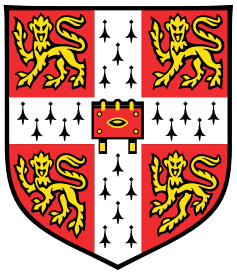
Domain Generalisation for Robust Model-Based Offline Reinforcement Learning

MLMI MPhil 2021-22



Alan Clark

Supervisor: Dr. David Krueger

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Jesus College

August 2022

Dedicated to Jake – I couldn't have done it without you.

Declaration

I, Alan Clark of Jesus College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Word Count: 14,418 words

Software Declaration Standard Python packages for machine learning applications were employed such as `TensorFlow`, `scikit-learn`, and `numpy`. All plots were produced using `matplotlib`.

The official **MOPO** repository (https://github.com/tianheyu927/mopo - Yu et al. (2020)) was used as the starting point for this work. Specifically, for both training dynamics models (Chapter 4) and policies (Chapter 5). In addition to adapting the dynamics model training procedure to incorporate Risk Extrapolation (REx) (Krueger et al., 2020), further updates that were made include: allowing any source of rollout data to be used in training; fixing the process of loading pre-trained dynamics models for use in policy training; and enabling the evaluation of any learned policy using any learned dynamics model.

The following packages/repositories were additionally directly leveraged:

- **Softlearning** (https://github.com/rail-berkeley/softlearning - Haarnoja et al. (2018a)): This repository was used, without modification, to train demonstrator policies in Chapter 4.
- **D3RLPY** (https://github.com/takuseno/d3rlpy - Seno and Imai (2021)): As above, this repository was used, without modification, to train demonstrator policies in Chapter 4.

- **OpenAI Gym** (https://github.com/openai/gym - (Brockman et al., 2016)): Source of the HalfCheetah task used in this work - see Chapter 3. Trained policies were executed and visualised in this environment.

Alan Clark
August 2022

Acknowledgements

I would like to thank my supervisor, Dr David Krueger, for all of his support, patience and guidance. I would further like to thank Shoaib Ahmed Siddiqui for his suggestions and insights, for always being happy to lend an ear, and for simply being such a nice person to spend many, many long days with in the lab. Many thanks also to Robert Kirk for all of his guidance and suggestions, and to Katie Collins for her endless enthusiasm and willingness to hear me talk about the project and provide suggestions.

Many thanks to Professor Richard Turner and Anne Debenham for their leadership of the MLMI course, and their personal support. Thank-you also to the entire MLMI cohort – I've learnt so much from all of you and feel privileged to have spent the last year in your company.

Finally, enormous thanks to my partner, Jake, for their endless support and encouragement.

Abstract

In many industries it is too dangerous or expensive to learn via online interaction with the real-world. The goal of offline reinforcement learning (RL) is therefore to learn a policy solely from collected data. Using this data, offline model-based RL (MBRL) methods employ supervised learning to train a model of the world (a dynamics model), against which a learning agent can safely interact. In our work, we define domain generalisation with reference to the logging/behavioural policies employed during data collection, such as when different human demonstrators have performed a given task, each following their own decision process. Diverse demonstrators give rise to varied data distributions, and so domain-generalisation techniques can, in theory, be used to learn dynamics models that are *robust* to distributional shifts encountered during policy training.

To this end, we apply Risk Extrapolation (REx) (Krueger et al., 2020) to the process of training dynamics models. As a proxy for data collected from real-world demonstrators, we train policies using online RL methods and use these to generate datasets. We demonstrate that models trained with REx exhibit improved domain generalisation performance, and achieve greater equality of risks across out-of-distribution domains. Further, we find that these models enable superior policies to be learned in the offline MBRL setting, and increase the stability of the policy learning process.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

How do we learn to complete new tasks? In many cases the answer is by trial-and-error: we attempt the task and make mistakes repeatedly, but we learn in the process, and eventually succeed and gain a new skill. However, we are also often able to learn by being *shown*; typically through the observation of others attempting to perform the same task. Can this approach of learning through demonstration be similarly adopted in reinforcement learning (RL)?

In the standard *online* RL paradigm, *agents* often require hundreds-of-thousands or millions of interactions with the environment in order to learn an optimal *policy* for completing a task. In many settings, however, the process of exploration in the real world is undesirable, impractical or unsafe (Levine et al., 2020; Prudencio et al., 2022). Executing the actions suggested by a learning agent could present significant danger; for instance, in the fields of autonomous driving, robotics, and manufacturing. Alternatively, there may be expensive consequences for mistakes, such as in the financial markets and business decision making.

In suitably simple scenarios, and with appropriate domain knowledge, a simulated version of the real environment can be created and used to train agents with superhuman performance – as demonstrated by AlphaGo (Silver et al., 2016). However, most real-world scenarios are too complex for sufficiently high fidelity simulators to be developed within reasonable time and cost (Dulac-Arnold et al., 2019; Zhao et al., 2020).

It would therefore be preferable to enable learning from *demonstration*, rather than *interaction*. In industrial settings and healthcare there are often significant volumes of data already available, which represent demonstrations of how a task can be completed. These are provided by one or more *demonstrators*, each executing their own *behavioural policy* – their method of achieving the task. For example, the computer logs of a power plant controlled by several operators will reflect their combined behavioural policies.

The field of *offline* RL aims to develop techniques for learning from static data without any interaction with the real environment (Levine et al., 2020; Prudencio et al., 2022). It holds significant promise for enabling large, pre-collected datasets to be turned into powerful *decision-making engines* (Levine et al., 2020). While there are various methods by which this goal can be achieved, our work focuses on those that perform *supervised learning* to train a *model* of the environment, against which a learning agent can safely interact.

A limiting factor in the ability of offline model-based RL methods to learn optimal policies is *distributional shift* (Levine et al., 2020; Yu et al., 2020). Data generated by a demonstrator will be sampled from the state visitation distribution induced by their behavioural policy. Unless steps are taken to avoid it, models trained on this data are likely to overfit this distribution and perform poorly when presented with data from another distribution – they will fail to *generalise*. When an RL agent attempts to use the model to learn a policy it will typically start with some form of random policy, and so immediately induce a new state visitation distribution, different from that observed during training. Unless constrained towards the demonstrator's behaviour policy (a method employed by many of the current SOTA offline RL algorithms (Kidambi et al., 2020; Yu et al., 2020)), the various iterations of the policy being learned will continue to induce state distributions differing from that of the behaviour policy. If the model has indeed overfit to the training distribution, it will likely perform poorly under these distributional shifts. This may inhibit learning entirely, or may be exhibited as *model exploitation*, whereby the learned policy obtains higher reward using the dynamics model than it does when deployed to the real environment (Cang et al., 2021; Clavera et al., 2018; Janner et al., 2019).

Rather than constraining the learning policy's exploration, the aim of our work is to increase the *robustness* of dynamics models to distributional shift by applying *Risk Extrapolation (REx)* (Krueger et al., 2020) during training. REx assumes that training data from multiple sources is available, each representing a domain. While the domain that generated each training record is known, *no knowledge about the domain itself is required*. By enforcing the training losses, or risks, across the domains present in the training data to be equal, REx aims to achieve the same risk on out-of-distribution domains, even if the distributional shift is more extreme than those observed at training/test time (Krueger et al., 2020).

In our work, each behavioural policy, or demonstrator, is considered to be a domain. By collecting data from multiple demonstrators and applying REx, we hypothesise that dynamics models can be trained which exhibit improved domain-generalisation performance and are therefore able to support the learning of more optimal policies.

1.1 Contributions

The main contributions of this work can be summarised as follows:

1. We implement an end-to-end pipeline to support the training and evaluation of offline model-based reinforcement learning algorithms in the multi-demonstrator setting.
2. To mimic the collection of data from multiple demonstrators, we train a collection of policies using online RL algorithms. We use each of these to generate individual datasets, and combine them into multi-demonstrator datasets.
3. We show that dynamics models trained with REx exhibit improved domain-generalisation performance, and achieve a greater equality of risks across out-of-distribution domains.
4. In our experiments, we observe that dynamics models trained with REx enable superior policies to be learned in the offline model-based reinforcement learning setting, and increase the stability of the policy learning process.
5. Although dynamics models trained with REx displayed increased *robustness* to distributional shifts, we find that incorporating methods from existing SOTA methods for *limiting* distributional shift can be beneficial to policy training.

1.2 Overview

This thesis is structured as follows:

- **Chapter 2 - Background:** We provide a technical overview of offline model-based reinforcement learning and domain-generalisation, and review the key algorithms in each space.
- **Chapter 3 - Experiment Pipeline and Overview:** The end-to-end nature of this work, from data generation to training policies offline, necessitated the design and implementation of a suitable experiment pipeline. We describe this pipeline and provide a roadmap to the experiments run.
- **Chapter 4 - Demonstrator Dataset Creation:** We define a set of data requirements for this work, and evaluate existing benchmarks against these. Having determined that none are suitable, we use online RL algorithms to generate proxy demonstrators, from which we create individual- and multi-demonstrator datasets.

- **Chapter 5 - Dynamics Model Training and Evaluation:** Using the datasets created in the previous chapter, we train dynamics models and assess their domain generalisation performance.
- **Chapter 6 - Policy Training and Evaluation:** We use the dynamics models trained in the previous chapter to learn policies offline. We evaluate the performance of these policies, and identify the benefits provided by dynamics models trained with REx.
- **Chapter 7 - Conclusions:** We provide an overview of what has been learned in this work, and recommend potential future avenues of work.

Chapter 2

Background

In our work, we draw together methods from the fields of reinforcement learning (RL) and domain generalisation. We first provide a recap of the goals and learning paradigms of RL, before focusing on the issue of *distributional shift* and reviewing the existing methods in literature which aim to limit its extent. Rather than *minimise* distributional shift, our goal is to improve *robustness* to shifts by improving domain generalisation performance. We give a high level overview of domain generalisation theory and techniques, focusing on Risk Extrapolation (REx) – the method we apply to the training of dynamics models in offline model-based RL.

2.1 Reinforcement Learning

Reinforcement learning offers a mathematical formalism for learning-based decision making and control. It encompasses an ever-growing variety of algorithms with the common goal of achieving a desired task, or sequence of tasks, by automatically acquiring behavioural skills, represented by *policies*, that maximise a reward signal (Levine et al., 2020; Sutton and Barto, 2018). It offers the potential to automate a wide variety of tasks and reveal novel ways of achieving them. Consequently, RL methods are increasingly finding use in industrial applications, such as robotics (Gu et al., 2016; Ibarz et al., 2021), autonomous vehicles (El Sallab et al., 2017; Kiran et al., 2020) and the optimisation of smart energy grids (Sogabe et al., 2018).

2.1.1 RL Definitions and Notation

RL is concerned with learning to control dynamical systems described as *Markov decision processes (MDPs)*. An MDP $\mathcal{M}$ is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, \rho_0, \gamma)$, where states

$s \in \mathcal{S}$ and actions $a \in \mathcal{A}$ are members of the state and action spaces $\mathcal{S}$ and $\mathcal{A}$ respectively, $T(s'|s,a)$ denotes the transition distribution, $r(s,a)$ denotes the reward function, $\rho_0(s)$ denotes the initial state distribution, and $\gamma \in (0,1]$ denotes a discount factor commonly applied to infinite-horizon processes. We consider continuous state and action spaces and finite horizon tasks in our work.

We assume that the *Markov property* holds, which dictates that the state encompasses all information about agents' previous interactions with the environment that are pertinent to predicting future states (Sutton and Barto, 2018). The transition distribution, $T(s'|s,a)$, then completely characterises the dynamics of the environment.

Within an MDP, our aim is to learn a policy, $\pi(a|s)$, which denotes the probability of taking action a when in state s. The expected return of policy π under MDP $\mathcal{M}$, $J(\pi,\mathcal{M})$, is given by Equation 2.1.

$$J(\pi,\mathcal{M}) = \mathbb{E}_{a\sim\pi(\cdot|s),s_0\sim\rho_0,s'\sim T(\cdot|s,a)}\left[\sum_{t=0}^{H} r(s_t,a_t)\right] \tag{2.1}$$

Trajectories, τ, are sequences of states and actions, $(s_t,a_t,s_{t+1},a_{t+1},\ldots,s_H)$, where the horizon, H, could be infinite for non-episodic tasks (Levine et al., 2020). The trajectory distribution, p_π, induced by policy π under MDP $\mathcal{M}$ is given by Equation 2.2.

$$p_\pi(\tau) = \rho_0(s_0)\prod_{t=0}^{H}\pi(a_t|s_t)T(s_{t+1}|s_t,a_t) \tag{2.2}$$

The marginal distributional over states induced by policy π will be denoted by $d^\pi(s)$, and that over state-action pairs by $d^\pi(s,a)$. The objective of reinforcement learning is to find an optimal policy, $\pi^*(a|s)$, that maximises the expected sum of rewards under the trajectory distribution, as shown by Equation 2.3.

$$\pi^* = \arg\max_\pi J(\pi,\mathcal{M}) = \arg\max_\pi \mathbb{E}_{\tau\sim\rho_\pi(\tau)}\left[\sum_{t=0}^{\infty} r(s_t,a_t)\right] \tag{2.3}$$

2.1.2 Online Reinforcement Learning

Inspired by diagrams presented by Levine et al. (2020) and Prudencio et al. (2022), Figure 2.1 illustrates a collection of learning paradigms that can be used as a taxonomy to classify RL algorithms.

Online RL involves the iterative collection of experience through interaction with the environment. In *on-policy* RL we employ the latest iteration of the policy being learned to decide which actions to take, and use the experience collected to make further improvements

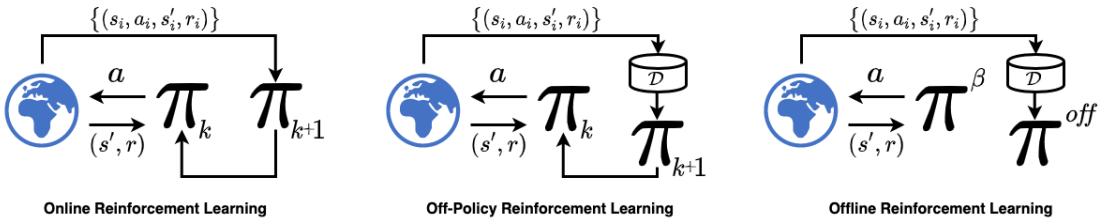


Fig. 2.1 Illustration of three common RL learning paradigms: online, off-policy and offline, inspired by diagrams presented by Levine et al. (2020) and Prudencio et al. (2022). $\mathcal{D}$ is a buffer containing transitions collected during or before training. π_k and π_{k+1} are iterations of a policy being learned, π^{β} is a behaviour policy, and π^{off} is a policy learned offline using a static dataset.

to the policy (Sutton and Barto, 2018). The need to continuously collect fresh experience leads to one of the major issues with online RL techniques: *sample efficiency*.

In *off-policy RL*, an agent is trained using experience collected from policies other than the one currently being improved. When this is a completely separate policy it is referred to as a *behaviour policy*, π^{β}. On- and off-policy methods can be used together, whereby newly collected online experiences are augmented with interactions sampled from a buffer populated during the course of training, $\mathcal{D}$. This can lead to significant improvements in sample efficiency.

2.1.3 Offline Reinforcement Learning

Offline RL, often also referred to as *batch RL*, algorithms are a subset of off-policy algorithms, whereby one or more behaviour policies interact with the environment to collect a static dataset of experiences that are later used to learn a policy, π^{off}, without any further interaction with the environment (Levine et al., 2020; Prudencio et al., 2022). The behaviour policies may themselves be RL agents, but more commonly will be another form of control policy (e.g., a rule-based control program) or a human (expert or otherwise). In our work, we consider independent agents acting to achieve the same task in the same environment to be independent behaviour policies.

Prudencio et al. (2022) propose a taxonomy to categorise offline RL methods which is briefly summarised here. Methods are grouped based on how they utilise the collected rollout data: learning a dynamics model, learning a trajectory distribution, or using the data directly in model-free approaches, which learn direct mappings from states to actions (Janner et al., 2019). *Planning* methods utilise either a learned dynamics model or trajectory distribution to determine the optimal actions to take at each time step. Alternatively, dynamics models can be used to generate additional data, $\mathcal{D}_{\text{model}}$, with which Q-learning and actor-critic based

algorithms can then be used to learn a policy. This offline model-based RL (MBRL) approach is the focus of our work.

While the transition distribution of an MDP is independent of the policy, the state and state-action visitation distributions induced by behavioural policy π^β, $d^{\pi^\beta}(s)$ and $d^{\pi^\beta}(s,a)$, are not. The pool of data collected for offline policy training is typically assumed to be composed of *iid* samples drawn from $d^{\pi^\beta}(s,a)$, and is likely to cover only a fraction of the total state-action space. In order to learn optimal policies, we would like our learning algorithm to *generalise* to other areas of this space; departing from the support of the training data in order to learn good behaviours that are not exhibited in the static dataset (Wu et al., 2021; Yu et al., 2020). The learning policy would therefore induce new state and state-action visitation distributions, $d^{\pi^{\text{off}}}(s)$ and $d^{\pi^{\text{off}}}(s,a)$, which may differ significantly from those of the behaviour policy. Numerous other factors will additionally contribute to changes in the visitation distributions, such as the initial state distribution used. Central to the development of many off-policy and offline RL algorithms are methods for limiting these *distributional shifts*.

2.1.4 The Impact of Distributional Shift

Errors made by the learned dynamics model, such as those arising from poor generalisation performance under distributional shift, can result in *model exploitation*. The policy being trained learns to take advantage of model errors when optimising the reward, leading to poor performance in the true environment (Cang et al., 2021; Clavera et al., 2018; Janner et al., 2019; Levine et al., 2020; Rajeswaran et al., 2016). However, in more extreme cases, model exploitation causes significant instability in training (Kurutach et al., 2018; Matsushima et al., 2020).

Q-learning and actor-critic algorithms aim to learn the optimal state-action value function, $Q^*(a|s)$, by iteratively applying the Bellman optimality operator, $\mathcal{T}$, given by Equation 2.4 (Kumar et al., 2019; Sutton and Barto, 2018; Wu et al., 2021).

$$\mathcal{T}\hat{Q}(s,a) := r(s,a) + \gamma \mathbb{E}_{T(s'|s,a)}[\max_{a'} \hat{Q}(s',a')] \tag{2.4}$$

For large state spaces, the Q-function approximator $\hat{Q}(s,a)$, the *critic*, is typically a neural network, which is trained using dynamic programming by minimising the Bellman Squared Error $\mathbb{E}[(\mathcal{Q} - \mathcal{T}\hat{Q})^2]$ (Haarnoja et al., 2018a,b; Kumar et al., 2019; Lillicrap et al., 2015; Sutton and Barto, 2018). In continuous action spaces $\max_{a'} \hat{Q}(s',a')$ is typically intractable – a problem which actor-critic methods resolve by additionally learning an *actor*, π_θ, to maximise the Q-function (Haarnoja et al., 2018a,b; Kumar et al., 2019; Lillicrap et al., 2015).

Performing $\max_{a'} \hat{Q}(s',a')$ during training may lead to $\hat{Q}(s',a')$ being evaluated for an action that does not appear in the training data, which can result in significant errors that propagate through the Bellman bootstrapping, destabilising training on other states (Kumar et al., 2019; Wu et al., 2021).

Fujimoto et al. (2018) demonstrate that standard off-policy deep RL algorithms DQN (Mnih et al., 2013) and DPPG (Lillicrap et al., 2015) are unable to learn without data that is correlated to the distribution induced by the current policy, and Kumar et al. (2019) make the same observation for SAC (Haarnoja et al., 2018a,b).

The standard approach adopted in prior work is to try to directly limit the divergence of the learned policy from the behavioural policy. BCQ (Fujimoto et al., 2018) attempt to minimises the distance between the two distributions, while BEAR (Kumar et al., 2019) relaxes the constraint by only requiring that the actions output by the learned policy lie within the support of the training distribution. UWAC (Wu et al., 2021) highlights that state-action pairs not present in the training data can still lie within the training distribution, and so use a form of *uncertainty quantification* to avoid areas of the state-action space where there is high uncertainty while allowing more exploration. Although such approaches reduce the potential for distributional shift, by preventing areas of the state-action space from being visited we also preclude the identification of optimal policies.

2.1.5 Existing Offline MBRL Approaches

The model-free algorithms described in the previous section are constrained to learn using only the states and actions in the collected dataset. Because this dataset is static, there is no possibility of spurious predictions being corrected through the collection of further data. Model-based methods present an opportunity to rectify this, however will only be beneficial if they limit the impact of distributional shift. In this space, a number of algorithms have been proposed which, like UWAC, aim to develop an awareness of the uncertainty in the predictions made by dynamics models and use this to guide the policy learning process.

PILCO (Deisenroth et al., 2015) learns a probabilistic dynamics model using Gaussian processes to allow the model's uncertainty to be expressed and incorporated into planning and policy evaluation. Errors introduced by the smoothness assumptions made about environments' dynamics (inherent to standard GP kernels) can be alleviated through the learning of suitable data representations (Calandra et al., 2014). However, GP methods struggle to scale to high dimensional state spaces (Chua et al., 2018).

