PerSim: Data-Efficient Offline Reinforcement Learning with Heterogeneous Agents via Latent Factor Representation

by

Cindy X. Yang

Bachelor of Science in Electrical Engineering and Computer Science,
Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2021

Certified by .....
Devavrat Shah
Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

PerSim: Data-Efficient Offline Reinforcement Learning with Heterogeneous Agents via Latent Factor Representation

by

Cindy X. Yang

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Offline reinforcement learning, where a policy is learned from a fixed dataset of trajectories without further interaction with the environment, is one of the greatest challenges in reinforcement learning. Despite its compelling application to large, real-world datasets, existing RL benchmarks have struggled to perform well in the offline setting. In this thesis, we consider offline RL with heterogeneous agents (i.e. varying state dynamics) under severe data scarcity where only one historical trajectory per agent is observed. Under these conditions, we find that the performance of state-of-the-art offline and model-based RL methods degrade significantly. To tackle this problem, we present PerSim, a method to learn a personalized simulator for each agent leveraging historical data across all agents, prior to learning a policy. We achieve this by positing that the transition dynamics across agents are a latent function of latent factors associated with agents, actions, and units. Subsequently, we theoretically establish that this function is well-approximated by a "low-rank" decomposition of separable agent, state, and action latent functions. This representation suggests a simple, regularized neural network architecture to effectively learn the transition dynamics per agent, even with scarce, offline data. In extensive experiments performed on RL methods and popular benchmark environments from OpenAI Gym and Mujoco, we show that PerSim consistently achieves improved performance, as measured by average reward and prediction error.

Thesis Supervisor: Devavrat Shah
Title: Professor

Acknowledgments

I want to express deep gratitude to my advisor, Devavrat Shah, and lab members Abdullah Alomar, Varkey Alumootil, Anish Agarwal, Dennis Shen, Zhi Xu, for their continued support, guidance, encouragement, patience, and cooperation. I am grateful for the unique opportunity to work with them this past year. This work would not exist without them and I have learned a tremendous amount through it. The dedication and focus that they pour into their work never fail to inspire me, and I can only hope to carry the same spirit in my future work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Reinforcement learning algorithms have achieved impressive results in a wide range of domains including robotics and game-solving, largely in part to the introduction of deep neural networks as a tool for high-capacity function approximation (Mnih et al., 2015; Silver et al., 2017b;a). RL algorithms use an online learning paradigm, in which an agent learns through data collected from continuous interaction with the environment. However, this paradigm has become a primary obstacle to the adoption of RL methods to real-world settings, where online interaction is often impractical, whether due to physical limitations or high costs. As a result, there is a growing body of literature on "offline" reinforcement learning, in which an agent learns from a fixed dataset without further interactions with the environment. Effective offline RL would allow us to leverage the vast amount of existing datasets in many domains such as robotics, healthcare, and recommendation systems. Another relevant challenge is the generalizability of RL methods to different dynamics. Many existing datasets consist of diverse experiences collected under heterogeneous settings, but existing RL methods still struggle to generalize well to heterogeneous environments.

Broadly, this thesis aims to tackle these challenges to real-world RL deployment and improve upon the limitations of existing work on offline RL methods. In order to tackle such a wide problem, we focus on the following setting which we believe

most effectively encapsulates the challenges described into one problem. We consider an offline RL setting where we are provided fixed data from heterogeneous agents, i.e. each agent has different transition dynamics. Further, we limit our focus to the case of *severe data scarcity*, where for each agent, we observe only one historical trajectory produced under an unknown, potentially suboptimal policy. Note that the covariates characterizing the source of heterogeneity across agents are unknown, posing a further challenge. We choose this setup in order to combine offline learning and generalizability into a single problem. We focus on severe data scarcity as it is the most "extreme" and imaginably most difficult case of agent heterogeneity. Being able to use such limited, heterogeneous data to learn a "personalized" policy for each agent would be powerful to real-world applications, yet has received limited attention in literature.

Motivating examples of the described challenges are abundant. Consider a pre-existing clinical dataset of patients (agents). Our goal is to design a personalized treatment plan (policy) for each patient moving forward. Notable challenges include the following: First, each patient only provides a single trajectory of their medical history. Second, each patient is heterogeneous in that s/he may have a varied response for a given treatment under similar medical conditions; further, the underlying reason for this heterogeneous response across patients is likely unknown and thus not recorded in the dataset. Third, in the absence of an accurate personalized "forecasting" model for a patient's medical outcome given a treatment plan, the treatment assigned is likely to be sub-optimal. This is particularly true for complicated medical conditions like T-Cell Lymphoma.

We aim to address the challenges laid out above (offline scarce data, heterogeneity, and sub-optimal policies) so as to develop a personalized "forecasting" model for each patient under any given treatment plan. Doing so will then naturally enable ideal personalized treatment policies for each patient. Tackling a scenario like the one described above in a principled manner is the focus of this work.

1.2 Contributions

The problem that this thesis focuses on is encapsulated in the question: "Can we use fixed, scarce data collected across heterogeneous agents under unknown policies to build personalized policies for each agent?". We believe the answer is "yes", and this thesis lays out a structured framework for doing so. The framework developed, which we refer to as PerSim, provides methodological, theoretical, and algorithmic contributions to offline RL research, and are benchmarked against competitive RL methods through extensive experiments. The main contributions are as follows.

1.2.1 Methodological

We propose the methodological use of personalized simulators. Taking inspiration from the model-based RL literature, our approach is to first build a personalized simulator for each agent. Our novel framework, PerSim, learns personalized simulators which can simulate trajectories under dynamics unique to each agent, using offline data collectively available across heterogeneous agents. Notably, we must learn the personalized simulators without knowledge of the covariates which characterize their heterogeneity. By building a simulator, we can effectively mimic the online RL setting and generate simulated data for each agent. Then, we can apply any online RL method to learn a personalized decision-making policy. In this thesis, we learn personalized policies using online model predictive control (MPC) (Garcia et al., 1989; Camacho and Alba, 2013) for our experiments.

1.2.2 Theoretical

We address the problem of learning across heterogenous agents by positing that transition dynamics across agents have a latent factor representation. We establish theoretically that this latent representation can be well-approximated by a "low-rank" decomposition of separable agent, state, and action latent functions (Theorem 1). This theoretical framework draws inspiration from literature on collaborative filtering for recommendation systems, where low-rank approximation under a latent factor model

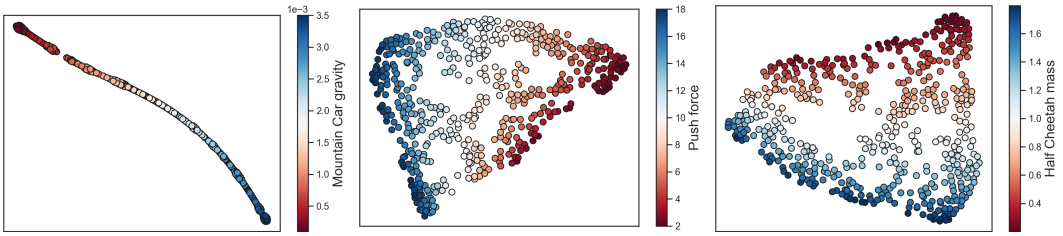
has remained among the most successful methods despite the advent of deep learning networks. As described before, learning a personalized policy for a particular agent when only a single trajectory is observed is a challenging task. This is because each agent likely explores a very small subset of the entire state-action space. By learning from the collective experiences of a multitude of agents, we potentially have access to a much richer subset of the state-action space. However, in order to pool together such data for learning, we must account for the large degree of heterogeneity that potentially exists between agents. As is the case in many real-world settings, we do not have access to the characteristics which produce heterogeneity. Generalizability is a key concern for RL methods, and existing model-based and off-policy methods often struggle to generalize well to unseen environments and suffer from overfitting. Our theoretical contribution is to reduce this problem – accurate model learning from any finite sampling of the state-action space under any policy under varying dynamics – into a matter of estimating a latent, low-rank tensor corresponding to agents, states, and actions. As such, low-rank tensors can provide a useful algorithmic lens to enable model learning with offline data in RL and we hope this work leads to further research studying the relationship between these seemingly disparate fields.

1.2.3 Algorithmic

Given our theoretical results on low-rank representation, we propose a natural neural network architecture that respects the constraints induced by the factorization across agents, states, and actions (Chapter 4.2). It is this principled structure, which accounts for agent heterogeneity and regularizes the model learning, that ensures the success of our approach despite access to only scarce and heterogeneous data.

1.2.4 Experimental

Using standard environments from the OpenAI gym, we extensively benchmark PerSim against two state-of-the-art methods: a model-based online RL method Lee et al. (2020) and a model-free offline RL method Fujimoto et al. (2019). The rationale



(a) MountainCar (b) CartPole (c) HalfCheetah

Figure 1-1: t-SNE visualization of the learned latent factors for the 500 heterogeneous agents. Colors indicate the value of the modified parameters in each environment (e.g., gravity in MountainCar). There is an informative low-dimensional manifold induced by the latent factors, and there is a natural direction on the manifold along which the parameters that characterizes the heterogeneity varies continuously and smoothly.

behind choosing these benchmarks is in Chapter 2. We reach four main conclusions from our experiments:

1. Despite access to only a single trajectory from each agent (and no access to the covariates that drive agent heterogeneity), PerSim produces accurate personalized simulators for each agent.

2. Both benchmarking algorithms perform sub-optimally for the data availability we consider, even on simple baseline environments such as MountainCar and Cartpole, which are traditionally considered to be "solved".

3. PerSim is able to robustly extrapolate outside the policy used to generate the offline dataset, even if the policy is highly sub-optimal (e.g., actions are sampled uniformly at random).

4. As corroboration of our latent factor representation, we find that across all environments, the learned agent-specific latent factors correspond very closely with the latent source of heterogeneity amongst agents; we re-emphasize this is despite PerSim not getting access to the agent covariates.

To visualize the above discussion, see Figure 1-1 for a depiction of the learned latent agent factors and Figure 1-2 for the relative prediction accuracy of the learned model using PerSim versus Lee et al. (2020).

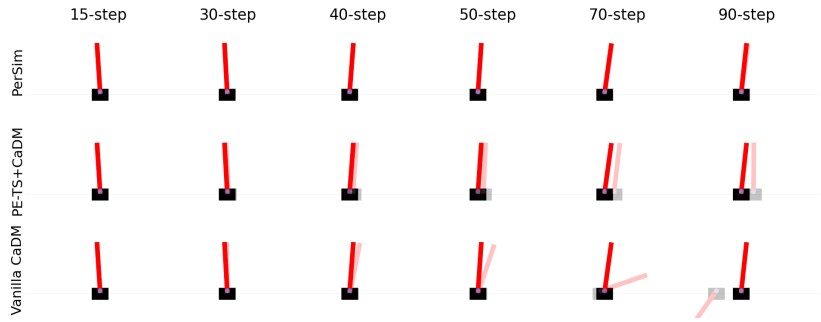


Figure 1-2: Visualization of the prediction accuracy of the various learned models for CartPole. Actual and predicted states are denoted by the opaque and translucent objects, respectively.

1.3 Thesis Outline

Chapter 2 provides background on the fields of work that our work draws inspiration from and highlights related work. In Chapter 3 we formalize the problem statement, and in Chapter 4 we lay out the proposed framework. We start with introducing the latent low-rank factor representation and its theoretical properties, then explain the PerSim algorithm. Chapter 5 describes the setup of a variety of experiments on different benchmarks.

