BATS three times, using different learned dynamics models, to produce three different stitched MDPs. When using CEM to plan new stitches, we use the mean output of each member of the dynamics model and check if the 80^{th} quantile is under some planning threshold. This planning threshold was set to 0.425 with for the mazes and 10 (after normalization) for Mujoco tasks. All experiments imposed a restriction of $k = 1$ number of actions that could be taken for stitching, except for halfcheetah which we set to $k = 5$. Additionally, when forming the nearest neighbor graph for finding potential stitches, we consider neighbors up to 0.225 away for umaze and medium maze and 0.15 away for large maze. These were set as large as memory constraints would allow for. For Mujoco tasks we found it easier to instead use the 25 closest neighbors.

We assume for these experiments that we have access to the start state distribution. For Mujoco tasks we simply label the beginning of trajectories in the logged dataset as start states. Since there is only one trajectory for each of the maze tasks, we label every state that is in the support of the start state distribution as a start state. The large maze dataset does not contain possible start states for all cells. For cells in which there is not a start state in the dataset, we widen the distribution slightly so that enough starts are included.

For the maze tasks we attempt $50,000$ stitches every iteration, and we run BATS for 10 iterations for umaze and medium maze and 20 iterations for the large maze. For the Mujoco experiments, we attempt to make $5,000$ stitches ever iteration, and we run BATS for 40 iterations.

To increase exploration for the stitches to consider, we apply Boltzmann exploration when selecting next actions to perform in the stitched MDP. That is, we select actions according to:

$$\mathbb{P}(a|s) =\propto \exp\left(Q(s,a)/T\right)$$

where T is a temperature parameter, which we choose to set to 0.25 for all experiments.

After running BATS, we searched for good hyperparameters by relabeling or removing stitched edges accordingly. The best found parameters are shown in Table B.1. When looking at the distribution of returns from trajectories in the resulting MDPs, there is a clear value for the returns that separates successfully stitched trajectories from those that were not able to be stitched to high value areas. As such, we only behavior clone on trajectories above 100 (umaze), 200 (medium maze), 300 (large maze), $1,000$ (hopper and walker2d), and $4,000$ (halfcheetah). These thresholds were selected by inspection.

Mountaincar.

For the mountain car example, we found that doing 20 iterations of stitching (each trying to make 100 stitches) was sufficient. We use the nearest 25 neighbors to determine which stitches can be made, and we allow for up to $k = 5$ actions to be used when stitching between states. We found that smaller dynamics models were sufficient for this problem, and in particular, each member of the ensemble had 3

	Mazes			Mujoco Mixed Tasks		
	umaze	**medium**	**large**	**hopper**	**walker2d**	**halfcheetah**
Planning Error Threshold	0.425			2.25	0.5	2
Penalty Coefficient	20		10		50	
Policy Layer Sizes	$64, 64$	$256, 256$	$256, 256, 256$	$256, 256$		
Batch Size	256					
Batch Updates	$10,000$	$20,000$		$10,000$		

Table B.1: **Table of BATS Hyperparameters.**

layers with 64 hidden units each. Lastly, we set the the temperature for Boltzmann exploration to $T = 0.1$. We behavior clone using a policy network with two hidden layers with 256 units each.

B.4.2 Bisimulation Implementation

The model architecture that we use for bisimulation is that of Zhang et al. [2020]. That is, we have a network that takes state observations as inputs and outputs a latent representation, and we have a dynamics model that operates on this latent representation. For the encoder network, we use three hidden layers with 256, 128, and 64 units, respectively, and we set the latent dimension to be 6. Unlike the model in Zhang et al. [2020]. however, we have one network that predicts both next transitions in bisimulation space and next rewards (the same dynamics model as described in Appendix B.4.1). We also use the same loss function as described in Zhang et al. [2020]. In particular, we draw batches of pairs of state observations and optimize according to

$$J(\phi) = \left(\|z_i - z_j\| - |r_i - r_j| - \gamma W_2 \left(\hat{P}(\cdot|\bar{z}_i, a_i), \hat{P}(\cdot|\bar{z}_j, a_j) \right) \right)^2$$

where z_k, r_k, a_k are the latent encoding, the predicted reward, and the observed action for the k^{th} sample, respectively. $\hat{P}$ is the learned dynamics model for the latent space, and we use a bar over z to signify that we stop gradients. For more details, please refer to Zhang et al. [2020]. Although in their work they iteratively update their model to reflect a changing policy, in our work we train with respect to a fixed policy. The on-policy nature of our training procedure resembles Castro [2019].

B.4.3 COMBO Implementation

Because the original COMBO [Yu et al., 2021] paper did not include results on the maze environments and does not yet have a public implementation, we made our best attempt at reimplementing their method in order to properly compare results. For running COMBO, we based our dynamics model training on Yu et al. [2020] and used all of their associated hyperparameters. For the conservative Q-learning and policy learning components, we mostly followed the public implementation of Kumar et al. [2020], but had to make some small tweaks to make it consistent with the descriptions in Yu et al. [2021].

For the COMBO hyperparameters, we did a grid search over conservative coefficient β in $\{0.5, 1.0, 5.0\}$ and rollout length h in $\{1, 5\}$ for all the maze tasks. We found the best parameters to be $h = 1, \beta = 1$ on umaze, $h = 1, \beta = 0.5$ on medium, and $h = 1, \beta = 0.5$ on large. For all the other hyperparameters, we followed the halcheetah parameters used in Yu et al. [2021]. Specifically, 3-layer feedforward neural networks with 256 hidden units for Q-networks and policy, Q-learning rate $3.e-4$, policy learning rate $1.e-4$, $\rho(a|s)$ being the soft-maximum of the Q-values and estimated with log-sum-exp, $\mu(a|s) = \pi(a|s)$, and deterministic backups.

Appendix C

Appendix for Chapter 4

C.1 Additional GPIDE Details

C.1.1 Forming the P, I, and D Features with GPIDE

As mentioned in Section 8.3, the P, I, and D features associated with a PID controller can be easily formed with GPIDE, and in this Appendix we show this concretely. Assume that the observations take the form $o_t = \left(x_t^{(1)}, \dots, x_t^{(M)}, \sigma_t^{(1)}, \dots, \sigma_t^{(M)}\right)$, and assume that the actions and rewards are not given to GPIDE for simplicity. Then the linear projection for each head, f_θ^h, takes in $x_{t-1}^{(1)}, \dots, x_{t-1}^{(M)}, \sigma_{t-1}^{(1)}, \dots, \sigma_{t-1}^{(M)}, (x_t^{(1)} - x_{t-1}^{(1)}), \dots, (x_t^{(M)} - x_{t-1}^{(M)}), (\sigma_t^{(1)} - \sigma_{t-1}^{(1)}), \dots, (\sigma_t^{(M)} - \sigma_{t-1}^{(M)})$ at each time step t. We use a GPIDE architecture with three heads and where g_θ is the identity function. The linear projections for each head is as follows:

$$f_\theta^1(o_{t-1}, o_t - o_{t-1}) = f_\theta^2(o_{t-1}, o_t - o_{t-1}) = \begin{bmatrix} (x_t^{(1)} - x_{t-1}^{(1)}) + x_{t-1}^{(1)} - (\sigma_t^{(1)} - \sigma_{t-1}^{(1)}) + \sigma_{t-1}^{(1)} \\ \vdots \\ (x_t^{(M)} - x_{t-1}^{(M)}) + x_{t-1}^{(M)} - (\sigma_t^{(M)} - \sigma_{t-1}^{(M)}) + \sigma_{t-1}^{(M)} \end{bmatrix}$$

$$f_\theta^3(o_{t-1}, o_t - o_{t-1}) = \begin{bmatrix} (x_t^{(1)} - x_{t-1}^{(1)}) - (\sigma_t^{(1)} - \sigma_{t-1}^{(1)}) \\ \vdots \\ (x_t^{(M)} - x_{t-1}^{(M)}) - (\sigma_t^{(M)} - \sigma_{t-1}^{(M)}) \end{bmatrix}$$

Note that f_θ^1 and f_θ^2 form the current error at time t (i.e. the P term), and f_θ^3 forms the change in error (i.e. the D term). For accumulation strategies, using exponential smoothing with $\alpha = 1$ for the first and third heads and a summation head for the second head will recover the P, I, and D terms for heads 1, 2, and 3, respectively.

Note that the above assumes $dt = 1$, but the linear projections can be adjusted to take different dt values into account.

C.2 Implementation Details

Code Release All code for implementations are provided in the supplemental material along with instructions for how to run experiments. The code can also be found at `https://github.com/IanChar/GPIDE`. The only experiment that cannot be run are the "real" cases for tokamak control.

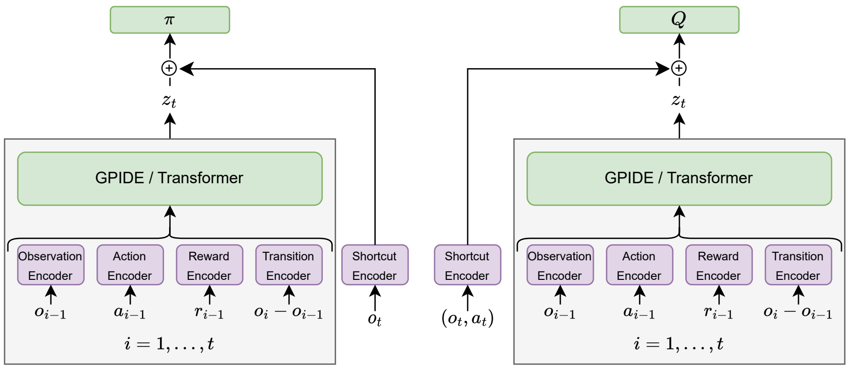


Figure C.1: **General Policy and Q Function Architectures.** This architecture is heavily inspired by Ni et al. [2022]. The gray box shows the history encoder modules, and this is the only thing that changes between baseline methods in the tracking problems. Note that there are two encoders: one for the policy function and one for the Q value function. The purple boxes show the input encoders, and hyperaparmeters for these can be found in Table C.2. We found the shortcut encoders to be essential to good performance. The architecture when using GRU is nearly identical; however, there is no "Transition Encoder" since Ni et al. [2022] encodes (o_i, a_{i-1}, r_{i-1}) for each time step instead.

Architecture We use the same general architecture for each of the RL methods in this paper (see Figure C.1). Each input to the history encoders, policy functions, and Q-value functions have corresponding encoders. This setup closely follows what was done in Ni et al. [2022]. The encoders are simply linear projections; however, in the case of our GRU history encoder we do linear projections followed by a ReLU activation (as done in Ni et al. [2022]). Although hypothetically the policy only needs to take in history encoding, z_t, since int includes the current observation, we found it essential for the current observation to be passed in independently and have its own encoder.

C.2.1 GPIDE Implementation Details

In addition to what is mentioned in Section 8.3, we found that there were several choices that helped with training. First, there may be some scaling issues because $o_t - o_{t-1}$ may be small or the result of summation type heads may result in large encodings. To account for this, we use batch normalization layers [Ioffe and Szegedy, 2015] before each input encoding and after each ℓ^h.

There are very few nonlinear components of GPIDE. The only one that remains constant across all experiments is that a tanh activation is used for the final output of the encoder. For tracking tasks, the decoder g_θ has 1 hidden layer with 64 units and uses a ReLU activation function. For PyBullet tasks, g_θ is a linear function.

C.2.2 Recurrent and Transformer Baseline Details

Recurrent Encoder. For the recurrent encoder, we tried to match as many details as Ni et al. [2022] as possible. We double checked our implementation against theirs and confirmed that it achieves similar performance.

Transformer Encoder. We follow the GPT2 architecture [Radford et al., 2019] for inspiration, and particularly the code provided in Karpathy [2022–]. In particular, we use a number of multi-headed self-attention blocks in sequence with residual connections. We use layer normalization [Ba et al., 2016] before multi-headed attention and out projections; however, we do not use dropout. The out projection for each multi-headed self-attention block has one hidden layer with four times the number of units as the embedding dimension. Although Melo [2022] suggests using T-Fixup weight initialization, we found that more reliably high performance was achieved with the weight initialization of Radford et al. [2019]. Lastly, we used the same representation for the history as GPIDE, i.e. $(o_{t-1}, a_{t-1}, r_{t-1}, o_t - o_{t-1})$, since it results in better performance.

C.2.3 PID Baseline

To tune our PID baseline, we used Bayesian Optimization over the three (for SISO) or six (for MIMO) dimensional space. Specifically we use the library provided by Nogueira [2014–]. The output of the blackbox optimization is the average over 100 different settings (independent from the 100 settings used for testing). We allow the optimization procedure to collect as many samples as the RL methods. The final performance reported uses the PID controller with the best gains found during the optimization procedure. The bounds for each of the tracking tasks were eyeballed to be appropriate, which potentially preferably skews performance.

C.3 Hyperparameters

Because of resource restrictions, we were unable to do full hyperparameter tuning for each benchmark presented in this paper. Instead, we focused on ensuring that all history encoding methods were roughly comparable, e.g. dimension of encoding, number of parameters, etc. Tables C.1 show selected hyperparameters, and the following subsections describe how an important subset of these hyperparameters were picked. Any tuning that was done was over three seeds using 100 fixed settings (different from the 100 settings used for testing).

Task Type	Learning Rate	Batch Size	Discount Factor	Policy Network	Q Network	Path Length Encoding
Tracking	$3e^{-4}$	32 (256 for PIDE)	0.95	[24]	[256, 256]	100
PyBullet	$3e^{-4}$	32 (256 for PIDE)	0.99	[256, 256]	[256, 256]	64

Table C.1: **SAC Hyperparameters.** The "Path Length Encoding" is the amount of history each encoder gets to observe besides PIDE which, because of the nature of it, uses the entire episode.

	Observation	Action	Reward	Transition	Policy Shortcut	Q Shortcut	History Encoding
GPIDE (Tracking)	8	N/A	N/A	8	8	64	64
GRU (Tracking)	8	N/A	N/A	N/A	8	64	64
Transformer (Tracking)	16	N/A	N/A	16	8	64	64
GPIDE (PyBullet)	32	16	16	64	8	64	128
Transformer (PyBullet)	48	16	16	48	8	64	128

Table C.2: **Dimension for the Input Encoders and Final History Encoding**. The input encoders correspond to the output dimensions of the purple boxes in Figure C.1. By "History Encoding" size we mean the dimension of z_t.

Task Type	D	g_θ Hidden Size
Tracking	16	[64]
PyBullet	32	[]

Table C.3: **GPIDE Specific Hyperparamters.** Recall that D corresponds to the output dimension of f_θ. Empty brackets for the hidden size means that g_θ is a linear function.

Task Type	Number of Layers	Number of Heads	Embedding Size per Head
Tracking	2	4	8
PyBullet	4	8	16

Table C.4: **Transformer Specific Hyperparamters**

Encoder	SISO Tracking	MIMO Tracking (2D)	PyBullet
Transformer	25,542	25,644	793,868-795,026
GRU	14,240	14,264	74,816-75,248
GPIDE	13,228	13,288	75,296-76,486
GPIDE-ES	12,204	12,264	50,720-51,910
GPIDE-ESS	12,204	12,264	50,720-51,910
GPIDE-Attention	15,276	15,336	99,872-101,062

Table C.5: **Number of Parameters in History Encoder Modules**. The number of parameters corresponds to the gray boxes in Figure C.1. The difference in SISO vs MIMO and the PyBullet tasks is due to the different observation and action space dimensionalities.

C.3.1 Hyperparamters for Tracking Tasks

For tracking tasks, we tried using a history encoding size of 32 and 64 for GRU, and we found that performance was better with 64. This is surprising since PIDE can perform well in these environments even though its history encoding is much smaller (3 or 6 dimensional). To make it a fair comparison, we set the history encoding dimension for GPIDE and transformer to be 64 as well. We use one layer for GRU. For the transformer-specific hyperparameters we choose half of what appears in the PyBullet tasks.

C.3.2 Hyperparameters for PyBullet Task

For the PyBullet tasks, we simply tried to emulate most of the hyperparameters found in Ni et al. [2022]. For the transformer, we choose to use similar hyperparameters to those found in Melo [2022]. However, we found that, unlike the tracking tasks, positional encoding hurts performance. As such, we do not include it for PyBullet experiments.

C.3.3 Hyperparameters for Ablations

For the ablations of GPIDE, we use $\alpha = 0.01, 0.1, 0.25, 0.5, 0.9, 1.0$ for the smoothing parameters when only exponential smoothing is used. When using exponential smoothing and summation, the $\alpha = 0.01$ head is replaced with a summation head. The attention version of GPIDE replaces all six of these heads with attention.

C.4 Computation Details

We used an internal cluster of machines to run these experiments. We mostly leveraged Nvidia Titan X GPUs for this, but also used a few Nvidia GTX 1080s. It is difficult to get an accurate estimate of run time since job loads vary drastically on our cluster from other users. However, to train a single policy on DMSD to completion (1 million transitions collected, or 1,000 epochs) using PIDE takes roughly 4.5 hours, using GPIDE takes roughly 17.25 hours, using a GRU takes roughly 14.5 hours, and using a transformer takes roughly 21 hours. This is similar for other tracking tasks. For PyBullet tasks, using GPIDE took roughly 43.2 hours and using a transformer took roughly 64.2 hours. We note that our implementation of GPIDE is somewhat naive and could be vastly improved. In particular, for exponential smoothing and summation heads, w_t can be cached to save on compute, which is not being done currently. This is a big advantage in efficiency that GPIDE (especially one without attention heads) has over transformers.

C.5 Environment Descriptions

C.5.1 Mass Spring Damper

For both MSD and DMSD, the observations include the current mass position(s), the target reference position(s), and the last action played. Each episode lasts for 100 time steps. For all RL methods, the action is a difference in force applied to the mass, but for the PID the action is simply the force to be applied to the mass at that time. The force is bounded between -10 and 10 N for MSD and -30 and 30 N for DMSD. Each episode, system parameters are drawn from a uniform distribution with bounds shown in Table C.6 (they are the same for both MSD and DMSD). Targets are drawn to uniformly at random to be -1.5 to 1.5 m offset from the masses' resting positions.

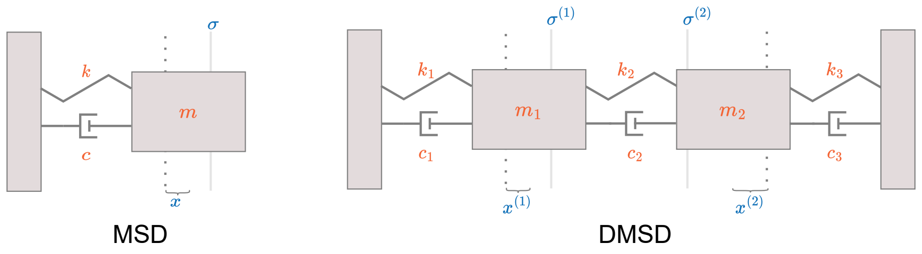


Figure C.2: **Diagram of the Mass Spring Damper Environments.** The diagram on the left the Mass Spring Damper (MSD) environment, and the diagram on the right shows the Double Mass Spring Damper (DMSD) environment. In the diagram, we have labelled the system parameters and the parts of the observation. The dotted line shows where the center of the mass is located with no force applied, and the current position of the mass is measured with respect to this point.

System Parameter	Fixed	Small	Large
Damping Constant	$\mathcal{U}(4.0, 4.0)$	$\mathcal{U}(3.5, 5.5)$	$\mathcal{U}(2.0, 10.0)$
Spring Constant	$\mathcal{U}(2.0, 2.0)$	$\mathcal{U}(1.75, 3.0)$	$\mathcal{U}(0.5, 6.0)$
Mass	$\mathcal{U}(20.0, 20.0)$	$\mathcal{U}(17.5, 40.0)$	$\mathcal{U}(10.0, 100.0)$

Table C.6: **MSD and DMSD System Parameter Distributions.** Each episode system parameters are uniformly at random drawn from these bounds.

C.5.2 Navigation Environment

Like the MSD and DMSD environments, the navigation experiment lasts 100 time steps each episode. Additionally, the observation includes position signal, target locations, and the last action. For all methods we set the action to be the change in force, and the total amount of force is bounded between -10 and 10 N. The penalty on the reward is equal to 0.01 times the magnitude of the change in force. In addition, the maximum magnitude of the velocity for the agent is bounded by $1.0 m/s$. The agent always starts at the location $(0, 0)$, and the target is picked uniformly at random to be within a box of length 10 centered around the origin.

Every episode, the mass, kinetic friction coefficient, and static friction coefficient is sampled, The friction is sampled by first sampling the total amount of friction in the system, and then sampling what proportion of the total friction is static friction. All distributions for the system parameters are uniform, and we show the bounds in Table C.7.

System Parameter	No Friction	Friction
Total Friction	$\mathcal{U}(0.0, 0.0)$	$\mathcal{U}(0.05, 0.25)$
Static Friction (Proportion)	$\mathcal{U}(0.0, 0.0)$	$\mathcal{U}(0.25, 0.75)$
Mass	$\mathcal{U}(15.0, 25.0)$	$\mathcal{U}(5.0, 35.0)$

Table C.7: **Navigation System Parameter Distributions.** Each episode system parameters are uniformly at random drawn from these bounds. The static friction parameter drawn is the proportion of the total friction that is static friction.

C.5.3 Tokamak Control Environment

Simulator Our simulator version of the tokamak control is inspired by equations used by Boyer et al. [2019], Scoville et al. [2007]. In particular, we use the following relations for stored energy, E, and rotation, v_{rot}:

$$\dot{E} = P - \frac{E}{\tau_E}$$
$$\dot{v}_{\text{rot}} = C_{\text{rot}}T - \frac{v_{\text{rot}}}{\tau_m}$$

where P is the total power, T is the total torque, τ_E is the energy confinement time, τ_m is the momentum confinement time, and C_{rot} is a quantity relying on the ion density and major radius of the plasma. We treat τ_m and C_{rot} as constants with values of 0.1 and 80.0 respectively.