In their algorithm PETS, Chua et al. (2018) model the transition function using an ensemble of neural networks whose outputs parameterise a multivariate Gaussian distribution with diagonal covariance matrix: $\hat{T}^i_{\theta,\phi}(s_{t+1}|s_t,a) = \mathcal{N}(\mu_\theta(s,a),\Sigma_\phi(s,a))$. PETS uses planning to

learn an optimal sequence of actions, rather than training a policy. Further, it periodically retrains the dynamics models using freshly collected data, making it an online approach. MBOP (Argenson and Dulac-Arnold, 2020) extends PETS into an offline method by removing interactions with the environment and learning a model of the behaviour policy to guide the action sampling process, in an attempt to avoid large distributional shifts.

The dynamics model training process from PETS is further used in the online method MBPO (Janner et al., 2019) to train policies. Yu et al. (2020) subsequently adapted MBPO into the offline method MOPO. As the SOTA offline MBRL algorithm used as the basis for our work, we take time below to discuss MOPO in more detail.

Other offline MBRL methods include MOReL (Kidambi et al., 2020), COMBO (Yu et al., 2021), BREMEN (Matsushima et al., 2020) and MABE (Cang et al., 2021). While MOReL continues to use uncertainty quantification, the authors of COMBO (Yu et al., 2021) highlight that this approach can be challenging and unreliable when using deep neural networks (Ovadia et al., 2019). They propose a method which combines the rollout generation process from MOPO with the conservative value function estimation of CQL (Kumar et al., 2020). Concretely, conservatism is achieved by down-weighting Q-values on state-action tuples from the model rollouts and up-weighting those from the offline dataset. Although BREMEN is frequently included in the category of model-based offline RL, the authors' focus was increasing the *deployment efficiency* of policy learning – that is, *minimising* the number of times the policy interacts with the environment during training (Matsushima et al., 2020). This can be reduced to zero to make it an offline method. The learned policy is biased towards the data-collection policy by initialising it to an estimate of the behaviour policy learned via behaviour cloning (BC) and conditioning learning on the KL divergence from the estimated behaviour policy to be less than a user-specified maximum.

Model-Based Offline Policy Optimisation (MOPO)

Unlike PETS, MOPO does not assume the reward function is known, and so the ensemble of N neural networks learned parameterise a multivariate Gaussian distribution, with diagonal covariance matrix, over both the next state and reward, based on the current state and action (Yu et al., 2020).

$$p^i_{\theta,\phi}(s_{t+1}, r|s_t, a) = \mathcal{N}(s_{t+1}, r; \mu^i_\theta(s,a), \Sigma^i_\phi(s,a)) \tag{2.5}$$

The policy being learned is used in conjunction with the trained models to generate short rollouts, with the collected transitions stored in a replay buffer. Samples from the offline

dataset and replay buffer are then used to update the policy with the SAC algorithm (Haarnoja et al., 2018a,b).

Further, MOPO uses *uncertainty quantification* to estimate the risk associated with leaving the support of the training data (Yu et al., 2020). The authors define the uncertainty penalised reward given by Equation 2.6, where $\hat{r}(s,a)$ is the mean predicted reward output by the ensemble, $||\Sigma^i_\phi(s,a)||_{\mathrm{F}}$ is the Frobenius norm of the diagonal matrix of standard deviations predicted by member i of the ensemble, and λ is the MOPO penalty coefficient, a user-specified hyperparameter. The authors state they use the maximum norm to be more conservative (Yu et al., 2020).

$$\tilde{r}(s,a) = \hat{r}(s,a) - \lambda \max_{i=1,\dots,N} ||\Sigma^i_\phi(s,a)||_{\mathrm{F}} \tag{2.6}$$

Therefore, if any of the models predict high variance for a given (s,a) tuple, the predicted reward will receive a large penalty, and so the policy will likely learn to avoid taking the given action in the given state. The updated reward yields the *uncertainty-penalised MDP* $\tilde{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \hat{T}, \tilde{r}, \rho_0, \gamma)$, under which the authors show that their method maximises a lower bound on the return of the true MDP (although the use of the maximum predicted standard deviation as the uncertainty penalty was empirically justified) (Yu et al., 2020).

2.1.6 Summary

Policy-induced distributional shift is a significant challenge in offline RL. The previous works we've identified attempt to *limit* its impact by directly or indirectly guiding the learning policy to either not deviate from the behavioural policy or to avoid certain areas of the state-action space. These approaches therefore constrain exploration, which may prevent the learning of optimal policies if the method is either overly-conservative or erroneously prevents exploration in areas where the model error is actually low. In our work we instead propose an approach that attempts to be *robust* to distributional shift by improving the *domain generalisation* performance of the trained dynamics models.

2.2 Domain Generalisation

Distributional shift is not an issue unique to RL. In traditional supervised learning it is common for the *iid* – independent and identically distributed – assumption to be made (Bishop, 2006). That is, it is assumed that both the training and test data have been drawn from the same distribution. However, when models are presented with truly novel data they often suffer a deterioration in performance caused by distributional shift (Beery et al., 2018;

Gururangan et al., 2018; Subbaswamy et al., 2019). Concretely, there is often a gap between the domain that the data used the train and test the model was drawn from, and the ones it encounters when deployed. For example, if we have inputs, or covariates, X and targets Y, two forms of distributional shift that can occur between domains are: shifts in the distribution of $P(X)$, or *covariate shift*; and changes in $P(Y|X)$, or *concept shift* (Moreno-Torres et al., 2012). We cannot reasonably expect to collect data from all possible domains to train our model, and so instead may look to improve the ability of the model *generalise* to unseen domains.

2.2.1 Empirical Risk Minimisation

We first describe Empirical Risk Minimisation (ERM): the standard training approach taken in supervised learning, and baseline against which domain generalisation methods can be compared (Shen et al., 2021; Vapnik, 1991). Consider an input space $\mathcal{X}$, output space $\mathcal{Y}$, a fixed loss function ℓ and a model family Θ. Further assume that we have a dataset, $\mathcal{D}_e$, of training data drawn from domain e with distribution $P_e(X,Y)$ over the inputs and outputs: $(x^e, y^e) \sim P_e(X,Y)$. Using model $\theta \in \Theta$, the training risk for domain e is given by Equation 2.7.

$$\mathcal{R}_e(\theta) = \mathbb{E}_{(x^e,y^e)\sim P_e(X,Y)}[\ell(f_\theta(x^e), y^e)] \tag{2.7}$$

The typical goal in supervised learning is to identify $\theta^* \in \Theta$ which minimises the training risk. In ERM, we minimises the average loss over the training dataset (Vapnik, 1991).

$$\theta^*_{ERM} := \arg\min_\theta \mathbb{E}_{(x^e,y^e)\sim \mathcal{D}_e}[\ell(f_\theta(x^e), y^e)] \tag{2.8}$$

Consider now that we have a set of N training domains $\mathcal{E} \in \{e_1, \ldots, e_N\} \subset \mathcal{F}$, from each of which a training dataset, $\mathcal{D}_e$, has been collected. Each domain is assumed to posses a different distribution over the inputs and outputs: $P_e(X,Y) \neq P_f(X,Y) \forall e, f \in \mathcal{E}$. The ERM approach to learning $\theta^* \in \Theta$ would be to pool the training data together, $\mathcal{D} = \bigcup_{e=1}^N \mathcal{D}_e$, and minimise the average loss across the training examples from all the domains (Arjovsky et al., 2019).

However, this approach leads models to learn all correlations present in the training data, including those that are spurious – misleading heuristics which do not hold generally (Arjovsky et al., 2019; Sagawa et al., 2019). Arjovsky et al. (2019) highlight that by pooling data from different domains we discard the information about how the data distribution changes when the data source or collection specifics are varied; information that could tell us whether a feature in the data is spurious or stable.

2.2.2 Domain Generalisation Goal

The goal in domain generalisation is to learn the model $\theta^*_{DG} \in \Theta$ which minimises the risk on an unseen set of target domains $\mathcal{T} \in \{t_1, \ldots, t_T\} \subset \mathcal{F}, \mathcal{T} \not\subset \mathcal{E}$, where $P_t(X,Y) \neq P_e(X,Y) \forall t \in \mathcal{T}, e \in \mathcal{E}$.

$$\theta^*_{DG} := \arg\min_{\theta} \mathbb{E}_{t\sim\mathcal{T}} \left[\mathbb{E}_{(x^t,y^t)\sim P_t(X,Y)} [\ell(f_\theta(x^t), y^t)] \right] \tag{2.9}$$

Wang et al. (2021) give the illustrative example of a training set comprising images of dogs, but where each image may be a sketch, cartoon, art painting or photo. If we were to train a classification model using the first three of these, the model would exhibit good domain generalisation performance if it achieves minimal prediction error on photos of dogs.

2.2.3 Domain Generalisation Techniques

There is a wealth of literature on methods that aim to increase the robustness of trained models to distributional shift; interested readers are referred to the recent surveys produced by Wang et al. (2021), Shen et al. (2021) and Zhou et al. (2021). Wang et al. (2021) propose the following taxonomy, which categorises methods into the the following three groups:

- **Data manipulation:** Methods which employ either *data augmentation* or *data generation* to promote the learning of general representations.
- **Representation learning:** Methods for *feature disentanglement* or learning *domain-invariant representations/predictors*.
- **Learning strategy:** A wide category, including methods for performing *distributionally robust optimisation*.

Each of these categories contains methods that could be applied to the learning of dynamics models in offline MBRL. In our work, we use the Risk Extrapolation (REx) method (Krueger et al., 2020), which falls across *distributionally robust optimisation* and *domain-invariant prediction* categories (Wang et al., 2021).

Distributionally Robust Optimisation

The goal of distributionally robust optimisation (DRO) (Ben-Tal et al., 2013; Duchi et al., 2020) is to minimise the worst case risk over an uncertainty set of *possible* target domains, $\tilde{\mathcal{T}}$.

$$\theta^*_{DRO} := \arg\min_{\theta} \left[\sup_{t \sim \tilde{\mathcal{T}}} \left[\mathbb{E}_{(x,y)\sim t}[\ell(f_\theta(x), y)] \right] \right] \tag{2.10}$$

In Group DRO, Sagawa et al. (2019) propose minimising the worst case risk over groups in the training data, which is mathematically equivalent to considering convex combinations of the training risks (Krueger et al., 2020). While Sagawa et al. (2019) focus on users having prior knowledge about how best to group the training data, in our work we have assumed training data has been collected from the set of domains $\mathcal{E}$. Given it makes no assumptions about the types of shift that may occur, Group DRO can provide robustness to both covariate shift and concept shift.

Domain-Invariant Prediction

As it is most relatable to REx, we focus our analysis of domain-invariant prediction techniques on Invariant Risk Minimisation (IRM) (Arjovsky et al., 2019), which looks to learn correlations in the data that are invariant across domains, $\mathcal{F}$. IRM derives from causal inference methods, where it is assumed that an invariant relationship exists between the target variable and it's direct causes – readers are referred to the works of Peters et al. (2015, 2017) and Bühlmann (2018) for further background.

IRM considers a data representation $\Phi : \mathcal{X} \to \mathcal{H}$ to elicit a *domain-invariant predictor* $w \cdot \Phi$ if there exists a classifier $w : \mathcal{H} \to \mathcal{Y}$ that is simultaneously optimal for all domains, $\mathcal{F}$ (Arjovsky et al., 2019). That is, the same classifier w is used across domains, and IRM looks to identify a representation Φ that both: enables good predictions to be made, and elicits an invariant predictor across domains. In its practical implementation, a scalar, fixed "dummy" classifier of $w = 1$ is used such that Φ becomes the entire invariant predictor. A term is included in the loss function to encourage the predictor $1 \cdot \Phi(x)$ to be optimal across all training domains (Arjovsky et al., 2019).

One issue with IRM is that if there is only covariate shift (i.e., only P(X) varies across domains), then $\Phi(x) = x$ is already suitable data representation for invariant prediction, given P(Y|X) does not change. Thus, IRM is not expected to be robust to covariate shifts.

Risk Extrapolation (REx)

Risk Extrapolation (REx) (Krueger et al., 2020) seeks to improve domain generalisation performance by enforcing the equality of risks across training domains. As shown by Figure 2.2, encouraging equality of the training risks flattens what Krueger et al. (2020) refer to as the "risk plane" – the affine span of training risks. We can consider this affine span on the

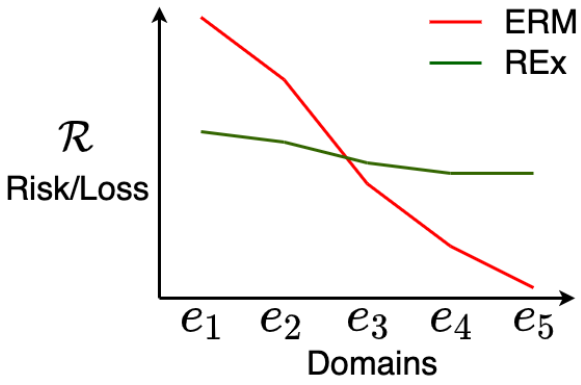


Fig. 2.2 When applying REx we expect to decreases the risks across training domains whilst also increasing their similarity, which may lead to an increased risk for certain domains. The goal of ERM is to reduce average risk, resulting in a decrease in risk for those domains that most directly enable this goal (Duchi et al., 2020)

.

training domains to contain theoretical domains representing distributional shifts that may be more extreme than those encountered in the training data (Krueger et al., 2020). Therefore, if training losses were equalised, it is hoped that the loss on any domain encountered in reality would be approximately equal to the loss on training domains.

The *Minimax-REx (MM-REx)* method, Equation 2.11, can be thought of as an extension of Group DRO to *affine* combinations of the training risks (Krueger et al., 2020). Hyperparameter λ_{min} controls the amount of extrapolation.

$$\mathcal{R}_{\text{MM-REx}} = (1 - N\lambda_{\text{min}}) \max_e \mathcal{R}_e(\theta) + \lambda_{\text{min}} \sum_{e=1}^{N} \mathcal{R}_e(\theta) \tag{2.11}$$

In practice, Krueger et al. (2020) found that V-REx, Equation 2.12, offered better performance, and so this variant was used in our work. When $\beta = 0$, and if all datasets contain an equal number of examples, we effectively recover ERM, although we now have the sum of losses.

$$\mathcal{R}_{\text{V-REx}} = \beta \text{Var}(\{\mathcal{R}_1(\theta), \ldots, \mathcal{R}_N(\theta)\}) + \sum_{e=1}^{N} \mathcal{R}_e(\theta) \tag{2.12}$$

Krueger et al. (2020) show that REx can uncover *invariant relationships* between inputs X and targets Y, and that it optimises for robustness to the forms of distributional shift which have the largest impact on performance across a collection of training domains. The authors further highlight the inability of REx to distinguish between underfitting in a training domain (i.e., epistemic uncertainty) and inherent noise (aleatoric uncertainty), which can encourage

the model being trained to make equally bad predictions everywhere, even if the noise in certain domains is lower than in others. IRM therefore has an advantage in settings where some domains are intrinsically harder than others. However, unlike IRM, REx is better able to solve invariant prediction tasks where covariate shift occurs Krueger et al. (2020).

2.2.4 Summary

While many different domain-generalisation techniques could be applied to dynamics model training, the robustness of REx to multiple forms of distributional shift make it a strong candidate for our investigations. We expect to see the flattening of the "risk plane" shown in Figure 2.2 when evaluating models on out-of-distribution domains, and anticipate that this increased robustness to distributional shift will be beneficial for learning policies in the offline MBRL setting.

In the next chapter, we introduce and discuss our experiment pipeline.

Chapter 3

Experiment Pipeline and Overview

In this chapter we introduce the experiment pipeline created for this work, and provide a roadmap to our experiments. Important notation and terminology will be defined and used consistently in subsequent Chapters. We discuss the environment and task that are used across all experiments, and provide baseline results for MOPO – the offline model-based RL algorithm used as the foundation of this work.

3.1 Environment and Task

All experiments in this work use OpenAI Gym's MuJoCo HalfCheetah. OpenAI Gym presents a standardised API and wide range of environments for training and evaluating RL algorithms (Brockman et al., 2016). This includes a set of environments implemented with MuJoCo: a physics engine for simulating the interaction of articulated structures within an environment (Todorov et al., 2012). MuJoCo environments are used extensively in RL research and benchmarks (Fu et al., 2020; Gulcehre et al., 2020). The HalfCheetah, shown in Figure 3.1, is a 2-dimensional robot comprising 9 links and 8 joints. The standard goal of this task, as used in our work, is to apply torques on the joints to make the cheetah run forward as fast as possible, while minimising the magnitude of the actions taken. The 17 dimension continuous, unbounded observation space includes the position and velocities of the individual parts of the cheetah, while the 8 dimension continuous action space consists of the torques applied to each joint, bounded in the range [-1,1]. Readers are directed to OpenAI's documentation for further information[1].

[1] OpenAI Gym HalfCheetah: https://www.gymlibrary.ml/environments/mujoco/half_cheetah

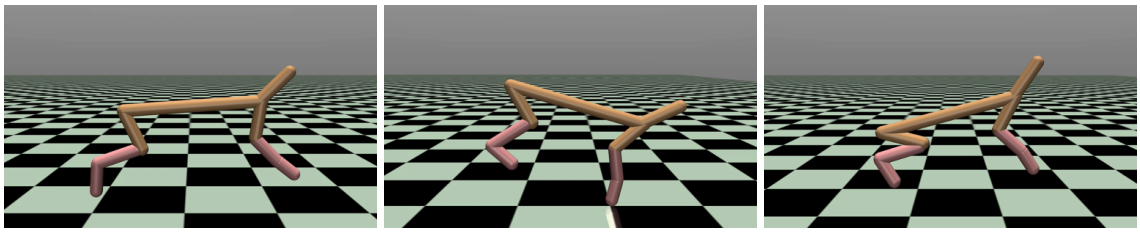


Fig. 3.1 An OpenAI Gym MuJoCo HalfCheetah running forwards.

3.2 Experiment Pipeline

Our experiment pipeline, illustrated in Figure 3.2, has been divided into three sections, aligned with the organisation of this thesis. There are a number of interacting components, and so an overview is provided here that can easily be referred back to.

3.2.1 Demonstration Dataset Generation

Initially, a selection of demonstrator policies, π^e, are trained using online RL algorithms. The sole purpose of these policies is to generate individual demonstrator datasets, $\mathcal{D}_e$, to proxy for those generated by real life demonstrators (whether humans, or pre-existing control policies). The term **steps** will be used to refer to the amount of online training a demonstrator received. To increase data diversity, additional demonstrator datasets are created using a random policy, $\pi^{\mathcal{N}}$. Each dataset comprises N_e transition tuples of the form: (s^e, a^e, s'^e, r^e, e). The unique demonstrator identifier e is used during the training of dynamics models with Risk Extrapolation (REx) to identify individual domains. The identifier is simply a number, and provides no information about the demonstrator. Independently sampled evaluation datasets are also created for each demonstrator. The term **transitions** will be used to refer to the size of the demonstration dataset – for example, "demonstration dataset X with 1M transitions was used in training." Collections of M individual datasets are combined to produce multi-demonstrator datasets: $\mathcal{D}_{\mathcal{E}} = \{(s_i^e, a_i^e, s_i'^e, r_i^e, e)_{i=1}^{N_e}\}_{e=1}^{M}$.