1.4 Additional Note

The work presented in this thesis is the culmination of efforts across multiple students under Devavrat Shah's lab, and contains information largely overlapping with (Agarwal et al., 2021).

Chapter 2

Background and Related Work

In this section, we provide some relevant background highlighting different fields of research and related work which our work draws from. The strengths and drawbacks of existing methods have greatly motivated our proposed framework, and select methods are used as benchmarks in our experiments.

2.1 Offline RL

In offline learning, we use a prerecorded dataset to learn an optimal policy, without further interaction with the environment. Offline RL is challenging for multiple reasons. The most obvious is that we are unable to explore the state-action space beyond existing data. Second is that we are unable to predict counterfactual trajectories correctly. Counterfactual queries ask "what if" questions about what would happen if an agent took a sequence of actions different from the one provided. This question is complicated when (i) the policy used to generate historical data is different from the learned policy and (ii) agent heterogeneity exists through different transition dynamics. We can't apply most supervised learning methods, which generally assume data is independently and identically distributed.

To understand existing methods, we start with a brief overview of online RL methods, which can be partitioned into two broad subfields: model-based and model-free RL, and describe their success in the offline setting and at tackling agent heterogeneity.

2.1.1 Model-based RL

Model-based reinforcement learning methods attempt to learn a model of the transition dynamics of the environment. This model can then be used to either produce data for learning a policy, or produce predictions to optimize the policy with respect to the expected reward. Model-based methods have been popular due to their superior data efficiency compared to its model-free counterparts (Wang et al., 2019; Chua et al., 2018; Clavera et al., 2018; Kurutach et al., 2018; Kaiser et al., 2019; Hafner et al., 2019), which require larger amounts of interaction to learn successful policies due to the lack of a model of the transition dynamics. However, model-based methods struggle to (i) extend to the offline setting and (ii) learn a global model which generalizes well to different dynamics.

Model-based online RL

Although model-based online RL algorithms are highly data efficient, they still struggle to adapt to different agents when heterogeneity exists. Previously, empirical results have shown that vanilla deep RL algorithms generalize better than specialized deep RL algorithms designed to adapt to new environments, suggesting that learning a single model for all environments so far is the most effective method for handling agent heterogeneity. A vanilla model leaves much to be desired, as it is unable to produce personalized policies adapted to the heterogeneity of each agent. Recent online model-based strategies, though, have shown promising results in considering agent heterogeneity (Lee et al., 2020; Nagabandi et al., 2018). (Nagabandi et al., 2018) suggest a model-based meta-learning algorithm where a meta-training process learns a dynamics model prior which can rapidly adjust to the local contexts of new data, resulting in online adaptation. (Lee et al., 2020) propose using a context encode to learn a context latent vector using recent trajectories for each agent, which is then used in the forward dynamics model. We use (Lee et al., 2020) as a benchmark against PerSim, given its strong performance in handling heterogeneous agents.

Model-based Offline RL

Model-based RL in the offline setting is challenging due to the deviation between state visitation distribution of the policy used to generate the data and the policy being evaluated. This problem is often exacerbated as the learned policy increasingly deviates from the data-generating policy as training progresses. This deviation is called distribution shift, and constitutes a major challenge in offline RL. In particular, it can lead to model exploitation, in which the policy inaccurately predicts out-of-distribution successive states that lead to higher rewards than the actual successive states would in the real MDP. Therefore, most model-based offline RL require measures of uncertainty to safeguard against model inaccuracies. Due to these challenges, the vast majority of offline RL methods are model-free and designed for a single setting, i.e., no agent heterogeneity (Fujimoto et al., 2019; Kumar et al., 2019; Laroche et al., 2019; Liu et al., 2020; Wu et al., 2019; Agarwal et al., 2020b; Kumar et al., 2020). Recent works (Kidambi et al., 2020; Yu et al., 2020) have shown that model-free offline RL methods can outperform model-free methods by minimally adapting online model-based methods, but they are restricted to the setting of a single agent with a large number of observations. Extending these model-based offline methods to handle agent heterogeneity is an interesting area of future work, but due to the relatively nascence of this field, we do not use a model-based offline RL method as a benchmark in our experiments.

2.1.2 Model-free RL

Within model-free RL literature, we can focus on offline methods only as there are existing offline model-free RL methods which already achieve strong performance (Fujimoto et al., 2019; Kumar et al., 2019; Laroche et al., 2019; Liu et al., 2020; Wu et al., 2019; Agarwal et al., 2020b; Kumar et al., 2020). Many model-free RL methods are adapted from off-policy RL methods, which can learn from datasets collected using actions taken under a policy other than the current policy. Though off-policy RL methods still require additional interaction during the learning process, they can theoretically

be used for offline RL by simply learning without any further interaction. However, these methods suffer from distribution shift and are limited in practice. To overcome this challenge, offline RL methods design policies that are "close", in an appropriate sense, to the observed behavioural policy in the offline dataset (Kumar et al., 2019; Wu et al., 2019; Fujimoto et al., 2019). They normally do so by directly regularizing the learnt policy (e.g. parameterized via the Q-function) based on the quantified level of uncertainty for a given (state, action)-pair. Among the vast literature in this field, one of the promising ones is batch-constrained Q-learning (Fujimoto et al., 2019), which uses a generative model to produce plausible actions given the dataset. To study how much offline methods suffer if agent heterogeneity is introduced, we compare with (Fujimoto et al., 2019) given its strong performance with offline data. Despite the success of these offline methods, they are designed for settings with no agent heterogeneity and allowing access to numerous trajectories from a single agent. Because they do not build a model of the transition dynamics, they cannot explicitly make use of the rich contextual information potentially encoded in each environment.

2.2 Latent factor model

Latent factor models have remained the state-of-the-art methodology for collaborative filtering in recommendation systems, which deal with the general problem of recommending different items to unique users given each users past history of ratings for a subset of items. The basic assumption is that there exists an latent low-dimensional representation of users and items, and matrix factorization methods can be used for a low-rank approximation of the latent representation. A latent representation is attractive because of its interpretability and reduction of dimensionality. This is especially relevant as the use of high-capacity deep networks in many successful online RL methods are vulnerable to issues of distribution shift in offline settings due to their high dimensionality. As such, low-rank tensors can be a useful tool for enabling model-based offline RL and we hope this work further contributes to research studying the relationship between these two fields.

Chapter 3

Problem Statement

3.1 Problem Setup

In this thesis, we consider a Markov Decision Process (MDP) with N heterogeneous agents, indexed with $n \in [N]$. Formally, our problem is as a tuple $(S, A, P_n, R_n, \gamma_n, \mu_n)$. S and A denote the state and action spaces, respectively, which are common across agents. For every agent n, $P_n(s'|s, a)$ is the unknown transition kernel, $R_n(s, a)$ is the immediate reward received, $\gamma_n \in (0, 1)$ is the discounting factor, and μ_n is the initial state distribution. We consider the offline RL setting where a single trajectory of length T is observed for every agent $n \in [N]$. For each agent n and time step t, let $s_n(t) \in \mathcal{S}$, $a_n(t) \in \mathcal{A}$, and $r_n(t) \in \mathbb{R}$ denote the observed state, action, and reward. The set of all observations is denoted as $\mathcal{D} = (s_n(t), a_n(t), r_n(t)) : n \in [N], t \in [T]$.

3.2 Goals

We have two primary goals:

1. **Build Personalized Simulators**: Use observations $\mathcal{D}$ to build a "personalized" simulator for trajectory prediction. For a given agent n and a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, our goal is to estimate $\mathbb{E}[s'_n|(s_n, a_n) = (s, a)]$, the expected values of the transition function.

2. **Learn Personalized Policies**: To test the efficacy of the personalized simulator, we would like to subsequently use it to learn a good decision-making policy for agent n, denoted as $\pi_n : \mathcal{S} \rightarrow \mathcal{A}$, which takes as input a given state and produces a corresponding action.

Chapter 4

Methods

This section describes our proposed framework. First, we introduce the latent factor model for transition dynamics used in PerSim. Next, we detail the PerSim algorithm for building personalized simulators and learning personalized policies.

4.1 Latent Low-Rank Factor Representation

To address the goals of this work, a key first step is introducing a latent factor model for the transition dynamics. This model guides our method of building the personalized simulator and generating personalized policies. We choose such a model because leveraging latent factors have been successful in recommendation systems for overcoming heterogeneity of users. Such models have also been shown to provide a "universal" representation for multi-dimensional exchangeable arrays, cf. Aldous (1981); Hoover (1979). In fact, our latent model holds for known environments such as MountainCar.

Assume $\mathcal{S} \subseteq \mathbb{R}^D$, i.e., the state is D-dimensional. Let s_{nd} refer to the d-th coordinate of s_n. We posit the transition dynamics (in expectation) obey the following model: for every agent n and state-action pair (s, a),

$$\mathbb{E}[s'_{nd} | (s_n, a_n) = (s, a)] = f_d(\theta_n, \rho_s, \omega_a), \tag{4.1}$$

where s'_{nd} denotes the d-th state coordinate after taking action a. Here, $\theta_n \in \mathbb{R}^{d_1}$, $\rho_s \in \mathbb{R}^{d_2}$, $\omega_a \in \mathbb{R}^{d_3}$ for some $d_1, d_2, d_3 \geq 1$ are latent feature vectors capturing relevant information specific to the agent, state, and action; $f_d : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \times \mathbb{R}^{d_3} \to \mathbb{R}$ is a latent function capturing the model relationship between these latent feature vectors. We assume f_d is L-Lipschitz and the latent features are bounded.

Assumption 1. Suppose $\theta_n \in [0,1]^{d_1}, \rho_s \in [0,1]^{d_2}, \omega_a \in [0,1]^{d_3}$, and f_d is L-Lipschitz with respect to its arguments, i.e., $|f_d(\theta_{n'}, \rho_{s'}, \omega_{a'}) - f_d(\theta_n, \rho_s, \omega_a)| \leq L(\|\theta_{n'} - \theta_n\|_2 + \|\rho_{s'} - \rho_s\|_2 + \|\omega_{a'} - \omega_a\|_2)$.

For notational convenience, let $\tilde{f}_d : [N] \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be such that $\tilde{f}_d(n, s, a) = \mathbb{E}[s'_{nd} | (s_n, a_n) = (s, a)]$.

Theorem 1. Suppose Assumption 1 holds and without loss of generality, let $d_1, d_3 \leq d_2$. Then for all $d \in [D]$ and any $\delta > 0$, there exists $h_d : [N] \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, such that $h_d(n, s, a) = \sum_{\ell=1}^{r} u_\ell(n) v_\ell(s, d) w_\ell(a)$ with $r \leq C\delta^{-(d_1+d_3)}$ and $\|h_d - \tilde{f}_d\|_\infty \leq 2L\delta$, where C is an absolute constant.

Theorem 1 suggests that under the model in (4.1), the transition dynamics are well approximated by a low-rank order-three functional tensor representation. In fact, as we show below, for a classical non-linear dynamical system, it is exact.