We base the energy confinement time off of the ITERH-98 scaling [Transport et al., 1999]. This uses many measurements of the plasma, but we focus on a subset of these and treat the rest as constants. In particular,

$$\tau_E = C_E I^{0.95} B^{0.15} P^{-0.69}$$

where C_E is a constant value we set to be 200, I is the plasma current, and B is the toroidal magnetic field. To relate the stored energy to β_N we use the rough approximation

$$\beta_N = C_\beta \left(\frac{aB}{I}\right) E$$

where C_β is a constant we set to be 5, and a is the minor radius of the plasma. For a, I, and B, we sample these from the distribution described in Table C.8 for each episode. Lastly, we add momentum to the stored energy. That is, the stored energy derivative at time t, $\dot{E}_t$, is

$$\dot{E}_t = 0.5\left(P_t - \frac{E_t}{\tau_E}\right) + 0.5\dot{E}_{t-1}$$

Minor Radius (m)	Plasma Current (MA)	Toroidal Magnetic Field (T)
$\mathcal{N}(0.589, 0.02)$	$\mathcal{N}(1e6, 1e5)$	$\mathcal{N}(2.75, 0.1)$

Table C.8: **Tokamak Control Simulator Distributions.**

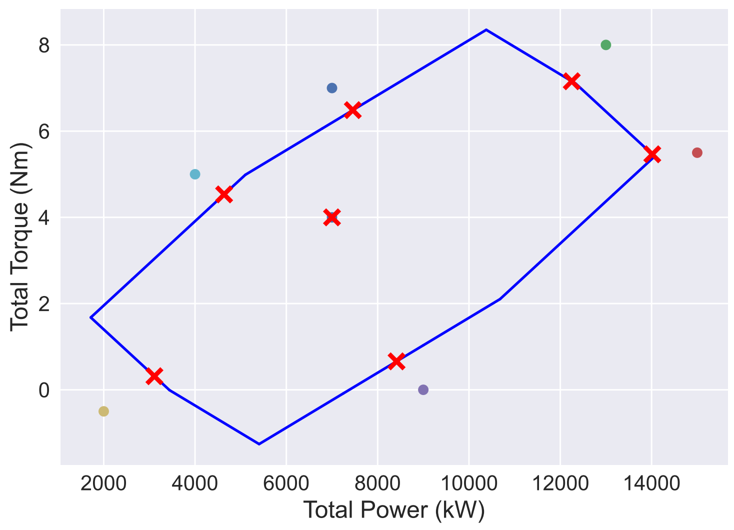


Figure C.3: **Power and Torque Bounds.** The region outlined in blue shows the possible power-torque configurations. The dots show possible requests, and the corresponding red X marks show the actual achieved power-torque setting.

The actions for all control methods is the amount of change for the power and torque. Because the total amount of power and torque injected rely on the beams, they are not totally disentangled. In Figure C.3, we show the bounds for the action space. Furthermore, we bound the amount that power and torque can be changed by roughly $40MW/s$ and $35Nm/s$, respectively. Each step is 0.025 seconds.

Each episode lasts for 100 increments of 0.025 seconds. The observations are the current β_N and rotation values, their reference values, and the current power and torque settings. We make the initial β_N and rotation relatively small in order to simulate the plasma ramping up. We let the β_N and rotation targets be distributed as $\mathcal{U}(1.75, 2.75)$ and $\mathcal{U}(25.0, 50.0)$ rad/s, respectively.

"Real" For the real versions of the tokamak control experiments, most of the previous (such as action bounds and target distributions) stays the same. This data-driven simulator is based on the one from Char et al. [2023a], and we refer the reader there for more details. That being said, there are some differences between the architecture presented there and the one used in this work. Our network is a recurrent network that uses a GRU, has four hidden layers with 512 units each, and outputs the mean and log variance of a normal distribution describing how β_N and rotation will change. In addition to power and torque, it takes in measurements for the plasma current, the toroidal magnetic field, n1rms (a measurement related to the plasma' stability), and 13 other actuator requests for gas control and plasma shaping. In addition to sampling from the normal distribution outputted by the network, we train an ensemble of ten networks, and an ensemble member is selected every episode. We use five of these models during training and the other five during testing. Along with an ensemble member being sampled each episode, we also sample a historical run, which determines the starting conditions of the plasma and how the other inputs to the neural network which are not modelled evolve over time. Recall that 100 fixed settings are used to evaluate the policy every epoch of training. In this case, a setting consists of targets, an ensemble member, and a historical run.

C.6 Additional Experiments

C.6.1 Experiments Using VIB + GRU

As shown in this work, using a GRU for a history encoder often results in a policy that is ill-equipped to handle changes in the dynamics not seen at train time. One may wonder whether using other robust RL techniques is able to mask this inadequacy of GRU. To test this, we look at adding Variational Information Bottlenecking (VIB) to our GRU baseline [Alemi et al., 2016]. Previous works applying this concept to RL usually do not consider the same class of POMDPs as us [Lu et al., 2020, Igl et al., 2019]; however, Eysenbach et al. [2021] does have a baseline that uses VIB with a recurrent policy.

To use VIB with RL, we alter the policy network so that it encodes input to a latent random variable, and the decodes into an action. Following the notation of Lu et al. [2020], let this latent random variable be Z and the random variable representing the input of the network be S. The goal is to learn a policy that maximizes $J(\pi)$ subject to $I(Z, S) \leq I_C$, where $I(Z, S)$ is the mutual information between Z and S, and I_C is some given threshold. In practice, we optimize the Lagrangian. Where β is a Lagrangian multiplier, $p(Z|S)$ is the conditional density of Z outputted by the encoder, and $q(Z)$ is the prior, the penalizer is $-\beta \mathbb{E}_S \left[D_{\text{KL}}\left(p(Z|S)||q(Z)\right)\right]$. Like other works, we assume that $q(Z)$ is a standard multivarite normal.

We alter our GRU baseline for tracking tasks so that the policy uses VIB. This is not entirely straightforward since our policy network is already quite small. We choose to keep as close to original policy architecture as possible and set the dimension of the latent variable, Z, to be 24. Note that this change has no affect on the history encoder; this only affects the policy network. For our experiments, we set $\beta = 0.1$, but we note that we may be able to achieve better performance through more careful tuning or annealing of β.

In any case, we do see that VIB helps with robustness in many instances (see Table C.9). However, the cases where there are improvements are instances where the GRU policy already did a good job at generalizing to the test environment. These are primarily the MSD and DMSD environments where the system parameters drawn during training time are simply a subset of those drawn during testing time (interestingly, this notion of dynamics generalization matches the set up of the experiments presented in Lu et al. [2020]). Surprisingly, in the navigation and tokamak control experiments, where there are more complex differences between the train and test environments, VIB can sometimes hurt the final performance.

	PID Controller	GRU	GRU+VIB	Transformer	PIDE	GPIDE
MSD Fixed / Fixed	-6.14 ± 0.02	-5.76 ± 0.02	-5.73 ± 0.01	-5.75 ± 0.01	$\mathbf{-5.69 \pm 0.00}$	-5.76 ± 0.01
MSD Fixed / Large	-11.39 ± 0.09	-12.52 ± 0.11	-12.50 ± 0.14	$\mathbf{-10.87 \pm 0.05}$	-11.44 ± 0.03	-11.61 ± 0.07
MSD Small / Small	-7.49 ± 0.03	-7.02 ± 0.01	$\mathbf{-7.01 \pm 0.01}$	-7.15 ± 0.02	-7.14 ± 0.01	-7.12 ± 0.04
MSD Small / Large	-11.18 ± 0.09	-9.82 ± 0.07	$\mathbf{-9.57 \pm 0.03}$	-10.01 ± 0.03	-10.88 ± 0.04	-10.43 ± 0.14
DMSD Fixed / Fixed	-15.33 ± 0.14	-16.20 ± 0.31	-15.83 ± 0.28	-15.41 ± 0.13	$\mathbf{-12.64 \pm 0.04}$	-13.49 ± 0.22
DMSD Fixed / Large	-27.59 ± 0.44	-37.21 ± 0.35	-35.34 ± 0.28	-28.16 ± 0.17	$\mathbf{-25.29 \pm 0.18}$	-27.54 ± 0.33
DMSD Small / Small	-21.78 ± 0.14	-22.49 ± 0.34	-22.51 ± 0.24	-20.56 ± 0.16	$\mathbf{-18.09 \pm 0.04}$	-18.67 ± 0.17
DMSD Small / Large	-26.57 ± 0.22	-31.27 ± 0.36	-30.93 ± 0.34	-26.04 ± 0.24	-23.82 ± 0.13	$\mathbf{-23.65 \pm 0.20}$
Nav Sim / Sim	-17.23 ± 0.18	-13.82 ± 0.01	-14.69 ± 0.02	-13.68 ± 0.01	-13.74 ± 0.00	$\mathbf{-13.65 \pm 0.00}$
Nav Sim / Real	-23.87 ± 0.29	-29.85 ± 0.55	-39.57 ± 0.24	-22.84 ± 0.11	$\mathbf{-20.37 \pm 0.08}$	-21.23 ± 0.12
β_N Sim / Sim	-8.09 ± 0.00	$\mathbf{-7.19 \pm 0.00}$	-7.24 ± 0.01	-7.22 ± 0.00	-8.71 ± 0.02	-7.21 ± 0.01
β_N Sim / Real	$\mathbf{-16.41 \pm 0.30}$	-31.21 ± 1.67	-32.19 ± 1.19	-31.49 ± 3.66	-43.78 ± 6.46	-26.83 ± 1.36
β_N-Rotation Sim / Sim	-27.56 ± 0.08	-18.53 ± 0.02	-18.61 ± 0.12	-18.79 ± 0.09	-21.36 ± 0.05	$\mathbf{-18.45 \pm 0.03}$
β_N-Rotation Sim / Real	$\mathbf{-30.08 \pm 0.95}$	-45.91 ± 2.10	-44.24 ± 1.33	-48.23 ± 0.25	-60.23 ± 3.20	-41.86 ± 0.69
Average	-18.33	-20.12	-21.14	-18.71	-19.58	-17.51

Table C.9: **Tracking Experiments with GRU+VIB**. We use green and red text to highlight significant improvements and deteriorations in performance over vanilla GRU. We only highlight a subset of configurations since we are focused on the robustness properties. This table shows average (unnormalized) returns.

C.6.2 Lookback Size Ablations

To better understand the role of the maximum lookback size (i.e. the amount of history used to form the encoding) of GPIDE, we repeat the PyBullet experiments using a lookback size of 4, 16, 64, and 128 with and without attention (labelled GPIDE and GPIDE-ESS, respectively). Figures C.4 and C.5 show performance curves for GPIDE and GPIDE-ESS respectively. It is clear that there is a massive increase in improvement when expanding the maximum size of the lookback from 4 to 16. For the most part, this trend continues expanding the lookback size from 16 to 64; however, it seems pushing from 64 to 128 yields mixed results. For some tasks, such as HalfCheetah-P, expanding the lookback to 128 results in noticeable improvements both with and without attention. For other tasks, such as Hopper-V, this expansion yields slightly worse performance, possibly because of training stability issues.

One interesting observation is that, when attention is included, this decrease in performance from expanding lookback can occur when increasing from 16 to 64 (see HalfCheetah-V and Walker-V). At the same time, however, it appears the GPIDE can sometimes maintain good performance when expanding the lookback to 128 when GPIDE-ESS cannot (Walker-P).

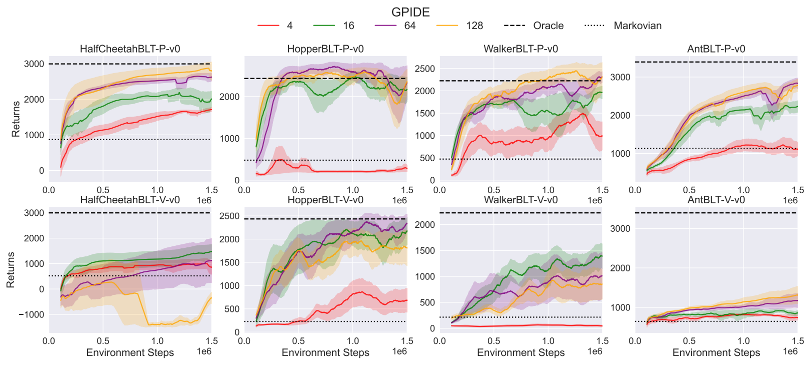


Figure C.4: **GPIDE Lookback Ablation.** Each curve shows the average over four seeds and the standard error of each.

C.7 Further Results

In this Appendix, we give further evaluation of the evaluation procedure. In addition, we give full tables of results for normalized and unnormalized scores for all methods. We also show performance traces. Note that the percentage changes in Table 4.4 do not necessarily reflect tables in this section since they report all combinations of environment variants.

C.7.1 Evaluation Procedure

As stated in the main paper, for tracking tasks, we fix 100 settings (each comprised of targets, start state, and system parameters) that are used to evaluate the policy for

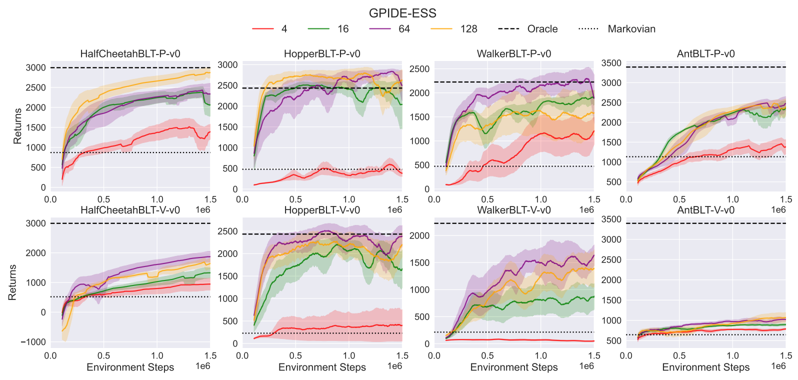


Figure C.5: **GPIDE-ESS Lookback Ablation.** Each curve shows the average over four seeds and the standard error of each.

every epoch of training (i.e. for every epoch the evaluation returns is the average over all 100 settings returns). We use a separate 100 settings when tuning. For the final returns, we average over the last 10% of recorded evaluations.

For the PyBullet tasks, we use ten different rollouts for evaluation following Ni et al. [2022]. We also average over the last 20% of recorded evaluations like they do.

Normalized Table Scores. We now give an in-depth explanation of how the scores in the table are computed. Let $\pi_{(b,i)}$ be the policy trained with baseline method b (e.g. with GPIDE, transformer, or GRU encoder) on environment variant i (e.g. fixed, small, or large). Let $J_j(\pi_{(b,i)})$ be the evaluation of policy $\pi_{(b,i)}$ on environment variant j, i.e. the average returns over all seeds and episodes. The normalized score for policy $\pi_{(b,i)}$ on variant j is then

$$\frac{J_j(\pi_{(b,i)}) - \min_{b',i'} J_j(\pi_{(b',i')})}{\max_{b',i'} J_j(\pi_{(b',i')}) - \min_{b',i'} J_j(\pi_{(b',i')})}$$

Note that we only min and max over baseline methods presented in the table.

For PyBullet tasks, we do the same procedure but normalize by the oracle policy's performance (sees both position and velocity and has no history encoder) and the Markovian policy's performance (sees only position or velocity and has no history encoder). For both of these policies, we use what was reported from Ni et al. [2022]. Note the our normalized scores differ slightly from those used in Ni et al. [2022] since they normalize based on the best and worst returns of any policy;

however, we believe our scheme gives a more intuitive picture of how any given policy is performing.

C.7.2 MSD and DMSD Results

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Fixed / Fixed	-6.14 ± 0.02	-5.76 ± 0.02	-5.75 ± 0.01	$\mathbf{-5.69 \pm 0.00}$	-5.76 ± 0.01	-5.75 ± 0.01	-5.73 ± 0.01	-5.83 ± 0.02
Fixed / Small	-7.51 ± 0.04	-7.56 ± 0.03	$\mathbf{-7.29 \pm 0.01}$	-7.37 ± 0.01	-7.33 ± 0.04	-7.37 ± 0.01	-7.32 ± 0.03	-7.39 ± 0.03
Fixed / Large	-11.39 ± 0.09	-12.52 ± 0.11	$\mathbf{-10.87 \pm 0.05}$	-11.44 ± 0.03	-11.61 ± 0.07	-11.48 ± 0.05	-12.50 ± 0.19	-11.52 ± 0.10
Small / Fixed	-6.26 ± 0.06	$\mathbf{-5.80 \pm 0.00}$	-5.92 ± 0.01	-5.95 ± 0.01	-5.93 ± 0.05	-5.89 ± 0.01	-5.92 ± 0.02	-5.91 ± 0.02
Small / Small	-7.49 ± 0.03	$\mathbf{-7.02 \pm 0.01}$	-7.15 ± 0.02	-7.14 ± 0.01	-7.12 ± 0.04	-7.09 ± 0.02	-7.15 ± 0.02	-7.12 ± 0.02
Small / Large	-11.18 ± 0.09	$\mathbf{-9.82 \pm 0.07}$	-10.01 ± 0.03	-10.88 ± 0.04	-10.43 ± 0.14	-10.42 ± 0.13	-10.43 ± 0.12	-10.07 ± 0.14
Large / Fixed	-6.78 ± 0.16	$\mathbf{-6.08 \pm 0.01}$	-6.28 ± 0.03	-6.27 ± 0.01	-6.27 ± 0.03	-6.23 ± 0.04	-6.25 ± 0.04	-6.28 ± 0.05
Large / Small	-7.78 ± 0.12	$\mathbf{-7.25 \pm 0.02}$	-7.44 ± 0.05	-7.43 ± 0.02	-7.45 ± 0.03	-7.44 ± 0.05	-7.44 ± 0.04	-7.48 ± 0.06
Large / Large	-11.12 ± 0.05	$\mathbf{-9.44 \pm 0.02}$	-9.67 ± 0.05	-10.37 ± 0.02	-9.66 ± 0.04	-9.68 ± 0.05	-9.70 ± 0.05	-9.69 ± 0.06
Average	-8.41	-7.92	**-7.82**	-8.06	-7.95	-7.93	-8.05	-7.92

Table C.10: **Unnormalized MSD Results**.

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Fixed / Fixed	58.09 ± 1.66	93.18 ± 1.46	94.04 ± 0.83	$\mathbf{100.00 \pm 0.27}$	93.18 ± 1.20	93.77 ± 1.26	96.20 ± 1.22	87.16 ± 1.78
Fixed / Small	36.41 ± 5.36	29.74 ± 3.82	$\mathbf{64.96 \pm 1.38}$	54.90 ± 0.73	59.54 ± 5.78	54.35 ± 1.35	60.89 ± 3.48	51.84 ± 3.38
Fixed / Large	36.58 ± 2.86	0.00 ± 3.42	$\mathbf{53.70 \pm 1.71}$	34.92 ± 0.93	29.55 ± 2.32	33.62 ± 1.71	0.60 ± 6.25	32.51 ± 3.29
Small / Fixed	46.87 ± 5.88	$\mathbf{89.05 \pm 0.32}$	78.21 ± 1.31	75.81 ± 0.79	77.27 ± 4.66	81.41 ± 1.20	78.82 ± 1.44	79.64 ± 1.80
Small / Small	38.25 ± 3.44	$\mathbf{100.00 \pm 0.98}$	83.49 ± 3.07	84.88 ± 0.81	87.78 ± 5.31	90.66 ± 2.02	83.97 ± 2.40	87.57 ± 2.65
Small / Large	43.52 ± 2.82	$\mathbf{87.63 \pm 2.28}$	81.44 ± 0.82	53.21 ± 1.31	68.03 ± 4.43	68.09 ± 4.10	67.78 ± 3.84	79.57 ± 4.71
Large / Fixed	0.00 ± 15.12	$\mathbf{63.36 \pm 1.17}$	45.01 ± 3.18	46.37 ± 1.29	46.68 ± 3.06	49.86 ± 3.84	48.52 ± 3.69	45.03 ± 4.72
Large / Small	0.00 ± 15.75	$\mathbf{70.44 \pm 3.30}$	45.45 ± 6.93	45.73 ± 2.47	43.66 ± 4.47	44.71 ± 6.45	45.21 ± 5.42	39.64 ± 7.82
Large / Large	45.60 ± 1.71	$\mathbf{100.00 \pm 0.61}$	92.60 ± 1.49	69.88 ± 0.69	93.03 ± 1.27	92.36 ± 1.62	91.67 ± 1.68	91.95 ± 1.80
Average	33.92	70.38	**70.99**	62.86	66.53	67.65	63.74	66.10

Table C.11: **Normalized MSD Results**.

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Fixed / Fixed	-15.33 ± 0.14	-16.20 ± 0.31	-15.41 ± 0.13	$\mathbf{-12.64 \pm 0.04}$	-13.49 ± 0.22	-13.92 ± 0.09	-13.35 ± 0.05	-16.77 ± 0.13
Fixed / Small	-21.29 ± 0.29	-25.21 ± 0.32	-21.37 ± 0.16	$\mathbf{-18.58 \pm 0.05}$	-19.77 ± 0.24	-21.31 ± 0.07	-20.09 ± 0.08	-23.29 ± 0.15
Fixed / Large	-27.59 ± 0.44	-37.21 ± 0.35	-28.16 ± 0.17	$\mathbf{-25.29 \pm 0.18}$	-27.54 ± 0.33	-31.14 ± 0.13	-28.14 ± 0.11	-31.84 ± 0.71
Small / Fixed	-18.15 ± 0.91	-17.75 ± 0.42	-15.86 ± 0.11	$\mathbf{-13.43 \pm 0.09}$	-14.37 ± 0.17	-14.35 ± 0.11	-13.57 ± 0.10	-16.85 ± 0.11
Small / Small	-21.78 ± 0.14	-22.49 ± 0.34	-20.56 ± 0.16	-18.09 ± 0.04	-18.67 ± 0.17	-18.93 ± 0.10	$\mathbf{-17.97 \pm 0.07}$	-21.77 ± 0.10
Small / Large	-26.57 ± 0.22	-31.27 ± 0.36	-26.04 ± 0.24	-23.82 ± 0.13	-23.65 ± 0.20	-23.66 ± 0.10	$\mathbf{-22.72 \pm 0.08}$	-28.26 ± 0.12
Large / Fixed	-21.96 ± 0.62	-22.41 ± 0.32	-18.37 ± 0.30	$\mathbf{-14.83 \pm 0.12}$	-15.75 ± 0.14	-16.79 ± 0.04	-15.23 ± 0.12	-18.89 ± 0.28
Large / Small	-22.30 ± 0.44	-26.63 ± 0.39	-22.00 ± 0.24	$\mathbf{-19.46 \pm 0.08}$	-19.99 ± 0.15	-21.14 ± 0.07	-19.71 ± 0.12	-23.19 ± 0.32
Large / Large	-25.29 ± 0.30	-29.34 ± 0.30	-24.43 ± 0.21	-24.06 ± 0.03	-22.08 ± 0.14	-23.06 ± 0.07	$\mathbf{-21.81 \pm 0.09}$	-25.32 ± 0.19
Average	-22.25	-25.39	-21.36	**-18.91**	-19.48	-20.48	-19.18	-22.91

Table C.12: **Unnormalized DMSD Results**.