3.2.2 Dynamics Model Training and Evaluation

Dynamics models $p_{\theta,\phi}(s_{t+1}, r_t | s_t, a_t)$ are trained using both individual demonstrator datasets, $\mathcal{D}_e$, and multi-demonstrator datasets $\mathcal{D}_{\mathcal{E}}$. The outputs of the models parameterise a Gaussian distribution with diagonal covariance matrix: $\mathcal{N}(s_{t+1}, r_t; \mu_\theta^i(s,a), \Sigma_\phi^i(s,a))$. The MOPO algorithm (Yu et al., 2020), outlined in Algorithm 1, was chosen as the starting point for our work[2], as this allowed the impact of applying domain generalisation techniques in dynamics

[2]https://github.com/tianheyu927/mopo

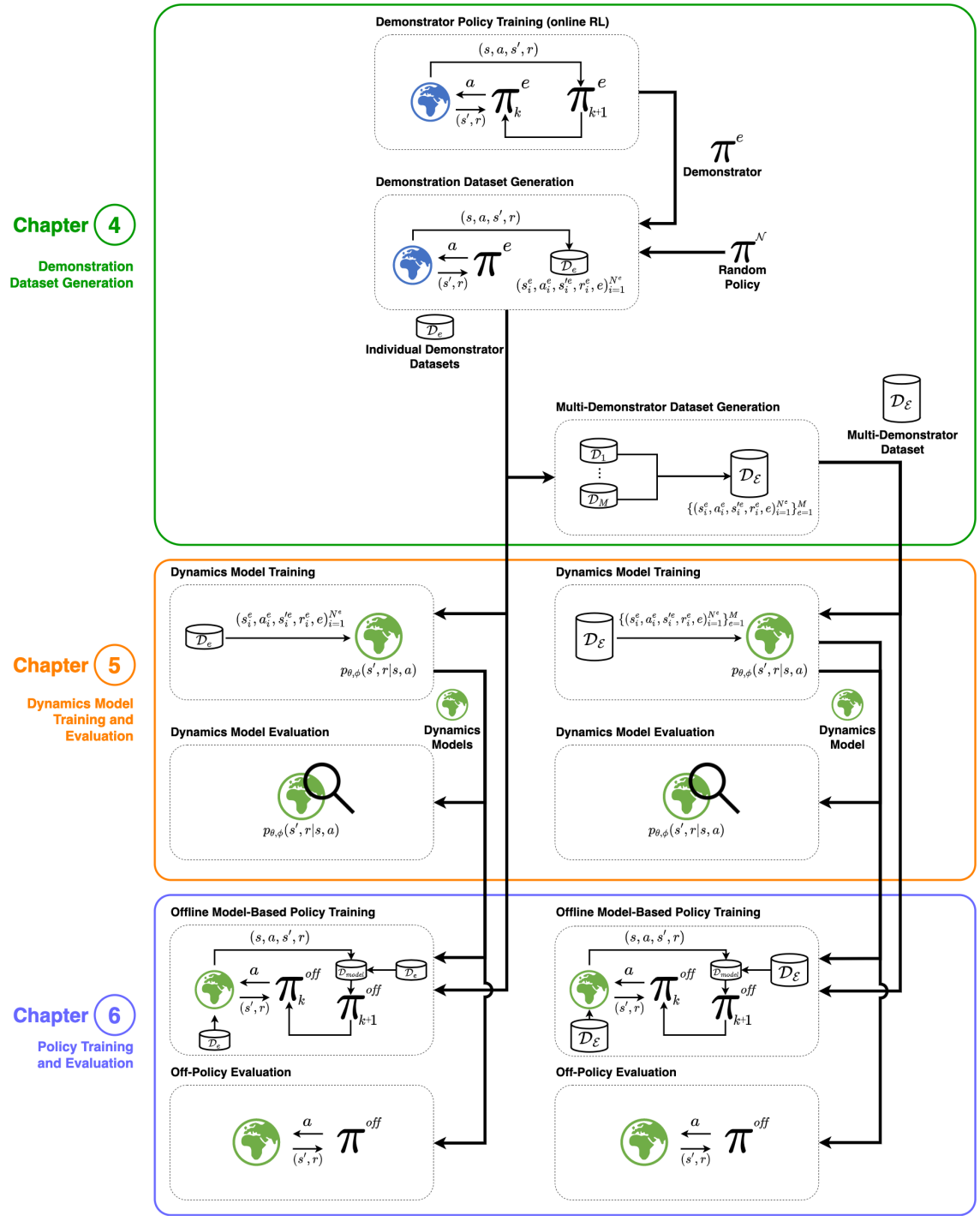


Fig. 3.2 The experiment pipeline implemented for this work. Blue globes represent the real world, while green globes indicate a learned model of the world: $p_{\theta,\phi}(s_{t+1}, r_t | s_t, a_t)$. Trained demonstrator policies are denoted by π^e, random policies by $\pi^{\mathcal{N}}$, and policies learned offline by π^{off}. The lower half of the diagram contains two parallel tracks: one relating to the use of individual demonstrator policies, and another concerning multi-demonstrator policies.

model training to be directly assessed against an existing SOTA algorithm. Further details of the dynamics model training procedure will be provided in Chapter 5.

Algorithm 1 MOPO - adapted from (Yu et al., 2020) to include REx

Require: : reward penalty coefficient λ, rollout length h, rollout batch size b, REx penalty coefficient β

1: Train on batch data $\mathcal{D}_{\text{env}}$ an ensemble of N probabilistic dynamics models $\{p_{\theta,\phi}(s_{t+1}, r_t | s_t, a_t) = \mathcal{N}(\mu^i_\theta(s,a), \Sigma^i_\phi(s,a))\}^N_{i=1}$ with REx penalty coefficient β
2: Initialise policy π^{off} and empty replay buffer $\mathcal{D}_{\text{model}} \leftarrow \varnothing$
3: **for** epoch $1, 2, \ldots$ **do**
4: **for** $1, 2, \ldots, b$ (in parallel) **do**
5: Sample state s_1 from $\mathcal{D}_{\text{env}}$ for the initialisation of the rollout
6: **for** $j = 1, 2, \ldots, h$ **do**
7: Sample an action $a_j \sim \pi(s_j)$
8: Randomly pick dynamics $\hat{T}$ from $\{\hat{T}^i\}^N_{i=1}$ and sample $s_{j+1}, r_j \sim \hat{T}(s_j, a_j)$
9: Compute $\tilde{r}_j = r_j - \lambda \max^N_{i=1} ||\Sigma^i_\phi(s_j, a_j)||_{\text{F}}$
10: Add sample $(s_j, a_j, \tilde{r}_j, s_{j+1})$ to $\mathcal{D}_{\text{model}}$
11: Drawing samples from $\mathcal{D}_{\text{env}} \bigcup \mathcal{D}_{\text{model}}$, use SAC to update π^{off}

The performance of trained dynamics models is assessed against the evaluation datasets created earlier, as well as the D4RL benchmark datasets (discussed in Section 4.2). We evaluate the impact of training demonstrator(s) and model hyperparameters on domain-generalisation performance. Average mean squared errors (MSEs) and log-likelihoods are calculated for both in-distribution evaluation datasets (i.e., those drawn from the demonstrator(s) used to train the model) and out-of-distribution datasets (all other demonstrators, plus the D4RL datasets).

3.2.3 Offline Model-Based Policy Training and Evaluation

As shown in Algorithm 1, policies are trained using the SAC algorithm (Haarnoja et al., 2018a,b). The required inputs are a dynamics model and a pool of collected transitions, $\mathcal{D}_{\text{env}}$, which is not required to be the same dataset used to train the dynamics model. The model and learning policy are used to generate short rollouts of length h from starting locations drawn from $\mathcal{D}_{\text{env}}$ once per epoch, and the experience collected stored in a model replay buffer, $\mathcal{D}_{\text{model}}$.

To assess whether the application of REx to dynamics model training has yielded sufficient domain generalisation performance to enable increased exploration, experiments are run where both: the rollout length h is increased from the maximum of 5 used in MOPO, and a variety of different starting locations are provided. Gradient updates are performed using a

combination of records from $\mathcal{D}_{\text{model}}$ and $\mathcal{D}_{\text{env}}$, with 95 % drawn from the former. At the end of each epoch the current policy is evaluated in the real environment.

MOPO Reward Penalty

As a reminder from Section 2.1.5, MOPO uses the uncertainty-penalised reward given by Equation 3.1, where λ is the user-specified MOPO penalty coefficient (Yu et al., 2020). To analyse whether continuing to incorporate the MOPO penalty was beneficial when using dynamics models trained with REx, policies were typically trained for $\lambda \in \{1, 5\}$ (those primarily investigated in the original MOPO paper) as well as $\lambda = 0$.

$$\tilde{r}(s,a) = \hat{r}(s,a) - \lambda \max_{i=1,\ldots,N} ||\Sigma_{\phi}^{i}(s,a)||_{\text{F}} \tag{3.1}$$

Policy Evaluation

Once a policy, π^{off}, has been trained offline we can use the MDP, $\tilde{\mathcal{M}}$, induced by a given transition distribution, $\tilde{T}(s'|s,a)$, and reward function, $\tilde{r}(s,a)$, to estimate the expected return of the policy, $J(\pi^{off}, \tilde{\mathcal{M}})$ (see Section 2.1.1). The transition distribution and reward function could be those from the real environment (in which case we'd be using the true MDP, $\mathcal{M}$), or we could employ any of the models we have trained. We use the initial state distribution of the real environment in all cases, although in a strictly offline setting one would need to be derived using the static dataset or prior knowledge. We obtain an empirical estimate of the expected return by using π^{off}, $\tilde{T}(s'|s,a)$ and $\tilde{r}(s,a)$ to generate episodes and take an average of the returns.

We evaluate each set of learned policies through a number of lenses to extract maximal information about the presence and impact of domain-dependencies.

- Initially, the returns of the policies in the real environment are considered. Dynamics models that are robust to distributional shifts should support increased exploration, and so yield policies with higher returns. This will be influenced by the choice of dataset that rollout starting locations are sampled from in policy training, hence we will run experiments where the same dataset is used for all policies.

- The prevalence of model exploitation (discussed in Section 2.1.4) is determined by evaluating policies against the dynamics model and reward function that were used to train them. When compared to the return obtained in the real environment, this *self-evaluation* return will be artificially inflated if the policy has exploited errors made by the models.

- Finally, if all learned dynamics models and reward functions made perfect (or at least identical) predictions across the entire state-action space then they would induce matching MDPs, and so would all yield the same expected return for a given policy. We evaluate policies across a range of models and use the disagreement between the expected returns as an indicator of domain-dependence.

We anticipate that the increased robustness of models trained with REx should reduce model exploitation and improve the agreement in the predicted returns for policies.

3.3 Baseline MOPO Results

To ensure reproducibility and provide a baseline for our work, experiments using the D4RL HalfCheetah datasets (Fu et al., 2020) were repeated both before and after modifying MOPO, using the same hyperparameters as the original paper (provided in Appendix A.1). In terms of our experiment pipeline, Figure 3.2, this is equivalent to each of the D4RL datasets being an individual demonstrator dataset, which are fed into the dynamics model training and offline model-based policy training components. The post-modification results are shown in Figure 3.3 and Table 3.1. Results are quoted using the metric proposed by Fu et al. (2020), which normalises the undiscounted average evaluation returns during the final iteration of policy training to lie roughly between 0 and 100 (see Appendix A.2). The reproduced results are in good alignment with those provided in the paper.

When compared to a range of SOTA algorithms, MOPO outperforms model-free methods and MBPO on all but the medium dataset. Yu et al. (2020) hypothesise that this is due to a lack of action diversity in the dataset, and that model-free methods are more suitable in such scenarios. Most of the D4RL datasets contain one million transition records, however the medium-replay dataset, which provided the second highest average return, contains 101 thousand records, highlighting the content of the dataset can be more important than its size.

3.4 Conclusion

By providing a holistic overview of the experiment pipeline and evaluations conducted we hope to have made the content of, and relationships between, the remaining chapters clear. In the next chapter we train individual demonstrators, and combine these into multi-demonstrator datasets.

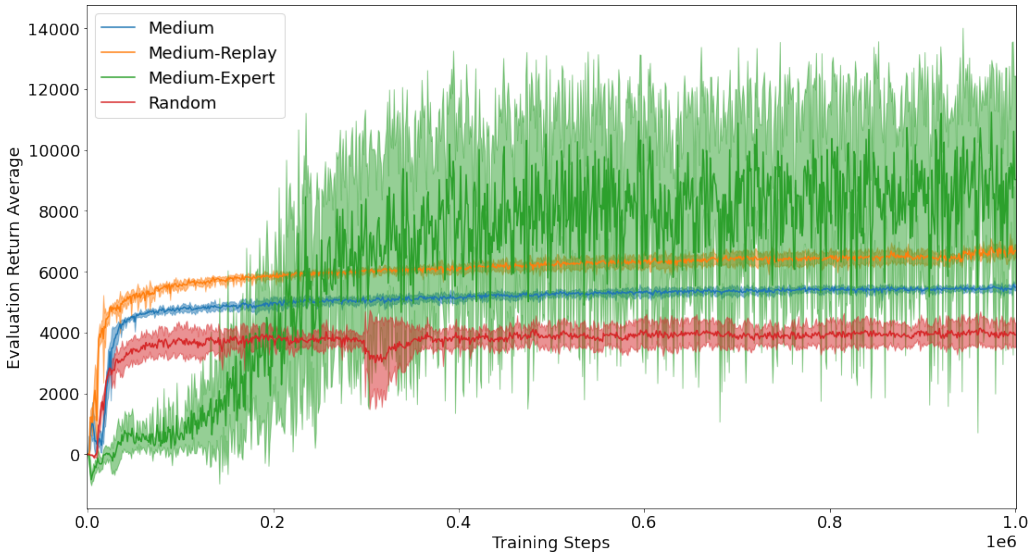


Fig. 3.3 Baseline MOPO runs against D4RL datasets (Fu et al., 2020). The mean and standard deviation over six random seeds are shown.

D4RL Dataset Type	Original Paper	Reproduction	
	Normalised Score	Normalised Score	Raw Returns
Random	35.4 ± 2.5	33.9 ± 3.2	3903 ± 316
Medium	42.3 ± 1.6	46.5 ± 0.8	5489 ± 100
Medium-Replay	53.1 ± 2.0	54.8 ± 1.9	6527 ± 230
Medium-Expert	63.3 ± 38.0	75.0 ± 27.5	9030 ± 3411

Table 3.1 Baseline MOPO runs against D4RL datasets (Fu et al., 2020). The mean and standard deviation of the normalised score over six random seeds are shown, along with the raw returns obtained when reproducing the results.

Chapter 4

Demonstrator Dataset Creation

It was necessary for us to obtain or generate suitable data to proxy for real-world demonstration datasets. We therefore define a set of data requirements, and evaluate popular offline RL benchmarks against these. We found no benchmark that met our requirements, and so we produce our own set of demonstration policies using online RL algorithms. From these, individual demonstrator and multi-demonstrator datasets are created.

4.1 Data Requirements

We define the following set of data requirements for our work:

- **Unmodified HalfCheetah Environment:** Data must have been collected in the HalfCheetah environment (Brockman et al., 2016), without any modifications to the original MDP. While changes in the dynamics and initial state distribution represent interesting sources of diversity that are likely to arise in the real world (e.g., changes in weather conditions when collecting driver data), in our work we seek to limit variation solely to changes in the policies used to collect the data.
- **Diversity:** The data must capture a broad range of experience levels and modes of behaviour. Drawing an analogy with human demonstrators for training autonomous driving agents, different levels of experience could be obtained by collecting data from learners, regular drivers, and those with advanced driving qualifications. To capture different modes of behaviour, we would collect data from multiple individuals within each of these groups, given that each demonstrator will exhibit unique behaviours.
- **Identifiability:** The source of the data must be known. While this information is not used when training dynamics models, it is needed to assess the impact of the inclusion

or emission of each source. For example, we may wish to determine if a policy with the performance of an advanced driver can be learned using a dynamics model that was trained with only data collected from learner drivers.

- **Granularity:** To enable granular control over the types of data included in multi-demonstrator datasets, rollouts collected using specific policy snapshots are preferred over data collected throughout training. This is analogous to collecting a new dataset from a learner driver each lesson, rather than recording a single dataset spanning several months of lessons. The former can be combined into the latter when desired.
- **Randomness:** The option to include data from a random policy should also be present, allowing the impact of its presence or absence to be evaluated. Given that the OpenAI gym has pre-defined random policies, benchmarks were not required to provide this.

If no individual benchmark met all of the above requirements then we would have been happy to combine several.

4.2 Benchmark Review

Using our defined data requirements, we assess the suitability of popular RL benchmarks for use in this work. Only those which include the HalfCheetah task are discussed.

- **D4RL (Fu et al., 2020):** D4RL offers 5 HalfCheetah datasets (see Appendix A.3 for full descriptions), three of which were generated from individual policies: *random*, *medium*, and *expert*. These could be used, however do not collectively provide the desired **diversity**, and so would need to be combined with other datasets. Additionally, a dataset comprising the replay buffer of an agent trained to medium level, *medium-replay*, and one which combines medium and expert data, *medium-expert*, are provided. While the *medium-replay* dataset violates the requirement for **granularity**, *medium-expert* does not meet the **identifiability** condition.
- **RL Unplugged (Gulcehre et al., 2020):** This benchmark includes datasets collected during the training of three independent D4PG agents (Barth-Maron et al., 2018), and thus represents a collection of replay buffers. These therefore do not meet the **granularity** requirement. Further, given the datasets come from the same implementation of an online RL algorithm, they're likely to exhibit similar modes of behaviour, and so do not offer the **diversity** desired.

- **DOPE (Fu et al., 2021):** Rather than individual datasets, the Deep Off-Policy Evaluation (DOPE) benchmark provides 11 snapshoted policies collected during the training of a SAC agent in the MuJoCo HalfCheetah environment. However, these appear to have been generated during the training of a single policy, and so may again not present significant **diversity**.

While rollouts using the snapshoted DOPE policies could have been generated and used alongside the D4RL random, medium and expert datasets, this still did not represent a significant **diversity** of data. We therefore decided to train a custom collection of demonstrator policies.

Given that Yu et al. (2020) benchmark the performance of MOPO against the D4RL datasets, these were still used when evaluating the domain generalisation capabilities of learned dynamics models.

4.3 Demonstrator Generation

We use two different implementations of the SAC algorithm to create demonstrators: **Softlearning**, the official SAC implementation (Haarnoja et al., 2018a,b); and **D3RLPY**, which implements a version of SAC with delayed policy updates (Seno and Imai, 2021). Policies were trained using the default hyperparameters provided in their respective repositories.

Snapshots of each policy were taken at regular intervals during training, as summarised in Table 4.1[1] and Figure 4.1. The performance of the Softlearning agents is aligned with the original SAC paper, whereas the D3RLPY results are lower than anticipated. Visual inspection of rollouts generated using D3RLPY policies showed that the expected task was being performed (i.e. the HalfCheetah was running forward), and so we did not reject these policies. The use of two different SAC implementations increased the diversity of the data, as we desired.

4.4 Demonstrator Dataset Generation

Using the demonstrator policies trained in the previous section, we created demonstration datasets consisting of 100 and 1000 episodes of 1000 steps each (the standard length of the HalfCheetah task), resulting in datasets with either 100,000 (0.1M) or 1,000,000 (1M) transition records. The selection of datasets produced allowed the impact of both demonstrator

[1] D3RLPY agents were not trained for 3 million steps due to memory issues experienced on the HPC, even when using high-memory nodes.

Training Time Steps (million)	Softlearning Policies	D3RLPY Policies
0.1	5647 ± 74	3789 ± 187
0.2	-	5172 ± 93
0.25	7387 ± 332	-
0.5	8981 ± 435	7298 ± 635
1	10534 ± 366	8691 ± 1108
2	13834 ± 1087	9710 ± 1193
3	14718 ± 545	-

Table 4.1 Mean evaluation return ± one standard deviation over three policies trained using Softlearning (Haarnoja et al., 2018a,b) and D3RLPY (Seno and Imai, 2021) SAC implementations.

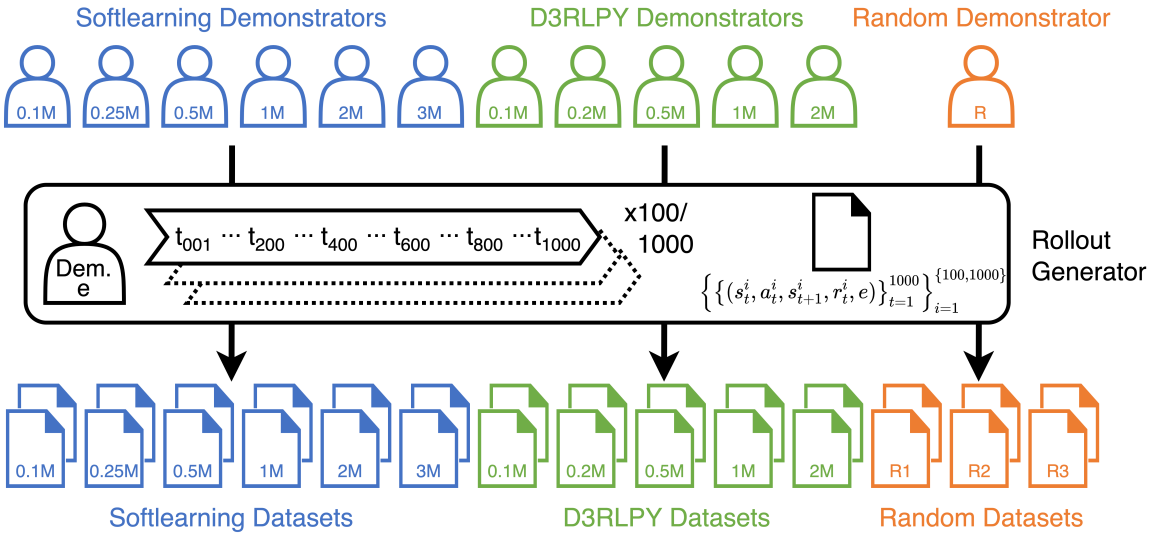


Fig. 4.1 Overview of the demonstrators created, and the process of generating individual demonstration datasets from these. One random demonstrator is used to create multiple random datasets.

experience (i.e., number of online training steps received) and data quantity on domain-generalisation performance and policy training to be evaluated. Figure 4.1 provides a visual representation of the generation process. Evaluation datasets consisting of 10 episodes were also independently generated for each policy. Finally, three 100 episode datasets were generated using a random policy.

4.5 Multi-Demonstrator Dataset Generation

We additionally use the trained demonstrators to create two-multi-demonstrator datasets, which we call **Novice** and **Mixed**. Each dataset contains a total of 0.1M transitions, with 20,000 records contributed by each of the five individual demonstrators shown in Table 4.2. Figure 4.2 illustrates the generation process followed for each dataset. Demonstrators were chosen based on the goal of each dataset:

1. **Novice:** We looked to answer the question: starting from a largely random training dataset with only limited examples of good behaviour, could dynamics models support the learning of a policy with returns exceeding those of the demonstrators used to create the training dataset? To achieve this, the learning policy will need to explore areas of the state-action space not captured by the training data, which dynamics models with good domain-generalisation performance are more likely to enable. Further, to reach high reward areas of the state-action space, exploration is likely to result in distributional shifts that are more extreme than those observed in training – allowing us to evaluate the extrapolation capabilities of REx. We exclude Softlearning demonstrators from this dataset so that we can evaluate how well models generalise to this class of demonstrators. In total, 40 % of the total transitions were generated using the D3RLPY 0.1M step demonstrator – 20 episodes of 1000 steps (i.e., complete episodes) and 100 episodes of 200 steps (i.e., the time during which the HalfCheetah is accelerating). A further 20 % of transitions were obtained from an agent trained for 20,000 steps. The remaining 40 % of the data is from a random policy.