An example. We show the MountainCar transition dynamics Brockman et al. (2016) exactly satisfies this low-rank tensor representation. In MountainCar, the state $s_n = [s_{n1}, s_{n2}]$ consists of car (agent) n's position and velocity, i.e., $\mathcal{S} \subseteq \mathbb{R}^2$; the action a_n is a scalar that represents the applied acceleration, i.e., $\mathcal{A} \subseteq \mathbb{R}$. For car n, parameterized by gravity g_n, the (deterministic) transition dynamics given action a_n are

$$\begin{aligned} s'_{n1} &= s_{n1} + s_{n2} - \frac{g_n \cos(3 s_{n1})}{2} + \frac{a_n}{2}, \\ s'_{n2} &= s_{n2} - g_n \cos(3 s_{n1}) + a_n. \end{aligned}$$

Proposition 1. In MountainCar, $r = 3$.

Model Learning & Tensor Estimation. Consider any finite sampling of the states $\tilde{\mathcal{S}} \subset \mathcal{S}$ and actions $\tilde{\mathcal{A}} \subset \mathcal{A}$. Let $\mathcal{X} = [X_{nsad}] \in \mathbb{R}^{N \times |\tilde{\mathcal{S}}| \times |\tilde{\mathcal{A}}| \times D}$ be the order-four tensor, where $X_{nsad} = f_d(\theta_n, \rho_s, \omega_a)$. Hence, to learn the model of transition dynamics for all the agents over $\tilde{\mathcal{S}}$, $\tilde{\mathcal{A}}$, it is sufficient to estimate the tensor, $\mathcal{X}$, from observed data. The offline data collected for a given policy induces a corresponding observation pattern of this tensor. Whether the complete tensor is recoverable, is determined by this induced sparsity pattern and the rank of $\mathcal{X}$. Notably, Theorem 1 suggests that $\mathcal{X}$ is low-rank under mild regularity conditions. Therefore, the question of model identification, i.e., completing the tensor $\mathcal{X}$, boils down to conditions on the offline data in terms of the observation pattern that it induces in the tensor. In the existing tensor estimation literature, there are few sparsity patterns for which the underlying tensor can be provably recovered, provided $\mathcal{X}$ has a low-rank structure. They include: (i) each entry of the tensor is observed independently at random with sufficiently high-probability Barak and Moitra (2016); Montanari and Sun (2018); Shah and Yu (2019); (ii) the entries that are observed are *block-structured* Agarwal et al. (2020a); Shah et al. (2020). However, the most general set of conditions on the sparsity pattern under which the underlying tensor can be faithfully estimated, i.e., the model is identified for our setup, remains an important and active area of research. In addition, when the state and action spaces are not finite (e.g., MountainCar), we are interested in estimating a low-rank tensor function, which is generally not considered in the literature.

4.1.1 Proof of Theorem 1

Proof. We will construct a low-rank order-four tensor $\mathcal{T} \in \mathbb{R}^{N \times |\mathcal{S}| \times |\mathcal{A}| \times D}$ by partitioning the latent parameter spaces associated with agents, states, actions, and the d-th coordinate of the expected next state. As we will see, it would be helpful to think of $\mathcal{T}$ as a composition of D order-three tensors, denoted as $\mathcal{T}^{(d)} \in \mathbb{R}^{N \times |\mathcal{S}| \times |\mathcal{A}|}$, corresponding to each coordinate d. In doing so, we will demonstrate that each frontal slice of $\mathcal{T}^{(d)}$ is a low-rank matrix and only a subset of the $|\mathcal{A}|$ frontal slices of $\mathcal{T}^{(d)}$ are distinct. Similarly, we will prove that a subset of the D order-three tensors $\mathcal{T}^{(d)}$ will be distinct

as well. Together, these observations establish the low-rank property of $\mathcal{T}$. Finally, we will complete the proof by showing that $\mathcal{X}$ is entry-wise arbitrarily close to $\mathcal{T}$.

Partitioning the latent spaces to construct $\mathcal{T}$. Fix some $\delta_1, \delta_3, \delta_4 > 0$. Since the latent row parameters θ_n come from a compact space $[0,1]^{d_1}$, we can construct a finite covering or partitioning $P_1(\delta_1) \subset [0,1]^{d_1}$ such that for any $\theta_n \in [0,1]^{d_1}$, there exists a $\theta_{n'} \in P_1(\delta_1)$ satisfying $\|\theta_n - \theta_{n'}\|_2 \leq \delta_1$. By the same argument, we can construct a partitioning $P_3(\delta_3) \subset [0,1]^{d_3}$ such that $\|\omega_a - \omega_{a'}\|_2 \leq \delta_3$ for any $\omega_a \in [0,1]^{d_3}$ and some $\omega_{a'} \in P_3(\delta_3)$; similarly, $P_4(\delta_4) \subset [0,1]^{d_4}$ such that $\|\alpha_d - \alpha_{d'}\|_2 \leq \delta_4$ for any $\alpha_d \in [0,1]^{d_4}$ and some $\alpha_{d'} \in P_4(\delta_4)$.

For each θ_n, let $p_1(\theta_n)$ denote the unique element in $P_1(\delta_1)$ that is closest to θ_n. At the same time, we define $p_3(\omega_a)$ and $p_4(\alpha_d)$ as the corresponding elements in $P_3(\delta_3)$ and $P_4(\delta_4)$ that are closest to ω_a and α_d, respectively. We now construct our tensor $\mathcal{T} = [T_{nsad}]$ by defining its (n,s,a,d)-th entry as $T_{nsad} = f(p_1(\theta_n), \rho_s, p_3(\omega_a), p_4(\alpha_d))$.

Establishing the low-rank property of $\mathcal{T}$. Let us fix a coordinate d to consider $\mathcal{T}^{(d)} = [T_{nsad}]_{n\in[N],s\in[\mathcal{S}],a\in[\mathcal{A}]} \in \mathbb{R}^{N\times|\mathcal{S}|\times|\mathcal{A}|}$, and fix a frontal slice a of $\mathcal{T}^{(d)}$. Now, consider any two rows of $\mathcal{T}^{(d)}_{\cdot,\cdot,a}$, say n and n'. If $p_1(\theta_n) = p_1(\theta_{n'})$, then rows n and n' of $\mathcal{T}^{(d)}_{\cdot,\cdot,a}$ are identical. Hence, there at most $|P_1(\delta_1)|$ distinct rows in $\mathcal{T}^{(d)}_{\cdot,\cdot,a}$, and thus $\text{rank}(\mathcal{T}^{(d)}_{\cdot,\cdot,a}) \leq |P_1(\delta_1)|$. In words, each frontal slice of $\mathcal{T}^{(d)}$ is a low-rank matrix with its rank bounded above by $|P_1(\delta_1)|$. Next, consider any two frontal slices a and a' of $\mathcal{T}^{(d)}$. If $p_3(\omega_a) = p_3(\omega_{a'})$, then for all (n,s), we have $T_{nsad} = f(p_1(\theta_n), \rho_s, p_3(\omega_a), p_4(\alpha_d)) = f(p_1(\theta_n), \rho_s, p_3(\omega_{a'}), p_4(\alpha_d)) = T_{nsa'd}$. In words, the a-th frontal slice of $\mathcal{T}^{(d)}$ is equivalent to the a'-th frontal slice of $\mathcal{T}^{(d)}$. Hence, $\mathcal{T}^{(d)}$ has at most $|P_3(\delta_3)|$ distinct frontal slices.

Following the arguments above, consider any two coordinates d and d'. If $p_4(\alpha_d) = p_4(\alpha_{d'})$, then $T_{nsad} = f(p_1(\theta_n), \rho_s, p_3(\omega_a), p_4(\alpha_d)) = f(p_1(\theta_n), \rho_s, p_3(\omega_a), , p_4(\alpha_{d'})) = T_{nsad'}$. This establishes that $\mathcal{T}$ can be decomposed into at most $|P_4(\delta_4)|$ distinct order-three tensors. To recap, we have established that all of the frontal slices $\mathcal{T}^{(d)}_{\cdot,\cdot,a}$ of $\mathcal{T}^{(d)}$ are low-rank matrices, and only a subset of the frontal slices of $\mathcal{T}^{(d)}$ are distinct. We have also proven that only a subset of the order-three tensors $\mathcal{T}^{(d)}$ are distinct. Therefore, it follows that the canonical polyadic (CP) rank of $\mathcal{T}$ is bounded by the

product of the maximum matrix rank of any slice of $\mathcal{T}^{(d)}$ with the number of distinct slices in $\mathcal{T}^{(d)}$ and the number of distinct order-three tensors in $\mathcal{T}$. More specifically, $\text{rank}(\mathcal{T}) \leq |P_1(\delta_1)| \cdot |P_3(\delta_3)| \cdot |P_4(\delta_4)|$.

$\mathcal{X}$ is well approximated by $\mathcal{T}$. Here, we bound the maximum difference of any entry in $\mathcal{X}$ from $\mathcal{T}$. Using the Lipschitz property of f (Assumption 1), we obtain for any (n, s, a, d),

$$\begin{aligned}
|X_{nsad} - T_{nsad}| &= |f(\theta_n, \rho_s, \omega_a, \alpha_d) - f(p_1(\theta_n), \rho_s, p_3(\omega_a), p_4(\alpha_d))| \\
&\leq L \cdot (\|\theta_n - p_1(\theta_n)\|_2 + \|\omega_a - p_3(\omega_a)\|_2 + \|\alpha_d - p_4(\alpha_d)\|_2) \\
&\leq L \cdot (\delta_1 + \delta_3 + \delta_4).
\end{aligned}$$

This proves $\mathcal{X}$ is entry-wise arbitrarily close to $\mathcal{T}$.

Concluding the proof. By the Lipschitz property of f and the compactness of the latent spaces, it follows that $|P_1(\delta_1)| \leq C\delta_1^{-d_1}, |P_3(\delta_3)| \leq C\delta_3^{-d_3}, |P_1(\delta_4)| \leq C\delta_4^{-d_4}$, where C is an absolute constant. As such, we can bound the tensor rank $r = \text{rank}(\mathcal{T})$ as $r \leq C\delta_1^{-d_1}\delta_3^{-d_3}\delta_4^{-d_4}$. Importantly, r does not depend on the dimensions of $\mathcal{T}$. Further, by definition of CP-rank, this allows us to write any entry of $\mathcal{T}$ as

$$T_{nsad} = \sum_{\ell=1}^{r} u_{n\ell} v_{s\ell} w_{a\ell} q_{d\ell} = \sum_{\ell=1}^{r} u_{n\ell} v_{s\ell}^{(d)} w_{a\ell},$$

where $v_{s\ell}^{(d)} = v_{s\ell} q_{d\ell}$. Consequently, for any coordinate d, we can write

$$\mathcal{T}^{(d)} = \sum_{\ell=1}^{r} u_\ell \otimes v_\ell^{(d)} \otimes w_\ell,$$

where $u_\ell \in \mathbb{R}^N$, $v_\ell^{(d)} \in \mathbb{R}^{|\mathcal{S}|}$, and $w_\ell \in \mathbb{R}^{|\mathcal{A}|}$. Observing that $\|\mathcal{X}^{(d)} - \mathcal{T}^{(d)}\|_{\max} \leq \|\mathcal{X} - \mathcal{T}\|_{\max}$ and $\text{rank}(\mathcal{T}^{(d)}) \leq \text{rank}(\mathcal{T})$, and setting $\delta = \delta_1 = \delta_3 = \delta_4$ completes the proof.