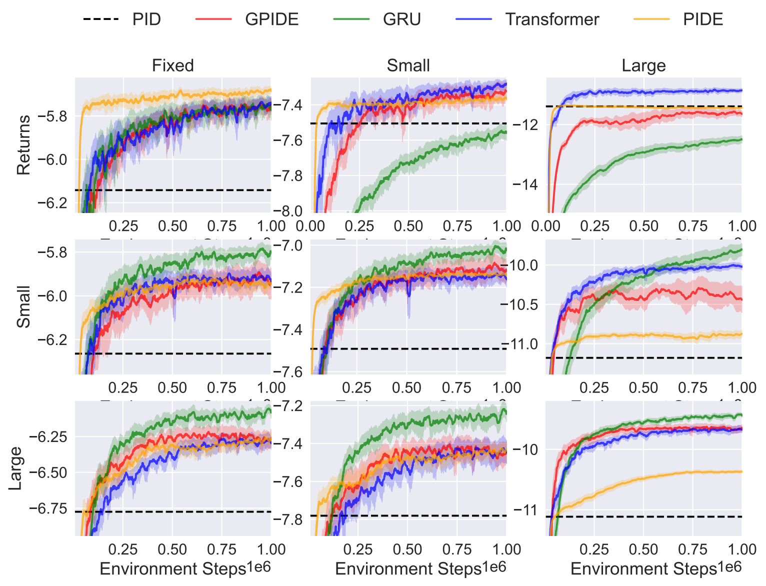


Figure C.6: **MSD Performance Curves.** Each row corresponds to a training environment, and each column corresponds to a testing environment.

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Fixed / Fixed	72.45 ± 1.44	63.59 ± 3.16	71.62 ± 1.30	$\mathbf{100.00 \pm 0.39}$	91.35 ± 2.28	86.93 ± 0.89	92.74 ± 0.55	57.75 ± 1.31
Fixed / Small	61.66 ± 3.35	16.43 ± 3.75	60.71 ± 1.80	$\mathbf{93.01 \pm 0.60}$	79.26 ± 2.81	61.50 ± 0.81	75.51 ± 0.97	38.55 ± 1.77
Fixed / Large	62.47 ± 2.86	0.00 ± 2.24	58.78 ± 1.11	$\mathbf{77.38 \pm 1.14}$	62.76 ± 2.13	39.41 ± 0.83	58.92 ± 0.73	34.84 ± 4.61
Small / Fixed	43.59 ± 9.27	47.76 ± 4.25	67.02 ± 1.10	$\mathbf{91.92 \pm 0.90}$	82.32 ± 1.72	82.52 ± 1.14	90.46 ± 0.99	56.98 ± 1.16
Small / Small	56.04 ± 1.57	47.82 ± 3.96	70.07 ± 1.88	98.69 ± 0.48	91.94 ± 2.00	88.95 ± 1.18	$\mathbf{100.00 \pm 0.78}$	56.17 ± 1.11
Small / Large	69.11 ± 1.42	38.57 ± 2.33	72.51 ± 1.58	86.96 ± 0.82	88.08 ± 1.31	87.99 ± 0.64	$\mathbf{94.09 \pm 0.51}$	58.08 ± 0.80
Large / Fixed	4.64 ± 6.34	0.00 ± 3.30	41.37 ± 3.09	$\mathbf{77.62 \pm 1.24}$	68.16 ± 1.45	57.60 ± 0.36	73.51 ± 1.24	36.06 ± 2.85
Large / Small	50.02 ± 5.07	0.00 ± 4.56	53.45 ± 2.80	$\mathbf{82.77 \pm 0.98}$	76.66 ± 1.75	63.36 ± 0.85	79.93 ± 1.43	39.74 ± 3.65
Large / Large	77.38 ± 1.93	51.09 ± 1.98	82.96 ± 1.38	85.37 ± 0.18	98.23 ± 0.90	91.86 ± 0.44	$\mathbf{100.00 \pm 0.56}$	77.22 ± 1.21
Average	55.26	29.47	64.28	**88.19**	82.08	73.35	85.02	50.60

Table C.13: **Normalized DMSD Results**.

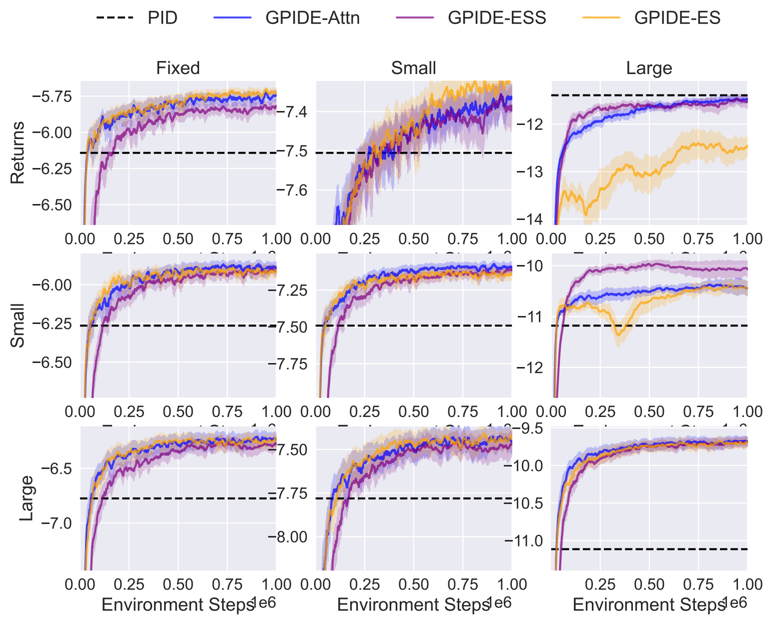


Figure C.7: **MSD Performance Curve for Ablations.** Each row corresponds to a training environment, and each column corresponds to a testing environment.

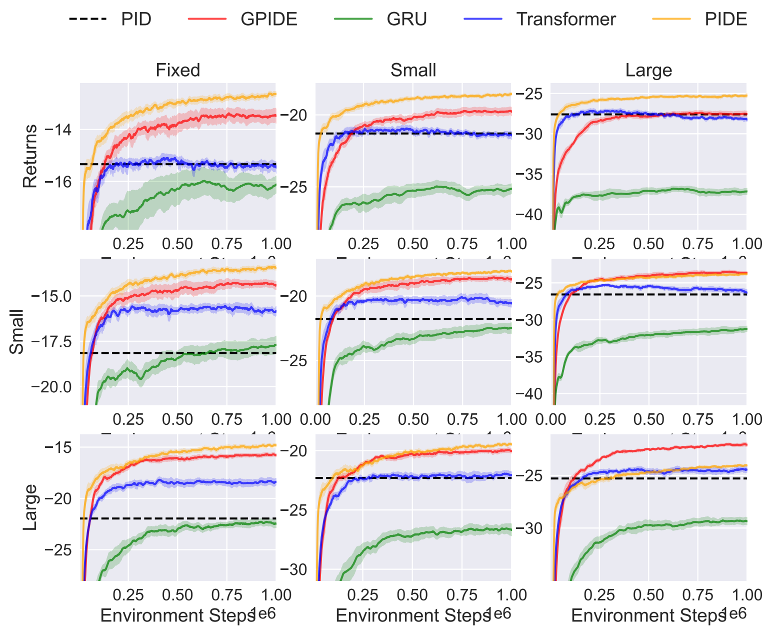


Figure C.8: **DMSD Performance Curves.** Each row corresponds to a training environment, and each column corresponds to a testing environment.

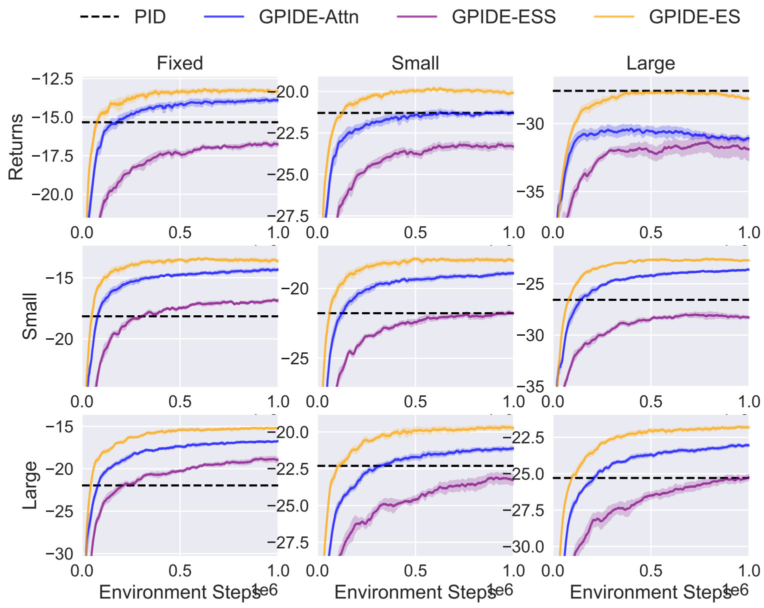


Figure C.9: **DMSD Performance Curve for Ablations.** Each row corresponds to a training environment, and each column corresponds to a testing environment.

C.7.3 Navigation Results

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Sim / Sim	28.94 ± 3.63	96.76 ± 0.15	99.57 ± 0.12	98.33 ± 0.06	$\mathbf{100.00 \pm 0.07}$	99.64 ± 0.06	99.66 ± 0.06	99.81 ± 0.09
Sim / Real	43.12 ± 2.08	0.00 ± 3.94	50.55 ± 0.78	$\mathbf{68.34 \pm 0.57}$	62.16 ± 0.89	63.17 ± 0.57	59.21 ± 1.15	52.52 ± 0.50
Real / Sim	0.00 ± 4.09	57.49 ± 1.17	68.03 ± 0.40	59.54 ± 0.85	$\mathbf{74.88 \pm 0.61}$	72.84 ± 0.64	74.75 ± 0.68	71.13 ± 0.72
Real / Real	67.28 ± 2.05	97.29 ± 0.20	99.20 ± 0.14	95.94 ± 0.04	$\mathbf{100.00 \pm 0.21}$	99.19 ± 0.09	99.11 ± 0.21	99.67 ± 0.17
Average	34.83	62.89	79.34	80.54	**84.26**	83.71	83.18	80.78

Table C.14: **Normalized Navigation Results**. Note that these results are after 1 million collected samples.

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Sim / Sim	-17.23 ± 0.18	-13.82 ± 0.01	-13.68 ± 0.01	-13.74 ± 0.00	$\mathbf{-13.65 \pm 0.00}$	-13.67 ± 0.00	-13.67 ± 0.00	-13.66 ± 0.00
Sim / Real	-23.87 ± 0.29	-29.85 ± 0.55	-22.84 ± 0.11	$\mathbf{-20.37 \pm 0.08}$	-21.23 ± 0.12	-21.09 ± 0.08	-21.64 ± 0.16	-22.57 ± 0.07
Real / Sim	-18.69 ± 0.21	-15.79 ± 0.06	-15.26 ± 0.02	-15.69 ± 0.04	$\mathbf{-14.92 \pm 0.03}$	-15.02 ± 0.03	-14.93 ± 0.03	-15.11 ± 0.04
Real / Real	-20.52 ± 0.28	-16.36 ± 0.03	-16.09 ± 0.02	-16.55 ± 0.01	$\mathbf{-15.98 \pm 0.03}$	-16.09 ± 0.01	-16.11 ± 0.03	-16.03 ± 0.02
Average	-20.08	-18.96	-16.97	-16.59	**-16.45**	-16.47	-16.59	-16.84

Table C.15: **Unnormalized Navigation Results**. Note that these results are after 1 million collected samples.

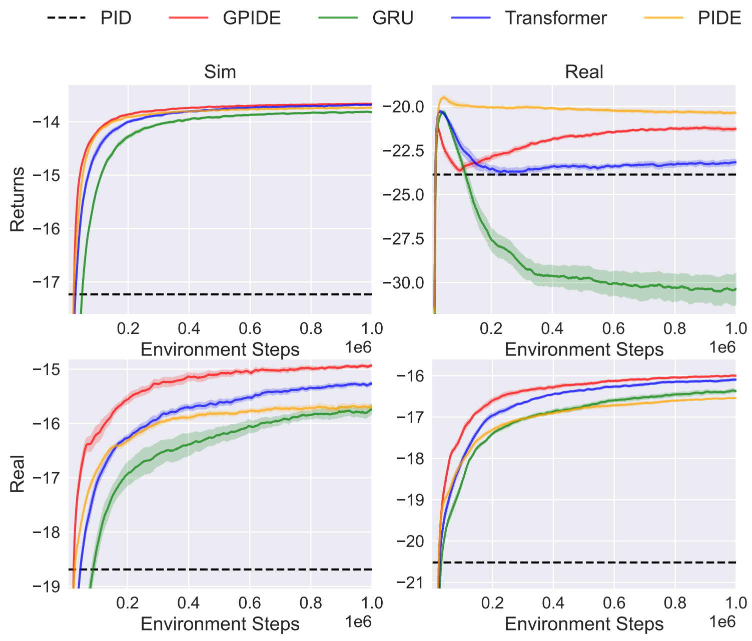


Figure C.10: **Navigation Performance Curves.** Each row corresponds to a training environment, and each column corresponds to a testing environment. Note that these runs are only done for one million transitions.

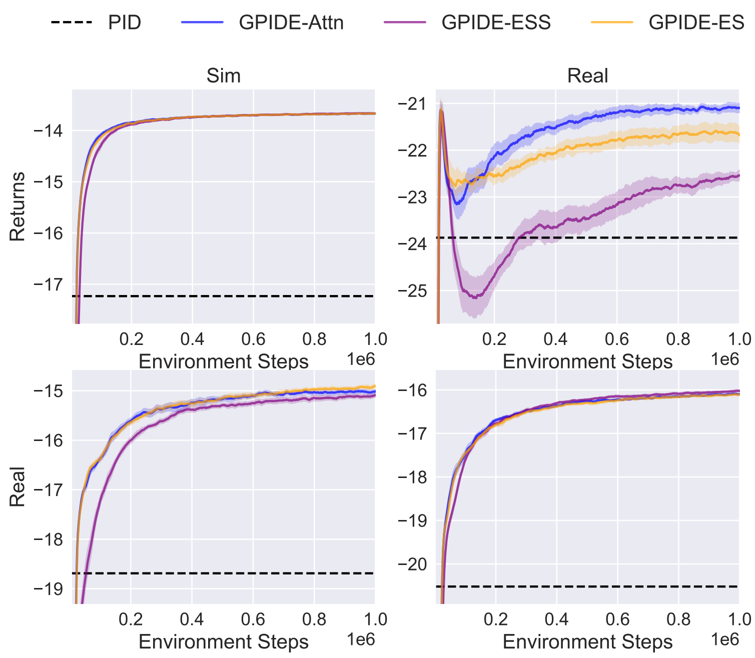


Figure C.11: **Navigation Performance Curve for Ablations.** Each row corresponds to a training environment, and each column corresponds to a testing environment. Note that these runs are only done for one million transitions.

C.7.4 Tokamak Control Results

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Sim / Sim	90.95 ± 0.05	$\mathbf{100.00 \pm 0.03}$	99.63 ± 0.03	84.74 ± 0.16	99.75 ± 0.06	99.91 ± 0.02	99.90 ± 0.02	99.47 ± 0.04
Sim / Real	$\mathbf{89.15 \pm 0.99}$	40.96 ± 5.45	40.05 ± 11.91	0.00 ± 21.04	55.21 ± 4.44	61.56 ± 7.40	65.65 ± 5.66	35.66 ± 4.41
Real / Sim	50.62 ± 3.96	36.33 ± 3.61	35.26 ± 2.22	0.00 ± 3.48	48.40 ± 4.04	52.62 ± 1.38	$\mathbf{56.30 \pm 2.25}$	16.33 ± 5.98
Real / Real	98.45 ± 0.77	98.24 ± 0.38	98.74 ± 0.29	$\mathbf{100.00 \pm 0.23}$	99.30 ± 0.64	98.39 ± 0.33	98.55 ± 0.33	98.27 ± 0.37
Average	**82.29**	68.88	68.42	46.18	75.67	78.12	80.10	62.43

Table C.16: **Normalized β_N Tracking Results**.

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Sim / Sim	-8.09 ± 0.00	$\mathbf{-7.19 \pm 0.00}$	-7.22 ± 0.00	-8.71 ± 0.02	-7.21 ± 0.01	-7.19 ± 0.00	-7.20 ± 0.00	-7.24 ± 0.00
Sim / Real	$\mathbf{-16.41 \pm 0.30}$	-31.21 ± 1.67	-31.49 ± 3.66	-43.78 ± 6.46	-26.83 ± 1.36	-24.88 ± 2.27	-23.63 ± 1.74	-32.83 ± 1.35
Real / Sim	-12.12 ± 0.40	-13.55 ± 0.36	-13.66 ± 0.22	-17.18 ± 0.35	-12.34 ± 0.40	-11.92 ± 0.14	$\mathbf{-11.55 \pm 0.22}$	-15.55 ± 0.60
Real / Real	-13.56 ± 0.23	-13.62 ± 0.12	-13.47 ± 0.09	$\mathbf{-13.08 \pm 0.07}$	-13.30 ± 0.20	-13.58 ± 0.10	-13.53 ± 0.10	-13.61 ± 0.11
Average	**-12.55**	-16.39	-16.46	-20.69	-14.92	-14.39	-13.98	-17.31

Table C.17: **Unnormalized β_N Tracking Results**.

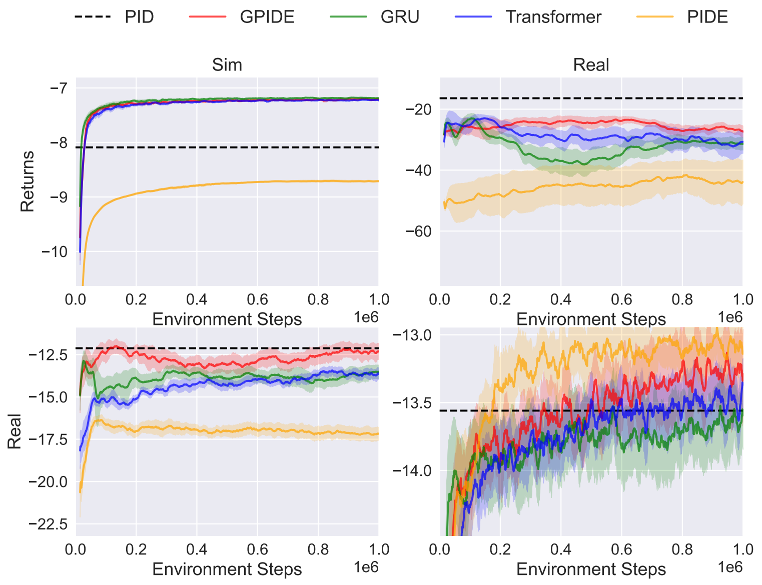


Figure C.12: β_N **Tracking Performance Curves.** Each row corresponds to a training environment, and each column corresponds to a testing environment.

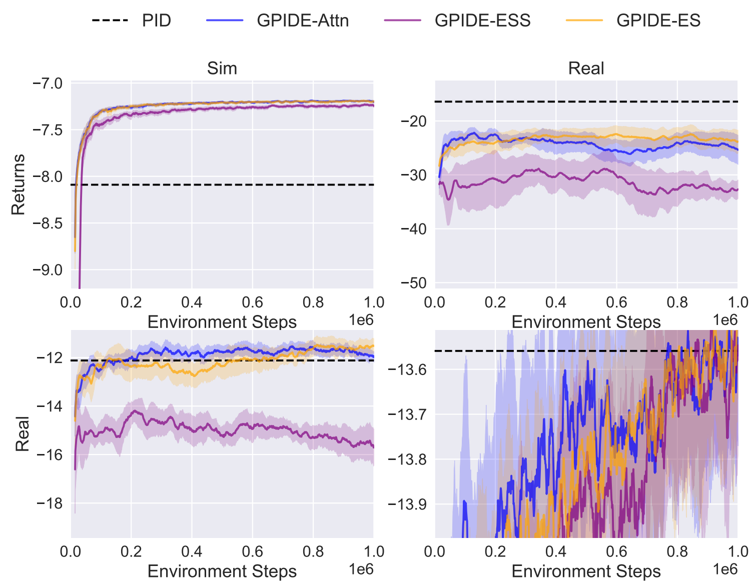


Figure C.13: β_N **Tracking Performance Curve for Ablations.** Each row corresponds to a training environment, and each column corresponds to a testing environment.