2. **Mixed:** In contrast to the **Novice** dataset, the purpose of this dataset was to evaluate the benefits REx could bring to a highly diverse dataset with many examples of good behaviour. Two D3RLPY and two Softlearning demonstrators were used; in each case one of these had been trained for a limited number of steps, while the other had been trained for significantly more. Data was additionally generated from a random demonstrator to further increase diversity.

Compared to individual demonstrator policies, both of the above datasets decrease the amount of training data drawn from individual demonstrators, while increasing the diversity of starting locations for rollouts generated during policy training.

Novice Dataset			Mixed Dataset		
Demonstrator	**Episodes**	**Steps per Episode**	**Demonstrator**	**Episodes**	**Steps per Episode**
Random	20	1000	Random	20	1000
			D3RLPY 0.2M	20	1000
D3RLPY 0.02M	20	1000	D3RLPY 2M	20	1000
D3RLPY 0.1M	20	1000	SAC 0.1M	20	1000
D3RLPY 0.1M	100	200	SAC 1M	20	1000

Table 4.2 The demonstrators used to generate the **Novice** and **Mixed** datasets, including the number of episodes contributed to the dataset and the steps per episode.

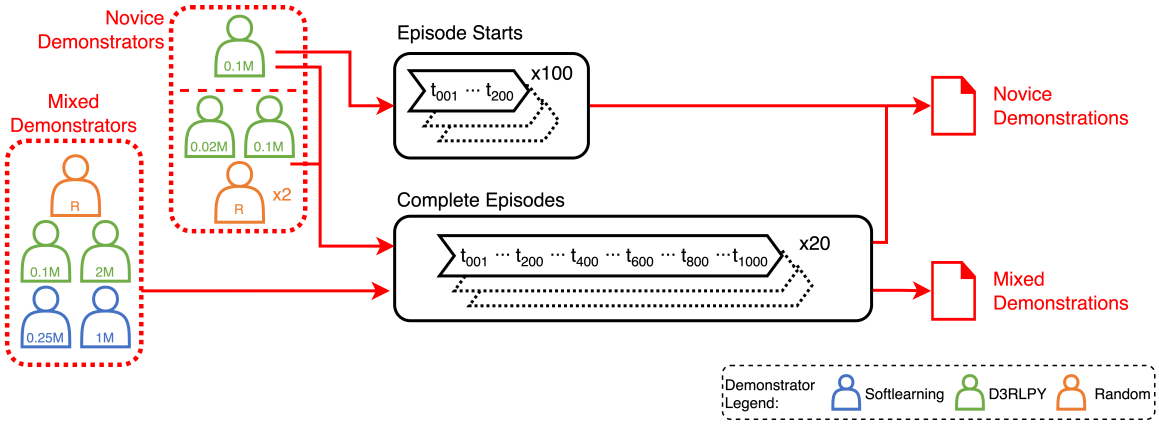


Fig. 4.2 The demonstrators used to create the **Novice** and **Mixed** datasets, and the process followed to generate demonstration datasets from these.

4.6 Conclusion

In this chapter, we defined a set of data requirements for our work, and, having identified no existing benchmarks which fulfil the criteria, created demonstrators from which individual- and multi-demonstrator datasets have been generated.

Ideally a wider variety of online RL algorithms – such as DQN (Mnih et al., 2013), DDPG (Lillicrap et al., 2015) and PPO (Schulman et al., 2017) – would have been used to train demonstrators. We attempted to use the RLlib (Liang et al., 2017) implementation of PPO to train agents, however only a fraction of the evaluation return quoted by the library's authors was achieved when running training with the same hyperparameters. This is believed to be an example of the reproduciblity crisis in RL (Gulcehre et al., 2020; Henderson et al., 2017). Future work should look to make use of a more diverse range of datasets, and generate a broader variety of multi-demonstrator datasets.

In the next chapter, we will use the datasets created here to train dynamics models and assess their domain-generalisation performance.

Chapter 5

Dynamics Model Training and Evaluation

Having generated demonstrator datasets in the previous chapter, we now use these to train dynamics models and analyse their domain-generalisation performance. As discussed in Chapter 3, the MOPO codebase (Yu et al., 2020) was used as the foundation for this work. We first provide an overview of the original dynamics model training procedure and loss function, before detailing the modifications made to incorporate risk extrapolation (REx) into training. Significant instability was observed during the REx phase of training – we discuss the methods employed in an attempt to reduce this.

We initially evaluate models trained on individual demonstrator datasets, which we use to illustrate the impact of distributional shift on model performance. We further evaluate the impact that demonstrator experience (i.e., the number of **steps** used in online policy training) and dataset size (the number of **transition** records) have on performance. We show that models trained on multi-demonstrator datasets generally exhibit improved generalisation capabilities, which are further enhanced by increasing the REx penalty coefficient applied in training. We observe that, as discussed in Section 2.2.3, REx "flattens the risk plane," by lowering risk (whether this be the log-likelihood or mean squared error) on certain out-of distribution domains while increasing it for others.

5.1 MOPO Dynamics Model Training

As discussed in Section 2.1.5, MOPO (Yu et al., 2020) models the environment dynamics and reward function using an ensemble of neural networks whose outputs parameterise a multivariate Gaussian distribution (with diagonal covariance matrix) over the next state and reward, conditioned on the current state and action.

$$p^i_{\theta,\phi}(s_{t+1}, r|s_t, a) = \mathcal{N}(s_{t+1}, r; \mu^i_\theta(s,a), \Sigma^i_\phi(s,a)) \tag{5.1}$$

Models are composed of 4 hidden layers with 200 units each. Training is performed via supervised learning, using mini-batch gradient descent with a batch size of 256 transition records, which are sampled uniformly at random from a static dataset of collected transitions, $\mathcal{D}_{\text{env}}$. In our work, $\mathcal{D}_{\text{env}}$ is either an individual demonstrator's dataset, $\mathcal{D}_e$, or a multi-demonstrator dataset, $\mathcal{D}_{\mathcal{E}}$. A set of 1000 records are chosen at random to be used as an evaluation dataset, with the remainder used for training. The loss function used when training the dynamics models, Equation 5.2, can be expressed as the sum of three components:

1. **Negative Log-Likelihood**, $\mathcal{R}_{nll}$: The first term is simply the negative log-likelihood (excluding constants), where N denotes the number of records in $\mathcal{D}_{\text{env}}$ and D is the dimensionality of each record. $\mu_d(\mathbf{x_n})$ and $\sigma_d^2(\mathbf{x_n})$ are the mean and variance of dimension d predicted for record $\mathbf{x}_n$.

2. **Weight Regularisation**, $\mathcal{R}_{wr}$: The second term corresponds to ℓ_2 weight regularisation, where L denotes the number of layers in the model, w_l denotes the weights in layer l, and ν_l is a weight decay term for layer l. ν_l values were not adjusted from those used in the original implementation.

3. **Variance Bounding**, $\mathcal{R}_{vb}$: Chua et al. (2018) highlight that outside of the training distribution the predicted variance can assume arbitrary values, and can both collapse to zero or explode to infinity (in contrast to models like GPs where variance values are better behaved). The authors found that bounding the output variance such that it cannot exceed the range seen in the training data was beneficial, and therefore include the final two terms in the loss function, where $\sigma^2_{\text{max,d}}$ and $\sigma^2_{\text{min,d}}$ are the learned maximum and minimum variance of dimension d respectively.

$$\mathcal{R}_{\text{MOPO}} = \mathcal{R}_{nll} + \mathcal{R}_{wr} + \mathcal{R}_{vb} \tag{5.2}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{D}\sum_{d=1}^{D}\left(\frac{(x_{n,d}-\mu_d(\mathbf{x_n}))^2}{\sigma_d^2(\mathbf{x_n})} + \log(\sigma_d^2(\mathbf{x_n}))\right) + \sum_{l=1}^{L}\nu_l||w_l||_2 + \left[0.01\sum_{d=1}^{D}\log\sigma^2_{\text{max},d} - 0.01\sum_{d=1}^{D}\log\sigma^2_{\text{min},d}\right]$$

The model's variance predictions are updated to account for the learned bounds as follows:

$$\log \sigma_d^2 = \log \sigma_{\max,d}^2 - \text{softplus}(\log \sigma_{\max,d}^2 - \log \sigma_d^2)$$
$$\log \sigma_d^2 = \log \sigma_{\min,d}^2 + \text{softplus}(\log \sigma_d^2 - \log \sigma_{\min,d}^2)$$

At the end of each epoch, the mean squared errors (MSE) for both training and evaluation datasets are calculated. Training terminates when the loss on the evaluation dataset does not decrease by more than 1 % for any model in the ensemble for five consecutive epochs.

5.1.1 Single Demonstrator Training Performance

No modifications were made to the original MOPO loss function or termination condition when training models on individual demonstrator datasets, except where explicitly highlighted. Figure 5.1 shows the evolution of the training and evaluation MSEs for models trained against individual D3RLPY demonstrator 0.1M transition datasets. For each dataset, we trained three sets of models independently with different random seeds. The results show that the termination condition can lead to significant variability in training times. The impact that training time has on domain generalisation performance will be shown.

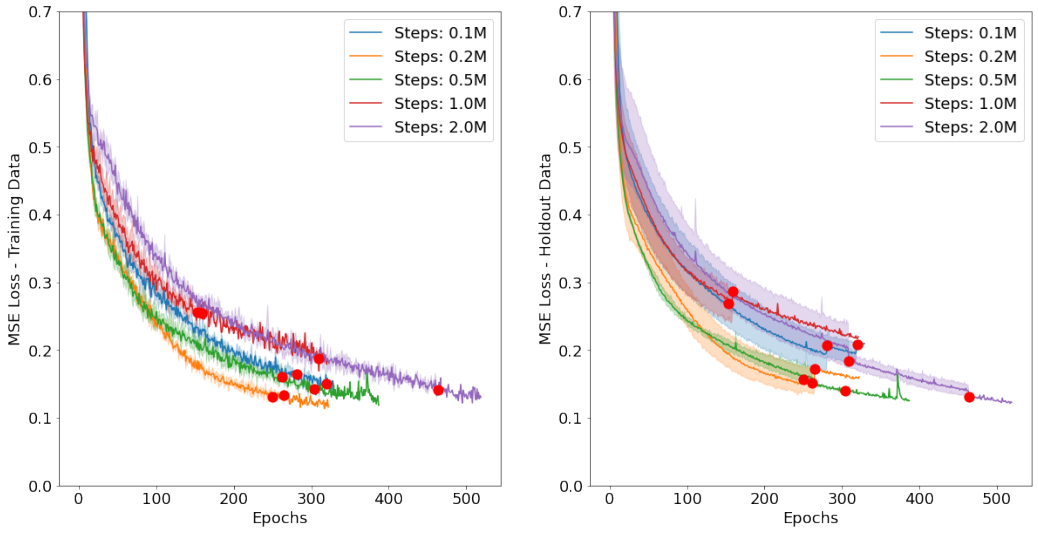


Fig. 5.1 Training and evaluation MSEs during the training of dynamics models on D3RLPY demonstrator 0.1M transition datasets. Colours denote the number of steps the demonstrator was trained for. The mean and standard deviation over three seeds are shown. The MOPO termination condition introduces variation in training times – red dots indicate when individual runs completed training.

Figure 5.2 shows that lower terminal training and validation losses were obtained when using 1M transition datasets, even though the number of training epochs was approximately the same as for the 0.1M transition datasets. To enable comparison of the impacts of dataset size and training time on domain-generalisation performance, we trained an additional set of models on the 0.1M transition datasets for four times the original number of epochs. This allowed the losses to approach those of models trained on 1M transitions. Although the validation losses typically remained significantly higher, there was no indication of overfitting.

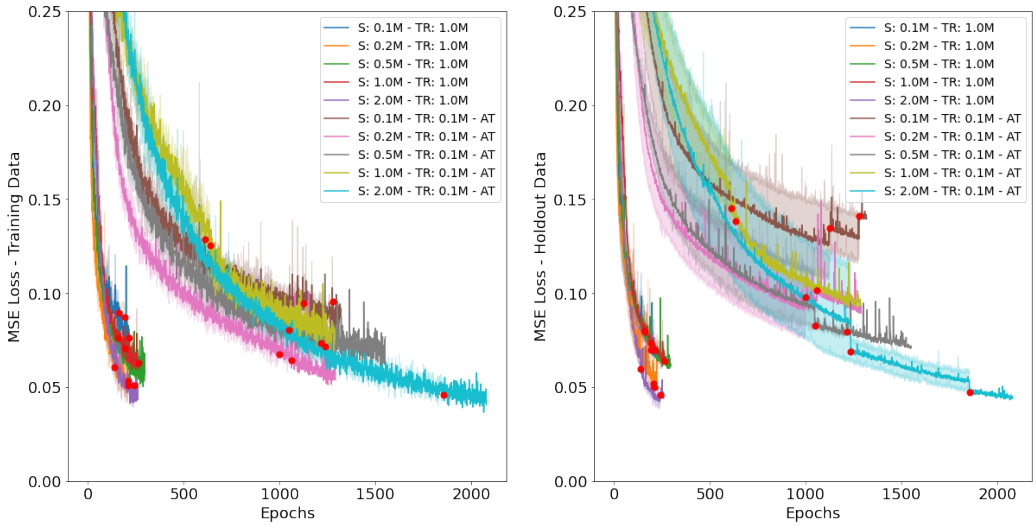


Fig. 5.2 Training and evaluation MSEs during dynamics model training on D3RLPY demonstration datasets containing 0.1M and 1M transitions. Colours denote the number of demonstrator steps (S) and transition records (TR). When the MOPO termination condition was reached, training on datasets containing 0.1M transitions was allowed to continuing until the number of completed epochs had increased four-fold. The mean and standard deviation over three seeds are shown. Red dots indicate when individual runs finished training, which can cause the mean to suddenly shift.

5.2 Dynamics Model Training with REx

The multi-demonstrator datasets represent data collected from M demonstrators, $\mathcal{D}_{\mathcal{E}} = \bigcup_{e=1}^{M} \mathcal{D}_e$. Individual negative log-likelihood values, $\mathcal{R}_{nll}^{e}$, are calculated for each demonstrator's data, $\mathcal{D}_e$. These are then used to calculate the MOPO V-REx loss given by Equation 5.3, where β is the REx penalty coefficient and τ_t is a learning rate decay term dependent on the training epoch, t.

$$\mathcal{R}_{MOPO\ V\text{-}REx} = \tau_t \left(\sum_{e=1}^{M} [\mathcal{R}^e_{nll}] + \beta \cdot \mathrm{Var}\left(\mathcal{R}^1_{nll}, \mathcal{R}^2_{nll}, \ldots, \mathcal{R}^M_{nll}\right) + \mathcal{R}_{wr} + \mathcal{R}_{vb} \right) \tag{5.3}$$

Model training for multi-demonstrator datasets was split into two phases:

1. **ERM:** Hyperparameter values $\beta = 0$ and $\tau_t = 1$ are used until the original MOPO termination condition is reached. This is the training that was performed for individual demonstration datasets.

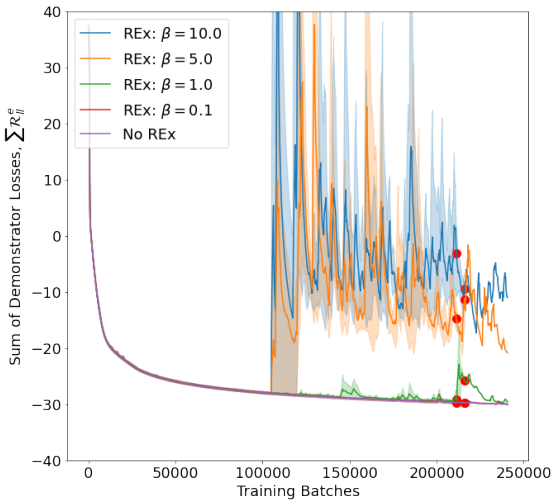
2. **REx:** Training was then allowed to continue for a pre-defined multiple of the number of epochs completed in the ERM phase (typically either 1 or 3), with β and τ_t now set to user-specified values.

To assess the impact that varying the strength with which the equality of risks across training domains is enforced had on both domain-generalisation performance and the training of policies, models were trained against the **Novice** and **Mixed** datasets using REx penalty coefficients $\beta \in \{0, 0.1, 1, 5, 10\}$.

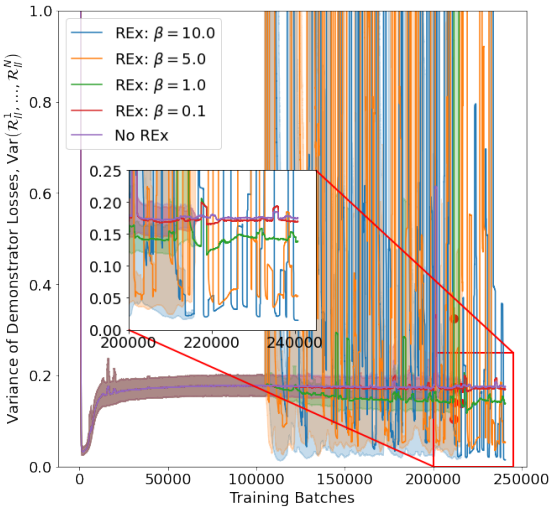
5.2.1 Multi-Demonstrator Training Performance

Figure 5.3 plots the evolution of the individual loss terms in Equation 5.3 during the training of models against the **Mixed** multi-demonstrator dataset. The **REx** phase of training was run for an equal amount of time as the initial **ERM** phase, with $\tau_t = 1$. While the training loss decreases monotonically and smoothly during the **ERM** phase, the point at which the **REx** phase begins is marked by significant instability in the sum and variance of demonstrator losses. The inset in Figure 5.3b suggests that higher REx penalty coefficients may have decreased the variance of training losses, however the noise precludes making definitive statements. Figure 5.3d also shows that the use of higher REx penalty coefficients led to a reduction in the variance bounding loss term, $\mathcal{R}_{vb}$, and therefore a narrowing of the values the predicted variance can take. This, in-turn, will limit the values that the MOPO penalty can take, given that penalties are determined using the predicted variance (see Section 2.1.5).

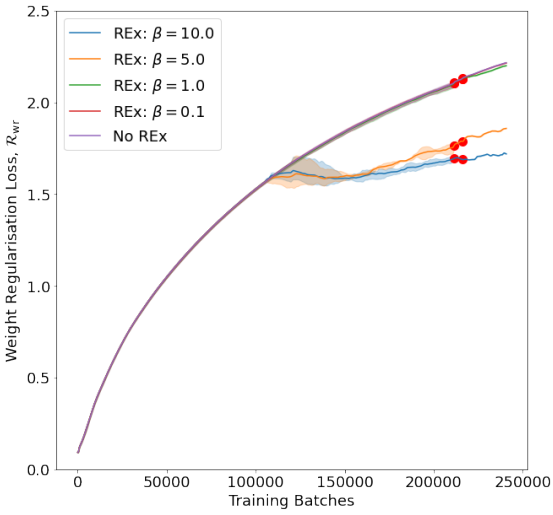
Several attempts were made to increase the stability of training, such as: 1) setting the learning rate decay term to $\tau_t = 10$ during the REx phase of training, and 2) increasing the mini-batch size five-fold in the hope that the empirical variance of the demonstrator losses would approach the population value, and that this would be less variable. However, as shown in Figure 5.4, these were not observed to offer a notable improvements. The runs using larger mini-batch sizes were allowed to train for a roughly equal total number of batches as the



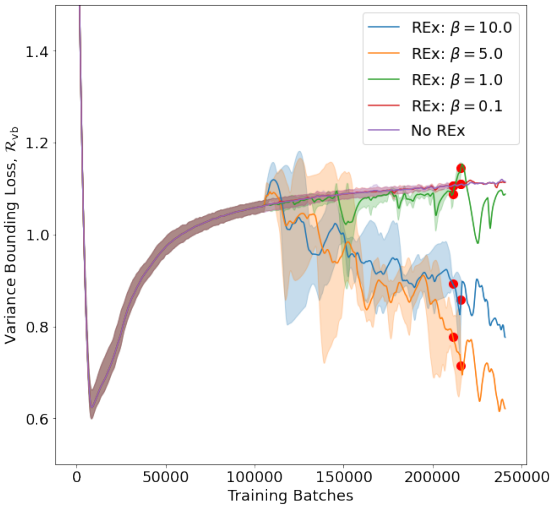
(a) Sum of Demonstrator Losses



(b) Variance of Demonstrator Losses

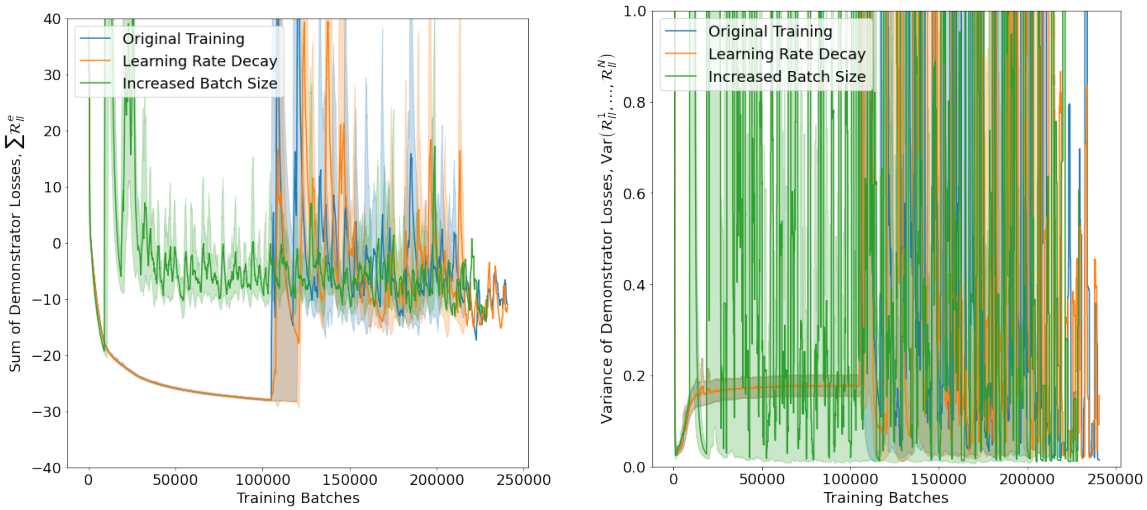


(c) Weight Regularisation Loss



(d) Variance Bounding Loss

Fig. 5.3 Evolution of the individual MOPO V-REx loss terms (Equation 5.3) when training models against the **Mixed** dataset. Colours denote the REx penalty coefficient used. The mean and standard deviation over three seeds are shown. The early stopping criterion introduces variation in training times – red dots indicate when individual runs completed training.