□

4.1.2 Proof of Proposition 1

Proof. To show that $\text{rank}(\mathcal{X}^{(d)}) = 3$ for $d \in \{1, 2\}$, it suffices to find $u_{n\cdot}, v_{s\cdot}^{(1)}, v_{s\cdot}^{(2)}, w_{a\cdot} \in \mathbb{R}^3$ such that $X_{nsad} = \sum_{\ell=1}^{r} u_{n\ell} v_{s\ell}^{(d)} w_{a\ell}$ for any $n \in [N]$, $s = [s_1, s_2] \in \mathcal{S}$, and $a \in \mathcal{A}$. Further, we require that $u_{n\cdot}$ can only depend on agent n, i.e., not on the action or state. Analogously, $v_{s\cdot}^{(1)}, v_{s\cdot}^{(2)}$ can only depend on the state, and $w_{a\cdot}$ can only depend the action. Towards this, consider the following factors:

$$u_{n\cdot} = \begin{bmatrix} 1 & g_n & 1 \end{bmatrix}, w_{a\cdot} = \begin{bmatrix} 1 & 1 & a \end{bmatrix},$$
$$v_{s\cdot}^{(1)} = \begin{bmatrix} s_1 + s_2 & -\frac{\cos(3s_1)}{2} & \frac{1}{2} \end{bmatrix}, v_{s\cdot}^{(2)} = \begin{bmatrix} s_2 & -\cos(3s_1) & 1 \end{bmatrix}.$$

Recalling that $X_{nsa1} = s_1 + \ s_2 - \frac{g_n \cos(3s_1)}{2} + \frac{a}{2}$ and $X_{nsa2} = s_2 - g_n \cos(3s_1) + a$ completes the proof. □

4.2 PerSim Algorithm

The Persim algorithm can be broken into two steps: (i) building a personalized simulator for each agent n given offline observations $\mathcal{D}$ as described in Chapter 3, and (ii) learning a personalized decision-making policy using MPC.

4.2.1 Building a "personalized" simulator

Theorem 1 suggests that the transition dynamics can represented as a low-rank tensor function with latent functions associated with the agents, states, and actions. This can be designed under a simple, regularized neural network architecture: we use three separate neural networks to learn the agent, state, and action latent functions, i.e., we remove any edges between these three neural networks. See Figure 1 for a visual depiction of the proposed architecture. Specifically, to estimate the next state for a given agent n and state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, we learn the following functions:

1. An **agent encoder** $g_u : [N] \to \mathbb{R}^r$, parameterized by ψ, which estimates the latent function associated with an agent, i.e., $\widehat{u}(n) = g_u(n; \psi)$.

2. A **state encoder** $g_v : \mathcal{S} \to \mathbb{R}^{D \times r}$ parameterized by ϕ, which estimates D latent functions, where each vector is associated with the corresponding state coordinate, i.e., $\widehat{v}(s) := (\widehat{v}(s,1),\ \ldots,\ \widehat{v}(s,D))^T = g_v(s;\phi)$.

3. An **action encoder** $g_w : \mathcal{A} \to \mathbb{R}^r$ parameterized by θ, which estimates the latent function associated with the action, i.e., $\widehat{w}(a) = g_w(a;\theta)$.

Then, our estimate of the expected the d-th coordinate of the next state is given by

$$\widehat{\mathbb{E}}[s'_{nd}|(s_n, a_n) = (s,a)] = \sum_{\ell=1}^{r} \widehat{u}_\ell(n)\widehat{v}_\ell(s,d)\widehat{w}_\ell(a).$$

We optimize our agent, state, and action encoders by minimizing the squared loss:

$$\mathcal{L}(s,a,n,s';\psi,\phi,\theta) = \sum_{d=1}^{D}\left(s'_{nd} - \sum_{\ell=1}^{r}\widehat{u}_\ell(n)\widehat{v}_\ell(s,d)\widehat{w}_\ell(a)\right)^2.$$

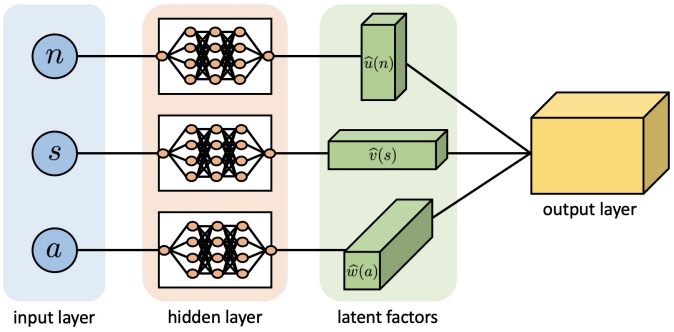


Figure 4-1: Neural Network Architecture

4.2.2 Learning a "personalized" decision-making policy

We use MPC to select the next action, as is done in Lee et al. (2020). Specifically, we sample C candidate action sequences of length h, which we denote as $\{a_1^{(i)}, \ldots, a_h^{(i)}\}_{i=1}^{C}$. The actions are sampled using the cross entropy method in environments with continuous action spaces and the random shooting method in environments with discrete action space Camacho and Alba (2013); Botev et al. (2013).

Since the offline data may not span the entire state-action space, planning using a learned simulator without any regularization may result in "model exploitation" Levine et al. (2020). To overcome this issue, we gauge the model uncertainty, as is common in the literature, as follows. We train an ensemble of M simulators $\{g_u^{(m)}, g_s^{(m)}, g_a^{(m)}\}_{m=1}^{M}$. Then, for $i \in [C]$, we evaluate the average reward of performing the i-th action sequence, which we denote by $r^{(i)}$, using the estimates across the M simulators. Specifically,

$$r^{(i)} = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{h} R\Big(\widehat{s}_t^{(m,i)}, a_t^{(i)}\Big),$$

where $\widehat{s}_t^{(m,i)}$ is the predicted trajectory according to the m-th simulator and the sequence of actions $\{a_1^{(i)}, \ldots, a_h^{(i)}\}$, and R is the reward function (which we assume is known, as is done in prior works Lee et al. (2020); Kidambi et al. (2020)). Finally, we choose the first element from the sequence of actions with the best *average* reward, i.e., the sequence $\{a_1^{(i^*)}, \ldots, a_h^{(i^*)}\}$, where $i^* = \arg\max_{i \in [C]} \; r^{(i)}$.

In our experiments, we find that the simple technique of using the state difference instead of the raw state improves performance. For a detailed description of the algorithm, including the pseudo-code, please refer to Appendix A.

Chapter 5

Experiments and Discussion

Through a systematic collection of experiments on a variety of benchmark environments, we demonstrate that PerSim consistently outperforms state-of-the-art model-based and offline model-free RL algorithms, in terms of prediction error and reward, for the data regime we consider. First we describe the setup and results of the main experiments comparing PerSim's performance against other benchmarks. Then we explain various additional experiments performed to better understand PerSim's sensitivity to factors like data availability.

5.1 Experiment Setup

We evaluate PerSim on benchmark environments from the OpenAI Gym Brockman et al. (2016). A detailed description of each environment can be found in Appendix A.3. For conciseness, we focus on MountainCar, CartPole, and HalfCheetah in this section. However, the results in these environments are indicative of the overall conclusions.

5.1.1 Heterogeneous Agents

We introduce agent heterogeneity across the various environments by modifying the covariates that characterize the transition dynamics of an agent. This is in line with what has been done in the literature Lee et al. (2020) to study algorithmic robustness

to agent heterogeneity. The range of parameters we consider for each of the three environments is given in Table 5.1; for example, we create heterogeneous agents in MountainCar by varying the gravity parameter of a car (agent) within the interval $[0.0001, 0.0035]$. See Appendix A.3 for a detailed description of how we vary agent covariates in each environment.

For each environment, we uniformly sample 500 covariates (i.e., heterogeneous agents) from the parameter ranges displayed in Table 5.1. We then select five of the 500 agents to be our "test" agents, i.e., these are the agents for which we report the prediction error and eventual reward for the various RL algorithms. We select one of the five covariates to be the default covariate parameter in an environment; the other four are selected so as to cover the "extremes" of the parameter range. Below, we show results for three test agents in this section; the results for the remaining two agents can be found in Appendix B. The conclusions we draw from our experiments continue to hold over all test agents we evaluate on.

Table 5.1: Environment parameters used for experiments.

Environment	Parameter range	Test agents	Policy training agents
MountainCar	gravity ∈ [0.0001, 0.0035]	{0.0001, 0.0005, 0.0010, 0.0025, 0.0035}	{0.0003, 0.00075, 0.00175, 0.0025, 0.0030}
CartPole	length ∈[0.15, 0.85] force ∈ [2.0, 18]	{(2.0,0.5), (10.0,0.5), (18.0, 0.5), (10.0,0.85), (10.0,0.15)}	{(6.0,0.5), (14.0,0.5), (10.0,0.5), (10.0,0.675), (10.0, 0.325)}
HalfCheetah	relative mass ∈ [0.2,1.8] relative damping ∈ [0.2,1.8]	{(0.3,1.7), (1.7,0.3), (0.3, 0.3), (1.7,1.7), (1.0,1.0)}	{(0.6,1.4), (1.4,0.6), (0.6, 0.6), (1.4,1.4)}

5.1.2 Offline Data from Sub-optimal Policies

For each environment, we create four offline datasets of 500 trajectories (one per agent) to study how robust the various RL algorithms are to the optimality of the sampling policy used to generate the historical trajectories. The four types of policies used are:

1. *Pure policy*: For each agent, actions are sampled according to a fixed policy that has been trained to achieve "good" performance. Specifically, for each environment, we first train a policy in an online fashion for the five agents shown in Table 5.1. We pick the training agents to be uniformly spread throughout

the training range to ensure reasonable performance for all agents. See Appendix A.3 for details about the average reward achieved across all agents using this procedure. For MountainCar and CartPole, we use DQN (Mnih et al., 2015) to train the sampling policy; for HalfCheetah, we use TD3 (Fujimoto et al., 2018). The policies are trained to achieve rewards of approximately -200, 120, and 3000 for MountainCar, CartPole, and HalfCheetah, respectively. Then for each of the 500 agents, we sample one trajectory using the policy trained on the training agent with the closest parameter value.

2. *Random*: Actions are selected uniformly at random.

3. *Pure-ε-20*: With probability 0.2, actions are selected uniformly at random; otherwise, selected via the pure policy.

4. *Pure-ε-40*: With probability 0.4, actions are selected uniformly at random; otherwise, selected via the pure policy.

The "pure policy" dataset has relatively optimal transition dynamics for each agent compared to the other three policies. It's easy to imagine that a "pure policy" is the ideal scenario when we only have access to limited offline data. As a result, traditional offline RL algorithms learn policies which tend to imitate the policy used to sample the observed offline trajectories. However, such an ideal sampling procedure is hardly met in practice; real-world data often contains at least some amount of "trial and error" in terms of how actions are chosen. We model this by sampling a fraction of the trajectory at random. We find that PerSim performs well across all datasets, and an element of randomness in the policy used to generate data can actually improve performance due to increased richness of the dataset.

5.1.3 Benchmarking Algorithms

We compare with two state-of-the-art RL algorithms, one from the online model-based literature and the other from the offline model-free literature. The rationale for

choosing these are explained in Chapter 2. In Appendix A, we give implementation details.