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Sim / Sim	46.78 ± 0.44	99.50 ± 0.12	97.99 ± 0.50	82.96 ± 0.27	$\mathbf{100.00 \pm 0.15}$	99.64 ± 0.19	99.97 ± 0.12	96.18 ± 1.35
Sim / Real	$\mathbf{83.48 \pm 2.63}$	39.65 ± 5.83	33.22 ± 0.69	0.00 ± 8.87	50.86 ± 1.92	54.36 ± 2.07	52.56 ± 2.44	42.51 ± 2.97
Real / Sim	0.00 ± 8.79	21.31 ± 2.45	7.23 ± 3.86	22.49 ± 1.84	19.02 ± 3.88	$\mathbf{22.70 \pm 4.42}$	5.20 ± 20.06	15.35 ± 8.29
Real / Real	91.76 ± 0.84	98.07 ± 0.52	96.05 ± 0.31	97.94 ± 0.23	99.73 ± 0.46	97.62 ± 0.46	$\mathbf{100.00 \pm 0.28}$	96.33 ± 0.47
Average	55.51	64.63	58.62	50.85	67.40	**68.58**	64.43	62.59

Table C.18: **Normalized β_N-Rotation Tracking Results**.

	PID Controller	GRU	Transformer	PIDE	GPIDE	GPIDE-ES	GPIDE-ESS	GPIDE-Attn
Sim / Sim	-27.56 ± 0.08	-18.53 ± 0.02	-18.79 ± 0.09	-21.36 ± 0.05	$\mathbf{-18.45 \pm 0.03}$	-18.51 ± 0.03	-18.45 ± 0.02	-19.10 ± 0.23
Sim / Real	$\mathbf{-30.08 \pm 0.95}$	-45.91 ± 2.10	-48.23 ± 0.25	-60.23 ± 3.20	-41.86 ± 0.69	-40.60 ± 0.75	-41.25 ± 0.88	-44.88 ± 1.07
Real / Sim	-35.57 ± 1.50	-31.92 ± 0.42	-34.33 ± 0.66	-31.72 ± 0.32	-32.31 ± 0.66	$\mathbf{-31.68 \pm 0.76}$	-34.68 ± 3.43	-32.94 ± 1.42
Real / Real	-27.09 ± 0.30	-24.81 ± 0.19	-25.54 ± 0.11	-24.86 ± 0.08	-24.21 ± 0.16	-24.98 ± 0.17	$\mathbf{-24.12 \pm 0.10}$	-25.44 ± 0.17
Average	-30.08	-30.29	-31.72	-34.54	-29.21	**-28.94**	-29.62	-30.59

Table C.19: **Unnormalized β_N-Rotation Tracking Results**.

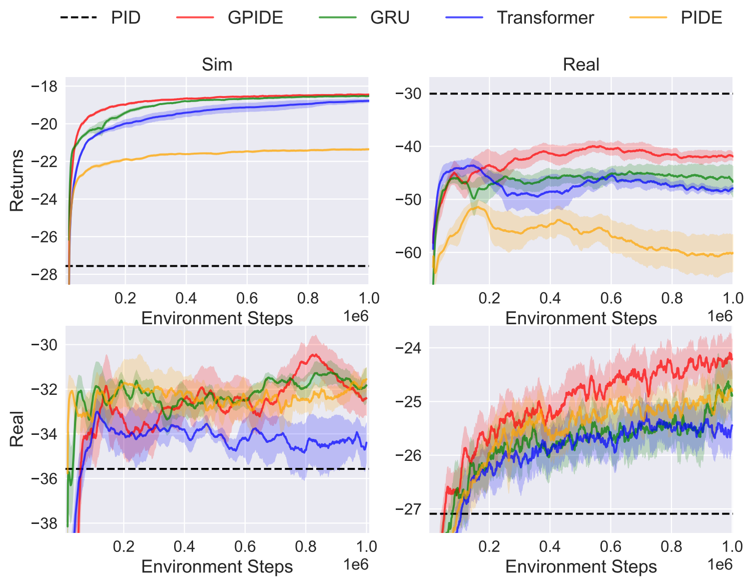


Figure C.14: β_N**-Rotation Tracking Performance Curves.** Each row corresponds to a training environment, and each column corresponds to a testing environment.

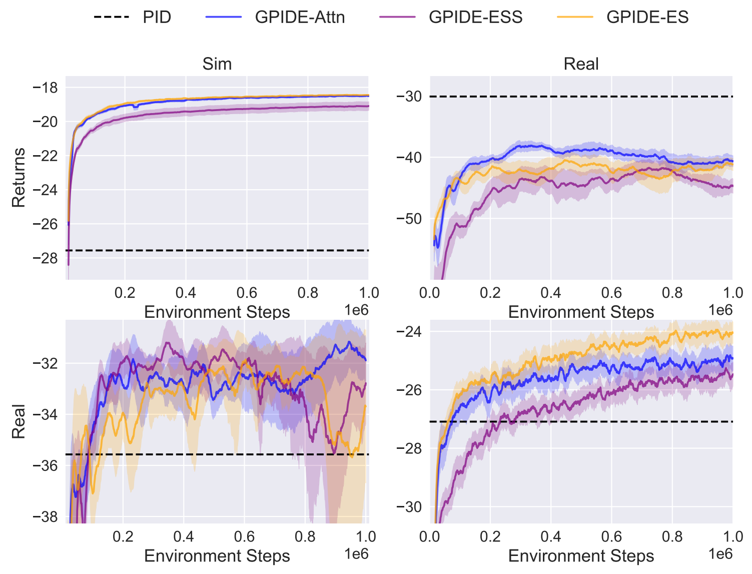


Figure C.15: β_N**-Rotation Tracking Performance Curve for Ablations.** Each row corresponds to a training environment, and each column corresponds to a testing environment.

C.7.5 PyBullet Results

For these results, SAC encodes observations, actions and rewards. TD3 encodes observations and actions since it is the best performing on average.

	PPO-GRU	A2C-GRU	SAC-LSTM	TD3-GRU	VRM	SAC-Transformer	SAC-GPIDE	SAC-GPIDE-ES	SAC-GPIDE-ESS	SAC-GPIDE-Attn
HalfCheetah-P	27.09 ± 7.85	-22.00 ± 5.13	77.06 ± 7.96	$\mathbf{85.80 \pm 5.15}$	-107.00 ± 1.39	37.00 ± 9.97	82.63 ± 3.46	85.45 ± 4.83	71.08 ± 6.95	77.63 ± 4.60
Hopper-P	49.00 ± 5.22	-2.29 ± 3.33	64.36 ± 9.94	84.63 ± 8.33	3.53 ± 1.63	59.54 ± 19.64	93.27 ± 13.56	111.48 ± 4.20	$\mathbf{113.95 \pm 4.67}$	104.87 ± 8.76
Walker-P	1.67 ± 4.39	-10.48 ± 1.73	40.92 ± 15.56	29.08 ± 9.67	-3.89 ± 1.25	24.89 ± 14.80	$\mathbf{96.61 \pm 1.60}$	76.58 ± 5.47	94.58 ± 11.00	71.36 ± 6.57
Ant-P	39.48 ± 3.74	-13.06 ± 6.52	60.97 ± 3.54	-36.36 ± 3.35	-36.39 ± 0.17	-10.57 ± 2.34	$\mathbf{66.66 \pm 2.94}$	64.73 ± 3.82	57.78 ± 3.78	63.19 ± 5.32
HalfCheetah-V	19.68 ± 11.71	-50.13 ± 9.50	18.54 ± 33.09	$\mathbf{59.03 \pm 2.88}$	-80.49 ± 2.97	-41.31 ± 26.15	20.39 ± 29.60	51.03 ± 13.93	53.14 ± 5.86	-54.70 ± 19.89
Hopper-V	13.86 ± 4.80	-0.60 ± 3.33	16.26 ± 12.44	57.43 ± 8.63	10.08 ± 3.51	0.28 ± 8.49	$\mathbf{90.98 \pm 4.28}$	72.63 ± 19.28	90.09 ± 2.50	30.73 ± 1.60
Walker-V	8.12 ± 5.43	-8.02 ± 0.57	-1.57 ± 1.88	-4.63 ± 1.30	-1.80 ± 0.70	-8.21 ± 1.31	36.90 ± 16.59	$\mathbf{68.30 \pm 4.33}$	67.54 ± 3.60	14.85 ± 11.26
Ant-V	1.43 ± 3.26	-13.67 ± 1.83	-16.95 ± 1.29	17.03 ± 6.55	-13.41 ± 0.12	0.81 ± 1.31	$\mathbf{18.03 \pm 5.10}$	4.56 ± 5.20	12.85 ± 1.67	-1.84 ± 5.76
Average	20.04	-15.03	32.45	36.50	-28.67	7.80	63.18	66.84	**70.13**	38.26

Table C.20: **Normalized PyBullet Scores**.

	PPO-GRU	A2C-GRU	SAC-LSTM	TD3-GRU	VRM	SAC-Transformer	SAC-GPIDE	SAC-GPIDE-ES	SAC-GPIDE-ESS	SAC-GPIDE-Attn
HalfCheetah-P	1445.81 ± 166.79	403.35 ± 108.97	2506.88 ± 168.93	$\mathbf{2692.53 \pm 109.43}$	-1401.67 ± 29.62	1656.13 ± 211.75	2625.13 ± 73.49	2684.98 ± 102.57	2379.79 ± 147.67	2519.06 ± 97.72
Hopper-P	1436.43 ± 102.09	433.19 ± 65.09	1736.81 ± 194.51	2133.42 ± 162.93	546.93 ± 31.81	1642.63 ± 384.10	2302.31 ± 265.21	2658.48 ± 82.18	$\mathbf{2706.81 \pm 91.39}$	2529.31 ± 171.41
Walker-P	501.06 ± 76.99	288.10 ± 30.39	1189.28 ± 272.77	981.63 ± 169.46	403.60 ± 21.85	908.17 ± 259.52	$\mathbf{2165.52 \pm 28.10}$	1814.40 ± 95.91	2129.91 ± 192.87	1722.81 ± 115.22
Ant-P	2025.52 ± 84.58	837.57 ± 147.53	2511.54 ± 80.13	310.72 ± 75.68	310.24 ± 3.83	893.84 ± 52.83	$\mathbf{2640.16 \pm 66.46}$	2596.63 ± 86.26	2439.48 ± 85.37	2561.67 ± 120.21
HalfCheetah-V	1005.13 ± 289.84	-723.40 ± 235.29	977.02 ± 819.24	$\mathbf{1979.56 \pm 71.40}$	-1475.15 ± 73.42	-505.00 ± 647.43	1022.93 ± 732.93	1781.36 ± 344.95	1833.60 ± 145.14	-836.47 ± 492.45
Hopper-V	534.05 ± 105.85	215.22 ± 73.48	587.10 ± 274.42	1495.11 ± 190.42	450.77 ± 77.35	234.49 ± 187.36	$\mathbf{2235.02 \pm 94.45}$	1830.26 ± 425.16	2215.47 ± 55.16	906.05 ± 35.29
Walker-V	377.80 ± 109.11	53.25 ± 11.45	182.97 ± 37.89	121.44 ± 26.14	178.28 ± 14.09	49.32 ± 26.43	956.43 ± 333.46	$\mathbf{1587.56 \pm 87.15}$	1572.41 ± 72.46	513.07 ± 226.34
Ant-V	684.36 ± 89.48	269.32 ± 50.35	178.98 ± 35.57	1113.19 ± 179.93	276.33 ± 3.18	667.20 ± 35.98	$\mathbf{1140.73 \pm 140.22}$	770.51 ± 143.02	998.35 ± 46.04	594.54 ± 158.48
Average	1001.27	222.08	1233.82	1353.45	-88.84	693.35	1886.03	1965.52	**2034.48**	1313.76

Table C.21: **Unnormalized PyBullet Scores**.

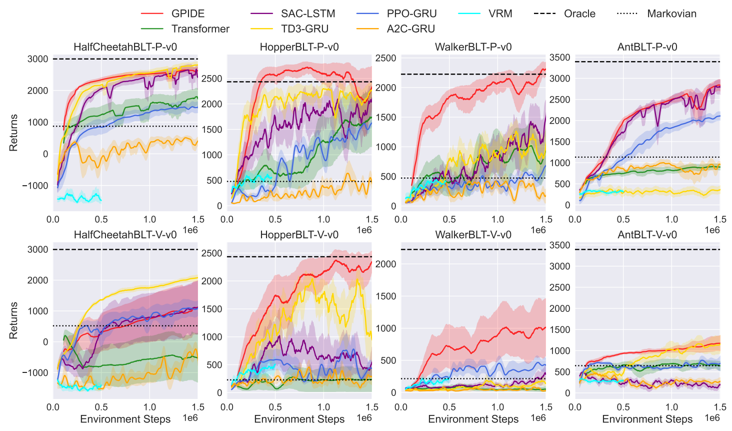


Figure C.16: **PyBullet Performance Curves.**

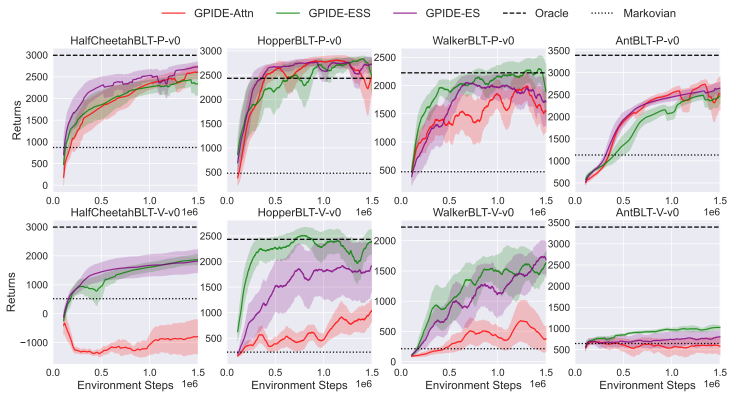


Figure C.17: **PyBullet Performance Curve for Ablations.**

Interestingly, we found that GPIDE policies often outperform the oracle policy on Hopper-P. While the oracle performance here was taken from Ni et al. [2022], we confirmed this also happens with our own implementation of an oracle policy. We hypothesize that this may be due to the fact the GPIDE policy gets to see actions and rewards and the oracle does not.

C.7.6 Attention Scheme Visualizations

We generate the attention visualizations (as seen in Figure 4.4) by doing a handful of rollouts with a GPIDE policy using only attention heads. During this rollout we collect all of the weighting schemes, i.e. softmax $\left(\frac{q_{1:t}k_{1:t}^T}{\sqrt{D}}\right)$, generated throughout the rollouts and average them together. Below, we show additional attention visualizations. In all figures/gpide, each plot shows one of the different six heads. For each of these, the policies were evaluated on the same version of the environment they were trained on.

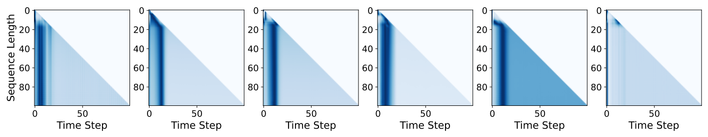


Figure C.18: **MSD-Fixed Attention**.

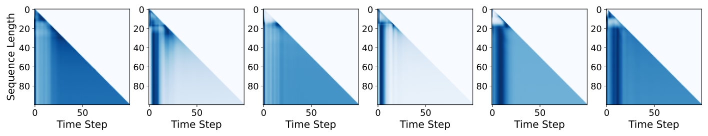


Figure C.19: **MSD-Small Attention**.

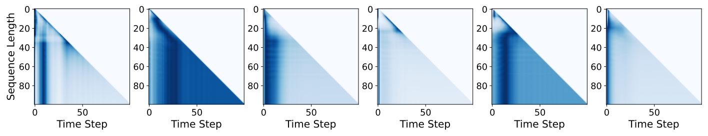


Figure C.20: **MSD-Large Attention**.

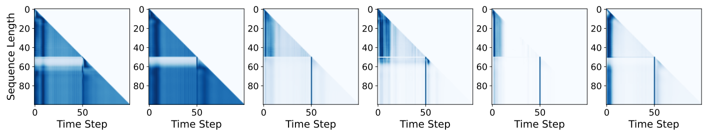


Figure C.21: **DMSD-Fixed Attention**.

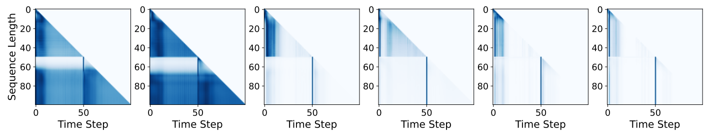


Figure C.22: **DMSD-Small Attention**.

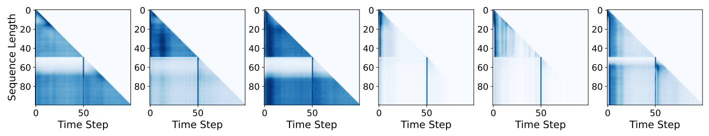


Figure C.23: **DMSD-Large Attention**.

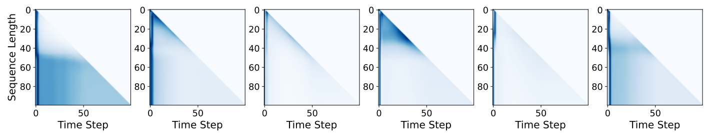


Figure C.24: **Navigation No Friction Attention**.

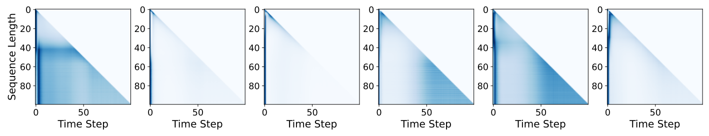


Figure C.25: **Navigation Friction Attention**.

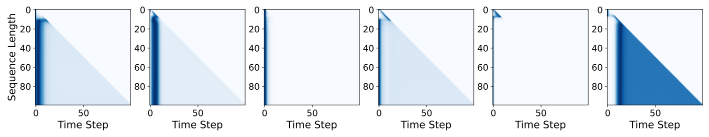


Figure C.26: β_N **Tracking Sim Attention**.

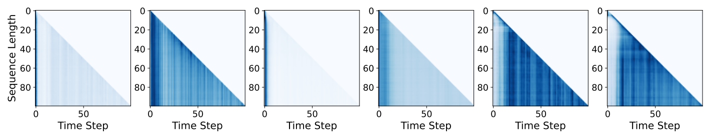


Figure C.27: β_N **Tracking Rotation Attention**.

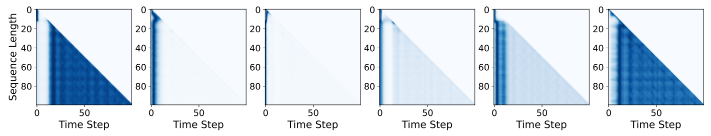


Figure C.28: β_N**-Rotation Tracking Sim Attention**.

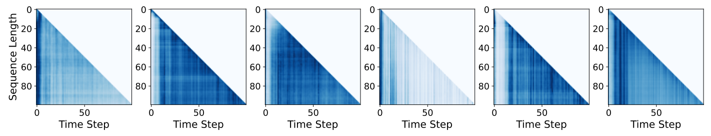


Figure C.29: β_N**-Rotation Tracking Rotation Attention**.

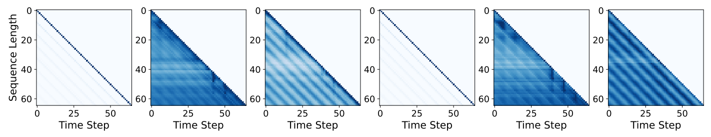


Figure C.30: **HalfCheetah-P Attention**.

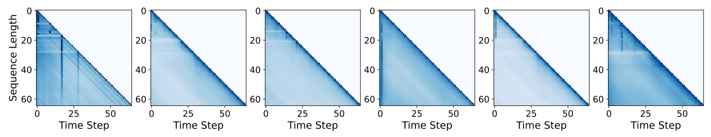


Figure C.31: **HalfCheetah-V Attention**.

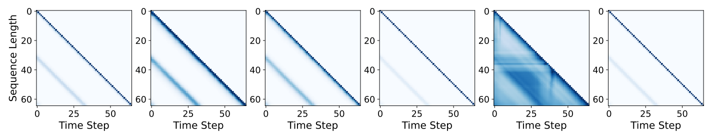


Figure C.32: **Hopper-P Attention**.

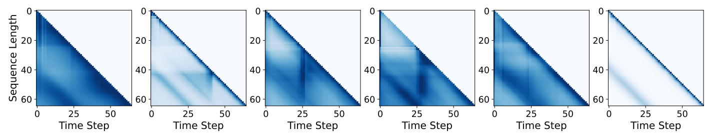


Figure C.33: **Hopper-V Attention**.

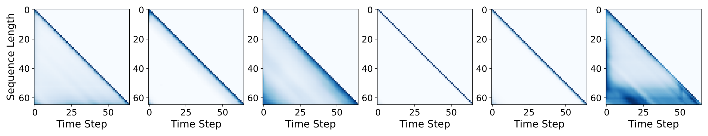


Figure C.34: **Walker-P Attention**.

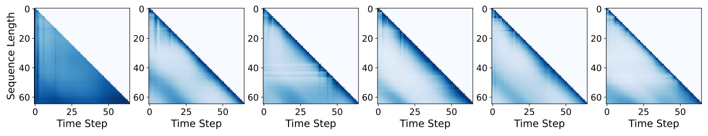


Figure C.35: **Walker-V Attention**.

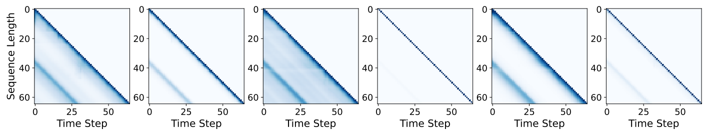


Figure C.36: **Ant-P Attention**.

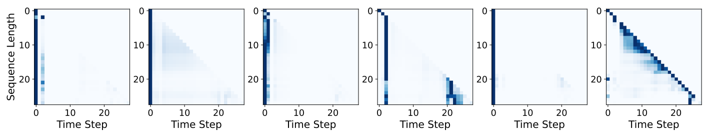


Figure C.37: **Ant-V Attention**. Note that total path length is less than 64 here since the agent falls down pretty fast.