(a) Sum of Demonstrator Losses

(b) Variance of Demonstrator Losses

Fig. 5.4 Sum and variance of demonstrator losses during the training of models against the **Mixed** dataset with REx penalty coefficient $\beta = 10$. Colours denote modifications to the training procedure. In the Learning Rate Decay run hyperparameter τ_t was set to 10 during the REx phase of training, while in the Increased Batch Size run the batch size was increased five-fold to 1,280. The Increased Batch Size run was allowed to train for an approximately equal number of batches as the other runs, indicating a five-fold increase in the amount of training received. The mean and standard deviation over three seeds are shown.

previous runs, and so effectively also received five times the amount of training. The sum of demonstrator losses was not further reduced, nor was the variance of the losses.

5.3 Domain Generalisation Performance

We now use the dynamics models trained in the previous section to demonstrate the impact of distributional shift on model performance. We analyse the influence of various model training choices, the use of multi-demonstrator datasets, and the application of REx. As discussed in Section 3.2.2, we determine the mean-squared errors (MSE) and log-likelihoods for both in-distribution (ID) and out-of-distribution (OOD) evaluation datasets. While ID datasets are those generated from the same demonstrators used to train the model, OOD datasets comprise those generated from all other demonstrators, a random policy, and the D4RL datasets.

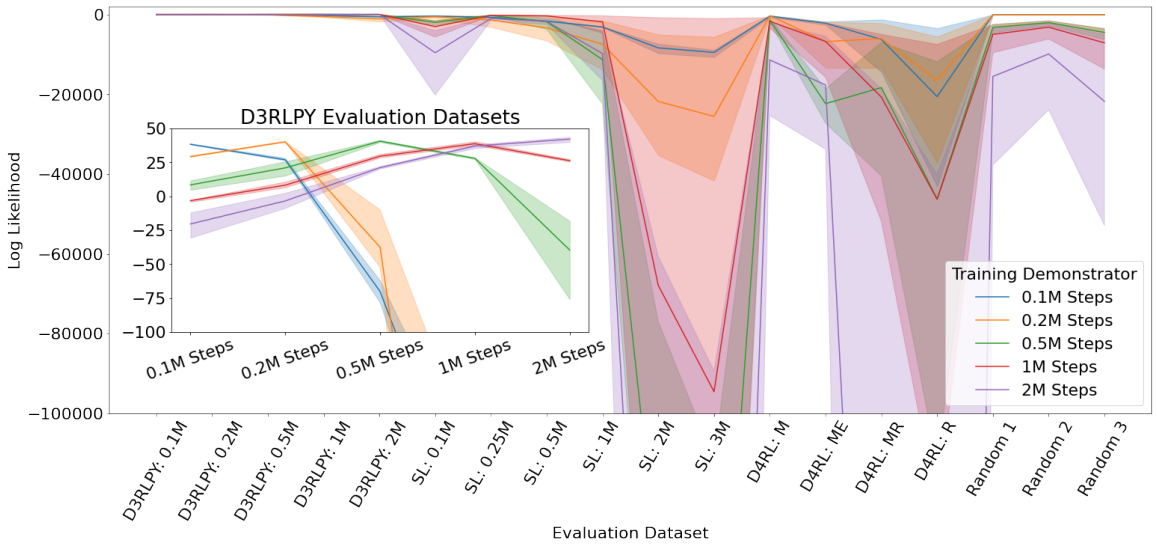
5.3.1 Individual Demonstrator Datasets

We initially demonstrate the domain-dependence exhibited by models trained against individual demonstrator datasets. Figure 5.5 shows the MSE and log-likelihood values for models trained using D3RLPY demonstrator 0.1M transition datasets. It's clear that the out-of-distribution performance is highly dependent on both training and evaluation domains, with the dataset type (e.g., Softlearning, D3RLPY, D4RL, Random) and number of demonstrator steps both having significant influence. The inset figures magnify the evaluation results for D3RLPY demonstrators and highlight that performance is highest in-distribution (i.e., when the evaluation dataset matches the training dataset). Performance decreases as the difference between the number of demonstrator steps in the training and evaluation datasets increases, likely due to larger distributional shifts.

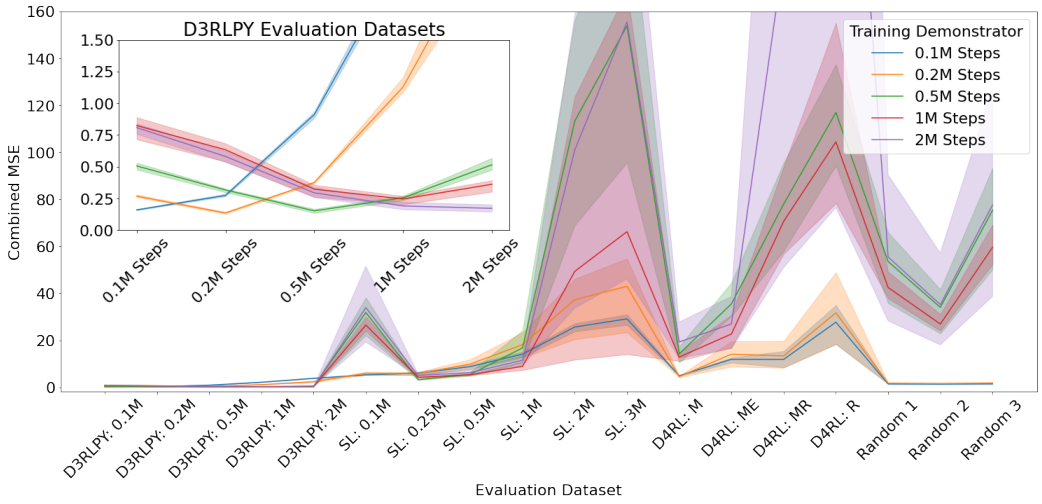
Looking across all evaluation datasets, models trained on 0.5M, 1M and 2M step demonstrators appear to exhibit lower generalisation capabilities than those trained with 0.1M and 0.2M steps. We hypothesise that demonstrators with extensive training will have more deterministic policies, and so their datasets will contain a narrower distribution of behaviours. Models trained on this data are therefore likely to have reduced generalisation capabilities. The inability of model-based offline RL methods to exceed the performance of model-free methods on datasets with limited diversity has been observed by many authors (Kidambi et al., 2020; Rafailov et al., 2020; Yu et al., 2020), and so the reduced OOD performance is anticipated to have a negative impact on policy training. This is evaluated in the Chapter 6.

In addition to those trained on 0.1M transitions, dynamics models had also earlier been trained using 1M transitions for approximately the same number of epochs, and had been observed to have lower training and evaluation losses. Figure 5.6 shows that the in-distribution performance of these models improves, while the OOD performance deteriorates. The same effect is seen for the models that had been trained for four times the original number of epochs while holding the number of transition records constant at 0.1M. This indicates overfitting to the training domain, which we predict will have a detrimental impact on policy training. This is also evaluated in Chapter 6.

Table 5.1 shows the average and worst-case MSE and log-likelihood values obtained across OOD datasets for all individual demonstrator models, including Softlearning demonstrators. ID results are also included, which highlight the significant gap between ID and average OOD performance. While OOD performance generally decreased with demonstrator steps for D3RLPY demonstrators, the same level of consistency was not observed for Softlearning demonstrators. Looking at the average values for each demonstrator type, D3RLPY demonstrators have both lower MSEs and log-likelihoods – the centre of their Gaussians are



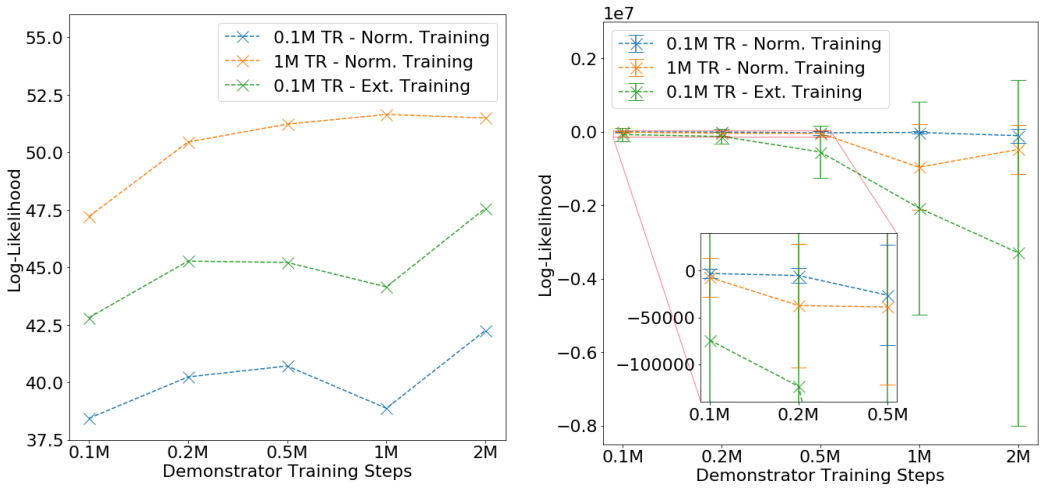
(a) Log-Likelihood



(b) MSE

Fig. 5.5 Log-likelihood and MSE values for models trained with 0.1M transition D3RLPY demonstration datasets, measured against a wide range of evaluation datasets. Colours denote the number of steps the demonstrator was trained online for. The mean over three independently trained models is shown, while the edges of the shaded regions convey the minimum and maximum values. Softlearning is abbreviated to SL.

closer to the true value, but the variances are higher. We will compare these results against those for models trained using multi-demonstrator datasets in the next section.



(a) Training Dataset

(b) OOD Datasets

Fig. 5.6 Log-likelihood values for models trained with D3RLPY demonstration datasets. Colours indicate the number of transitions (TR) in the dataset, and the training procedure: Norm. Training = the standard training procedure; Ext. Training = a four-fold increase in the number of training epochs. The mean and standard deviation over all OOD datasets is shown in Figure (b).

5.3.2 Multi-Demonstrator Datasets

In the previous section we showed that the domain-generalisation performance of models trained on individual demonstrators was dependent on both the training and evaluation domains. We now demonstrate that this dependence is generally reduced for models trained on multi-demonstrator datasets, and that further improvements are gained by applying REx during training.

The average and worst-case MSE and log-likelihood values for OOD datasets are shown in Table 5.2. The results for models trained without REx (i.e., $\beta = 0$) are compared against those for individual demonstrators, shown in Table 5.1. For the **Novice** dataset, the maximum MSE values are lower than 66 % of Softlearning models and 60 % of D3RLPY models, while minimum log-likelihoods are lower than 33 % of Softlearning models and the same 60 % of D3RLPY models. The improvements obtained across all OOD metrics using the **Mixed** dataset were far more substantial – domain-generalisation performance was significantly better than for any individual demonstrator model. We find the difference between the datasets to be unsurprising: the **Mixed** dataset contains transitions from a more diverse collection of demonstrators, so models trained against it would be expected to have higher domain generalisation performance.

Demonstrator		MSE			Log-Likelihood		
Type	**Steps (M)**	**ID**	**OOD Avg**	**OOD Max**	**ID**	**OOD Avg**	**OOD Min**
Softlearning	0.1	0.44	4.71	19.93	27	-519	-3424
Softlearning	0.25	0.14	477.94	1888.46	42	-55048	-199632
Softlearning	0.5	0.11	292.01	1177.09	39	-54414	-224586
Softlearning	1	0.20	3.35	18.53	36	-197	-1059
Softlearning	2	0.28	21.43	71.61	34	-1772	-6452
Softlearning	3	0.33	17.42	56.56	35	-1308	-4341
Average		**0.25**	**136.14**	**538.70**	**36**	**-18876**	**-73249**
D3RLPY	0.1	0.16	9.22	29.11	38	-3183	-20582
D3RLPY	0.2	0.14	11.36	42.98	40	-5384	-25543
D3RLPY	0.5	0.15	43.15	153.77	40	-26300	-193527
D3RLPY	1	0.25	29.54	104.4	38	-15209	-94565
D3RLPY	2	0.17	63.74	370.65	42	-103215	-714632
Average		**0.17**	**31.40**	**140.18**	**40**	**-30658**	**-209770**

Table 5.1 MSE and log-likelihood values for dynamics models trained using Softlearning and D3RLPY demonstrators. Each model was assessed against data drawn from its training distribution, and an average taken over data drawn from OOD datasets. Maximum MSE and minimum log-likelihood values are also shown, along with average statistics for both Softlearning and D3RLPY demonstrator types.

Demonstrator	REx Penalty Coefficient, β	MSE			Log-Likelihood		
		ID	**OOD Avg**	**OOD Max**	**ID**	**OOD Avg**	**OOD Max**
Novice	0	0.19	8.58	49.45	29	-4653	-47804
Novice	0.1	0.19	8.54	50.22	29	-3931	-37236
Novice	1.0	0.18	17.72	198.90	30	-3759	-32212
Novice	5.0	0.23	8.60	91.23	24	-2848	-38089
Novice	10.0	0.29	3.97	20.19	14	-432	-4660
Mixed	0	0.22	0.64	2.28	34	55	-183
Mixed	0.1	0.22	0.65	2.51	34	65	-222
Mixed	1.0	0.22	0.64	2.34	33	63	-214
Mixed	5.0	0.25	0.65	1.80	26	21	-55
Mixed	10.0	0.28	0.72	1.96	23	14	-28

Table 5.2 MSE and log-likelihood values for dynamics models trained using **Novice** and **Mixed** demonstrators. Each model was assessed against data drawn from its training distribution, and an average taken over data drawn from OOD datasets. Maximum MSE and minimum log-likelihood value are also shown.

As β was increased, ID performance decreased for both datasets: MSE values increased and log-likelihoods decreased. The maximum OOD MSE was lower at $\beta = 10$ than $\beta = 0$ in both cases, however there was significant variability in the region between these settings, and the average OOD MSE increased slightly for the **Mixed** dataset.

While there are outliers, the minimum OOD log-likelihood showed a positive correlation with β for both datasets, as can be seen clearly in Figure 5.8. However, while the average OOD log-likelihood for the **Novice** dataset also increased with β, for the **Mixed** dataset it decreased. Figure 5.7 plots the individual log-likelihoods across the evaluation datasets. As had been anticipated, it's clear that the "risk plane" was flattened for $\beta \in \{5, 10\}$. In the case of the **Novice** dataset this led to a decrease in log-likelihood for some domains (as can be seen from the inset in Figure 5.7b), however many domains saw a sizeable increase. This includes all Softlearning domains, which had been excluded from the **Novice** dataset. For the **Mixed** dataset the story is reversed: log-likelihoods decreased for most domains, and so the average is reduced. However, the average and minimum values are still significantly higher than for the **Novice** dataset.

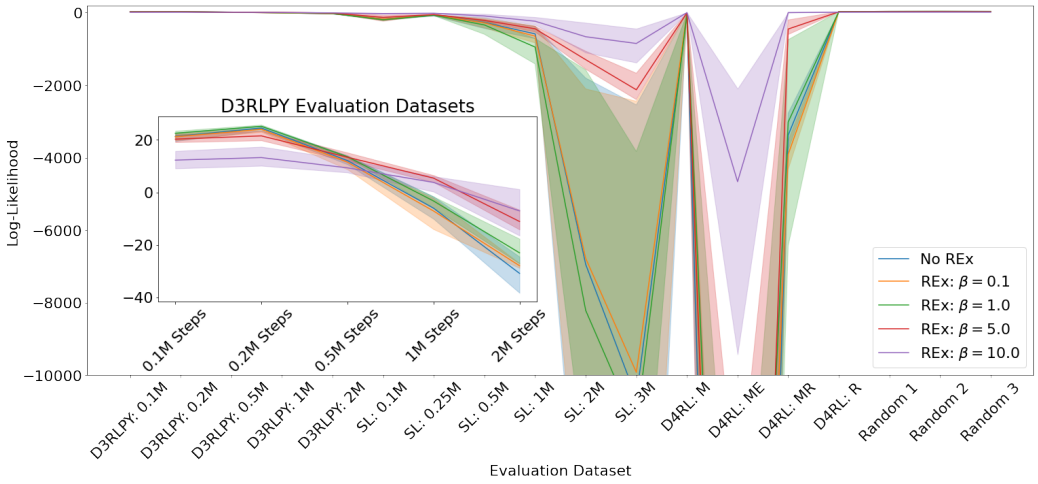
There was concern that REx may achieve equality of risks by simply learning to predict high variance regardless of the inputs to the model; thus making the predicted distributions more uniform. Figure 5.9 shows that high REx penalty coefficients did result in more uniform average predicted variances across the evaluation datasets, but this includes decreases for many domains, rather than universal increases. REx effectively exhibits greater certainty in domains with larger distributional shifts, at the expense of those domains with lowest distributional shift.

5.4 Conclusions

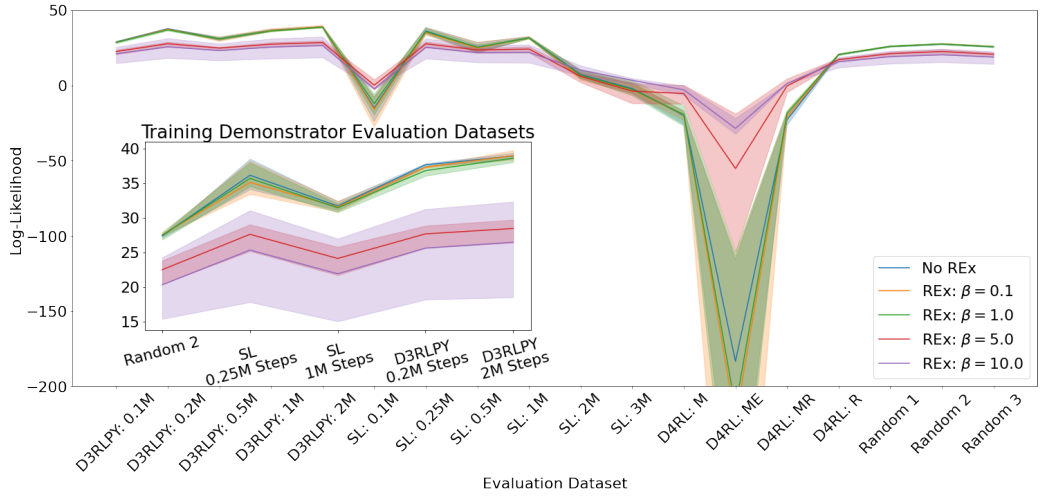
In our experiments, we found that the domain generalisation performance of dynamics models is highly dependent on the dataset used in training. We had expected that demonstrators trained online for more steps may have produced datasets sampled from narrower distributions, which might result in dynamics models with reduced domain generalisation performance. While there was some evidence for this, it was not consistent across demonstrator types. We found that OOD performance decreased when larger training datasets were used, and when training time increased. We expect positive correlation between the OOD performance of models and the return of policies trained with them – this is evaluated in the next chapter.

We observed that the **Novice** multi-demonstrator dataset did not contain sufficient diversity to improve domain generalisation performance beyond that of all models trained using individual demonstrators. However, REx penalty coefficients at the higher end of those investigated ($\beta \in \{5, 10\}$) were observed to improve average and worst-case OOD metrics. While we showed that REx flattened the risk plane, for the **Mixed** dataset this led to a reduction in the worst-case risk and an increase in the average risk. Although our

belief is that increased domain generalisation performance should be beneficial to policy training, whether improved average or improved worst-case performance is more favourable was unknown. Experiments performed in the next chapter allow us to analyse this.