Vanilla CaDM + PE-TS CaDM (Lee et al., 2020). CaDM tackles heterogeneity by learning a context vector using the recent trajectory of a given agent, with a common context encoder across all agents. It is designed for both the online and offline framework. Since CaDM is a model-based method, we compare against two CaDM variants discussed in (Lee et al., 2020) with respect to both model prediction error and eventual reward. Vanilla CaDM evaluates how well CaDM can learn the unique transition dynamics for each heterogeneous agent using only limited offline data, while PE-TS CaDM underscores the subsequent impact of the model fitting in learning a policy via MPC.

BCQ-P + BCQ-A (Fujimoto et al., 2019). Batch-constrained Q-learning (BCQ) is an offline model-free RL method that has been shown to exhibit excellent performance in the standard offline setting, i.e., numerous trajectories collected of a single agent. Two variants are considered: (i) BCQ-Population (or BCQ-P), where a *single* policy is trained using data from all available (heterogeneous) agents, i.e., all 500 observed trajectories; (ii) BCQ-Agent (or BCQ-A), where a separate policy is learned for each of the test agents using just the single observed trajectory associated with that test agent. We compare PerSim against both BCQ variants with respect to the eventual reward in order to study the effect that data scarcity and agent heterogeneity can have on standard offline RL methods.

5.2 Evaluation

We use two metrics of performance for PerSim and benchmark methods: prediction error is used to evaluate the model learning process, while average reward is used to evaluate the learned policy.

5.2.1 Model Learning: Prediction Error

PerSim Accurately Learns Personalized Simulators. PerSim outperforms both Vanilla CaDM and PE-TS CaDM for most test agents consistently across environments. RMSE for PerSim is notably lower by orders of magnitude in MountainCar and CartPole. In these two environments, R^2 for PerSim is nearly one (i.e., the maximum achievable) across most agents, while for the two CaDM variants it is notably lower. In a number of experiments, the two CaDM variants have negative R^2 values, i.e., their predictions are worse than simply predicting the average true state across the test trajectory. For HalfCheetah, which has a relatively high-dimensional state space (it is 18-dimensional), PerSim continues to deliver reasonably good predictions for each agent despite the challenging data availability. Though PerSim still outperforms CaDM in HalfCheetah, we note CaDM's performance is more comparable in this environment.

For a more visual representation of PerSim's prediction accuracy, refer back to Figure 1-2 in Section 1, which shows the predicted and actual state for different horizon lengths in CartPole; there, we see that our learned personalized simulator consistently produces state predictions that closely match the actual state observed from the true environment. Additional visualizations of the learned simulator across environments can be found in Appendix B.

Rethinking Model-based RL for Our Setting. Although it can be tempting to directly utilize existing model-based RL methods for an offline setting, our experiment results indicate that existing model-based RL methods cannot effectively learn a model of the transition dynamics simultaneously across *all* heterogeneous agents (e.g., CaDM sometimes has negative R^2 values) if only given access to sparse offline data. It has been exhibited (Janner et al., 2019; Yu et al., 2020; Levine et al., 2020) that certain model-based methods that were originally targeted for the online setting can potentially still deliver reasonable performance with offline data and minimal algorithmic change. Our experiments offer evidence that model-based RL methods, even those which are optimized to work with heterogeneous agents, do not provide a

uniformly good "plug-in" solution for the particular data availability we consider. In contrast, our latent factor approach consistently learns a reasonably good model of the dynamics across all agents given access to the exact same offline dataset.

Latent Factors Capture Agent Heterogeneity. The poor performance of standard model-based RL methods for the data availability setting we consider emphasizes the need for new principled approaches. Ours is one such approach, where we posit a low-rank latent factor representation of the agents, states, and actions. To further validate our approach, we visualize the learned latent agent factors associated with the 500 heterogeneous agents in each environment in Figure 1-1 of Chapter 1. We see that the latent factors correspond closely with the heterogeneity across agents, a reassuring sign that a latent factor representation can closely represent true covariates for each agent.

5.2.2 Policy Performance: Average Reward

In this section, we evaluate the average reward achieved by PerSim compared to several state-of-the-art model-based and model-free offline RL methods. For the model-based methods, we follow Lee et al. (2020) in utilizing MPC to make policy decisions on top of the learned model. We include an additional baseline, "True env + MPC", where we apply MPC on top of the actual ground-truth environment; this allows us to quantify the difference in reward when using MPC with the actual environment versus the learned simulators. For the two model-free BCQ methods, BCQ-P and BCQ-A, we can directly apply the policy resulted from the learned Q-value.

We evaluate the performance of each method using the average reward over 20 episodes for the model-based methods and the average reward over 100 episodes for the model-free methods. We repeat each experiment five times and report the mean and standard deviation. In Tables 5.2a, 5.2b, and 5.2c, we present the results for MountainCar, CartPole, and HalfCheetah, respectively. Further experimental results regarding reward can be found in Appendix B.2.

As a high-level summary, in most experiments, PerSim either achieves the best reward or close to it. This not only reaffirms our prediction results in Appendix B.1,

Table 5.2: Average reward results.

(a) Reward: MountainCar.

Data	Method	0.0001	0.0025	0.0035
Pure	PerSim	-56.80± 1.83	**-189.4**± 6.44	**-210.6**± 4.27
	Vanilla CaDM	-106.3± 44.1	-432.3± 117	-471.8± 43.0
	PE-TS CaDM	-74.23± 16.5	-492.3± 13.3	-500.0± 0.00
	BCQ-P	-67.60± 22.3	-267.8± 202	-295.1± 180
	BCQ-A	**-44.79**± 0.08	-380.7± 170	-500.0± 0.00
Random	PerSim	**-57.70**± 5.63	**-186.6**± 4.25	**-210.1**± 4.48
	Vanilla CaDM	-62.57± 5.11	-479.3± 21.7	-497.5± 4.39
	PE-TS CaDM	-82.00± 3.47	-500.0± 0.00	-500.0± 0.00
	BCQ-P	-500.0± 0.00	-500.0± 0.00	-500.0± 0.00
	BCQ-A	-500.0± 0.00	-500.0± 0.00	-500.0± 0.00
Pure-ε-20	PerSim	**-54.20**± 0.56	**-191.2**± 6.70	**-199.7**± 3.99
	Vanilla CaDM	-56.73± 4.20	-463.2± 57.5	-478.9± 35.8
	PE-TS CaDM	-107.6± 36.3	-500.0± 0.00	-500.0± 0.00
	BCQ-P	-71.21± 24.4	-286.6± 196	-328.3± 158
	BCQ-A	-364.5± 180	-260.6± 51.4	-204.5± 68.9
Pure-ε-40	PerSim	-54.60± 0.55	**-189.7**± 7.14	**-200.3**± 2.26
	Vanilla CaDM	-55.23± 0.76	-481.7± 25.3	-496.2± 4.31
	PE-TS CaDM	-102.3± 20.3	-500.0± 0.00	-500.0± 0.00
	BCQ-P	**-50.01**± 7.50	-373.6± 180	-352.0± 211
	BCQ-A	-94.87± 0.88	-358.7± 201	-486.5± 20.6
	True env+MPC	-53.95± 4.10	-182.9± 22.9	-197.5± 20.7

(b) Reward: CartPole.

Data	Method	(2/0.5)	(10/0.85)	(10/0.15)
Pure	PerSim	**199.7**± 0.58	**193.8**± 4.28	**192.0**± 2.28
	Vanilla CaDM	168.0± 19.7	190.8± 6.80	58.10± 10.7
	PE-TS CaDM	92.30± 44.8	193.6± 8.30	127.5± 9.50
	BCQ-P	166.2± 39.3	181.2± 13.5	182.8± 15.0
	BCQ-A	65.40± 67.5	79.20± 69.5	132.1± 85.0
Random	PerSim	**197.7**± 7.82	193.0± 6.60	**185.7**± 3.49
	Vanilla CaDM	150.5± 15.7	175.6± 5.80	65.10± 16.3
	PE-TS CaDM	88.60± 18.5	**196.1**± 3.00	171.0± 21.7
	BCQ-P	44.80± 34.0	57.91± 53.0	36.40± 36.0
	BCQ-A	43.90± 16.4	21.10± 5.81	39.50± 12.1
Pure-ε-20	PerSim	**199.8**± 0.24	**199.1**± 1.30	**197.8**± 1.68
	Vanilla CaDM	171.1± 38.1	193.4± 2.10	64.20± 10.0
	PE-TS CaDM	98.30± 42.9	198.6± 0.40	141.1± 12.0
	BCQ-P	98.90± 30.2	162.1± 15.5	86.10± 72.1
	BCQ-A	67.30± 62.2	65.60± 51.8	140.0± 80.6
Pure-ε-40	PerSim	**199.9**± 0.18	**198.0**± 1.21	**197.4**± 1.72
	Vanilla CaDM	160.6± 46.6	191.9± 6.40	79.60± 31.6
	PE-TS CaDM	91.90± 67.6	197.0± 1.40	143.5± 17.5
	BCQ-P	28.90± 6.80	31.80± 25.9	18.50± 11.1
	BCQ-A	34.60± 1.55	47.71± 48.7	23.20± 9.44
	True env+MPC	200.0± 0.00	198.4± 7.20	200.0± 0.00

(c) Reward: HalfCheetah.

Data	Method	(0.3/1.7)	(1.7/0.3)	(0.3/0.3)
Pure	PerSim	**1984** ± 763	997.0± 403	**714.7**± 314
	Vanilla CaDM	50.31± 71.7	-134.0± 81.1	11.39± 171
	PE-TS CaDM	481.1± 252	503.7± 181	553.0± 127
	BCQ-P	549.8± 322	**2006** ± 153	-65.18± 92.8
	BCQ-A	-262.7± 96.6	-139.0± 236	165.6± 83.1
Random	PerSim	**2124** ± 518	**2060** ± 900	472.0± 56.9
	Vanilla CaDM	288.4± 32.4	362.9± 55.4	351.8± 34.9
	PE-TS CaDM	754.6± 242	744.5± 281	**767.4**± 214
	BCQ-P	-1.460± 0.16	-1.750± 0.22	-1.690± 0.19
	BCQ-A	279.6± 160	-44.05± 24.9	-75.03± 44.0
Pure-ε-20	PerSim	**3186** ± 604	1032.4±232	**1120.9**± 243
	Vanilla CaDM	412.0± 152	31.92± 109	460.2± 159
	PE-TS CaDM	1082 ± 126	**1125** ± 132	1067 ± 64.3
	BCQ-P	254.6± 352	406.7± 71.1	385.9± 57.1
	BCQ-A	376.8± 102	84.66± 53.3	230.1± 10.0
Pure-ε-40	PerSim	**2590** ± 813	1016 ± 283	**1365** ± 582
	Vanilla CaDM	465.6± 49.2	452.7± 130	720.0± 74.9
	PE-TS CaDM	1500 ± 246	**1218** ± 221	1339 ± 54.8
	BCQ-P	78.25± 200	173.8± 189	417.1± 155
	BCQ-A	269.2± 60.7	-181.5± 57.4	193.0± 31.8
	True env+MPC	7459 ± 171	42893± 6959	66675± 9364

but also is particularly encouraging for PerSim since the policy utilized is simply MPC and not optimized as done in model-free approaches. Furthermore, these results corroborate the appropriateness of our low-rank latent factor representation and our overall methodological framework as a principled solution to this challenging yet

meaningful setting within offline RL. Next, we highlight interesting conclusions from these sets of experiments.