Appendix D

Appendix for Chapter 5

D.1 Related Works

Reinforcement Learning The experimental setting considered in this work falls under the so-called "offline" reinforcement learning setting [Levine et al., 2020]. In this setting, a policy must be learned from a fixed dataset of environment interactions and additional environment queries cannot be made. In this setting, Yu et al. [2020] the MOPO algorithm that uses an ensemble of PNNs and relies on penalties to make sure the policy does not steer out of the offline dataset's support. Chen et al. [2021b] extend this idea in the algorithm MAPLE, which incorporates an adaptive policy into the learning procedure. Our reinforcement learning experiments are similar to the set up in MAPLE, except we do full episodes from start states (drawn from the start distribution which is assumed to be known). In contrast, MAPLE does short rollouts of 10 steps starting states randomly selected in the offline dataset.

Gaussian Processes in RL Hypothetically, one could use a GP for the dynamics model and draw posterior samples when generating rollouts with the policy, and this has been done for several low dimensional tasks by previous works [Deisenroth and Rasmussen, 2011, Mehta et al., 2021b, 2022] However, the non-parametric nature of GPs makes scaling up to higher dimensional tasks a challenge. Perhaps an even greater problem comes from the fact that these posterior samples are much more computationally expensive than their PNN alternative. Because algorithms such as MBPO, MOPO, and MAPLE require millions if not billions of model samples to train a neural network policy to convergence, GPs are often too big of a computational burden to use.

D.2 PNN Toy Example

To help guide intuition on why the PNN is useful even in deterministic environments, consider a toy regression problem in which we wish to model the function $f(x) = \cos(3x)$. Our training dataset consists of 100 (X, Y) data points where X is distributed as an exponential random variable. As such, there will be a high concentration of X data around 0, but the concentration of training data quickly tapers off. We train a PNN on this toy problem and show the results in Figure D.1. As one would hope, the PNN is confident in regions where data is plentiful, and the model produces wide predictive distributions in regions lacking data. Why does this happen instead of the network producing highly confident predictive distribution where the mean goes through each training point? We hypothesize that this is due to both the capacity of the network and the property of neural networks to produce generally smooth solutions. Similar observations were also made in Seitzer et al.

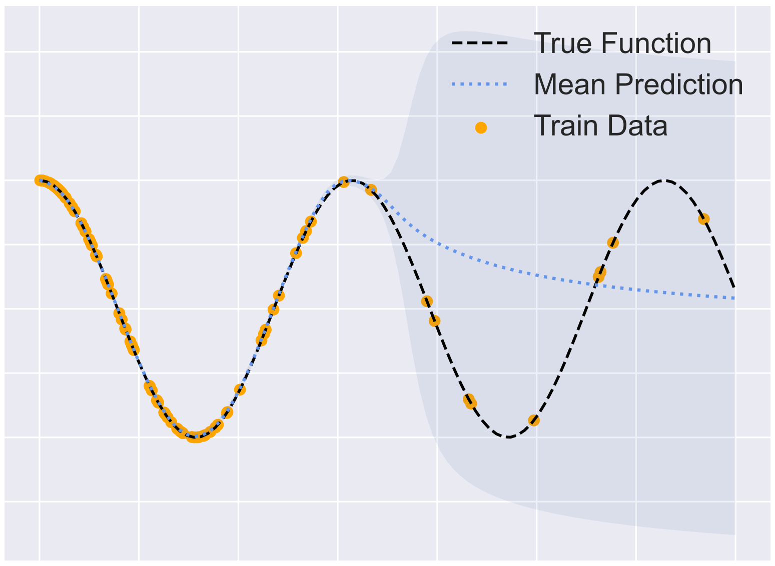


Figure D.1: **PNN Trained on a Toy Example**. The orange points make up the training dataset, the black dashed line is the true function, and the dotted blue line shows predicted mean. The blue shaded region shows three standard deviations of the predicted Gaussian distribution. We also use a validation set with 20 points to know when to stop training the model.

[2022], although they consider the setting in which aleatoric noise truly does exist.

D.3 Algorithm Details

Algorithm 4 SPNN Trajectory Sampling

```
1: Input: Policy $\pi$, initial state $s_1$, kernels $\{\kappa_d\}_{d=1}^{D}$, horizon $H$, and number of bases $B$.
2: Sample function $g$ by sampling $\phi_{b,d} \sim p_{\kappa_d}$ and $\tau_{b,d} \sim \mathcal{U}(0, 2\pi)$ for $b \in \{1, \dots, B\}$ and $d \in \{1, \dots, D\}$.
3: for $t \leftarrow 1, \dots H$ do
4:     $a_t \sim \pi(s_t)$
5:     $x_t \leftarrow (s_t, a_t)$
6:     $s_{t+1} \leftarrow s_t + \mu_\theta(x_t) + \sigma_\theta(x_t) g(x_t)$
7: end for
8: Return $(x_1, \dots, x_H)$
```

D.4 Environment Details

Nuclear Fusion Environment As stated in the main body of the paper, this environment is adapted from Char and Schneider [2023]. This environment uses equations described in Boyer et al. [2019] and Scoville et al. [2007]. In particular, we use the following relations for stored energy, E, and rotation, v_{rot}:

$$\begin{aligned}\dot{E} &= P - \frac{E}{\tau_E} \\ \tau_E &= C_E I^{0.95} B^{0.15} P^{-0.69} \\ \beta_N &= C_\beta \left(\frac{aB}{I}\right) E\end{aligned}$$

where P is the total power, τ_E is the energy confinement time, I is the plasma current, a is the minor radius, B is the magnetic field, and C_E, C_β are constants set to 200 and 5, respectively. The second of these equations is ITERH-98 scaling [Transport et al., 1999]. For our version of the environment, we also fix $I = 10^6$, $a = 0.589$, and $B = 2.75$. Similar to Char and Schneider [2023], we include momentum in the energy update. The equation describing the evolution of the energy is

$$\dot{E}_t = 0.5\left(P_t - \frac{E_t}{\tau_E}\right) + 0.5\dot{E}_{t-1}$$

The observation space for the environment is three dimensional and consists of the current β_N measurement, the rate of change of β_N, and the current amount of power being injected into the system. The action space is one-dimensional and is simply the change in the power. We set the β_N limit to be 2.2 , and the reward function is

$$r(\beta_N, a) := \begin{cases} \left(\frac{2.5 - |\beta_N - 2.2|}{2.5}\right)^2 - \frac{\|a\|}{10} & \beta_N \leq 2.2 \\ -100 & \beta_N > 2.2 \end{cases}$$

where a is the action scaled to be between -1 and 1. We use a horizon length of 100 for each episode.

Mountain Ridge Environment The mountain ridge environment has five-dimensional observations space: $s_t = (x_t, \dot{x}_t, y_t, \dot{y}_t, \theta)$, where x_t is the x position, y_t is the y position, and θ is the angle of the thruster used to propel the agent. The action space is two-dimensional and consists of a^{thrust} and a^{angle}, which controls the amount of thrust and the change in angle of the thruster, respectively. The updates are as follows:

$$\begin{aligned}
x_{t+1} &= x_t + \dot{x}_t \Delta t \\
\dot{x}_{t+1} &= \dot{x}_t + \left(\text{sign}(x_t) x_t^2 + a^{\text{thrust}} \sin(\theta_t)\right) \Delta t \\
y_{t+1} &= y_t + \dot{y}_t \Delta t \\
\dot{y}_{t+1} &= \dot{y}_t + \left(a^{\text{thrust}} \cos(\theta_t) + 0.05 \exp(y_t)\right) \Delta_t \\
\theta_{t+1} &= \text{clip}\left(\theta_t + \frac{\pi}{6} f(a^{\text{angle}}), -\pi, \pi\right)
\end{aligned}$$

where the function f is defined as

$$f(x) = \begin{cases} 1 + \exp\left[-12.5\,(x - 0.5)\right] & x > 0 \\ 1 + \exp\left[12.5\,(x + 0.5)\right] & x \leq 0 \end{cases}$$

The reward function is simply $\frac{6 + y_t - \left\|a^{\text{thrust}}\right\|}{10}$ while the agent is on the cliff. The episode ends and the agent recieves a reward of -100 if $|x_t| > 3$, $y_t < -6$, or $y_t > 5$. We use a horizon length of 200 for each episode.

D.5 Additional Training Details

Model Training For each of the dynamics models, we use a network with 2 hidden layers, each with 512 units. We use two separate heads for the mean and standard deviation predictions. We find that we can get better uncertainty by adding an additional hidden layer with 256 units to the standard deviation head. Besides the change in architecture, the learning procedure follows what is done in Chua et al. [2018b], and we use the Adam [Kingma and Ba, 2014] optimizer with a learning rate of 3×10^{-4} and a batch size of 64. Lastly, we use 10% of the data as a validation set and early stop based on MSE (although we pick the checkpoint that achieves the best negative log likelihood).

Reinforcement Learning Training Our implementation of SAC with recurrent policies closely follows the implementation given by Ni et al. [2021] and uses a Gated Recurrent Unit (GRU) [Cho et al., 2014b]. We give hyperparameter settings in Table D.1. We train for 100, 250, and 1000 epochs for the Cart Pole, Fusion, and Mountain environments, respectively. Following other offline model-based reinforcement learning works [Yu et al., 2020, Chen et al., 2021b], we add a penalty to the reward to indicate when the policy is going out of distribution. When using PNN models, we use the maximum predicted standard deviation among the ensemble members, and, when using neural networks with point predictions, we use the standard deviation among the mean predictions. As per [Chen et al., 2021b] we scale this uncertainty by 0.25.

When simulating episodes with the model during training time, the trajectory can sometimes blow up and predict large values. While this is rare, we find that it helps training stability to cap the velocity components of the state space to reasonable values. Finally, for all methods of sampling, we choose a member of the ensemble to make predictions each episode, and we fix this member for the entire episode.

Hyperparameter	**Value**
Discount Factor	0.99
Learning Rate	3×10^{-4}
Batch Size	256
Target Soft Update Weight	5×10^{-3}
History Lookback Size	64
Exploration Steps per Epoch	1000
Gradient Steps per Epoch	1000

Table D.1: **Reinforcement learning hyperparameters.**

D.6 Model Metric Details

A widely accepted metric in uncertainty quantification to evaluate the validity of distributional predictions is *average calibration*. Given input covariates X, target variables Y, a predictive distribution with CDF $F_X : \mathcal{X} \to (\mathcal{Y} \to [0,1])$ and its corresponding quantile function $F_X^{-1} : \mathcal{X} \to ([0,1] \to \mathcal{Y})$, F is said to be average calibrated if

$$P\left(Y \leq F_X^{-1}(p)\right) = p, \forall p \in [0,1]. \tag{D.1}$$

Note that Eq. D.1 assesses the validity of the predictive quantile function F_X^{-1}, which is identical to a prediction interval between the probabilities $[0,p]$. We note that centered prediction intervals (e.g. a 95% prediction interval that spans the probabilities $[0.025, 0.975]$) can be more useful in practice, and we assess the average of centered prediction intervals, which is defined as:

$$P\left(F_X^{-1}(0.5 - p/2) \leq Y \leq F_X^{-1}(0.5 + p/2)\right) = p, \forall p \in [0,1]. \tag{D.2}$$

Miscalibration, i.e. error in average calibration of centered intervals, is then measured as

$$\int_0^1 \mid P\left(F_X^{-1}(0.5 - p/2) \leq Y \leq F_X^{-1}(0.5 + p/2)\right) - p \mid dp. \tag{D.3}$$

Given a dataset $\{x_i, y_i\}_{i=1}^N$, and a uniform draw of probabilities $\{p_k\}_{k=1}^K \in [0, 1]$, miscalibration of centered intervals can be estimated as

$$\frac{1}{K}\sum_{k=1}^{K}\Big|\texttt{(empirical coverage at } p_k\texttt{)} - p_k\Big|, \tag{D.4}$$

where $\texttt{(empirical coverage at } p_k\texttt{)}$ is defined as $\frac{1}{N}\sum_{i=1}^N \mathbb{I}\{F_{x_i}^{-1}(0.5 - p_k/2) \leq y_i \leq F_{x_i}^{-1}(0.5 + p_k/2)\}$ and $\mathbb{I}$ is the indicator function.

We compute miscalibration of centered intervals at each timestep of a trajectory, where the inputs are the current state-action pairs, and the targets are the state delta: i.e. from Eq. D.4, x_i would be the tuple $(s_{i,t}, a_{i,t})$ and y_i would be $s_{i,t+1} - s_{i,t}$. We used 19 equi-spaced probabilities: $\{p_k = \frac{k}{20}\}_{k=1}^{19}$. Since an ensemble does not provide a closed form quantile function, we use empirical quantiles for $F_{x_i}^{-1}$ by generating many trajectories for a single test sequence of states and actions.

To measure overconfidence, we performed the outer summation over probabilities in Eq. D.4 only if the empirical coverage was lower than p_k:

$$\frac{1}{K}\sum_{k=1}^{K}\min\left(0, \texttt{(empirical coverage at } p_k\texttt{)} - p_k\right), \tag{D.5}$$

D.7 Ignoring Error Correlation Can Lead to Overconfidence

In this section, we show that under certain assumptions, ignoring the correlation between consecutive residuals leads to overconfident predictions. While these assumptions make major simplifications to the problem, this result still gives insight into why overconfidence may grow over time. In what follows, assume that there is a fixed action sequence $a_1, \ldots, a_N$. The corresponding rollout using the true transition function, T, is then $x_1 \ldots, x_{N+1}$.

Ideally, we would compare this to the distribution of rollouts created by sampling autoregressively from $\hat{T}$, which we assume to be a PNN. However since this distribution is difficult to characterize, we focus on analyzing one-step errors. Towards this end, let $\delta_t := \mu_\theta(x_t) - T(x_t)$ and let $\Delta_N := \sum_{t=1}^N \delta_t$. Although the true amount of error after N steps is hard to reason about because of the predicted sequence's autoregressive nature, Δ_N can be thought of as a proxy. We also make the following assumptions:

1. The sequence of residuals is Markovian, i.e. $p(\delta_t|\delta_1, \ldots, \delta_{t-1}) = p(\delta_t|\delta_{t-1})$.
2. The distribution between consecutive residuals is

$$\begin{bmatrix} \delta_t \\ \delta_{t-1} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix} \right)$$

Note that these are assumptions that are made to the true underlying transition function (i.e. that there are correlations between residuals). We will now see that the standard deviation of Δ_N grows faster if positive correlation is present versus if an independence assumption is made (as is usually done with sampling procedure in PNNs).

Proposition 1. *(Deserno [2002]) Following the above assumptions*

$$\Delta_{N+1} = \sqrt{\frac{1+\rho}{1-\rho}} G_{N+1} - \frac{\rho}{1-\rho}\delta_{N+1} + \frac{1-\sqrt{1-\rho^2}}{1-\rho} g_1$$

where $N > 1$, $g_1, \ldots, g_{N+1} \overset{i.i.d}{\sim} \mathcal{N}(0, 1)$, and $G_{N+1} := \sum_{t=1}^{N+1} g_t$.

Note that the first term of Δ_N has standard deviation $\sqrt{\frac{1+\rho}{1-\rho} N}$, and the standard deviation for the rest of the terms does not grow as N increases. In contrast, if one were to model residuals as independent, the standard deviation of the sample trajectories from that model after N steps would be $\sqrt{N}$. Again, the assumptions made here prevent any statement from being made in the actual setting in which trajectories are autoregressively predicted; however, it does give intuition as to why PNNs may produce overconfidence predictive distributions over time.

We now restate the proof from Deserno [2002] for completeness.

Proof. Using the fact that $p(\delta_t|\delta_{t-1} = d) \sim \mathcal{N}(\rho d, 1 - \rho^2)$, we can express the sequence in terms of IID samples as follows:

$$\delta_1 = g_1; \qquad\qquad \delta_t = \rho\delta_{t-1} + \sqrt{1-\rho^2} g_t$$

We can expand the right definition of δ_t to get the following,

$$\delta_t = \rho^{t-1} g_1 + \sqrt{1-\rho^2} \sum_{i=2}^{t} g_i \rho^{t-i}$$

Summing this up to get Δ_N,

$$\begin{aligned}
\Delta_{N+1} &= \sum_{t=1}^{N+1}\left[\rho^{t-1}g_1 + \sqrt{1-\rho^2}\sum_{i=2}^{t} g_i\rho^{t-i}\right] \\
&= g_1\frac{1-\rho^{N+1}}{1-\rho} + \sqrt{1-\rho^2}\sum_{i=2}^{N+1} g_i \sum_{n=i}^{N+1}\rho^{n-i} \\
&= g_1\frac{1-\rho^{N+1}}{1-\rho} + \frac{\sqrt{1-\rho^2}}{1-\rho}\left(\sum_{i=2}^{N+1} g_i - \rho\sum_{i=2}^{N+1} g_i\rho^{N+1-i}\right).
\end{aligned}$$

The first term in the bracket is $G_{N+1} - g_1$, and the second term can be rewritten with δ_{N+1}.

$$\Delta_{N+1} = \sqrt{\frac{1+\rho}{1-\rho}}G_{N+1} - \frac{\rho}{1-\rho}x_N + \frac{1-\sqrt{1-\rho^2}}{1-\rho}g_1$$

$\square$

D.8 Empirical Correlations

We empirically compute temporal correlation between the residuals for models, specifically ensembles of PNNs trained on all our environments. Consider a rollout in the true dynamics - $(s_0, a_0, r_0), \ldots, (s_n, a_n, r_n)$. For a given ensemble member, let $b_0 = \frac{T(s_0,a_0)-\mu_\theta(s_0,a_0)}{\sigma_\theta(s_0,a_0)}, \ldots, b_n = \frac{T(s_n,a_n)-\mu_\theta(s_n,a_n)}{\sigma_\theta(s_n,a_n)}$ be the corresponding sequence of standardized residuals. We make the assumption that successive residuals b_i, b_{i+1} are sampled from a bivariate gaussian with correlation coefficient ρ, that is $\begin{bmatrix} b_i \\ b_{i+1}\end{bmatrix} \sim \mathcal{N}(\mu, \Sigma)$, where $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 1 & \rho \\ \rho & 1\end{bmatrix}$. Then, we compute the maximum likelihood estimator, $\hat{\rho}$, based on the observed residuals $b_1, \ldots, b_n$. The estimates in Table D.2 are averaged over five seeds and five ensemble members in each corresponding ensemble.

D.9 Additional Experimental Results

Additional Calibration Results To better understand how uncertainty estimates change with different sampling procedures, we provide additional plots of the miscalibration and overconfidence metrics. Figures D.2- D.9 show how both metrics change with respect to time for single models and an ensemble. Figures D.10- D.13 show how the metrics change with respect to ensemble size. In all of these, it is

Method	Fusion	Cart Pole	Mountain Random	Mountain Medium	Mountain Expert
Dimension 0	0.69 ± 0.13	0.96 ± 0.04	0.95 ± 0.02	0.94 ± 0.01	0.93 ± 0.02
Dimension 1	0.70 ± 0.04	0.95 ± 0.05	0.93 ± 0.01	0.97 ± 0.01	0.98 ± 0.01
Dimension 2	0.85 ± 0.08	0.96 ± 0.04	0.62 ± 0.10	0.57 ± 0.07	0.51 ± 0.11
Dimension 3	-	0.93 ± 0.07	0.45 ± 0.03	0.66 ± 0.02	0.68 ± 0.07
Dimension 4	-	-	-0.02 ± 0.03	0.13 ± 0.02	0.13 ± 0.03

Table D.2: **Empirical Correlations** Each value is the average over five seeeds and five ensemble members. Note entries in the table that are entry are due to the environment being lower dimensional (e.g. Fusion only has three dimensions).

clear that smooth samples mitigate against overconfidence over time; however, this can also cause uncertainty predictions to be slightly underconfident.

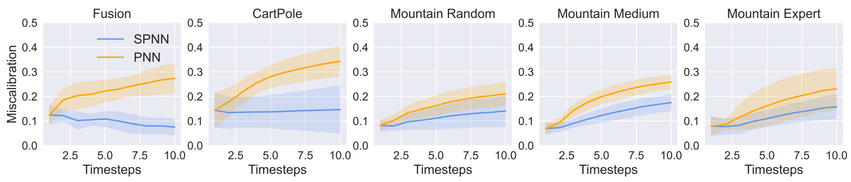


Figure D.2: **Average miscalibration for ID data using a single PNN with respect to rollout step.** The regions shows the standard error over the five seeds.

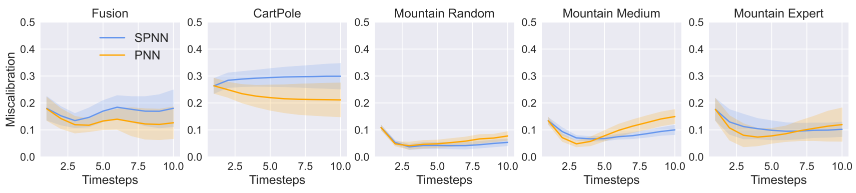


Figure D.3: **Average miscalibration for ID data using an ensemble of PNNs with respect to rollout step.** The regions shows the standard error over the five seeds.

Reinforcement Learning Training Curves We also provide plots of the average returns during training of the policy in Figures D.14- D.18. In general, we see that training with SPNN is less prone to overfitting and often more stable.

Can additional penalty help in the Mountain environment? To prevent the agent from falling off the cliff, it is possible that a higher scale on the penalty could be beneficial. In Table D.3, we test what happens when the penalty scaling is

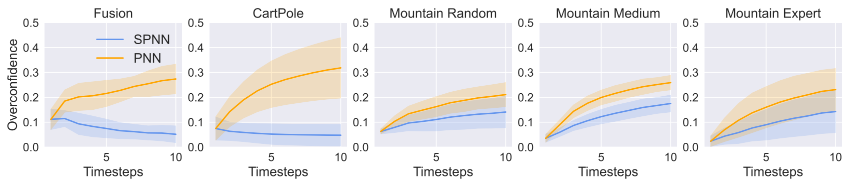


Figure D.4: **Average overconfidence for ID data using a single PNN with respect to rollout step.** The regions shows the standard error over the five seeds.