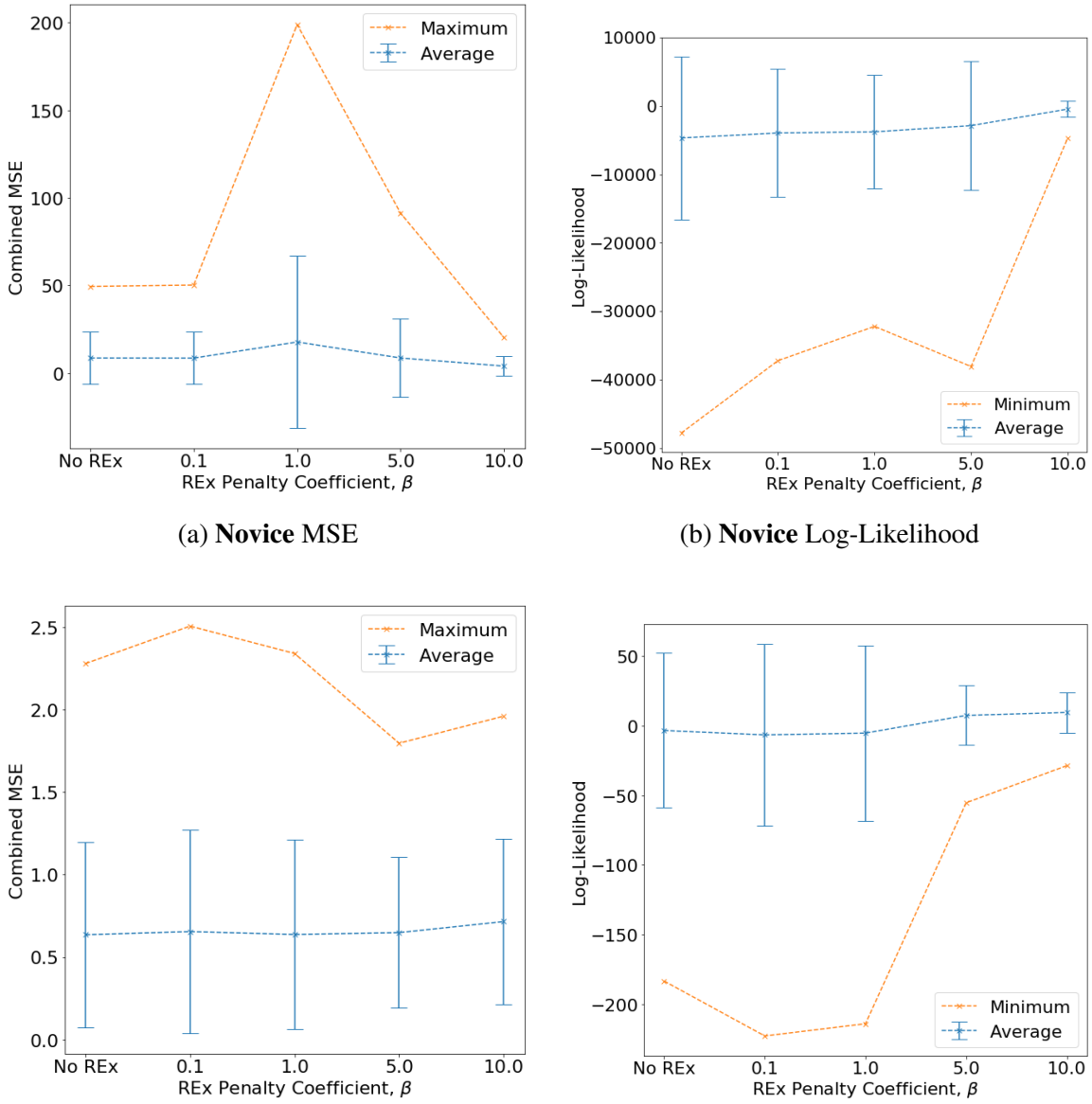


(a) **Novice** Dataset



(b) **Mixed** Dataset

Fig. 5.7 Log-likelihood values for models trained against the **Novice** and **Mixed** datasets, across a range of evaluation datasets. Colours denote the REx penalty coefficient used. The inset figure in (a) shows the results for the D3RLPY evaluation datasets, given that the D3RLPY 0.1M demonstrator was a contributor to the **Novice** dataset. The inset figure in (b) shows the results for the evaluation datasets of the training demonstrators that contributed to the **Mixed** dataset. Softlearning is abbreviated to SL



(a) **Novice** MSE

(b) **Novice** Log-Likelihood

(c) **Mixed** MSE

(d) **Mixed** Log Likelihood

Fig. 5.8 Average and worst-case MSE and log-likelihood values across OOD evaluation datasets for models trained with **Novice** and **Mixed** datasets using REx penalty coefficients $\beta \in \{0, 0.1, 1, 5, 10\}$. The worst-case MSE value is the maximum obtained across the OOD datasets, while the worst-case log-likelihood value is the minimum obtained across the OOD datasets. The error bars on the average values represent the standard deviation over the datasets.

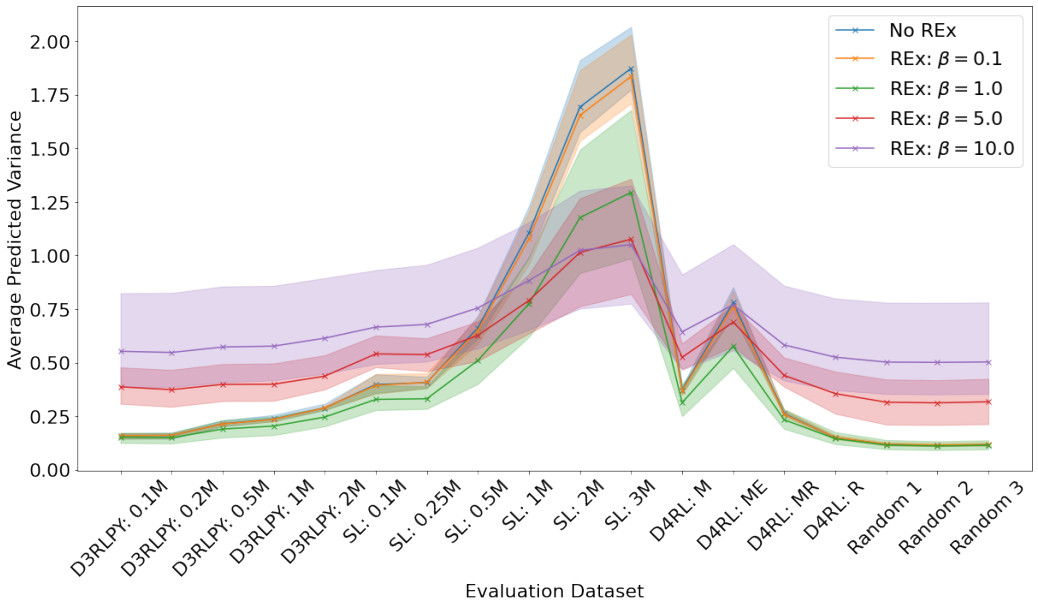


Fig. 5.9 Variance values predicted by dynamics models trained against the **Novice** dataset across a range of evaluation datasets. The mean and standard deviation over three independently trained models are shown. Colours denote the REx penalty coefficient used to train the models.

Chapter 6

Policy Training and Evaluation

In this chapter, we use dynamics models trained against both individual demonstrator, $\mathcal{D}_e$, and multi-demonstrator, $\mathcal{D}_\mathcal{E}$, datasets to learn and evaluate policies. Recalling from Section 3.2.3, policies are trained using the SAC algorithm (Haarnoja et al., 2018a,b). Rollouts are generated by running the current iteration of the policy being learned against a supplied dynamics model, with the collected experience used by SAC to update the policy. In an attempt to limit distributional shift, previous works (Janner et al., 2019; Yu et al., 2020) sample rollout starting locations from the same dataset used to train the dynamics model, and use horizons of at most five steps. Further, MOPO (Yu et al., 2020) uses the uncertainty-penalised reward given by Equation 6.1, where λ is the the user-specified MOPO penalty coefficient, to deter the learning policy from visiting areas of the state-action space where model uncertainty is high, as measured using the variance predictions of the model.

$$\tilde{r}(s,a) = \hat{r}(s,a) - \lambda \max_{i=1,\ldots,N} ||\Sigma_\phi^i(s,a)||_\text{F} \tag{6.1}$$

We first use dynamics models trained against individual demonstrator datasets to learn policies, with the goal of establishing the impact that domain-generalisation performance has on policy returns. This includes demonstrating that policies cannot be learned if rollout starting locations are sampled from OOD datasets.

We then show that the improved domain generalisation performance of dynamics models trained with risk extrapolation (REx) allows rollouts of more than five steps to be used in training, yields policies with higher average returns, and increases the robustness of policy training when sampling rollout starting locations from OOD datasets. While we expect REx to reduce model prediction errors, we know that such errors will never be entirely eliminated, and so we evaluate the impact that the MOPO penalty coefficient, λ, continues to play on policy training.

We analyse each set of trained policies with the goal of extracting maximal information about the presence and impact of domain dependence. As outlined in Section 3.2.3, this includes looking at the amount of model exploitation which occurs, and the agreement in expected policy returns across learned models. We show that, as anticipated, improved domain generalisation performance is beneficial, however we also observe that high MOPO penalty coefficients are important for reducing model exploitation.

6.1 Individual Demonstrator Policies

6.1.1 Policy Training

When using dynamics models trained against individual demonstrator datasets to learn policies, we observe that policy returns generally increase with the number of demonstrator training steps up to a certain point, before dropping – often significantly. While policy returns do not generally appear to be correlated with the OOD performance of the dynamics model used in training, we do observe that returns increase when the number of transition records used to train the dynamics models is increased. This was surprising, given we found in the last chapter that these models have lower domain-generalisation performance.

Replicating the approach taken by the MOPO algorithm (Yu et al., 2020), rollout starting locations were sampled from the same dataset used to train the dynamics model, and therefore varied across experiments. We used a MOPO penalty coefficient of $\lambda = 1$ and rollout length $h = 5$, as these were the most common hyperparameter settings for D4RL HalfCheetah datasets in the MOPO paper (Yu et al., 2020). We observed that policy training generally plateaued after 0.5M steps, and so all experiments were trained for this amount of time. The policy evaluation returns obtained in the final epoch of training are summarised in Table 6.1, and are compared with the OOD performance of the dynamics model they were trained with in Figure 6.1. Experiments had additionally been run using $\lambda = 0$ to assess the impact of discarding the MOPO penalties – this was observed to lower policy returns in almost all cases (see Appendix B.1).

While the results appear to show a misalignment between domain generalisation performance and the quality of the policies learned, other factors – such as not having tuned the hyperparameters used in policy training for each experiment individually, and the variation in rollout starting locations – will likely have had an impact.

Additionally, we consider **where** in the state-action space high model performance might have been most critical for these experiments. While larger quantities of transition records were found to be detrimental to average OOD performance, we know that in-

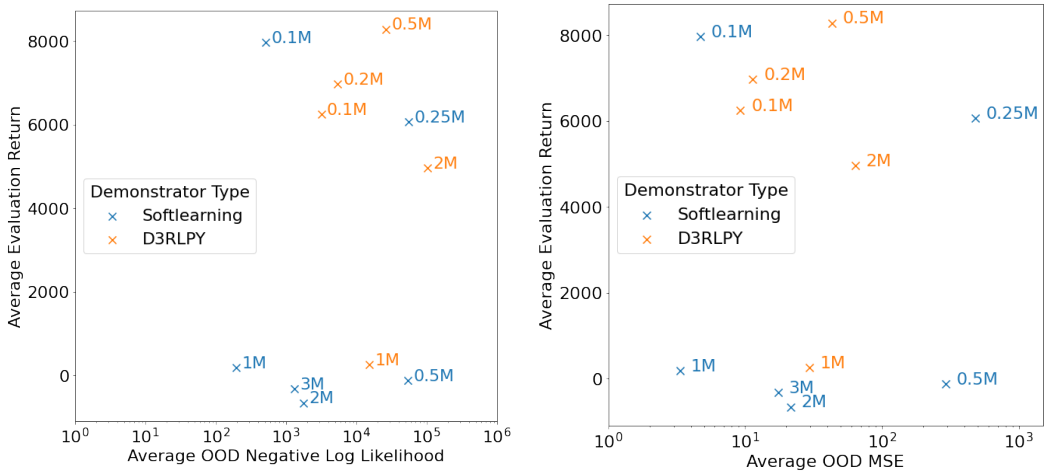


Fig. 6.1 No correlation is observed between the OOD performance of dynamics models and the average evaluation return of policies trained against it. Individual data point labels indicate the number of demonstrator steps. Models were trained on 0.1M transition records. OOD performance metrics were taken from Table 5.1.

distribution performance was improved. MBPO and MOPO use short rollouts from ID starting locations in an explicit attempt to limit distributional shift, therefore improved in-distribution performance may be beneficial for finding locally optimal policies. If this were the case, it might be expected that we would be constrained towards finding policies with similar returns as the demonstrator policy (see Table 4.1). However, there were many instances where the returns of the policies trained offline exceeded those of the demonstrator policy – such cases are bolded in Table 6.1.

To standardise the source of rollout starting locations and investigate the impact of using OOD datasets, we re-ran policy training for the D3RLPY 1M transition dynamics models with new starting locations. These were sampled from one of three datasets: D4RL Mixed-Replay, and transitions from both a Softlearning and random policy. When using in-distribution starting locations, all D3RLPY 1M transition dynamics models trained policies with evaluation returns > 6000. However, the results in Table 6.2 show that no policies with significantly positive evaluation returns could be learned using OOD starting locations, and many did not complete 0.5M steps of training. Training stopped prematurely when the policy model became degenerate and failed to make action predictions – which we discuss in Section 6.2.3.

Demonstrator Steps (M)	Softlearning Demonstrators		D3RLPY Demonstrators	
	0.1M Records	1M Records	0.1M Records	1M Records
0.1	**7961 ± 612**	**8956 ± 101**	**6244 ± 90**	**6961 ± 220**
0.2	-	-	**6963 ± 53**	**7622 ± 148**
0.25	6058 ± 3319	**10091 ± 533**	-	-
0.5	-124 ± 252	4334 ± 3830	**8280 ± 11**	**8354 ± 116**
1	181 ± 119	9124 ± 1573	258 ± 813	8219 ± 500
2	-657 ± 368	450 ± 721	4960 ± 3573	6879 ± 1431
3	-323 ± 49	-302 ± 123	-	-

Table 6.1 Policy returns initially increase with the number of demonstrator training steps, before decreasing sharply. Further, increasing the number of transition records increases policy returns. Policies were learned using dynamics models trained on individual Softlearning and D3RLPY demonstrators. The mean and standard deviation of the policy evaluation returns in the final epoch of training over three random seeds is shown. Returns which exceed those of the original demonstrator (see Table 4.1) have been bolded.

Demonstrator Steps (M)	D4RL Mixed-Replay	Random	Softlearning 0.25M
0.1	-130 ± 47 (3)	170 ± - (1)	-226 ± 328 (3)
0.2	-191 ± 136 (3)	-243 ± - (1)	115 ± 868 (3)
0.5	-182 ± 179 (3)	- (0)	-259 ± 35 (3)
1	-351 ± 470 (3)	- (0)	-395 ± 135 (2)
2	-89 ± 116 (2)	- (0)	-202 ± - (1)

Table 6.2 No policies with substantially positive returns are obtained when sampling rollout starting locations from OOD datasets. Policies were learned using dynamics models trained on D3RLPY demonstration datasets with 1M transitions. Each experiment was run for three seeds and 0.5M training steps. The format of the results is: mean ± std deviation (number of runs which completed training).

6.1.2 Policy Evaluation

We use the methods described in Section 3.2.3 to evaluate the learned policies. Our goal is to highlight the presence and extent of domain-dependencies for models trained on individual demonstrator datasets. We will later demonstrate the improvements obtained using multi-demonstrator datasets.

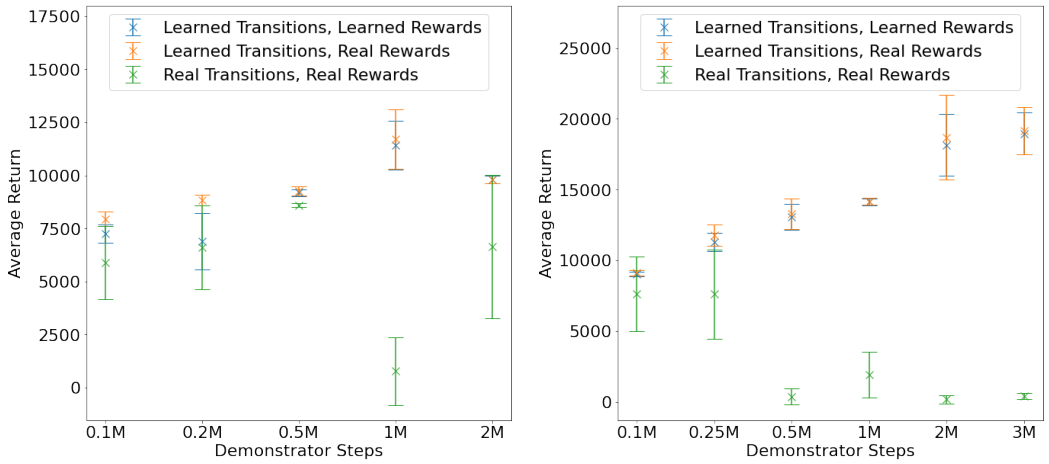
To determine the prevalence of model exploitation, in Figure 6.2 we compare the real environment returns (real transitions, real rewards) against those obtained with the learned dynamics model and reward function that were used in policy training (learned transitions, learned rewards). We refer to the latter as the *self-evaluation* returns. The initial state distribution of the real environment was used in both cases. While the trend is more consistent for Softlearning demonstrators, the results show a positive correlation between the number of demonstrator steps and the self-evaluation returns, which is not reflected by the returns in the real environment. This indicates that errors in the learned models have been exploited – spurious transitions and rewards have resulted in the policy learning behaviours that do not yield the same returns in the real environment.

Also shown in Figure 6.2 are the policy returns obtained using the learned dynamics model with the real reward function. Apart from the 0.1M and 0.2M step D3RLPY demonstrators, the distribution of returns using the learned reward function are in close agreement with those of the real reward function. Where there is disagreement, the reward has typically been under-predicted, although any errors will prevent optimal policies from being identified.

As a further example of the domain dependence exhibited by models trained on individual demonstrators, Figure 6.3 shows the evaluation of each policy against each learned model for both D3RLPY and Softlearning demonstrators. If all models made perfect (or at least identical) predictions across the entire state-action space then they would induce matching MDPs, and so would yield the same expected return for a given policy (see Section 2.1.1). We would therefore only see variation across the rows of Figure 6.3, as the policy being evaluated changes. However, there is significant variation across the columns, which highlights the poor domain-generalisation performance of the models.

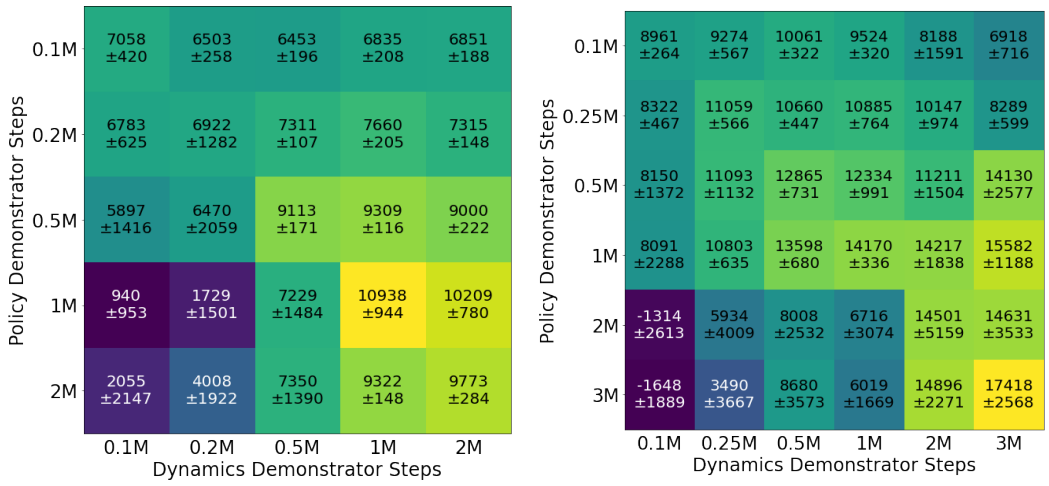
6.2 Multi-Demonstrator Policies

We now use the dynamics models trained on the **Novice** and **Mixed** multi-demonstrator datasets to highlight three key benefits to policy learning that are provided by models trained with REx, and specifically a REx penalty coefficient of $\beta = 10$ (the highest we evaluate). First, for both datasets, the highest mean evaluation returns were obtained using models trained with REx and $\beta = 10$. Secondly, when the rollout length was increased to $h = 10$,



(a) D3RLPY Demonstrators (b) Softlearning Demonstrators

Fig. 6.2 The expected policy returns obtained when using the learned dynamics and reward models are higher than in the real environment (i.e., when using the real dynamics and reward function). This indicates the presence of model exploitation. There is generally a close alignment between the returns using the learned and real reward functions. Policies were trained using a rollout length $h = 5$ and MOPO penalty coefficient $\lambda = 5.0$. The mean and standard deviation of the returns over 10 episodes is shown.