"Solved" Environments are Not Actually Solved. MountainCar and CartPole are commonly perceived as simple, "solved" environments within the RL literature. Yet, Tables 5.2a and 5.2b demonstrate that the offline setting with scarce and heterogeneous data impose unique challenges, and undoubtedly warrants a new methodology. We find state-of-the-art model-based and model-free methods perform poorly on some of the test agents in MountainCar and CartPole. In comparison, PerSim's performance in these environments is close to that of MPC planning using the ground-truth environment across all test agents, thereby confirming the success of learning the personalized simulators in Step 1 of our algorithm. In certain rare cases (e.g., MountainCar test agent 0.0001) where BCQ has a comparable performance to PerSim, it is likely due to the limitations of MPC for policy planning.

PerSim Robustly Extrapolates with Sub-optimal Data. Across all environments and offline data generating processes, PerSim remains the most consistent and robust winner. As such, it shows good promise in being applied to other environments and under different generating policies. In real-world applications, the policy used to generate the offline dataset is likely unknown. Thus, for broad applicability of a RL methodological framework, it is vital that it is robust to sub-optimality in how the dataset was generated, e.g., the dataset may contain a significant amount of "trial and error" (i.e, randomized actions). This is one of the primary motivations to use RL in such settings in the first place.

As mentioned earlier, the four offline datasets correspond to varying degrees of "optimality" of the policy used to sample agent trajectories. We highlight that PerSim achieves uniformly good reward, even with random data, i.e., the offline trajectories are produced using totally random actions. This showcases that by first learning a personalized simulator for each agent, PerSim is able to robustly *extrapolate* outside the policy used to generate the data, however sub-optimal that policy might be. In contrast, BCQ is not robust to such sub-optimality in the offline data generation. For example, for HalfCheetah, BCQ can achieve better results than PerSim only when

trained on "optimal" offline data (i.e., pure policy). This is because BCQ, by design, is conservative and is regularized to only pick actions that are close to what is seen in the offline data. However, when BCQ is provided sub-optimal data, its performance degrades substantially (and poorer compared to PerSim). Interestingly, in some cases, PerSim performs even better under policies with a degree of randomness than the pure policy. This may indicate that PerSim is able to leverage the additional richness in information and state-action space exploration that a randomized policy can provide. In summary, PerSim's ability to successfully extrapolate, even when given sub-optimal offline data, makes it a suitable candidate for real-world applications.

5.3 Ablation Study

We conduct an ablation study which explores the impact of two factors:

1. **Impact of less data.** With less data, we explore less of the state-action space and see a number of heterogeneous agents, making the dataset more challenging to learn a decision-making policy from. To see how well PerSim is able to handle less data compared to benchmarks, we evaluate the performance of each RL method on datases of size 250, 100, 50, and 25. We use a random policy to generate each dataset, to account for a less-than ideal setting where a pure policy is not available.

2. **Impact of Factorization.** Our current neural-network architectures models the transition dynamics as a latent function of latent factors corresponding to agents, state, and actions. We explore the impact of eliminating the latent function, and just learning separate latent factors by removing their connected edges in the neural network.

Figure B.12 shows the result of the ablation study on PerSim, unfactorized PerSim, Vanilla CaDM, PE-TS CaDM, and BCQ-P. For MountainCar, we tried an additional variant of PerSim that was "unpersonalized", i.e. did not distinguish between heterogeneous agents and learned a single simulator for all agents. The purpose was to see

value added from personalization through latent unit factors in PerSim.

We find that our previous conclusions hold for different dataset sizes. PerSim continues to perform best overall compared to benchmarks. We find that the unfactorized model performs better than CaDM and BCQ, but around the same as PerSim. Unpersonalized PerSim performs significantly worse than regular and unfactorized PerSim, demonstrating that PerSim is able to improve performance by learning the heterogeneity across agents. More experiments would allow to draw more concrete conclusions on the impact of less data and factorization.

Full experiment results can be found in Appendix B.4. For CartPole and HalfCheetah, we omit unpersonalized PerSim in our experiments because it would almost certainly perform strictly worse than PerSim.

Table 5.3: Ablation Study: MountainCar

N	Method	0.0001	0.0005	0.001	0.0025	0.0035
250	PerSim	**-53.70** ±0.41	-66.50 ±1.21	-116.6 ±3.18	-192.3 ±1.23	**-199.6** ±3.40
	Unfactorized	-54.00 ±0.22	**-65.90** ±3.23	**-104.2** ±2.10	**-189.4** ±12.1	-199.7 ±10.4
	Unpersonalized	-65.00 ±8.14	-66.60 ±4.65	-162.4 ±25.5	-320.2 ±5.84	-371.7 ±6.72
	Vanilla CaDM	-59.90 ±1.61	-78.13 ±7.11	-332.6 ±41.3	-467.8 ±16.2	-500.0 ±0.00
	PETS CaDM	-73.66 ±3.15	-106.8 ±7.15	-473.8 ±37.0	-500.0 ±0.00	-500.0 ±0.00
	BCQ-P	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00
100	PerSim	-55.00 ±0.70	**-67.02** ±1.76	**-110.3** ±3.46	-193.1 ±5.21	**-197.4** ±3.05
	Unfactorized	**-53.60** ±1.25	-69.60 ±0.50	-160.3 ±59.0	**-188.6** ±1.70	-198.1 ±3.50
	Unpersonalized	-67.50 ±10.2	-72.40 ±12.4	-149.4 ±27.4	-310.2 ±6.56	-321.1 ±11.4
	Vanilla CaDM	-56.60 ±1.77	-67.20 ±7.08	-264.0 ±110	-482.6 ±30.2	-497.1 ±5.08
	PETS CaDM	-79.33 ±4.36	-106.5 ±19.3	-418.4 ±27.3	-492.7 ±10.3	-499.9 ±0.14
	BCQ-P	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00
50	PerSim	-54.20 ±0.90	-66.40 ±0.17	**-110.2** ±8.65	**-188.7** ±5.25	**-199.6** ±3.23
	Unfactorized	**-53.70** ±1.13	-66.70 ±4.55	-157.3 ±46.8	-382.7 ±54.5	-329.6 ±125.0
	Unpersonalized	-66.50 ±9.78	-68.10 ±4.93	-152.0 ±10.6	-286.3 ±8.64	-337.1 ±5.00
	Vanilla CaDM	-58.80 ±1.40	**-66.37** ±0.35	-131.8 ±17.6	-497.2 ±4.85	-500.0 ±0.00
	PETS CaDM	-67.86 ±7.15	-79.73 ±6.37	-290.5 ±76.9	-458.4 ±26.8	-498.5 ±2.12
	BCQ-P	-214.3 ±202	-348.6 ±186	-428.1 ±103	-500.0 ±0.00	-500.0 ±0.00
25	PerSim	-56.80 ±1.81	**-67.50** ±2.81	**-139.9** ±29.5	**-246.8** ±39.1	**-243.8** ±47.1
	Unfactorized	**-54.50** ±0.62	-264.6 ±181	-194.7 ±30.6	-406.0 ±126	-343.1 ±114
	Unpersonalized	-64.10 ±5.46	-75.00 ±5.96	-184.6 ±57.7	-362.6 ±94.7	-412.9 ±62.4
	Vanilla CaDM	-62.33 ±3.34	-76.80 ±10.9	-275.7 ±63.9	-473.0 ±24.1	-496.5 ±4.90
	PETS CaDM	-67.26 ±6.95	-86.96 ±10.9	-331.0 ±121	-497.9 ±2.92	-500.0 ±0.00
	BCQ-P	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00

Chapter 6

Conclusion

6.1 Conclusion

In this work, we investigate RL in an offline setting, where we observe a single trajectory across heterogeneous agents under an unknown, potentially sub-optimal policy. This is particularly challenging for existing approaches even in "solved" environments such as MountainCar and CartPole. Our work, PerSim, offers a successful first attempt in simultaneously learning personalized policies across all agents under this data regime; we do so by first positing a principled low-rank latent factor representation, and then using it to build personalized simulators in a data-efficient manner. But there is much that remains unexplored. For example, in environments like HalfCheetah where the transition dynamics of each agent are harder to learn, the performance of PerSim is not comparable with the online setting, where one gets to arbitrarily sample trajectories for each agent. Of course, the considered data regime is fundamentally harder. Therefore, understanding the extent to which we can improve performance, using our low-rank latent factor approach or a different methodology altogether, remains to be established. Additionally, a rigorous finite-sample analysis for this setting, which studies the effect of the degree of agent heterogeneity, the diversity of the samples collected, etc. remains important future work. More generally, we believe that there are many fruitful inquiries under this challenging yet meaningful data regime for RL.

6.2 Next Steps

Given the promising results of PerSim on standard RL benchmark environments and across different types of datasets, the possible applications of PerSim to more complex environments and real-world applications is exciting.

Additional environments. The most crucial question is how well PerSim can extend to more complex environments. Some reasonable next environments would be Mujoco environments, including Ant, Walker, or SlimHumanoid, which will require higher rank in our latent representation. We could also branch out to new problem spaces which can be represented as an offline RL problem, such as the revenue management problem in econometric literature. Revenue management concerns the problem of dynamically manage demand over time for products and services through learning a decision-making policy. Most methods approach revenue management through a dynamic programming formulation. However, through preliminary experiments, we find that the revenue management problem can be modeled as an RL environment, and that PerSim can model the transition dynamics to high accuracy. More work in modeling such real-world environments is warranted and will be important step for bridging the gap from simulated environments to practical uses.

PerSim with other policy-learning algorithms. In our experiments we use MPC to learn from data generated by PerSim. However, PerSim can be paired with any other model-based RL method for learning a decision-making policy. For instance, exploring BCQ+PerSim would be natural extension given our experiments using BCQ as a benchmark. Comparing results of BCQ on the true environment against BCQ with simulated PerSim would help us further understand the ability of PerSim to model the true environment for the purposes of policy-learning.

Appendix A

Algorithm and Implementation Details

A.1 Our Method

Step 1 Details: Learning Personalized Simulators. As explained in Section 4.2, the personalized simulators are effectively trained by learning g_u, g_v, and g_w, which correspond to the agent, state, and action encoders, respectively. Below, we detail the architecture used for each function.

1. **Agent encoder: g_u.** We use a single layer that takes in a one-hot encoder of the agent and returns an r-dimensional latent factor.

2. **State encoder: g_v.** We use a multilayer perceptron (MLP) with 1 hidden layer of 256 ReLU activated nodes for both MountainCar and CartPole, and an MLP with 4 hidden layers each with 256 ReLU activated nodes for HalfCheetah.

3. **Action encoder: g_w.** In environments with discrete action spaces, i.e., MountainCar and CartPole, we use a single layer that takes in a one-hot encoder of the action and produce an r-dimensional latent factor. For HalfCheetah, we use an MLP with 2 hidden layers of (256,256) ReLU activated nodes.

We choose the tensor rank r to be 3, 5, and 15 for the MountainCar, CartPole, and

HalfCheetah environments, respectively. The choice is made via cross validation from the set $\{3, 5, 10, 15, 20, 30\}$. Specifically, 20% of the data points (selected randomly from different trajectories) are set aside for validation in the hyper-parameter selection process. We train our simulators with a learning rate of 0.001, 300 epochs, and a batch size of 512 for HalfCheetah and MountainCar and 64 for CartPole. Please refer to the pseudo-code in Algorithm 1 for a detailed description of the training procedure.