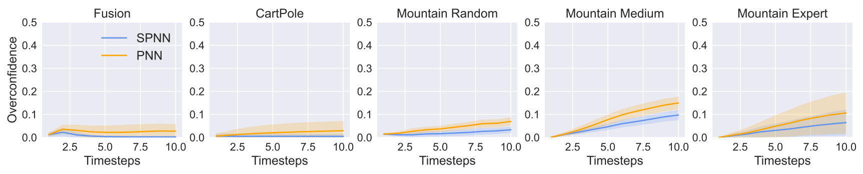


Figure D.5: **Average overconfidence for ID data using an ensemble of PNNs with respect to rollout step.** The regions shows the standard error over the five seeds.

changed to 0.0 or 1.0. We find that greatly increasing the penalty 4× does not have same affect on policy performance as intelligent sampling.

Method	**Mountain Random**	**Mountain Medium**	**Mountain Expert**	**Average**
SPNN Penalty=0.0	61.41 ± 4.47	29.67 ± 0.87	**88.00** ± **1.60**	59.69
PNN Penalty=0.0	63.64 ± 11.41	23.13 ± 1.15	68.77 ± 11.97	51.84
NN Penalty=0.0	31.26 ± 5.68	27.25 ± 0.67	40.71 ± 5.11	33.07
SPNN Penalty=0.25	65.93 ± 1.72	39.08 ± 8.24	84.72 ± 4.73	**63.24**
PNN Penalty=0.25	64.29 ± 4.01	23.39 ± 1.34	44.34 ± 13.45	44.01
NN Penalty=0.25	63.57 ± 9.80	26.79 ± 0.62	49.73 ± 2.50	46.7
SPNN Penalty=1.0	62.05 ± 1.07	**40.45** ± **8.91**	81.78 ± 3.63	61.42
PNN Penalty=1.0	53.06 ± 7.93	30.57 ± 2.33	46.28 ± 4.63	43.3
NN Penalty=1.0	**67.47** ± **10.56**	25.71 ± 0.07	55.07 ± 12.17	49.42

Table D.3: **Normalize policy performances for different penalties on the Mountain environment.** Each result is averaged over the last 20% of evaluations during training. Five seeds were used to compute the average scores, and we show the standard errors.

Mountain Environment Visualization To better understand what is happening in the Mountain environment, we plot the average path taken by each type of policy (see Figure D.19). While all policies are overconfident and have episdoes where the agent falls off the cliff, on average policies trained with SPNN stay within the support of the dataset and avoid falling off the cliff.

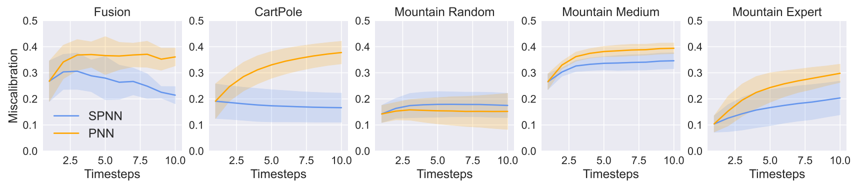


Figure D.6: **Average miscalibration for OOD data using a single PNN with respect to rollout step.** The regions shows the standard error over the five seeds.

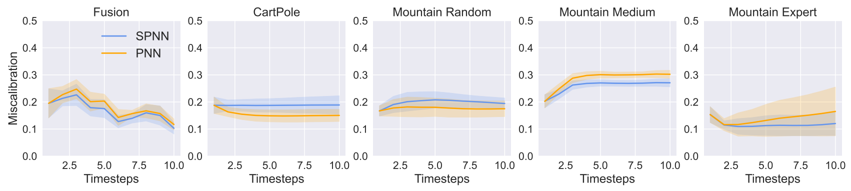


Figure D.7: **Average miscalibration for OOD data using an ensemble of PNNs with respect to rollout step.** The regions shows the standard error over the five seeds.

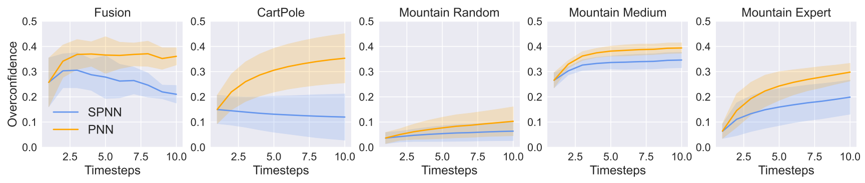


Figure D.8: **Average overconfidence for OOD data using a single PNN with respect to rollout step.** The regions shows the standard error over the five seeds.

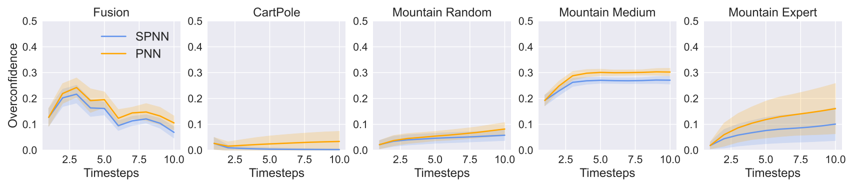


Figure D.9: **Average overconfidence for OOD data using an ensemble of PNNs with respect to rollout step.** The regions shows the standard error over the five seeds.

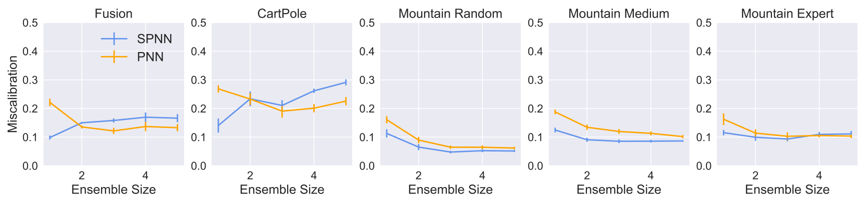


Figure D.10: **Average miscalibration over rollout for ID data with respect to ensemble size.** The error bars shows the standard error over the five seeds.

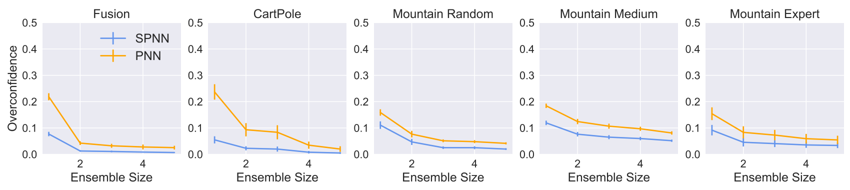


Figure D.11: **Average overconfidence over rollout for ID data with respect to ensemble size.** The error bars shows the standard error over the five seeds.

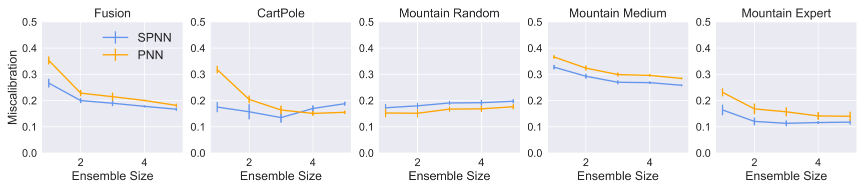


Figure D.12: **Average miscalibration over rollout for OOD data with respect to ensemble size.** The error bars shows the standard error over the five seeds.

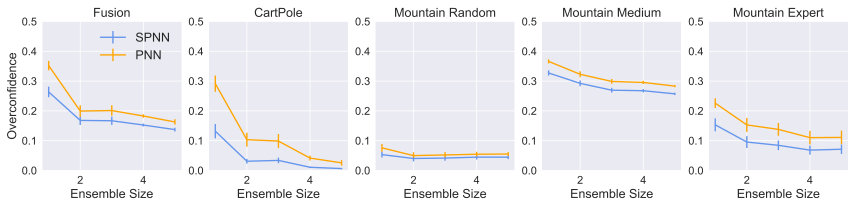


Figure D.13: **Average overconfidence over rollout for OOD data with respect to ensemble size.** The error bars shows the standard error over the five seeds.

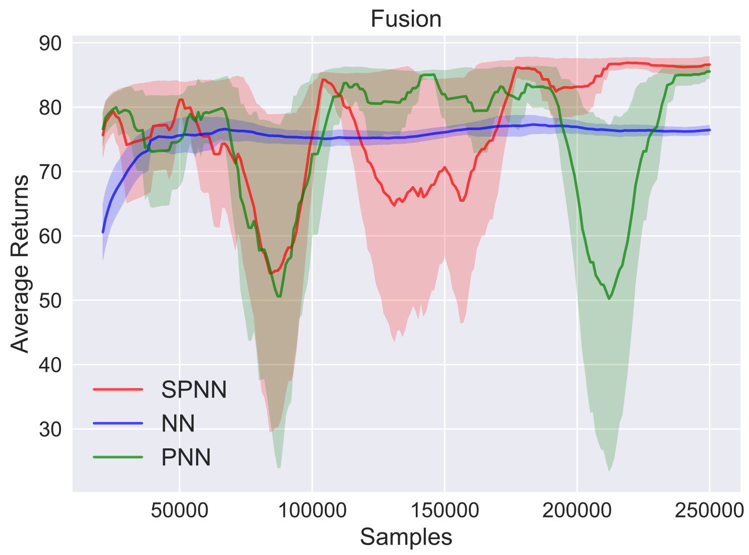


Figure D.14: **Average returns for the Fusion environment during training.** The regions shows the standard error over the five seeds.

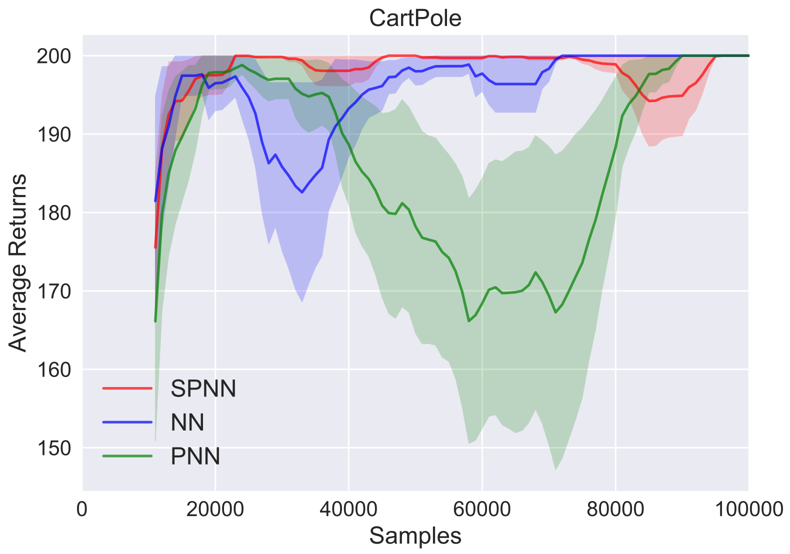


Figure D.15: **Average returns for the Cart Pole environment during training.** The regions shows the standard error over the five seeds.

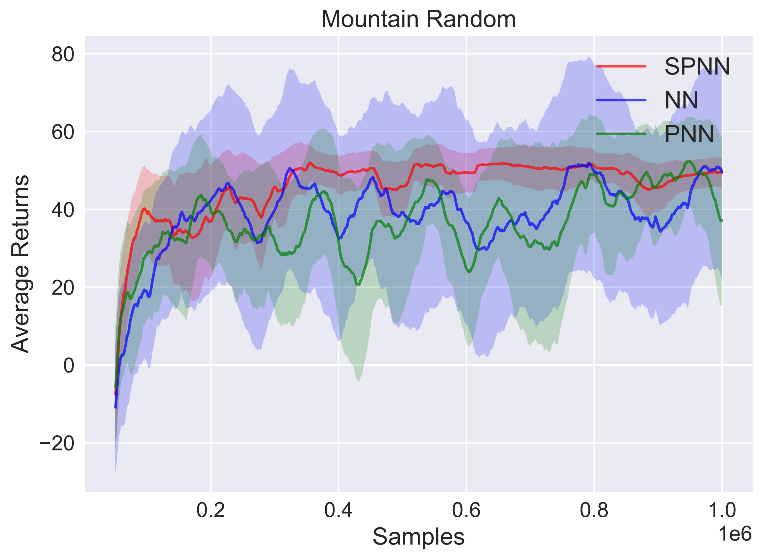


Figure D.16: **Average returns for the Mountain Random environment during training.** The regions shows the standard error over the five seeds.

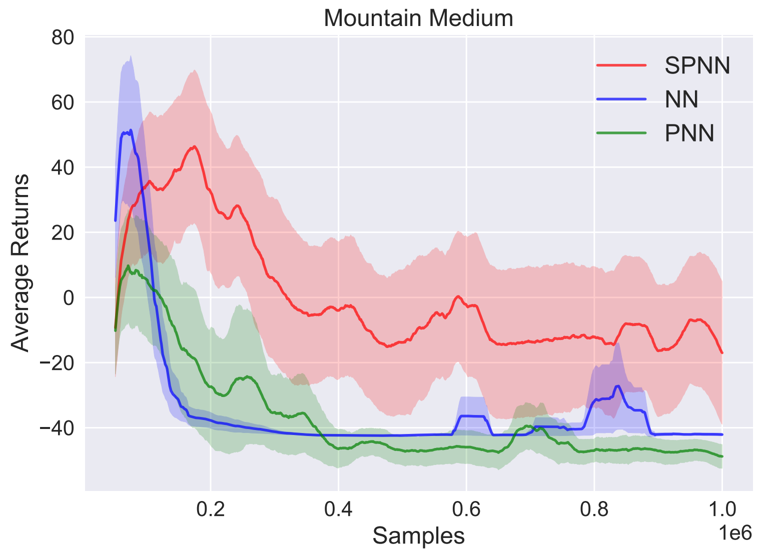


Figure D.17: **Average returns for the Mountain Medium environment during training.** The regions shows the standard error over the five seeds.

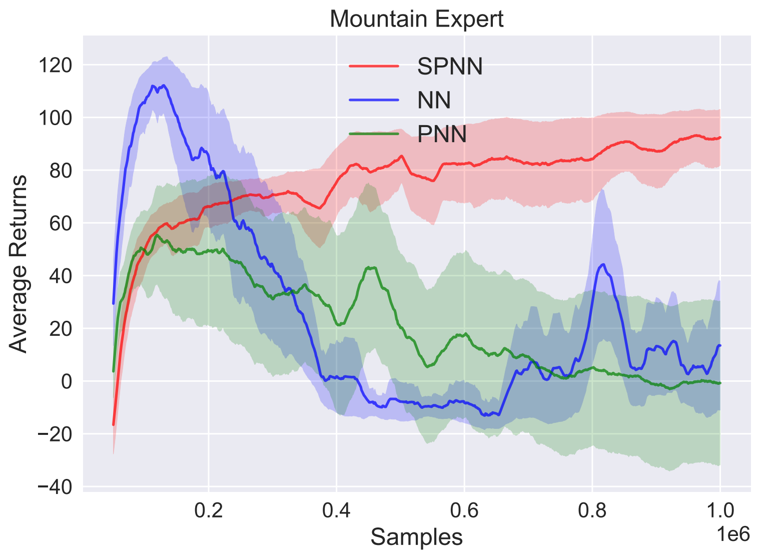


Figure D.18: **Average returns for the Mountain Expert environment during training.** The regions shows the standard error over the five seeds.

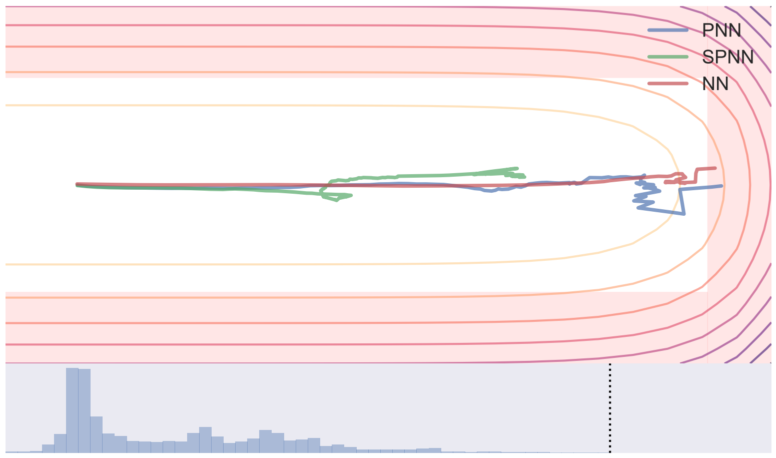


Figure D.19: **Average trajectory in medium mountain ridge environment.** To create this figure, we collect 100 paths in the true environment with each of the five policies corresponding to the random seeds. We then average each path together, and the average path can be seen in the top plot. The red regions represent terminal regions where the agent falls off the cliff, and the contour lines show the contours of the mountain ridge. Here, the x-axis shows the y position of the agent as described in the environment definition (Appendix D.4). The bottom plot shows a histogram of the y data in the Medium dataset, and the dashed black line shows the most extreme recorded y value.

Appendix E

Appendix for Chapter 6

E.1 Implementation

E.1.1 More GTNP Details

Algorithm 5 One Graph Transformer Block

```
Input: $V, E$
for $\nu_i^\ell \in V$ do
    $\nu_i^\ell = \nu_i^\ell + \texttt{MultiHeadedGraphAttention}_i($
            $\texttt{LayerNorm}(V), E)$
    $\nu_i^{\ell+1} = \nu_i^\ell + q_\theta(\texttt{LayerNorm}(\nu_i^\ell))$
end for
Return: $V$
```

Algorithm 5 shows one graph transformer block. Here q_θ is a neural network with one hidden layer neural network with a ReLU activation function. Before multi-headed graph attention and before q_θ we use `LayerNorm` [Ba et al., 2016]. Note that the architecture for this block is almost exactly the same as the blocks in GPT2 [Radford et al., 2019]; however, we use $2d$ hidden units in q_θ following Nguyen and Grover [2022] instead of $4d$.

Autoregressive Training Scheme. We use the same scheme as TNP-A in Nguyen and Grover [2022] in order to efficiently consider multiple context sizes at once per mini-batch update. To do this, instead of using $\mathbf{x}$ we use $\tilde{\mathbf{x}} = \mathbf{x} \oplus \mathbf{x}_{C+1:N}$. The corresponding node embeddings are then $\nu_i = f_\theta^V(\mathbf{y}_i)$ for $i \leq N$ and $\nu_i = f_\theta^V(\mathbf{0})$ if $i > N$. Let $M \in \mathbb{R}^{2N-C \times 2N-C}$ be a matrix of the masks for the attention where

$M_{i,j} = 1$ if the i^{th} vertex can attend to the j^{th} vertex and 0 otherwise. The entries of this matrix are as follows:

$$M_{i,j} = \begin{cases} 1 & j \leq C \\ 1 & i \leq N \text{ and } j \leq i \\ 1 & j - C < i - N \\ 0 & \text{else} \end{cases}$$

This is visualized on page four of Nguyen and Grover [2022]. Effectively, this masking scheme allows target points to attend to observations of other target points occuring earlier in the sequence. Alternatively, we could simply use **x**, the vertex encoding scheme described in Section 6.2.2, and a masking matrix of $M \in \mathbb{R}^{N\times N}$ where

$$M_{i,j} = \begin{cases} 1 & j \leq C \\ 0 & \text{else} \end{cases}$$

This would only consider one context during training (although at test time, the target set could still be predicted autoregressively). This is akin to TNP-D in Nguyen and Grover [2022], which was found to achieve worse log likelihood. We also find that this more straightforward masking scheme yields slightly worse results for GTNP.

E.1.2 More GEENP Details

Algorithm 6 One GEENP Block

```
**Input:** $E$
**for** $E^{\ell}_{i,j} \in E$ **do**
    $E^{\ell}_{i,j} = E^{\ell}_{i,j} + \texttt{MultiHeadedGEAttention}_{i,j}($
            $\texttt{LayerNorm}(E))$
    $E^{\ell+1}_{i,j} = E^{\ell}_{i,j} + q_\theta(\texttt{LayerNorm}(E^{\ell}_{i,j}))$
**end for**
**Return:** $E$
```

Algorithm 6 shows one GEENP block. Note that most of the details are indistinguishable from Algorithm 5 besides the type of attention used and the operation over edge embeddings instead of vertex embeddings.

E.2 Experiment Details

E.2.1 Reproducibility

We include an anonymized pytorch [Paszke et al., 2019b] implementation of GTNP and GEENP in the supplementary material, instructions for installing dependencies, and directions for launching synthetic experiments (i.e. the experiments in Sections 6.3.1 and 6.3.2). See README.md for more details.

E.2.2 Hyperparameters

Table E.1: **Hyperparameters for GTNP and GEENP.**

Batch Size	16
Learning Rate	5e-4
Learning Rate Scheduler	Cosine Annealing
d	64
H	4
Blocks	6
Activation Function	ReLU
f_θ^V and f_θ^E Depth	4
f_θ^V and f_θ^E Width	64
g_θ Depth	1
g_θ Width	128
q_θ width (see Algorithms 5 and 6	128

Table E.1 show the hyperparameters used for GTNP and GEENP. We selected these hyperparameters to be as close as possible to the ones used in TNP for a fair comparison. For each of the other baselines, we use the same hyperparameters they report. In particular, we use the same configurations that appear in `https://github.com/tung-nd/TNP-pytorch/tree/master` and `https://github.com/cambridge-mlg/convcnp`.

E.2.3 Additional Infinite Data Regime Details

Following the procedure in Nguyen and Grover [2022], all methods were trained on 1M batch updates, where each batch contains 16 sampled functions. The exception to this is in the 4D case, where it seemed that models were still learning after 1M batch updates. Thus we increase the number of batch updates to 10M for this setting. For test time, we generate 3,000 of these batches to evaluate on. Like during training, each batch has a random N and C chosen in the ranges of $[6, 50]$ and $[3, 47]$, respectively. We use the code in the official repository for Nguyen and Grover [2022] to generate the test set for the 1D experiment and our own code for the higher dimensions.