(a) D3RLPY Demonstrators (b) Softlearning Demonstrators

Fig. 6.3 The expected policy return is dependent on the model it is evaluated against. Trained models are denoted by columns, while policies are denoted by rows. In each case, the number of steps of online training received by the demonstrator used to trained the model/policy is shown. Policies were trained using a rollout length $h = 5$ and MOPO penalty coefficient $\lambda = 5$. An empirical estimation of the expected policy return was determined using all other learned models. The mean and standard deviation of the returns over 10 episodes is shown.

models trained without REx either failed to learn policies with positive returns, or failed to complete 0.5M steps of training at all. Those trained with $\beta = 10$ showed improved robustness and learned policies with positive mean returns. Finally, only models trained with $\beta = 10$ completed 0.5M steps of training when rollout starting locations were sampled from OOD datasets.

6.2.1 Policy Training

Policies were trained using MOPO penalty coefficients $\lambda \in \{0, 1, 5\}$ and rollout lengths $h \in \{5, 10\}$. While the MOPO paper does not report results for $h = 10$, we investigate longer rollout lengths to evaluate whether REx provided sufficient improvements in domain generalisation performance such that training remains stable under increased exploration, thus allowing more optimal policies to be learned. In preliminary experiments, dynamics models trained with REx penalty coefficients $\beta \in \{0.1, 1\}$ typically did not yield policies with notably differing performance from those trained without REx. Therefore, models trained with $\beta \in \{0, 5, 10\}$ were investigated.

The results in Table 6.3 show that the highest mean return for both datasets was obtained using models trained with $\beta = 10$. For the **Novice** dataset and a rollout length $h = 5$, dynamics models with REx penalty coefficient $\beta = 10$ achieved returns with the highest mean and lowest variance across all settings of λ. When the rollout length was increased to $h = 10$, the only runs which completed 0.5M steps of policy training were those which had used dynamics models trained with $\beta = 10$.

Conversely, for the **Mixed** dataset, the policy with the highest mean return was obtained for a rollout length $h = 10$. Models trained without REx could not learn a good policy, but did all complete training – an improvement over the **Novice** dataset, which likely arose due to the superior domain-generalisation performance of the **Mixed** models. For models trained with REx, the mean returns increased as β increased. We observed in the previous chapter that higher REx penalty coefficients increased worst-case OOD performance while decreasing average OOD performance, which appears to have been a beneficial trade-off for policy training.

Using the **Mixed** dataset, no good policies could be learned when rollout starting locations were sampled from OOD datasets, but models trained with REx and $\beta = 10$ were the only ones which completed training across all OOD datasets for a rollout length $h = 10$. Table 6.4 shows the policy returns for $\beta \in \{0, 10\}$. This result echos the observations from the **Novice** dataset, where models trained with a REx penalty coefficient $\beta = 10$ were the only ones able to complete training for $h = 10$.

Dataset	MOPO Penalty Coefficient, λ	Rollout Length, h	REx Penalty Coefficient, β		
			0.0	5.0	10.0
Novice	0.0	5	4936 ± 3752	**2580 ± 2984**	**7186 ± 227**
Novice	1.0	5	**5000 ± 3757**	61 ± 235	7123 ± 537
Novice	5.0	5	3424 ± 2520	746 ± 1314	4734 ± 561
Novice	0.0	10	- (0)	- (0)	4238 ± 2243 (2)
Novice	1.0	10	- (0)	- (0)	5454 ± 992 (2)
Novice	5.0	10	- (0)	- (0)	5474 (1)
Mixed	0.0	5	6834 ± 3191	3066 ± 914	3256 ± 1340
Mixed	1.0	5	**7642 ± 673**	3995 ± 3917	3577 ± 1758
Mixed	5.0	5	2212 ± 3433	**7063 ± 148**	7571 ± 680
Mixed	0.0	10	-308 ± 5	474 ± 794	2990 ± 715
Mixed	1.0	10	-316 ± 46	5330 ± 5716	4942 ± 2164
Mixed	5.0	10	-420 ± 50	4704 ± 3413	**8128 ± 1053**

Table 6.3 Dynamics models trained with REx and $\beta = 10$ yield policies with the highest mean evaluation returns for each multi-demonstrator dataset and improve training stability. Policies were learned using dynamics models that had been trained on the **Novice** and **Mixed** datasets. The mean and standard deviation (over three random seeds) of the policy evaluation returns in the final epoch of training are shown. For the **Novice** dataset and rollout length $h = 10$, the number of runs (out of the three attempted) that completed 0.5M steps of policy training are shown in brackets. The highest mean return for each (REx penalty coefficient, dataset) tuple have been bolded.

Rollout Length, h	Starting Location Dataset	REx Penalty Coefficient, β	
		0.0	10.0
5	D4RL Mixed-Replay	-16 ± 140	1173 ± 1004
10	D4RL Mixed-Replay	-333 ± 134	1050 ± 830
5	Random	-15 ± 339	-414 ± 85
10	Random	-	-391 ± 35
5	Softlearning 0.25M	-285 ± 157	-212 ± 50
10	Softlearning 0.25M	-	-310 ± 120

Table 6.4 Dynamics models trained with REx ($\beta = 10$) improved training stability when drawing rollout starting locations from OOD datasets. Policies were learned using dynamics models that had been trained on the **Mixed** dataset. The MOPO penalty coefficient was held constant at $\lambda = 5$; the value for which highest policy evaluation returns were obtained when sampling starting locations from the dynamics model's training dataset. The mean and standard deviation (over three random seeds) of the policy evaluation returns in the final epoch of training are shown. Blanks indicate no runs completed 0.5M steps of policy training.

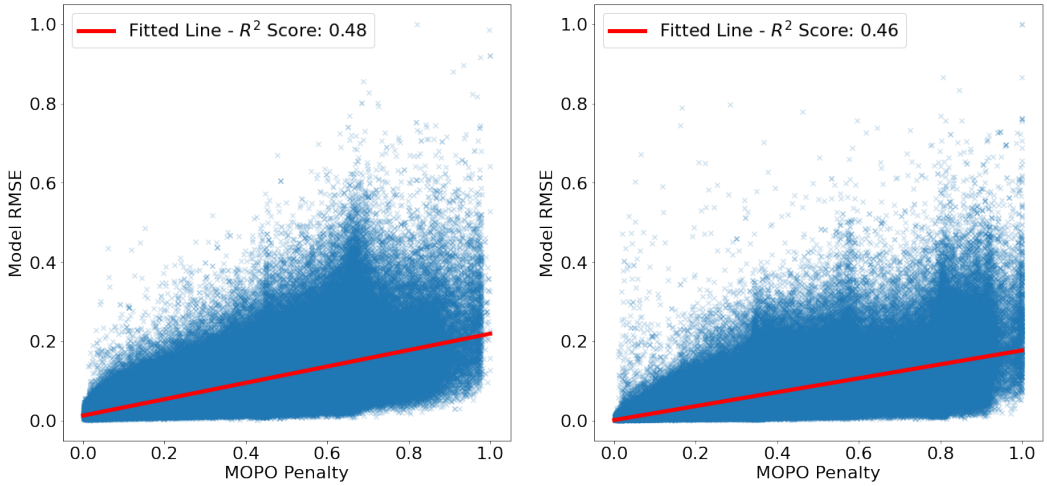
In the previous chapter, we anticipated that the improved domain-generalisation performance of models trained using the **Mixed** dataset would result in more optimal policies being learned than for models trained using the **Novice** dataset. While this is the case for models trained without REx, the reduction in the performance difference across the datasets for $\beta = 10$ is striking, and unexpected given the gap in domain-generalisation performance that still exists (see Table 5.2).

6.2.2 MOPO Penalties

In the previous section we largely neglected the influence of the MOPO penalty coefficient, λ, on the results, however it clearly had an impact. Values $\lambda \in \{0, 1, 5\}$ had been investigated.

For the **Novice** dataset and rollout length $h = 5$, policies with higher returns were typically obtained for $\lambda \in \{0, 1\}$. This suggests that the penalties may have been too conservative, and had overly constrained exploration. Replicating analysis conducted by Yu et al. (2021), Figure 6.4 plots the MOPO penalty against the true model error for transitions sampled from the rollout experience buffer populated during policy training. While the correlation is positive, the metric used by MOPO to quantify uncertainty (maximum predicted standard deviation across the model ensemble – see Section 3.2.3) is clearly an imperfect proxy for model error.

For dynamics models trained without REx on the **Mixed** dataset, policies with the highest returns were again obtained for $\lambda \in \{0, 1\}$. However, for models trained with REx, policies



(a) **Novice** dataset (b) **Mixed** dataset

Fig. 6.4 Replicating analysis conducted by Yu et al. (2021), MOPO penalties and model prediction RMSE values sampled from the model replay pool during policy training were normalised to lie between 0 and 1. Linear regression was performed on the resulting data and the line of best fit plotted. The R^2 score is additionally shown, and highlights the poor calibration between the MOPO penalty and model error.

with significantly higher return were obtained for $\lambda = 5$. This is despite the correlation between the MOPO penalty and model error being no better than for the **Novice** experiments (as shown in Figure 6.4).

Our results therefore appear to indicate that the MOPO penalties can still be beneficial, even when the domain generalisation performance of the dynamics models has been improved. However, the lack of strong correlation between the penalties and true model error make the results sensitive to the choice of penalty coefficient.

6.2.3 Impact of REx on Policy Training

For both datasets, we have observed that dynamics models trained with REx are necessary to learn policies with positive mean return when rollouts of length $h = 10$ are used. Policies not trained with REx models appear to be impacted by the issue of bootstrapped Bellman errors discussed in Section 2.1.4 – the common cause of training instability in offline model-based reinforcement learning algorithms that use Q-learning and actor-critic based algorithms.

To illustrate the problem, Figure 6.5 plots the average predicted Q-values over the rollout transition records collected in the model replay buffer during policy training for the **Mixed** dataset, with $\lambda = 5$ and $\beta = 10$. Note, while negative Q-values would usually indicate the HalfCheetah is primarily running backwards (i.e., has velocity in the opposite direction to

that expected), they are observed here because the MOPO penalties were larger than the original, positive rewards. If training is stable, we expect the policy evaluation returns to increase as the average Q-value increases, which we see for the REx models. However, for models trained without REx, the average Q-values tend toward negative infinity (either immediately, or later in training), while the evaluation returns remain around zero throughout.

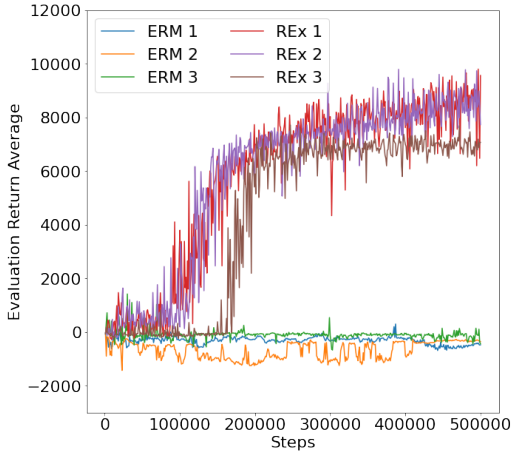
The observed degeneracy in the Q-values is caused when the Q-function is evaluated in OOD states and actions during training. The MOPO algorithm attempts to avoid this by using reward penalties to deter the learning policy from visiting areas of the state-action space where there is high model uncertainty. However we already observed that the penalties are not well correlated with model error (see Figure 6.4). Therefore, poor domain generalisation performance coupled with ineffective reward penalisation are the likely reasons that policy learning using models trained without REx failed. Further, given the poor penalty calibration, we believe the improved OOD performance of the REx dynamics models is the driving force behind the improved stability in policy training.

6.2.4 Policy Evaluation

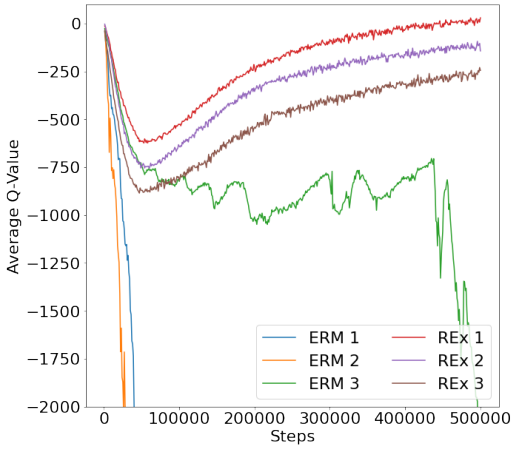
Using the same evaluation techniques we had applied to policies trained against individual demonstrators in Section 6.1.2, we now evaluate the multi-demonstrator policies. We continue to observe the presence of model exploitation – the expected policy returns obtained using the learned dynamics and reward model (which we refer to as the *self-evaluation* returns) exceed those that are achieved in the real environment (e.g., using the real dynamics and reward function). Surprisingly, the use of higher MOPO penalty coefficients was found to be the most important factor for reducing exploitation. When evaluating policies against dynamics and reward models that differed from the one they were trained with, models trained with the **Mixed** dataset consistently yielded expected policy returns within one standard deviation of the real environment returns. While improved model domain-generalisation performance appears to have been key, models trained with REx were not observed to further improve agreement with the real environment returns.

Figure 6.6 shows that exploitation was minimised when using a MOPO penalty coefficient of $\lambda = 5$, and that it was more prevalent for **Mixed** policies. For individual demonstrator policies we had observed positive correlation between the number of demonstrator steps and the self-evaluation returns. The **Mixed** datasets contains transitions from demonstrators trained with a higher number of steps than those used for the **Novice** dataset (see Section 4.5), and so the same observation appears to hold here.

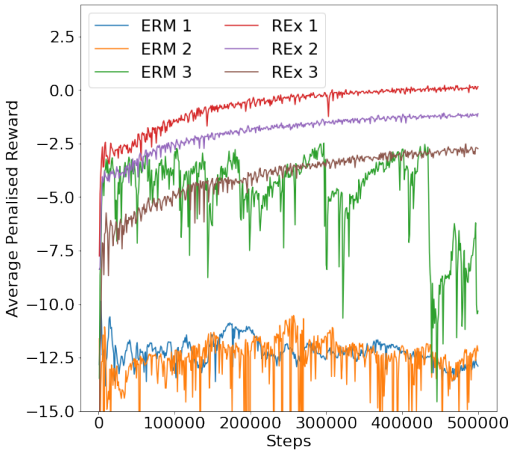
We further evaluated a selection of individual and multi-demonstrator policies against a range of trained models. This included models trained on both individual and multi-



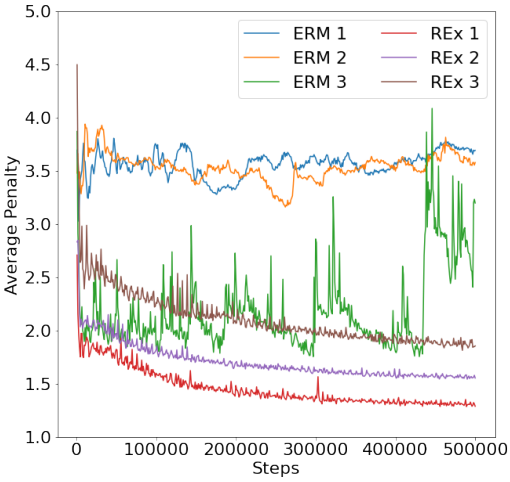
(a) Average Policy Evaluation Return



(b) Average Q-Value

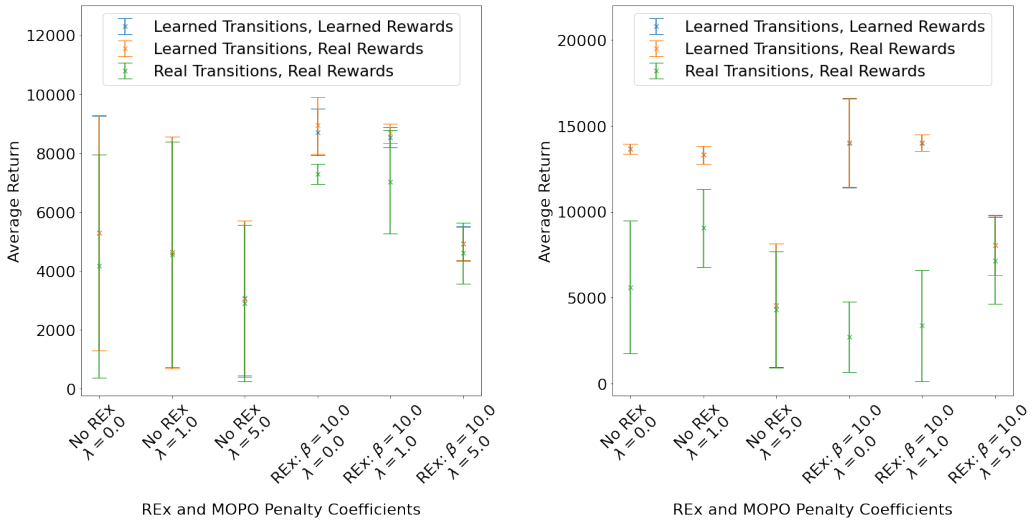


(c) Average Penalised Reward



(d) Average Penalty

Fig. 6.5 Evolution of the average evaluation return, Q-value, penalised reward and MOPO penalty during the training of policies using dynamics models trained with the **Mixed** dataset and a REx penalty coefficient of either $\beta = 0$ (ERM) or $\beta = 10$ (REx). The results from 3 different seeds are shown separately. MOPO penalty coefficient $\lambda = 5$ and rollout length $h = 10$ were used. The average penalties values shown have not been multiplied by the penalty coefficient.



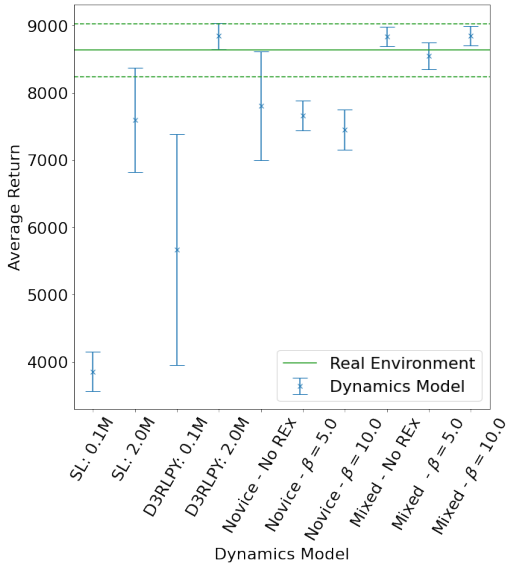
(a) **Novice** Dataset (b) **Mixed** Dataset

Fig. 6.6 The expected policy returns obtained when using the learned dynamics and reward models are higher than in the real environment (i.e., when using the real dynamics and reward function). This indicates the continued presence of model exploitation. Higher MOPO penalty coefficients, λ, appear to reduce exploitation. There is generally a close alignment between the returns using the learned and real reward. Policies were trained using a rollout length $h = 5$. The mean and standard deviation of the returns over 10 episodes is shown.

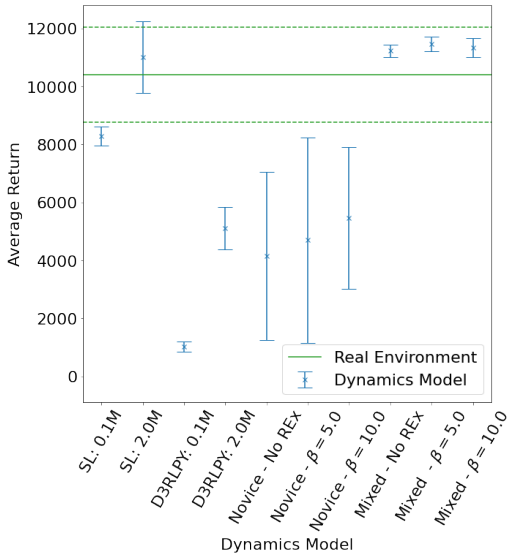
demonstrator datasets, and with varying REx penalty coefficients. The purpose of this was to evaluate how well the expected returns predicted by the learned models matched with those from the real environment, and so the choice of policy should be unimportant. Our expectation was that models with higher domain-generalisation performance, as evaluated in the previous chapter, would produce return estimations in closer agreement with the real environment. As shown in Figure 6.7, we observe that expected evaluation returns within one standard deviation of the true environment returns were consistently obtained when using dynamics models trained on the **Mixed** dataset. This was not typically the case for models trained against individual demonstrators, nor models trained on the **Novice** dataset. Given our observation in the previous chapter that the models trained on the **Novice** dataset were not observed to have higher domain generalisation performance than many of the models trained on individual demonstrators, we were not surprised by this result.