Algorithm 1 Training Personalized Dynamic Models

Input: Dataset $\mathcal{D}$, Rank r, Learning rate η, Batch size B, Number of epochs K
Output: $g_u(\cdot;\psi)$, $g_v(\cdot;\phi)$, $g_w(\cdot;\theta)$
Initialize ψ, ϕ, and θ
for each epoch **do** :
    for each batch **do** :
        for $i = 1$ to B **do** :
            Sample $\{s_i, a_i, s_i', n_i\} \sim \mathcal{D}$
            Compute $\Delta s_i \leftarrow s_i' - s_i$
            Get agent latent factor $\widehat{u}(n_i) \leftarrow g_u(n_i;\psi)$
            Get state latent factor $\widehat{v}(s_i) \coloneqq [\widehat{v}(s_i, d)]_{d\in[D]} \leftarrow g_v(s_i;\phi)$
            Get action latent factor $\widehat{w}(a_i) \leftarrow g_w(a_i;\theta)$
            Get the error estimate $\mathcal{L}_i \leftarrow \|\Delta s_i - \sum_{\ell=1}^{r} \widehat{u}_\ell(n_i) v_\ell(s_i) \widehat{w}_\ell(a_i)\|_2^2$
        end for
        Update $\psi \leftarrow \psi - \eta \nabla_\psi \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}_i$
        Update $\phi \leftarrow \phi - \eta \nabla_\phi \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}_i$
        Update $\theta \leftarrow \theta - \eta \nabla_\theta \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}_i$
    end for
end for

Step 2 Details: Learning a Decision-making Policy. As outlined in Section 4.2, we use MPC to select the best action. For MountainCar and CartPole, we use random shooting to sample 1000 candidate actions with a planning horizon of 50. For HalfCheetah, we use the cross entropy method to sample 200 candidate actions with a planning horizon of 30. For all environments, we train $M = 5$ simulators.

A.2 Benchmarking Algorithms

Vanilla CaDM + PE-TS CaDM. We use the implementation provided by the authors in Lee et al. (2020).[1] To train on offline data, we modify the sampling procedure in the implementation. Specifically, we change it to sample from a replay buffer containing the recorded trajectories. Similar to our method, we use MPC with a planning horizon of 30 for HalfCheetah, and 50 for MountainCar and CartPole. We train the forward dynamic model, the backward dynamic model, and the context encoder for 20 iterations each with a maximum of 200 epochs and a learning rate of 0.001. For PE-TS, as is done in Lee et al. (2020), we use an ensemble of five dynamics models, and use twenty particles for trajectory sampling.

BCQ-P +BCQ-A. We use the implementation provided by the authors in Fujimoto et al. (2019).[2] Specifically, we use discrete BCQ for MountainCar and CartPole, and continuous BCQ for HalfCheetah. For both BCQ-P and BCQ-A, we train the policy for 5.5×10^5 iterations.

A.3 Detailed Setup

A.3.1 Environments

In this paper, we carry out experiments on three environments from the OpenAI gym: two classical non-linear control environments, MountainCar and CartPole, and one Mujoco environment, HalfCheetah (Todorov et al., 2012). In what follows, we describe these three environments in detail.

MountainCar. In MountainCar, the goal is to drive a under-powered car to the top of a hill by taking the least number of steps.

- *Observation.* We observe $x(t)$, $\dot{x}(t)$: the position and velocity of the car, respectively.

[1] https://github.com/younggyoseo/CaDM
[2] https://github.com/sfujim/BCQ

- *Actions.* There are three possible actions $\{0, 1, 2\}$: (0) accelerate to the left; (1) do nothing; (2) accelerate to the right.

- *Reward.* The reward is defined as

$$R(t) = \begin{cases} 1, & x(t) \geq 0.5 \\ -1, & \text{otherwise.} \end{cases}$$

- *Environment modification.* We vary the gravity within the range $[0.0001, 0.0035]$. Note that with a weaker gravity, the environment is trivially solved by directly moving to the right. On the other hand, with a stronger gravity, the car must drive left and right to build up enough momentum. See Table 5.1 for details about the parameter ranges and the test agents.

CartPole. In CartPole, a pole is attached to a cart moving on a frictionless track. The goal is to prevent the pole from falling over by moving the cart to the left or to the right, and to do so for as long as possible (maximum of 200 steps).

- *Observation.* We observe $x(t), \dot{x}(t), \theta(t), \dot{\theta}(t)$: the cart's position, its velocity, the pole's angle, and its angular velocity, respectively.

- *Actions.* There are two possible actions $\{0, 1\}$: (0) push to the right; (1) push to the left.

- *Reward.* The reward is 1 for every step taken without termination. The environment terminates when the pole angle exceeds 12 degrees or when the cart position exceeds 2.4.

- *Environment modification.* As is done in Lee et al. (2020), we vary the length of the pole and push force within the ranges $[0.15, 0.85]$ and $[2.0, 18.0]$, respectively. See Table 5.1 for details about the parameter ranges and the test agents.

HalfCheetah. In HalfCheetah, the goal is to move the cheetah as fast as possible. The cheetah's body consists of 7 links connected via 6 joints.

- *Observation.* We observe an 18-dimensional vector that includes the angle and angular velocity of all six joints, as well as the 3-D position and orientation of the torso. Additionally, as is done in previous studies Lee et al. (2020); Kidambi et al. (2020), we append the center of mass velocity to our state vector to enable computing the reward from observations.

- *Actions.* The action $a(t) \in [-1, 1]^6$ represents the torque applied at the six joints.

- *Reward.* The reward is defined as

$$R(t) = v(t) - 0.05 \left\| a(t) \right\|^2,$$

  where $v(t)$ is the center of mass velocity at time t.

- *Environment modification.* As is done in Lee et al. (2020), we scale the mass of every link and the damping of every joint by factors m and d, respectively. Specifically, we vary both m and d within the range $[0.2, 1.8]$. See Table 5.1 for details about the parameter ranges and the test agents.

A.3.2 Offline Datasets

As stated in Section 5, we generate four offline datasets for each environment with varying "optimality" of the sampling policy. Specifically, we generate 500 trajectories (one per agent) for each environment as per the following sampling procedures:

(i) Pure. In the Pure procedure, actions are sampled according to a fixed policy (for each agent) that has been trained to achieve "good" performance. That is, for each environment, we first train a policy using online model-free algorithms for the training agents shown in Table 5.1. Specifically, we train these logging policies using DQN (Mnih et al., 2015) for MountainCar and CartPole, and using TD3 for and HalfCheetah. We train these policies to achieve rewards of approximately -200, 120, 3000, for MountainCar, CartPole, and HalfCheetah respectively. Then, to sample a

trajectory for each of the 500 agents, we use the policy trained on the training agent with the closest parameter value.

(ii) Random. Actions are selected uniformly at random.

(iii) Pure-ε-20. Actions are selected uniformly at random with probability of 0.2, and selected via the pure policy otherwise.

(iv) Pure-ε-40. Actions are selected uniformly at random with probability of 0.4, and selected via the pure policy otherwise.

See Table A.1 for details about the reward observed for the five test agents using these four sampling procedures, and the average reward and trajectory length achieved across all 500 agents.

Table A.1: Observed reward and trajectory length in the four sampled datasets in each environment. Agent 1 to Agent 5 refer to the five test agents. The average is taken across all 500 agents.

Environment	Data	Observed Reward						Average
		Agent 1	Agent 2	Agent 3	Agent 4	Agent 5	Average	Trajectory Length
MountainCar	Pure	-48.0	-50.0	-57.0	-171.0	-134.0	-112.926	113.914
	Random	-500.0	-500.0	-500.0	-500.0	-500.0	-496.324	496.344
	Pure-eps-2	-46.0	-54.0	-73.0	-165.0	-140.0	-155.252	156.214
	Pure-eps-4	-55.0	-64.0	-500.0	-264.0	-208.0	-227.278	228.18
CartPole	Pure	197	200	193	179	200	185.14	186.14
	Random	59	23	10	26	17	22.39	23.39
	Pure-eps-2	180	199	30	179	199	157.92	158.92
	Pure-eps-4	38	23	11	170	14	92.80	93.80
HalfCheetah	Pure	522.40	1246.32	251.25	2646.85	1011.99	1894.10	1000.00
	Random	-395.58	-65.77	-106.80	-150.01	-323.86	-251.75	1000.00
	Pure-eps-2	399.95	938.58	189.17	1742.85	1137.45	1121.52	1000.00
	Pure-eps-4	508.11	-115.94	128.64	1155.23	216.69	771.71	1000.00

Appendix B

Complete Experimental Results

In this section, we provide the complete experimental results for the prediction error (see Appendix B.1), and the average reward (see Appendix B.2) for all five test agents across the three benchmark environments. Further, we provide additional visualizations for the agent latent factors in Appendix B.3. We again emphasize that the additional results are consistent with the overall conclusions indicated in Section 5.

B.1 Prediction Error Results

In this section, we provide additional results for the prediction error experiments. As detailed in Section 5, we evaluate the accuracy of the learned transition dynamics for each of the five test agent, focusing on long-horizon model prediction. Specifically, we predict the next 50-step state trajectory given an initial state and an unseen sequence of 50 actions. The sequence of 50 actions are chosen according to an unseen test policy. Precisely , the test policies are fitted via DQN for MountainCar and CartPole, and via TD3 for and HalfCheetah, for the agent with the default covariate parameters. The test policies were trained to achieve an average rewards of -150, 150, 4000 for the MountainCar, CartPole, and HalfCheetah environments, respectively. As described in Section 5, we report the mean RMSE and the median R^2 across 200 trials. The results are summarized in Tables B.1, B.2, and B.3 for MountainCar, CartPole, and

Table B.1: Prediction error: MountainCar

Data	Method	Agent 1 0.0001	Agent 2 0.0005	Agent 3 0.001	Agent 4 0.0025	Agent 5 0.0035
Pure	PerSim	**0.025 (0.969)**	**0.090 (0.973)**	**0.031 (0.978)**	**0.014 (0.942)**	**0.039 (0.803)**
	Vanilla CaDM	0.149 (0.741)	0.126 (0.767)	0.075 (0.913)	0.177 (-18.426)	0.238 (-30.148)
	PE-TS CaDM	0.326 (-1.912)	0.288 (-3.449)	0.194 (-2.61)	0.154 (-1.114)	0.148 (-0.179)
Random	PerSim	**0.004 (0.999)**	**0.003 (1.000)**	**0.001 (1.000)**	**0.001 (1.000)**	**0.001 (1.000)**
	Vanilla CaDM	0.256 (0.428)	0.203 (0.264)	0.162 (-1.041)	0.134 (0.710)	0.217 (-1.767)
	PE-TS CaDM	0.242 (0.310)	0.177 (0.725)	0.156 (-0.259)	0.101 (0.868)	0.075 (0.967)
Pure-ε-20	PerSim	**0.004 (1.000)**	**0.003 (1.000)**	**0.001 (1.000)**	**0.002 (0.999)**	**0.004 (0.998)**
	Vanilla CaDM	0.227 (0.309)	0.201 (0.271)	0.131 (0.405)	0.101 (-0.369)	0.157 (-2.252)
	PE-TS CaDM	0.35 (-2.571)	0.282 (-3.829)	0.216 (-1.706)	0.139 (0.746)	0.13 (0.892)
Pure-ε-40	PerSim	**0.006 (0.999)**	**0.005 (1.000)**	**0.004 (1.000)**	**0.006 (0.992)**	**0.003 (0.999)**
	Vanilla CaDM	0.199 (0.54)	0.176 (0.542)	0.106 (0.703)	0.119 (0.384)	0.187 (-9.005)
	PE-TS CaDM	0.639 (-2.273)	0.283 (-2.299)	0.174 (-0.631)	0.192 (-6.056)	0.157 (0.546)