E.2.4 Additional Finite Data Regime Details

The GPs that generated the data for the finite datasets have fix When dealing with finite datasets, we shuffle the data every epoch in order to get different context-target splits in the data. We reserve 10% of the total data as a validation set. We train each model until 100 epochs after the best validation loss was observed (with a maximum of 5,000 epochs). We checkpoint the model that achieves the best validation loss and use this during evaluation.

We use a test set with 100,000 data points for each of the dimensions. The same test set is used regardless of how much training data is used. We now describe how to compute the normalized metrics found in Table 6.2. Let s_c be the average log joint likelihood that the trained model achieves on the test set given a context size of c. Furthermore, let f be the true GP that generated the data. We then define s_c^{min} and s_c^{max} as

$$s_c^{\text{min}} = \frac{1}{M}\sum_{m=1}^{M} f(\mathbf{y}_{c+1:N}^{(m)} | \mathbf{x}_{c+1:N}^{(m)})$$
$$s_c^{\text{max}} = \frac{1}{M}\sum_{m=1}^{M} f(\mathbf{y}_{c+1:N}^{(m)} | \mathbf{x}^{(m)}, \mathbf{y}_{1:c}^{(m)})$$

where $\mathbf{x}^{(m)}$ and $\mathbf{y}^{(m)}$ are the m^{th} set of test points. In other words, s_c^{min} is the joint log likelihood using the true prior and s_c^{max} is the joint log likelihood using the true posterior. The final reported score is then

$$100 \times \frac{1}{49}\sum_{c=1}^{49} \frac{s_c - s_c^{\text{min}}}{s_c^{\text{max}} - s_c^{\text{min}}}$$

Note that when M is large this score cannot be over 100; however, it is possible to get a score lower than 0.

E.2.5 Additional Nuclear Fusion Experiment Details

The dimensions of $\mathcal{X}$ for the nuclear fusion data are the current measurements of β_N, rotation, power injected from the neutral beams, torque injected from the neutral beams, and the backward difference of these four measurements. Additionally, because the power and torque injected from the neutral beams are actuators, we include the forward differences for the power and torque. Each shot on average has roughly 107 time steps.

The architecture for the PNN is the same as it appears in Chua et al. [2018b]. In particular, we use a fully connected network with 4 hidden layers each with 200 units. We use the swish activation function [Ramachandran et al., 2017] and have two dedicated heads for the mean and log variance predictions. We use mini-batches of size 256 and a learning rate of 1e-3.

E.2.6 Additional Lipophilicity Experiment Details

The RQ kernel is defined as

$$k_{RQ}(x, x') = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right).$$

Recall that the train split used to train our GTNP models contains sequences of molecule representations. For each of the 1400 sequences, we compute the optimal hyperparameters using L-BFGS that maximise the marginal log-likelihood of the context points in the sequence. The two histograms show the distribution of optimal hyperparameters over the train split. For α in particular, two clear peaks are present, which illustrate the inappropriateness of using a global GP model with a single choice of kernel hyperparameters to fit all the training data.

For the GP (fixed) baseline, α and ℓ were chosen to be right modes in their respective histograms, $\ell = 2.9$, $\alpha = \exp(7.5)$. The scale parameter σ^2 was set to 1 as we are using whitened data.

E.2.7 Compute Details

We train our models using Nvidia Titan Xp GPUs. Other baselines were trained using a mix of these GPUs and Titan X Pascals. Since the cluster we use to train on has many users, it is difficult to get a precise timing comparison; however, we give estimates in Table E.2. Note that TNP and GTNP are considerably faster at evaluation because only one forward pass is necessary to compute the joint log probability due to the masking scheme described in Appendix E.1.1.

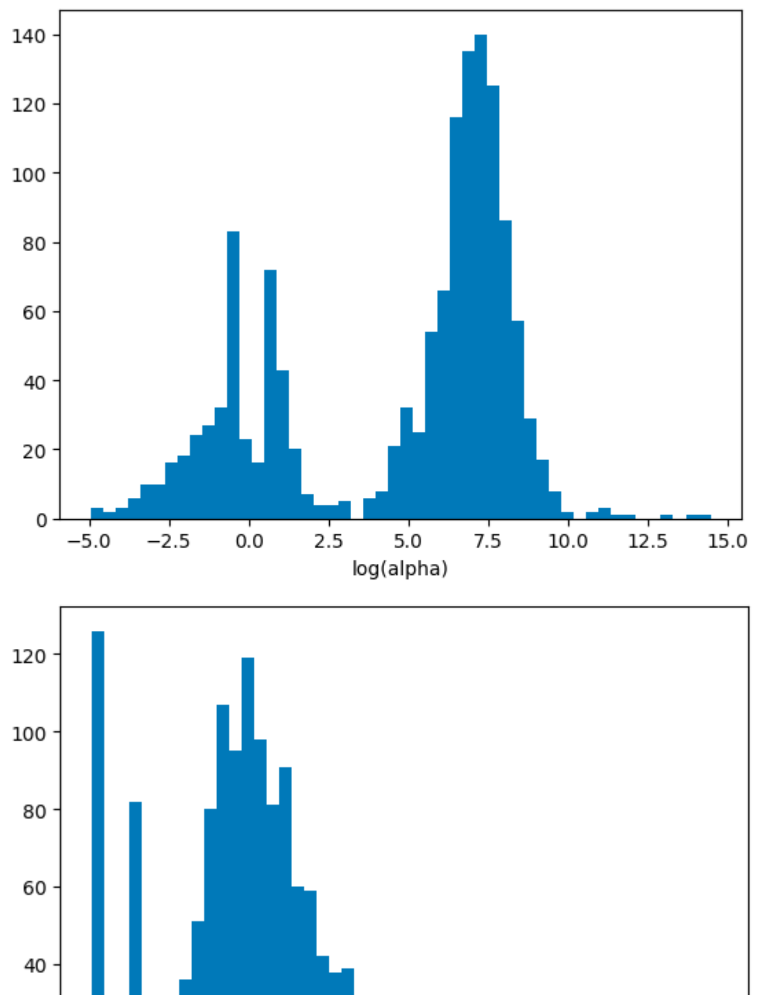


Figure E.1: **Histograms of Optimal** $\log \alpha$ **and** ℓ **RQ Kernel Hyperparameters for the Lipophilicity Task.** Note the two peaks illustrating the multi-modal nature of optimal kernel hyperparamters for our task.

Table E.2: **Approximate Run Times**. Each estimate is for one seed on the infinite 1D experiment.

Method	Train	Evaluation
AttnLNP	45m	2m 23s
AttnCNP	33m	1m 57s
ConvCNP	27m	2m 11s
TNP	52m	16s
GTNP	92m	39s
GEENP	89m	15m 32s

Bibliography

Joseph Abbate, R Conlin, and E Kolemen. Data-driven profile prediction for diii-d. *Nuclear Fusion*, 61(4):046027, 2021.

Joseph Abbate, Rory Conlin, Keith Erickson, and Kolemen Egemen. Data-driven control in tokamaks. *Journal of Plasma Physics (In Submission.)*, 2023.

Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In *Conference on Robot Learning*, pages 403–415. PMLR, 2023.

Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International Conference on Machine Learning*, pages 127–135, 2013.

Kei Akuzawa, Yusuke Iwasawa, and Yutaka Matsuo. Estimating disentangled belief about hidden state and hidden task for meta-reinforcement learning. In *Learning for Dynamics and Control*, pages 73–86. PMLR, 2021.

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

Lucas Alpi. Pid control: Breaking the time barrier, November 2019. URL https://www.novusautomation.com/en/article_PID_control#:~:text=In%201911%20the%20first%20PID,still%20widely%20used%20in%20automation.

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*, pages 6–20. PMLR, 2021.

Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. *Advances in neural information processing systems*, 34:7436–7447, 2021.

H Anand, S Coda, F Felici, C Galperti, and J-M Moret. A novel plasma position and shape controller for advanced configuration development on the tcv tokamak.

Nuclear Fusion, 57(12):126026, 2017.

Kavosh Asadi, Dipendra Misra, and Michael Littman. Lipschitz continuity in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 264–273. PMLR, 2018.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

EA Baltz, E Trask, M Binderbauer, M Dikovsky, H Gota, R Mendoza, JC Platt, and PF Riley. Achievement of sustained net plasma heating in a fusion experiment with the optometrist algorithm. *Scientific reports*, 7(1):6425, 2017.

Laszlo Bardoczi, NC Logan, and EJ Strait. Neoclassical tearing mode seeding by nonlinear three-wave interactions in tokamaks. *Physical Review Letters*, 127(5): 055002, 2021.

Gregory Benton, Wesley Maddox, Sanae Lotfi, and Andrew Gordon Gordon Wilson. Loss surface simplexes for mode connecting volumes and fast ensembling. In *International Conference on Machine Learning*, pages 769–779. PMLR, 2021.

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control, 2016.

Stephen Boyd, Martin Hast, and Karl Johan Åström. Mimo pid tuning via iterated lmi restriction. *International Journal of Robust and Nonlinear Control*, 26(8): 1718–1731, 2016.

Mark Boyer, Josiah Wai, Mitchell Clement, Egemen Kolemen, Ian Char, Youngseog Chung, Willie Neiswanger, and Jeff Schneider. Machine learning for tokamak scenario optimization: combining accelerating physics models and empirical models. In *APS Division of Plasma Physics Meeting Abstracts*, volume 2021, pages PP11–164, 2021a.

Mark D Boyer and Jason Chadwick. Prediction of electron density and pressure profile shapes on nstx-u using neural networks. *Nuclear Fusion*, 61(4):046024, 2021.

MD Boyer, KG Erickson, BA Grierson, DC Pace, JT Scoville, J Rauch, BJ Crowley, JR Ferron, SR Haskey, DA Humphreys, et al. Feedback control of stored energy and rotation with variable beam energy and perveance on diii-d. *Nuclear Fusion*, 59(7):076004, 2019.

MD Boyer, C Rea, and M Clement. Toward active disruption avoidance via real-time estimation of the safe operating region and disruption proximity in tokamaks. *Nuclear Fusion*, 62(2):026005, 2021b.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

Franklin H Branin. Widely convergent method for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research and Development*, 16(5):504–522, 1972.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Wessel P Bruinsma, James Requeima, Andrew YK Foong, Jonathan Gordon, and Richard E Turner. The gaussian neural process. *arXiv preprint arXiv:2101.03606*, 2021.

Wessel P Bruinsma, Stratis Markou, James Requiema, Andrew YK Foong, Tom R Andersson, Anna Vaughan, Anthony Buonomo, J Scott Hosking, and Richard E Turner. Autoregressive conditional neural processes. *arXiv preprint arXiv:2303.14468*, 2023.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12(May):1655–1695, 2011.

RJ Buttery, RJ La Haye, P Gohil, GL Jackson, H Reimerdes, EJ Strait, and DIII-D Team. The influence of rotation on the β n threshold for the 2/ 1 neoclassical tearing mode in diii-d. *Physics of Plasmas*, 15(5):056115, 2008.

Massimo Caccia, Jonas Mueller, Taesup Kim, Laurent Charlin, and Rasool Fakoor. Task-agnostic continual reinforcement learning: In praise of a simple baseline. *arXiv preprint arXiv:2205.14495*, 2022.

Barbara Cannas, Alessandra Fanni, A Murari, Alessandro Pau, Giuliana Sias, and JET EFDA Contributors. Automatic disruption classification based on manifold learning for real-time applications on jet. *Nuclear Fusion*, 53(9):093023, 2013.

Ignacio Carlucho, Mariano De Paula, and Gerardo G Acosta. An adaptive deep reinforcement learning approach for mimo pid control of mobile robots. *ISA transactions*, 102:280–294, 2020.

Andrew Carr and David Wingate. Graph neural processes: Towards bayesian graph neural networks. *arXiv preprint arXiv:1902.10042*, 2019.

Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic markov decision processes, 2019.

Ian Char and Jeff Schneider. Pid-inspired inductive biases for deep reinforcement learning in partially observable control tasks. *arXiv preprint arXiv:2307.05891*, 2023.

Ian Char, Youngseog Chung, Willie Neiswanger, Kirthevasan Kandasamy, Andrew O Nelson, Mark Boyer, Egemen Kolemen, and Jeff Schneider. Offline contextual bayesian optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Ian Char, Viraj Mehta, Adam Villaflor, John M Dolan, and Jeff Schneider. Bats: Best action trajectory stitching. *arXiv preprint arXiv:2204.12026*, 2022.

Ian Char, Joseph Abbate, László Bardóczi, Mark Boyer, Youngseog Chung, Rory Conlin, Keith Erickson, Viraj Mehta, Nathan Richner, Egemen Kolemen, et al. Offline model-based reinforcement learning for tokamak control. In *Learning for Dynamics and Control Conference*, pages 1357–1372. PMLR, 2023a.

Ian Char, Youngseog Chung, Rohan Shah, Willie Neiswanger, and Jeff Schneider. Correlated trajectory uncertainty for adaptive sequential decision making. In *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*, 2023b.

Ian Char, Youngseog Chung, Joseph Abbate, Egemen Kolemen, and Jeff Schneider. Full shot predictions for the diii-d tokamak via deep recurrent networks. *arXiv preprint arXiv:2404.12416*, 2024.

Hengwei Chen and Jürgen Bajorath. Meta-learning for transformer-based prediction of potent compounds. *Scientific Reports*, 13(1):16145, 2023.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021a.

Ruijie Chen, Ning Gao, Ngo Anh Vien, Hanna Ziesche, and Gerhard Neumann. Meta-learning regrasping strategies for physical-agnostic objects. *arXiv preprint arXiv:2205.11110*, 2022.

Xinyue Chen, Zijian Zhou, Zheng Wang, Che Wang, Yanqiu Wu, and Keith Ross. Bail: Best-action imitation learning for batch deep reinforcement learning, 2020.

Xiong-Hui Chen, Yang Yu, Qingyang Li, Fan-Ming Luo, Zhiwei Qin, Wenjie Shang, and Jieping Ye. Offline model-based adaptable policy learning. *Advances in Neural Information Processing Systems*, 34:8432–8443, 2021b.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio.

On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014a.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014b.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114, 2018a. URL http://arxiv.org/abs/1805.12114.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018b.

Youngseog Chung, Ian Char, Willie Neiswanger, Kirthevasan Kandasamy, Andrew Oakleigh Nelson, Mark D Boyer, Egemen Kolemen, and Jeff Schneider. Offline contextual bayesian optimization for nuclear fusion. *arXiv preprint arXiv:2001.01793*, 2020.

Youngseog Chung, Ian Char, Han Guo, Jeff Schneider, and Willie Neiswanger. Uncertainty toolbox: an open-source library for assessing, visualizing, and improving uncertainty quantification. *arXiv preprint arXiv:2109.10254*, 2021a.

Youngseog Chung, Willie Neiswanger, Ian Char, and Jeff Schneider. Beyond pinball loss: Quantile methods for calibrated uncertainty quantification. *Advances in Neural Information Processing Systems*, 34:10971–10984, 2021b.

S Coda, M Agostini, Raffaele Albanese, S Alberti, E Alessi, S Allan, J Allcock, R Ambrosino, H Anand, Y Andrèbe, et al. Physics research on the tcv tokamak facility: from conventional to alternative scenarios and beyond. *Nuclear Fusion*, 59(11):112023, 2019.

Rory Conlin, Keith Erickson, Joseph Abbate, and Egemen Kolemen. Keras2c: A library for converting keras neural networks to real-time compatible c. *Engineering Applications of Artificial Intelligence*, 100:104182, 2021.

Teodora Constantinescu, Claudiu Nicolae Lungu, and Ildiko Lung. Lipophilicity as a central component of drug-like properties of chalchones and flavonoid derivatives. *Molecules*, 24(8):1505, 2019.

Ronald James Cotton, Fabian Sinz, and Andreas Tolias. Factorized neural processes for neural processes: K-shot prediction of neural responses. *Advances in Neural Information Processing Systems*, 33:11368–11379, 2020.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

Iva Čvorović-Hajdinjak, Andjelka B Kovačević, Dragana Ilić, Luka Č Popović, Xinyu Dai, Isidora Jankov, Viktor Radović, Paula Sánchez-Sáez, and Robert Nikutta. Conditional neural process for nonparametric modeling of active galactic nuclei light curves. *Astronomische Nachrichten*, 343(1-2):e210103, 2022.

Ben Day, Cătălina Cangea, Arian R Jamasb, and Pietro Liò. Message passing neural processes. *arXiv preprint arXiv:2009.13895*, 2020.

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. *Nature*, 602(7897):414–419, 2022.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Markus Deserno. How to generate exponentially correlated gaussian random numbers. *Department of Chemistry and Biochemistry UCLA, USA*, 2002.

Ron Dorfman, Idan Shenfeld, and Aviv Tamar. Offline meta learning of exploration. *arXiv preprint arXiv:2008.02598*, 2020.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Robust predictable control. *Advances in Neural Information Processing Systems*, 34:27813–27825, 2021.

Alexander Fabisch and Jan Hendrik Metzen. Active contextual policy search. *The Journal of Machine Learning Research*, 15(1):3371–3399, 2014.

Alexander Fabisch, Jan Hendrik Metzen, Mario Michael Krell, and Frank Kirchner.

Accounting for task-difficulty in active multi-task robot control learning. *KI-Künstliche Intelligenz*, 29(4):369–377, 2015.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

F Felici and O Sauter. Non-linear model-based optimization of actuator trajectories for tokamak plasma profile control. *Plasma Physics and Controlled Fusion*, 54 (2):025002, 2012.

F Felici, O Sauter, S Coda, BP Duval, TP Goodman, JM Moret, JI Paley, TCV Team, et al. Real-time physics-model-based simulation of the current density profile in tokamak plasmas. *Nuclear Fusion*, 51(8):083052, 2011.

Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM J. Comput.*, 40(6):1662–1714, December 2011. ISSN 0097-5397. doi: 10.1137/10080484X. URL https://doi.org/10.1137/10080484X.

Norman Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. *CoRR*, abs/1207.4114, 2012. URL http://arxiv.org/abs/1207.4114.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020a.

Yichen Fu, David Eldon, Keith Erickson, Kornee Kleijwegt, Leonard Lupin-Jimenez, Mark D Boyer, Nick Eidietis, Nathaniel Barbour, Olivier Izacard, and Egemen Kolemen. Machine learning control for disruption and tearing mode avoidance. *Physics of Plasmas*, 27(2):022501, 2020b.

Fumitake Fujii, Akinori Kaneishi, Takafumi Nii, Ryu'ichiro Maenishi, and Soma Tanaka. Self-tuning two degree-of-freedom proportional–integral control system based on reinforcement learning for a multiple-input multiple-output industrial process that suffers from spatial input coupling. *Processes*, 9(3):487, 2021.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International conference on machine learning*, pages 1704–1713. PMLR, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Dibya Ghosh, Anurag Ajay, Pulkit Agrawal, and Sergey Levine. Offline rl policies should be trained to be adaptive. In *International Conference on Machine Learning*, pages 7513–7530. PMLR, 2022.

Mark N Gibbs. *Bayesian Gaussian processes for regression and classification*. PhD thesis, Citeseer, 1998.

David Ginsbourger, Jean Baccou, Clément Chevalier, Frédéric Perales, Nicolas Garland, and Yann Monerie. Bayesian adaptive reconstruction of profile optima and optimizers. *SIAM/ASA Journal on Uncertainty Quantification*, 2(1):490–510, 2014.

Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1):163–223, 2003. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(02)00376-4. URL https://www.sciencedirect.com/science/article/pii/S0004370202003764.

Tilmann Gneiting, Fadoua Balabdaoui, and Adrian E Raftery. Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):243–268, 2007.

P Gohil, KH Burrell, RJ Groebner, J Kim, WC Martin, EL McKee, and RP Seraydarian. The charge exchange recombination diagnostic system on the diii-d tokamak. Technical report, General Atomics, 1991.

Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

BA Grierson, MA Van Zeeland, JT Scoville, B Crowley, I Bykov, JM Park, WW Heidbrink, A Nagy, SR Haskey, and D Liu. Testing the diii-d co/counter off-axis neutral beam injected power and ability to balance injected torque. *Nuclear Fusion*, 61(11):116049, 2021.

Ryan-Rhys Griffiths, Leo Klarner, Henry Moss, Aditya Ravuri, Sang T. Truong, Yuanqi Du, Samuel Don Stanton, Gary Tom, Bojana Ranković, Arian Rokkum Jamasb, Aryan Deshwal, Julius Schwartz, Austin Tripp, Gregory Kell, Simon Frieder, Anthony Bourached, Alex James Chan, Jacob Moss, Chengzhi Guo, Johannes P. Dürholt, Saudamini Chaurasia, Ji Won Park, Felix Strieth-Kalthoff, Alpha Lee, Bingqing Cheng, Alan Aspuru-Guzik, Philippe Schwaller, and Jian Tang. GAUCHE: A library for gaussian processes in chemistry. In *Thirty-*

seventh Conference on Neural Information Processing Systems, 2023. URL https://openreview.net/forum?id=vzrA6uqOis.

Matthew Groves, Michael Pearce, and Juergen Branke. On parallelizing multi-task bayesian optimization. In *2018 Winter Simulation Conference (WSC)*, pages 1993–2002. IEEE, 2018.

Zhe Guan and Toru Yamamoto. Design of a reinforcement learning pid controller. *IEEJ Transactions on Electrical and Electronic Engineering*, 16(10):1354–1360, 2021.

Carlos Guestrin, Andreas Krause, and Ajit Paul Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 265–272. ACM, 2005.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018a. URL http://arxiv.org/abs/1801.01290.

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

Sang-hee Hahn, YJ Kim, BG Penaflor, JG Bak, H Han, JS Hong, YM Jeon, JH Jeong, M Joung, JW Juhn, et al. Progress and plan of kstar plasma control system upgrade. *Fusion Engineering and Design*, 112:687–691, 2016.

Dongqi Han, Kenji Doya, and Jun Tani. Variational recurrent models for solving partially observable control tasks. *arXiv preprint arXiv:1912.10703*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural* *computation*, 9(8):1735–1780, 1997.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

Hao Hu, Jianing Ye, Zhizhou Ren, Guangxiang Zhu, and Chongjie Zhang. Generalizable episodic memory for deep reinforcement learning. *CoRR*, abs/2103.06469, 2021. URL https://arxiv.org/abs/2103.06469.