6.3 Conclusions

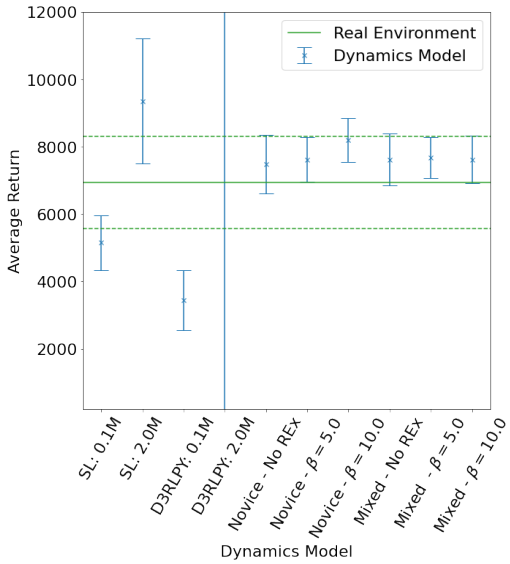
In our experiments, we found that dynamics models trained with REx could be used to learn better performing policies, and improve the stability of policy training when increasing the



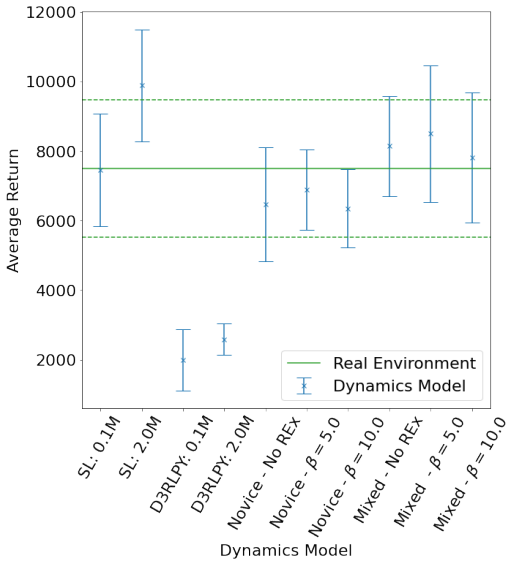
(a) **D3RLPY** policy



(b) **Softlearning** policy



(c) **Novice** policy



(d) **Mixed** policy

Fig. 6.7 Average returns obtained when evaluating policies against a range of learned models and the real environment. The training details of the policies evaluated are as follows: (a) model training dataset: D3RLPY 0.5M Step 1M Transitions; $\lambda = 1$; $h = 5$; (b) model training dataset: Softlearning 0.25M Step 1M Transitions; $\lambda = 1$; $h = 5$; (c) model training dataset: **Novice**; $\lambda = 0$; $h = 5$; (d) model training dataset: **Mixed**, $\lambda = 5$, $h = 5$. The means and standard deviations over 10 runs are shown for each of the learned models, along with the mean and standard deviation over 10 runs in the real environment, which do not depend on the learned model.

length of rollouts or sampling their starting locations from OOD datasets. However, it was not possible to learn *good* policies when rollout starting locations were sampled OOD, and we identified the presence of model exploitation.

For policies trained against individual demonstrators, we did not observe significant correlation between the domain generalisation performance of the dynamics model used and the return of the learned policy. The rollout length h and MOPO penalty coefficient λ should ideally be tuned for each dynamics model, however the variation in rollout starting locations is likely having a significant impact. Given we observed that individual demonstrator policies could not be learned using OOD rollout starting locations, we were unable to correct for this.

We observed that dynamics models trained using a larger number of transitions yielded policies with higher returns, even though these models were shown in the previous chapter to have lower domain generalisation performance. Given that rollout starting locations were sampled from the dynamics model training dataset, we theorise that improved performance in the locality of the training distribution may be the cause of these benefits.

We identified that the correlation between the MOPO penalty and true model error is positive, but very weak. However, for the **Mixed** dataset, a larger penalty was necessary to train policies with higher returns, and we observed larger penalties similarly reduced model exploitation. While we know REx improved robustness to distributional shifts, we don't expect to be able to entirely eliminate OOD errors. It therefore makes sense that combining REx trained models with existing SOTA methods that look to minimise distributional shift by estimating model uncertainty may be beneficial.

As is common across offline model-based RL literature, we assumed we could evaluate our learned policy in the real environment (Cang et al., 2021; Kidambi et al., 2020; Yu et al., 2020). However, in real-world scenarios, we may well have taken an offline approach because naive interaction with the environment is too dangerous or expensive, and so it's unlikely we'd be willing to test our policies online. We should, therefore, consider using *off-policy evaluation (OPE)* methods, which seek to evaluate policies, and perform hyper-parameter tuning, using only the data available to us (Prudencio et al., 2022). Voloshin et al. (2019), Le Paine et al. (2020) and Fu et al. (2021) survey and benchmark existing OPE methods. A subcategory of OPE methods are those which learn a dynamics model and reward function from the data, in the same way as we have done in our work. Therefore, models with improved robustness to distributional shift would potentially be a beneficial contribution to the OPE field. However, in our experiments we have seen that the issue of model exploitation, and the variability in expected policy returns obtained across learned models, persists when using REx. Future work may therefore look to evaluate policies learned offline using other

OPE methods, such as Fitted Q-Evaluation (FQE) (Le et al., 2019) or importance sampling with a learned behaviour policy (Fu et al., 2021; Kostrikov et al., 2020).

Finally, while all policies were trained across 3 random seeds, future works might consider running experiments for a larger number of seeds to evaluate the robustness of the results.

Chapter 7

Conclusions

In this final chapter we draw together our findings and make recommendations for future avenues of work.

7.1 Key Findings

In this work, we took what we believe is a novel approach to reducing the issue of distributional shift that commonly hinders policy training in offline model-based RL methods. Whereas existing SOTA algorithms (Kidambi et al., 2020; Yu et al., 2020) have attempted to *limit* distributional shift, our aim was to improve *robustness* to shifts by applying Risk Extrapolation (REx) (Krueger et al., 2020) to the training of dynamics models.

To this end, we demonstrated in Section 5.3.2 that dynamics models trained with REx showed improved domain generalisation performance, and achieved greater equality of risks across out-of-distribution domains. As Krueger et al. (2020) demonstrated may be necessary, we saw that achieving this greater equality led to increased risks for the domains with the lowest risk. In the case of our **Mixed** multi-demonstrator dataset, as we increased the strength with which we enforced the equality of the training risks (by increasing the penalty coefficient β), we observed that the worst-case OOD risk was reduced at the expense of the average OOD risk.

In Section 6.2 we demonstrated that dynamics models trained with the largest REx penalty coefficient investigated ($\beta = 10$) yielded the policies with the highest average return. Further, these models increased training stability when using longer rollouts and when sampling rollout starting locations from OOD datasets. However, good policies could not always be learned, and model exploitation was identified. Future works should determine whether REx penalty coefficients $\beta > 10$ further aid policy training.

Even though we showed them to be poorly correlated with true model error, we observed in certain experiments that MOPO penalties were beneficial to policy training, and that they reduced model exploitation. Future works could investigate the impact of using dynamics models trained with REx alongside other SOTA offline model-based RL methods, such as MOReL (Kidambi et al., 2020) or COMBO (Yu et al., 2021).

7.2 Future Work

In addition to those already mentioned, we identify the following possible future avenues of investigation.

Demonstrator Diversity

In Chapter 4 we used two implementations of the SAC algorithm to train policies online as proxies for real-word demonstrators. These were then used to generate demonstration datasets. As mentioned in Section 4.6, ideally a larger number of algorithms would have been employed to introduce greater diversity into the datasets. Further, while we only generated two multi-demonstrator datasets, we encourage the evaluation of our methods over a broader collection.

Domain Generalisation Method

While we chose to use REx in our work, as discussed in Section 2.2, there are a significant number of domain-generalisation methods which could have been used. Future works might consider adapting our experiment pipeline to investigate a broader range of these methods. A good starting point might be Group DRO (Sagawa et al., 2019). As discussed in Section 2.2.3, REx can be thought of as performing distributionally robust optimisation (DRO) over an extrapolated set of domains, whereas Group DRO is equivalent to considering convex combinations of the training risks (Krueger et al., 2020). Using Group DRO, we are unlikely to see the same flattening of the risk plane as was observed in Section 5.3.2. However, if the OOD domains with the worst-case risk can be expressed as a mixture of the training domains, then Group DRO may be able to improve worst-case risk without significantly deteriorating the performance in other domains.

Training Pipeline Improvements

Given our work involved learning both dynamics models and policies, the collection of hyperparameters to be tuned was significant. Further, our ultimate goal was to learn a policy

with the highest returns, but, as shown in Section 6.1.1, a number of factors beyond the domain-generalisation performance of the trained dynamics model can influence the policy obtained.

In our work, we focused on evaluating a diverse selection of design choices (dynamics model training time, REx penalty coefficient, etc.) to gain an appreciation of how each impacted offline policy returns. Future works may look to refine the training pipeline by, for example, streamlining the dynamics model selection process.

Gulrajani and Lopez-Paz (2020) propose multiple methods for evaluating domain generalisation algorithms, which could equally be applied to tuning the hyperparameters of dynamics models. We highlight the *leave-one-domain-out cross-validation* approach. Akin to the standard LOO-CV technique, we would train an equal number of models as we have training datasets, leaving one dataset out in each case. Each model would be evaluated on the domain that was held-out, and an average taken across models. The hyperparameters maximising this average would then be used to retrain the model using all training domains. Consistently training models with this method would be beneficial to the further investigation of the correlation between model domain-generalisation performance and the return of offline policies.

References

Argenson, A. and Dulac-Arnold, G. (2020). Model-Based Offline Planning.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. (2019). Invariant Risk Minimization.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Dhruva, T. B., Muldal, A., Heess, N., and Lillicrap, T. (2018). Distributed Distributional Deterministic Policy Gradients. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*.

Beery, S., Van Horn, G., and Perona, P. (2018). Recognition in Terra Incognita. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11220 LNCS:472–489.

Ben-Tal, A., den Hertog, D., De Waegenaere, A., Melenberg, B., and Rennen, G. (2013). Robust Solutions of Optimization Problems Affected by Uncertain Probabilities. *Management Science*, 59(2):341–357.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information science and statistics. Springer, New York.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Openai, W. Z. (2016). OpenAI Gym.

Bühlmann, P. (2018). Invariance, Causality and Robustness.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. (2014). Manifold Gaussian Processes for Regression. *Proceedings of the International Joint Conference on Neural Networks*, 2016-October:3338–3345.

Cang, C., Rajeswaran, A., Abbeel, P., and Laskin, M. (2021). Behavioral Priors and Dynamics Models: Improving Performance and Domain Transfer in Offline RL.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. *Advances in Neural Information Processing Systems*, 2018-Decem:4754–4765.

Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., and Abbeel, P. (2018). Model-Based Reinforcement Learning via Meta-Policy Optimization.

Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2015). Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423.

Duchi, J. C., Hashimoto, T., and Namkoong, H. (2020). Distributionally Robust Losses for Latent Covariate Mixtures.

Dulac-Arnold, G., Mankowitz, D., and Hester, T. (2019). Challenges of Real-World Reinforcement Learning.

El Sallab, A., Abdou, M., Perot, E., and Yogamani, S. (2017). Deep Reinforcement Learning framework for Autonomous Driving. *IS and T International Symposium on Electronic Imaging Science and Technology*, pages 70–76.

Fu, J., Kumar, A., Nachum, O., Brain, G., Google Brain, G. T., and Levine, S. (2020). D4RL: Datasets for Deep Data-Driven Reinforcement Learning.

Fu, J., Norouzi, M., Nachum, O., Tucker, G., Wang, Z., Novikov, A., Yang, M., Zhang, M. R., Chen, Y., Kumar, A., Paduraru, C., Levine, S., and Paine, T. L. (2021). Benchmarks for Deep Off-Policy Evaluation.

Fujimoto, S., Meger, D., and Precup, D. (2018). Off-Policy Deep Reinforcement Learning without Exploration. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:3599–3609.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2016). Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3389–3396.

Gulcehre, C., Wang, Z., Novikov, A., Le Paine, T., Colmenarejo, S. G., Zolna, K., Agarwal, R., Merel, J., Mankowitz, D., Paduraru, C., Dulac-Arnold, G., Li, J., Norouzi, M., Hoffman, M., Heess, N., and de Freitas, N. (2020). RL Unplugged: A Suite of Benchmarks for Offline Reinforcement Learning. *Advances in Neural Information Processing Systems*, 2020-December.

Gulrajani, I. and Lopez-Paz, D. (2020). In Search of Lost Domain Generalization.

Gururangan, S., Swayamdipta, S., Levy, O., Schwartz, R., Bowman, S. R., and Smith, N. A. (2018). Annotation Artifacts in Natural Language Inference Data. *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 2:107–112.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *35th International Conference on Machine Learning, ICML 2018*, 5:2976–2989.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018b). Soft Actor-Critic Algorithms and Applications.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2017). Deep Reinforcement Learning that Matters. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3207–3214.

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. (2021). How to Train Your Robot with Deep Reinforcement Learning; Lessons We've Learned. *International Journal of Robotics Research*, 40(4-5):698–721.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to Trust Your Model: Model-Based Policy Optimization. *Advances in Neural Information Processing Systems*, 32.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. (2020). MOReL : Model-Based Offline Reinforcement Learning. *Advances in Neural Information Processing Systems*, 2020-Decem.

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A., Yogamani, S., and Perez, P. (2020). Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926.

Kostrikov, I., Nachum, O., and Research, G. (2020). Statistical Bootstrapping for Uncertainty Estimation in Off-Policy Evaluation.

Krueger, D., Caballero, E., Jacobsen, J.-H., Zhang, A., Binas, J., Zhang, D., Priol, R. L., and Courville, A. (2020). Out-of-Distribution Generalization via Risk Extrapolation (REx).

Kumar, A., Fu, J., Tucker, G., and Levine, S. (2019). Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. *Advances in Neural Information Processing Systems*, 32.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative Q-Learning for Offline Reinforcement Learning. *Advances in Neural Information Processing Systems*, 2020-December.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-Ensemble Trust-Region Policy Optimization. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*.

Le, H. M., Voloshin, C., and Yue, Y. (2019). Batch Policy Learning under Constraints. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:6589–6600.

Le Paine, T., Paduraru, C., Michi, A., Gulcehre, C., Żołna, K., Novikov, A., Wang, Z., de Freitas, N., and Contributions, E. (2020). Hyperparameter Selection for Offline Reinforcement Learning. pages 2020–2027.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. (2017). RLlib: Abstractions for Distributed Reinforcement Learning. *35th International Conference on Machine Learning, ICML 2018*, 7:4768–4780.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Matsushima, T., Furuta, H., Matsuo, Y., Nachum Google Research, O., and Shane Gu Google Research, S. (2020). Deployment-Efficient Reinforcement Learning via Model-Based Offline Optimization.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning.

Moreno-Torres, J. G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. V., and Herrera, F. (2012). A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your mode's uncertainty? Evaluating predictive uncertainty under dataset shift. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Peters, J., Bühlmann, P., and Meinshausen, N. (2015). Causal inference using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 78(5):947–1012.

Peters, J., Janzing, D., and Schlkopf, B. (2017). *Elements of Causal Inference: Foundations and Learning Algorithms*. The MIT Press.

Prudencio, R. F., Maximo, M. R. O. A., and Colombini, E. L. (2022). A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems.

Rafailov, R., Yu, T., Rajeswaran, A., and Finn, C. (2020). Offline Reinforcement Learning from Images with Latent Space Models.

Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. (2016). EPOpt: Learning Robust Neural Network Policies Using Model Ensembles. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Sagawa, S., Wei Koh, P., Hashimoto Microsoft, T. B., and Liang, P. (2019). Distributionally Robust Neural Networks for Group Shifts: On the Importance of Regularization for Worst-Case Generalization.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Openai, O. K. (2017). Proximal Policy Optimization Algorithms.

Seno, T. and Imai, M. (2021). d3rlpy: An Offline Deep Reinforcement Learning Library.

Shen, Z., Liu, J., He, Y., Zhang, X., Xu, R., Yu, H., and Cui, P. (2021). Towards Out-Of-Distribution Generalization: A Survey.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature 2016 529:7587*, 529(7587):484–489.

Sogabe, T., Malla, D. B., Takayama, S., Shin, S., Sakamoto, K., Yamaguchi, K., Singh, T. P., Sogabe, M., Hirata, T., and Okada, Y. (2018). Smart Grid Optimization by Deep Reinforcement Learning over Discrete and Continuous Action Space. *2018 IEEE 7th World Conference on Photovoltaic Energy Conversion, WCPEC 2018 - A Joint Conference of 45th IEEE PVSC, 28th PVSEC and 34th EU PVSEC*, pages 3794–3796.

Subbaswamy, A., Schulam, P., and Saria, S. (2019). Preventing Failures Due to Dataset Shift: Learning Predictive Models That Transport. In Chaudhuri, K. and Sugiyama, M., editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 3118–3127. PMLR.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, MA, second edition edition.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. *IEEE International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Vapnik, V. (1991). Principles of Risk Minimization for Learning Theory. In Moody, J., Hanson, S., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann.

Voloshin, C., Le, H. M., Jiang, N., and Caltech, Y. Y. (2019). Empirical Study of Off-Policy Policy Evaluation for Reinforcement Learning.

Wang, J., Lan, C., Liu, C., Ouyang, Y., Qin, T., Member, S., Lu, W., Chen, Y., Zeng, W., and Yu, P. S. (2021). Generalizing to Unseen Domains: A Survey on Domain Generalization. *IJCAI International Joint Conference on Artificial Intelligence*, 14(8):4627–4635.

Wu, Y., Zhai, S., Srivastava, N., Susskind, J., Zhang, J., Salakhutdinov, R., and Goh, H. (2021). Uncertainty Weighted Actor-Critic for Offline Reinforcement Learning.

Yu, T., Kumar, A., Rafailov, R., Rajeswaran, A., Levine, S., and Finn, C. (2021). COMBO: Conservative Offline Model-Based Policy Optimization.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J., Levine, S., Finn, C., and Ma, T. (2020). MOPO: Model-based Offline Policy Optimization. *Advances in Neural Information Processing Systems*, 2020-Decem.

Zhao, W., Queralta, J. P., and Westerlund, T. (2020). Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744.

Zhou, K., Liu, Z., Qiao, Y., Xiang, T., and Loy, C. C. (2021). Domain Generalization: A Survey.

Appendix A

Implementation Details

A.1 MOPO Hyperparameters

The hyperparameters used for HalfCheetah MuJoCo environment D4RL datasets in the original MOPO paper (Yu et al., 2020) are provided in Table A.1.

Dataset Type	**Rollout Length,** h	**Penalty Coefficient,** λ
Random	5	0.5
Medium	1	1
Medium-Replay	5	1
Medium-Expert	5	1

Table A.1 Hyperparameters used in the MOPO paper (Yu et al., 2020) for the HalfCheetah MuJoCo environment D4RL datasets.

A.2 D4RL Score Normalisation

Fu et al. (2020) use Equation A.1 to normalise the undiscounted average evaluation returns during the final iteration of policy training to lie roughly between 0 and 100. A normalised score of 0 corresponds to the average returns (over 100 episodes) of an agent taking actions uniformly at random. A score of 100 corresponds to the average returns for a domain-specific expert, which, in the case of Gym-MuJoCo environments, corresponds to a SAC agent (Haarnoja et al., 2018b). For the HalfCheetah environment the random and expert scores are -280.2 and 12135.0 respectively.

$$\text{normalised score} = 100 \cdot \frac{\text{score} - \text{random score}}{\text{expert score} - \text{random score}} \tag{A.1}$$

A.3 D4RL MuJoCo Dataset Descriptions

Descriptions of the MuJoCo environment D4RL datasets are as follows (Fu et al., 2020):

1. **Random:** 1M samples from a randomly initialised policy.
2. **Expert:** 1M samples from a policy trained to completion with SAC.
3. **Medium:** 1M samples from a policy trained to 1/3 the performance of the expert.
4. **Medium-Replay:** the replay buffer of a policy trained to the performance of a medium agent. 101k samples in total.
5. **Medium-Expert:** slightly less than 2M samples with a 50-50 split of medium and expert data.

Appendix B

Additional Experiments

B.1 Individual Demonstrator MBPO Policies

Table B.1 shows the results for training policies on individual demonstrators using rollout length $h = 5.0$ and MOPO penalty coefficient $\lambda = 0$. When the MOPO penalty coefficient is removed, we are left with the MBPO algorithm (Janner et al., 2019).

Demonstrator	Softlearning		D3RLPY	
Steps (million)	0.1M Records	1M Records	0.1M Records	1M Records
0.1	6088 ± 1033	8991 ± 134	5792 ± 488	4759 ± 3797
0.2	-	-	6598 ± 394	7964 ± 114
0.25	218 ± 678	3196 ± 2753	-	-
0.5	-457 ± 65	2802 ± 2039	7601 ± 403	8674 ± 138
1	57 ± 39	3265 ± 1319	-340 ± 38	5363 ± 3466
2	-1044 ± 769	-43 ± 176	2458 ± 3956	6638 ± 1573
3	-279 ± 109	-314 ± 53	-	-

Table B.1 Policies were learned using dynamics models trained on individual Softlearning and D3RLPY demonstrators. The mean and standard deviation of the policy evaluation returns in the final epoch of training over three random seeds is shown. Datasets containing 0.1M and 1M transitions were used, and the first column indicates the number of steps the demonstrator was trained online for. The mean and standard deviation over three random seeds is shown.