David Humphreys, G Ambrosino, Peter de Vries, Federico Felici, Sun H Kim, Gary Jackson, A Kallenbach, Egemen Kolemen, J Lister, D Moreau, et al. Novel aspects of plasma control in iter. *Physics of Plasmas*, 22(2):021806, 2015.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*, pages 2117–2126. PMLR, 2018.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *Advances in neural information processing systems*, 32, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Clemens Isert, Jimmy C Kromann, Nikolaus Stiefl, Gisbert Schneider, and Richard A Lewis. Machine learning for fast, quantum mechanics-based approximation of drug lipophilicity. *ACS omega*, 8(2):2046–2056, 2023.

Azarakhsh Jalalvand, Joseph Abbate, Rory Conlin, Geert Verdoolaege, and Egemen Kolemen. Real-time and adaptive reservoir computing with application to profile prediction in fusion plasma. *IEEE Transactions on Neural Networks and Learning Systems*, 33(6):2630–2641, 2021.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019a.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization, 2019b.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Hozefa Jesawada, Amol Yerudkar, Carmen Del Vecchio, and Navdeep Singh. A model-based reinforcement learning approach for pid design. *arXiv preprint arXiv:2206.03567*, 2022.

Saurav Jha, Dong Gong, Xuesong Wang, Richard E Turner, and Lina Yao. The neural process family: Survey, applications and perspectives. *arXiv preprint arXiv:2209.00517*, 2022.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 652–661. PMLR, 2016.

Yuankun Jiang, Chenglin Li, Wenrui Dai, Junni Zou, and Hongkai Xiong. Monotonic robust policy optimization with model discrepancy. In *International Conference on Machine Learning*, pages 4951–4960. PMLR, 2021.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101 (1-2):99–134, 1998.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabás Póczos. Parallelised bayesian optimisation via thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pages 133–142, 2018.

Kirthevasan Kandasamy, Willie Neiswanger, Reed Zhang, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos. Myopic posterior sampling for adaptive goal oriented design of experiments. In *Proceedings of the 36th International Conference on Machine Learning*. JMLR. org, 2019a.

Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *arXiv preprint arXiv:1903.06694*, 2019b.

Andrej Karpathy. nanogpt, 2022–. URL https://github.com/karpathy/nanoGPT.

Julian Kates-Harbeck, Alexey Svyatkovskiy, and William Tang. Predicting disruptive instabilities in controlled fusion plasmas through deep learning. *Nature*, page 1, 2019.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

Makoto Kawano, Wataru Kumagai, Akiyoshi Sannai, Yusuke Iwasawa, and Yutaka Matsuo. Group equivariant conditional neural processes. *arXiv preprint arXiv:2102.08759*, 2021.

Zander Keith, Chirag Nagpal, Cristina Rea, and R Alex Tinguely. Risk-aware framework development for disruption prediction: Alcator c-mod and diii-d survival analysis. 2024.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020a.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21810–21823. Curran Associates, Inc., 2020b. URL https://proceedings.neurips.cc/paper/2020/file/f7efa4f864ae9b88d43527f4b14f750f-Paper.pdf.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *Int. J. Rob. Res.*, 32(11):1238–1274, September 2013. ISSN 0278-3649. doi: 10.1177/0278364913495721. URL https://doi.org/10.1177/0278364913495721.

Andreas Krause and Cheng S Ong. Contextual gaussian process bandit optimization. In *Advances in Neural Information Processing Systems*, pages 2447–2455, 2011.

Tipaluck Krityakierne and David Ginsbourger. Global optimization with sparse and local gaussian process models. In *International Workshop on Machine Learning, Optimization and Big Data*, pages 185–196. Springer, 2015.

Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. In *International conference on machine learning*, pages 2796–2804. PMLR, 2018.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

LL Lao, H St John, RD Stambaugh, AG Kellman, and W Pfeiffer. Reconstruction of current profile parameters and plasma shapes in tokamaks. *Nuclear fusion*, 25 (11):1611, 1985.

Nathan P Lawrence, Gregory E Stewart, Philip D Loewen, Michael G Forbes, Johan U Backstrom, and R Bhushan Gopaluni. Reinforcement learning based design of linear fixed structure controllers. *IFAC-PapersOnLine*, 53(2):230–235, 2020.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Shunjie Li, H Jiang, Zhigang Ren, and C Xu. Optimal tracking for a divergent-type parabolic pde system in current profile control. In *Abstract and Applied Analysis*, volume 2014. Hindawi, 2014.

Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye. A survey on transformers in reinforcement learning. *arXiv preprint arXiv:2301.03044*, 2023.

Xiang Li, Viraj Mehta, Johannes Kirschner, Ian Char, Willie Neiswanger, Jeff

Schneider, Andreas Krause, and Ilija Bogunovic. Near-optimal policy identification in active reinforcement learning. *arXiv preprint arXiv:2212.09510*, 2022a.

Yumeng Li, Ning Gao, Hanna Ziesche, and Gerhard Neumann. Category-agnostic 6d pose estimation with conditional neural processes. *arXiv preprint arXiv:2206.07162*, 2022b.

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. arxiv e-prints, page. *arXiv preprint arXiv:1712.09381*, 2017.

Huidong Liang and Junbin Gao. How neural processes improve graph link prediction. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3543–3547. IEEE, 2022.

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

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Lennart Ljung. Perspectives on system identification. *Annual Reviews in Control*, 34(1):1–12, 2010.

Christos Louizos, Xiahan Shi, Klamer Schutte, and Max Welling. The functional neural process. *Advances in Neural Information Processing Systems*, 32, 2019.

Xingyu Lu, Kimin Lee, Pieter Abbeel, and Stas Tiomkin. Dynamics generalization via information bottleneck in deep reinforcement learning. *arXiv preprint arXiv:2008.00614*, 2020.

James L Luxon. A design retrospective of the diii-d tokamak. *Nuclear Fusion*, 42 (5):614, 2002.

M Margo, B Penaflor, H Shen, J Ferron, D Piglowski, P Nguyen, J Rauch, M Clement, A Battey, and C Rea. Current state of diii-d plasma control system. *Fusion Engineering and Design*, 150:111368, 2020.

Henrik Marklund, Suraj Nair, and Chelsea Finn. Exact (then approximate) dynamic programming for deep reinforcement learning. *Bian and Invariances Workshop, ICML*, 2020.

Stratis Markou, James Requeima, Wessel P Bruinsma, Anna Vaughan, and Richard E Turner. Practical conditional neural processes via tractable dependent predictions. *arXiv preprint arXiv:2203.08775*, 2022.

La Marzocco. A brief history of the pid, October 2015. URL https://home.lamarzoccousa.com/history-of-the-pid/.

Dale Meade. 50 years of fusion research. *Nuclear Fusion*, 50(1):014004, 2009.

Viraj Mehta, Ian Char, Willie Neiswanger, Youngseog Chung, Andrew Nelson, Mark Boyer, Egemen Kolemen, and Jeff Schneider. Neural dynamical systems: Balancing structure and flexibility in physical prediction. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 3735–3742. IEEE, 2021a.

Viraj Mehta, Biswajit Paria, Jeff Schneider, Stefano Ermon, and Willie Neiswanger. An experimental design perspective on model-based reinforcement learning. *arXiv preprint arXiv:2112.05244*, 2021b.

Viraj Mehta, Ian Char, Joseph Abbate, Rory Conlin, Mark D Boyer, Stefano Ermon, Jeff Schneider, and Willie Neiswanger. Exploration via planning for information about the optimal trajectory. *arXiv preprint arXiv:2210.04642*, 2022.

Viraj Mehta, Joseph Abbate, Allen Wang, Andrew Rothstein, Ian Char, Jeff Schneider, Egemen Kolemen, Cristina Rea, and Darren Garnier. Towards llms as operational copilots for fusion reactors. In *NeurIPS 2023 AI for Science Workshop*, 2023a.

Viraj Mehta, Vikramjeet Das, Ojash Neopane, Yijia Dai, Ilija Bogunovic, Jeff Schneider, and Willie Neiswanger. Sample efficient reinforcement learning from human feedback via active exploration. 2023b.

Viraj Mehta, Ojash Neopane, Vikramjeet Das, Sen Lin, Jeff Schneider, and Willie Neiswanger. Kernelized offline contextual dueling bandits. *arXiv preprint arXiv:2307.11288*, 2023c.

Viraj Mehta, Jayson Barr, Joseph Abbate, Mark D Boyer, Ian Char, Willie Neiswanger, Egemen Kolemen, and Jeff Schneider. Automated experimental design of safe rampdowns via probabilistic machine learning. *Nuclear Fusion*, 64(4):046014, 2024.

Luckeciano C Melo. Transformers are meta-reinforcement learners. In *International Conference on Machine Learning*, pages 15340–15359. PMLR, 2022.

Lingheng Meng, Rob Gorbet, and Dana Kulić. Memory-based deep reinforcement learning for pomdps. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5619–5626. IEEE, 2021.

Jan Hendrik Metzen. Active contextual entropy search. *arXiv preprint arXiv:1511.04211*, 2015.

Randy R Miller, Maria Madeira, Harold B Wood, Wayne M Geissler, Conrad E Raab, and Iain J Martin. Integrating the impact of lipophilicity on potency and pharmacokinetic parameters enables the use of diverse chemical space during small molecule drug optimization. *Journal of Medicinal Chemistry*, 63(21): 12156–12170, 2020.

Nicolas Minorsky. Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309, 1922.

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

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

Kevin Joseph Montes, Cristina Rea, Robert Granetz, Roy Alexander Tinguely, Nicholas W Eidietis, O Meneghini, Dalong Chen, Biao Shen, Bingjia Xiao, Keith Erickson, et al. Machine learning for disruption warning on alcator c-mod, diii-d, and east. *Nuclear Fusion*, 2019.

Hirotomo Moriwaki, Yu-Shi Tian, Norihito Kawashita, and Tatsuya Takagi. Mordred: a molecular descriptor calculator. *Journal of cheminformatics*, 10(1):1–14, 2018.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*, 2021.

Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020a.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020b.

Marcel Nassar. Hierarchical bipartite graph convolution networks. *arXiv preprint arXiv:1812.03813*, 2018.

Marcel Nassar, Xin Wang, and Evren Tumer. Conditional graph neural processes: A functional autoencoder approach. *arXiv preprint arXiv:1812.05212*, 2018.

Tung Nguyen and Aditya Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. *arXiv preprint arXiv:2207.04179*, 2022.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl is a strong baseline for many pomdps. *arXiv preprint arXiv:2110.05038*, 2021.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free RL can be a strong baseline for many POMDPs. In *International Conference on Machine Learning*, pages 16691–16723. PMLR, 2022.

David A Nix and Andreas S Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 ieee international conference on neural networks (ICNN'94)*, volume 1, pages 55–60. IEEE, 1994.

Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–. URL https://github.com/fmfn/BayesianOptimization.

Junier B Oliva, Barnabás Póczos, and Jeff Schneider. The statistical recurrent unit. In *International Conference on Machine Learning*, pages 2671–2680. PMLR, 2017.

OpenAI. Gpt-4 technical report. *arXiv*, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Ari Pakman, Yueqi Wang, Catalin Mitelut, JinHyung Lee, and Liam Paninski. Neural clustering processes. In *International Conference on Machine Learning*, pages 7455–7465. PMLR, 2020.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pages 7487–7498. PMLR, 2020.

Young-Jin Park and Han-Lim Choi. A neural process approach for probabilistic reconstruction of no-data gaps in lunar digital elevation maps. *Aerospace Science and Technology*, 113:106672, 2021.

Matthew S Parsons. Interpretation of machine-learning-based disruption models for plasma control. *Plasma Physics and Controlled Fusion*, 59(8):085001, 2017.

Edoardo Pasolli and Farid Melgani. Gaussian process regression within an active learning scheme. In *2011 IEEE International Geoscience and Remote Sensing Symposium*, pages 3574–3577. IEEE, 2011.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019a. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-libr pdf.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019b.

M Pearce and J Branke. Efficient information collection on portfolios. *Warwick Research Archive Portal*, 2017.

Michael Pearce and Juergen Branke. Continuous multi-task bayesian optimisation with correlation. *European Journal of Operational Research*, 270(3):1074–1085, 2018.

PA Politzer, CC Petty, RJ Jayakumar, TC Luce, MR Wade, JC DeBoo, JR Ferron, P Gohil, CT Holcomb, AW Hyatt, et al. Influence of toroidal rotation on transport and stability in hybrid scenario plasmas in diii-d. *Nuclear Fusion*, 48(7):075001, 2008.

Alexander Pondaven, Märt Bakler, Donghu Guo, Hamzah Hashim, Martin Ignatov, and Harrison Zhu. Convolutional neural processes for inpainting satellite images. *arXiv preprint arXiv:2205.12407*, 2022.

Vitchyr Pong and Ashvin Nair. rlkit. https://github.com/rail-berkeley/rlkit, 2018–.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. *CoRR*, abs/1703.01988, 2017. URL http://arxiv.org/abs/1703.01988.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr, 2021.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2005. ISBN 9780262182539.

Christina Rea, KJ Montes, KG Erickson, RS Granetz, and RA Tinguely. A real-time machine learning-based disruption predictor in diii-d. *Nuclear Fusion*, 59(9): 096016, 2019.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

H Reimerdes, AM Garofalo, GL Jackson, M Okabayashi, EJ Strait, Ming-Sheng Chu, Y In, RJ La Haye, MJ Lanctot, YQ Liu, et al. Reduced critical rotation for resistive-wall mode stabilization in a near-axisymmetric configuration. *Physical review letters*, 98(5):055001, 2007.

P Rodriguez-Fernandez, AJ Creely, MJ Greenwald, D Brunner, SB Ballinger, CP Chrobak, DT Garnier, R Granetz, ZS Hartwig, NT Howard, et al. Overview of the sparc physics basis towards the exploration of burning-plasma regimes in high-field, compact tokamaks. *Nuclear Fusion*, 62(4):042003, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

Daniel Russo and Benjamin Van Roy. An information-theoretic analysis of thompson sampling. *The Journal of Machine Learning Research*, 17(1):2442–2471, 2016.

Johan Schoukens and Lennart Ljung. Nonlinear system identification: A user-oriented road map. *IEEE Control Systems Magazine*, 39(6):28–99, 2019.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

JT Scoville, DA Humphreys, JR Ferron, and P Gohil. Simultaneous feedback control of plasma rotation and stored energy on the diii-d tokamak. *Fusion engineering and design*, 82(5-14):1045–1050, 2007.

Maximilian Seitzer, Arash Tavakoli, Dimitrije Antic, and Georg Martius. On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. *arXiv preprint arXiv:2203.09168*, 2022.

J Seo, Y-S Na, B Kim, CY Lee, MS Park, SJ Park, and YH Lee. Development of an operation trajectory design algorithm for control of multiple 0d parameters using deep reinforcement learning in kstar. *Nuclear Fusion*, 62(8):086049, 2022.

Jaemin Seo, Y-S Na, B Kim, CY Lee, MS Park, SJ Park, and YH Lee. Feedforward beta control in the kstar tokamak by deep reinforcement learning. *Nuclear Fusion*, 61(10):106010, 2021.

Jaemin Seo, SangKyeun Kim, Azarakhsh Jalalvand, Rory Conlin, Andrew Rothstein, Joseph Abbate, Keith Erickson, Josiah Wai, Ricardo Shousha, and Egemen Kolemen. Avoiding fusion plasma tearing instability with deep reinforcement learning. *Nature*, 626(8000):746–751, 2024.

Sambu Seo, Marko Wallat, Thore Graepel, and Klaus Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Mustererkennung 2000*, pages 27–34. Springer, 2000.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

Ricardo Shousha, Keith Erickson, and Egemen Kolemen. Development and experimental demonstration of real-time kinetic profile fitting algorithm for improved equilibrium reconstruction (rtefit) and control on diii-d. *Journal of Plasma Physics (In Submission.)*, 2023.

Aayam Kumar Shrestha, Stefan Lee, Prasad Tadepalli, and Alan Fern. Deepaveragers: Offline reinforcement learning by solving derived non-parametric {mdp}s. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=eMP1j9efXtX.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR, 2019.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Hao Song, Tom Diethe, Meelis Kull, and Peter Flach. Distribution calibration for regression. In *International Conference on Machine Learning*, pages 5897–5906. PMLR, 2019.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. In *Advances in neural information processing systems*, pages 2004–2012, 2013.

William Tang, Matthew Parsons, Eliot Feibush, A Murari, J Vega, A Pereira, and J Choi. Big data machine learning for disruption predictions. In *26th IAEA Fusion Energy Conference-IAEA CN-234, Paper Number EX/P6–47*, 2016.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste

Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148. PMLR, 2016.

B Tobias, M Chen, IGJ Classen, CW Domier, R Fitzpatrick, BA Grierson, NC Luhmann Jr, CM Muscatello, M Okabayashi, KEJ Olofsson, et al. Rotation profile flattening and toroidal flow shear reversal due to the coupling of magnetic islands in tokamaks. *Physics of Plasmas*, 23(5):056107, 2016.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, et al. Xsede: accelerating scientific discovery. *Computing in science & engineering*, 16(5):62–74, 2014.

Brendan D Tracey, Andrea Michi, Yuri Chervonyi, Ian Davies, Cosmin Paduraru, Nevena Lazic, Federico Felici, Timo Ewalds, Craig Donner, Cristian Galperti, et al. Towards practical reinforcement learning for tokamak magnetic control. *arXiv preprint arXiv:2307.11546*, 2023.

ITER Physics Expert Group on Confinement Transport, , ITER Physics Expert Group on Confinement Modelling Database, , and ITER Physics Basis Editors. Chapter 2: Plasma confinement and transport. *Nuclear Fusion*, 39(12):2175–2249, 1999.

Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1009–1024. ACM, 2017.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

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

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

T Wakatsuki, T Suzuki, N Oyama, and N Hayashi. Ion temperature gradient control using reinforcement learning technique. *Nuclear Fusion*, 61(4):046036, 2021.

Michael L Walker, Peter De Vries, Federico Felici, and Eugenio Schuster. Introduction to tokamak plasma control. In *2020 American Control Conference (ACC)*, pages 2901–2918. IEEE, 2020.

ML Walker, DA Humphreys, and JR Ferron. Multivariable shape control development on the diii-d tokamak. In *17th IEEE/NPSS Symposium Fusion Engineering (Cat. No. 97CH36131)*, volume 1, pages 556–559. IEEE, 1997.

ML Walker, DA Humphreys, JA Leuer, JR Ferron, and BG Penaflor. Implementation of model-based multivariable control on diii-d. *GA-A23468*, 2000.

Allen M Wang, Darren T Garnier, and Cristina Rea. Hybridizing physics and neural odes for predicting plasma inductance dynamics in tokamak fusion reactors. *arXiv preprint arXiv:2310.20079*, 2023.

Allen M Wang, Oswin So, Charles Dawson, Darren T Garnier, Cristina Rea, and Chuchu Fan. Active disruption avoidance and trajectory design for tokamak ramp-downs with neural differential equations and reinforcement learning. *arXiv preprint arXiv:2402.09387*, 2024.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Junhang Wang, Jianing Zhu, Chao Zou, Linlin Ou, and Xinyi Yu. Robust adaptive pid control based on reinforcement learning for mimo nonlinear six-joint manipulator. In *2021 China Automation Congress (CAC)*, pages 2633–2638. IEEE, 2021a.

Rui Wang and Rose Yu. Physics-guided deep learning for dynamical systems: A survey. *arXiv preprint arXiv:2107.01272*, 2021.

Xuesong Wang, Lina Yao, Xianzhi Wang, Hye-young Paik, and Sen Wang. Global convolutional neural processes. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 699–708. IEEE, 2021b.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott E. Reed, Bobak Shahriari, Noah Y. Siegel, Josh Merel, Çaglar Gülçehre, Nicolas Heess, and Nando de Freitas. Critic regularized regression. *CoRR*, abs/2006.15134, 2020. URL https://arxiv.org/abs/2006.15134.

Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

Cristopher Williams and Carl Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT Press, 2005.

James Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc Deisenroth. Efficiently sampling functions from gaussian process posteriors. In *International Conference on Machine Learning*, pages 10292–10302. PMLR, 2020.

Mitchell Wortsman, Maxwell C Horton, Carlos Guestrin, Ali Farhadi, and Mohammad Rastegari. Learning neural network subspaces. In *International Conference on Machine Learning*, pages 11217–11227. PMLR, 2021.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. Elastic decision transformer. *Advances in Neural Information Processing Systems*, 36, 2024.

Zhihan Yang and Hai Huu Nguyen. Recurrent off-policy baselines for memory-based continuous control. In *Deep RL Workshop NeurIPS 2021*, 2021.

Yigit Yildirim and Emre Ugur. Learning social navigation from demonstrations with conditional neural processes. *Interaction Studies*, 23(3):427–468, 2022.

Yuan Yin, Vincent Le Guen, Jérémie Dona, Emmanuel de Bézenac, Ibrahim Ayed, Nicolas Thome, and Patrick Gallinari. Augmenting physical models with deep networks for complex dynamics forecasting. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124012, 2021.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34:28954–28967, 2021.

Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. corr abs/1703.06114 (2017). *arXiv preprint arXiv:1703.06114*, 2017.

Barbara Zdrazil, Eloy Felix, Fiona Hunter, Emma J Manners, James Blackshaw, Sybilla Corbett, Marleen de Veij, Harris Ioannidis, David Mendez Lopez, Juan F Mosquera, et al. The chembl database in 2023: a drug discovery platform spanning multiple bioactivity data types and time periods. *Nucleic Acids Research*, 52(D1):D1180–D1192, 2024.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *CoRR*, abs/2006.10742, 2020. URL https://arxiv.org/abs/2006.10742.

Guangxiang Zhu, Zichuan Lin, Guangwen Yang, and Chongjie Zhang. Episodic reinforcement learning with associative memory. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HkxjqxBYDB.

Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3651–3657. IEEE, 2019.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. *arXiv preprint arXiv:1910.08348*, 2019.