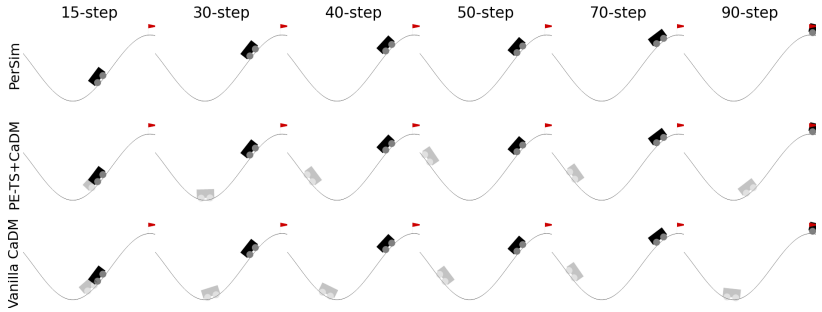
HalfCheetah, respectively. Additionally, Figure B-1 visualizes the prediction accuracy of PerSim up to 90-steps ahead predictions for two test agents in MountainCar and CartPole.

Table B.2: Prediction Error: CartPole

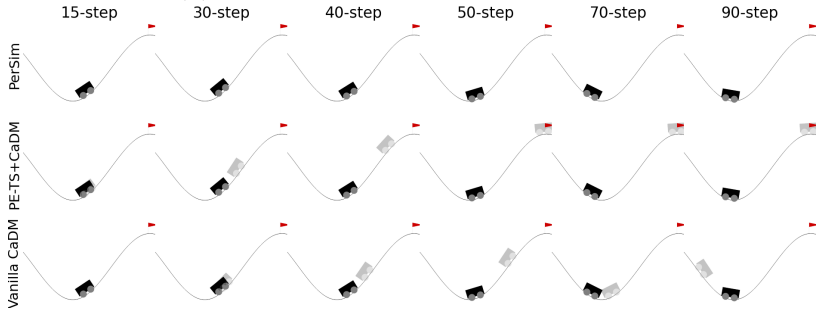
Data	Method	Agent 1 (2/0.5)	Agent 2 (10.0/0.5)	Agent 3 (18.0/0.5)	Agent 4 (10/0.85)	Agent 5 (10/0.15)
Pure	PerSim	**0.001 (1.000)**	**0.001 (1.000)**	**0.002 (1.000)**	**0.001 (1.000)**	**0.035 (0.995)**
	Vanilla CaDM	0.403 (0.712)	0.152 (0.425)	0.148 (0.664)	0.039 (0.975)	2.531 (-0.532)
	PE-TS CaDM	0.031 (0.993)	0.011 (0.995)	0.016 (0.997)	0.006 (1.000)	0.319 (0.651)
Random	PerSim	**0.014 (0.982)**	**0.022 (0.970)**	**0.030 (0.979)**	**0.006 (0.999)**	**0.152 (0.883)**
	Vanilla CaDM	0.172 (0.095)	0.282 (-0.483)	0.514 (-0.381)	0.098 (0.639)	3.307 (-0.734)
	PE-TS CaDM	0.564 (-1.357)	0.493 (-0.200)	0.891 (-0.458)	0.216 (0.450)	3.764 (-1.104)
Pure-ε-20	PerSim	**0.000 (1.000)**	**0.001 (1.000)**	**0.008 (0.999)**	**0.001 (1.000)**	**0.048 (0.984)**
	Vanilla CaDM	0.270 (-0.982)	0.037 (0.973)	0.046 (0.991)	0.058 (0.943)	2.193 (-0.432)
	PE-TS CaDM	0.639 (-1.206)	0.252 (0.000)	0.434 (-0.170)	0.148 (0.384)	2.680 (-0.561)
Pure-ε-40	PerSim	**0.000 (1.000)**	**0.002 (1.000)**	**0.011 (0.998)**	**0.000 (1.000)**	**0.018 (0.998)**
	Vanilla CaDM	0.233 (-0.883)	0.055 (0.956)	0.032 (0.993)	0.035 (0.987)	3.286 (-0.618)
	PE-TS CaDM	0.051 (0.983)	0.017 (0.992)	0.013 (0.998)	0.010 (0.999)	0.411 (0.553)

Table B.3: Prediction Error: HalfCheetah

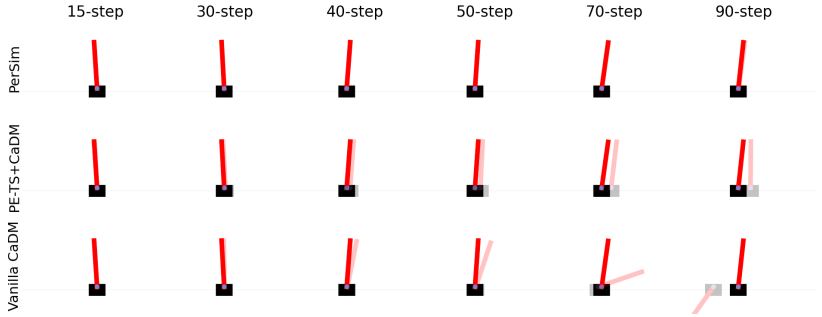
Data	Method	Agent 1 (0.3/1.7)	Agent 2 (1.7/0.3)	Agent 3 (0.3/0.3)	Agent 4 (1.7/1.7)	Agent 5 (1.0/1.0)
Pure	PerSim	**1.401 (0.890)**	4.766 (0.574)	**4.385 (0.791)**	**1.409 (0.840)**	2.505 (0.793)
	Vanilla CaDM	3.902 (0.472)	3.851(0.490)	6.308 (0.380)	2.881 (0.336)	**1.804 (0.829)**
	PE-TS CaDM	3.147 (0.614)	**3.080 (0.682)**	5.270 (0.572)	1.833 (0.647)	1.915 (0.784)
Random	PerSim	**1.194 (0.916)**	4.064 (0.670)	**4.070 (0.812)**	**1.291 (0.855)**	2.325 (0.812)
	Vanilla CaDM	4.030 (0.435)	4.121(0.430)	4.446 (0.672)	3.840 (-0.432)	4.270 (0.255)
	PE-TS CaDM	2.735 (0.731)	**2.756 (0.688)**	4.141 (0.767)	2.031 (0.601)	**1.957 (0.773)**
Pure-ε-20	PerSim	**1.172 (0.922)**	4.283 **(0.660)**	**3.832 (0.844)**	**1.123 (0.880)**	**2.057 (0.840)**
	Vanilla CaDM	3.613 (0.534)	3.455 (0.538)	6.046 (0.477)	2.256 (0.575)	2.070 (0.753)
	PE-TS CaDM	2.913 (0.699)	**2.959** (0.652)	4.818 (0.647)	1.970 (0.633)	2.073 (0.744)
Pure-ε-40	PerSim	**1.016 (0.940)**	4.021 **(0.664)**	**3.742 (0.853)**	**1.287 (0.868)**	**1.887 (0.836)**
	Vanilla CaDM	3.685 (0.493)	3.612 (0.480)	6.000 (0.392)	2.561 (0.521)	2.136 (0.738)
	PE-TS CaDM	3.021 (0.672)	**3.025** (0.614)	5.075 (0.615)	1.792 (0.668)	1.930 (0.779)



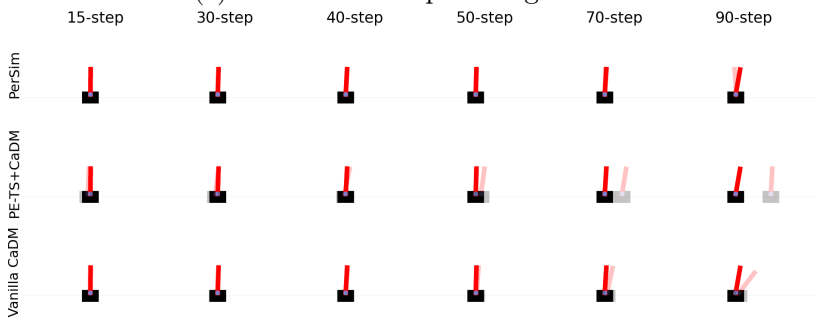
(a) MountainCar with gravity 0.0001



(b) MountainCar with gravity 0.0025



(c) CartPole with pole length 0.85



(d) CartPole with pole length 0.5

Figure B-1: Visualization of the prediction accuracy of PerSim for two heterogeneous agents in MountainCar and CartPole, and how it compares with the two CaDM variants. Specifically, given an initial state and a sequence of actions, we predict future states for the next 100 steps. Ground-truth states and predicted states are denoted by the opaque and translucent objects, respectively.

B.2 Detailed Average Reward Results

In this section, we show the full results for the experiments for the reward achieved in each environment. Specifically, we report the average reward achieved by PerSim and several state-of-the-art model-based and model-free offline RL methods on the three benchmark environments across 5 trials. We evaluate the performance of each method using the average reward over 20 episodes for the model-based methods and the average reward over 100 episodes for the model-free methods. We repeat each experiment five times and report the mean and standard deviation.

The results are summarized in Tables B.4, B.5, and B.6 for MountainCar, CartPole, and HalfCheetah, respectively.

Table B.4: Average Reward: MountainCar

Data	Method	Agent 1 0.0001	Agent 2 0.0005	Agent 3 0.001	Agent 4 0.0025	Agent 5 0.0035
Pure	PerSim	-56.80± 1.83	-74.30± 6.59	-114.1± 16.1	**-189.4**± 6.44	**-210.6**± 4.27
	Vanilla CaDM	-106.3± 44.1	-289.8± 195	-332.8± 193	-432.3± 117	-471.8± 43.0
	PE-TS CaDM	-74.23± 16.5	-119.1± 44.4	-361.3± 240	-492.3± 13.3	-500.0± 0.00
	BCQ-P	-67.60± 22.3	-68.60± 19.6	-79.20± 14.7	-267.8± 202	-295.1± 180
	BCQ-A	**-44.79**± 0.08	**-50.50**± 0.40	**-63.52**± 0.19	-380.7± 170	-500.0± 0.00
Random	PerSim	**-57.70**± 5.63	**-74.30**± 6.39	**-120.4**± 2.17	**-186.6**± 4.25	**-210.1**± 4.48
	Vanilla CaDM	-62.57± 5.11	-75.27± 4.59	-274.9± 74.2	-479.3± 21.7	-497.5± 4.39
	PE-TS CaDM	-82.00± 3.47	-115.7± 7.63	-472.1± 48.3	-500.0± 0.00	-500.0± 0.00
	BCQ-P	-500.0± 0.00	-500.0± 0.00	-500.0± 0.00	-500.0± 0.00	-500.0± 0.00
	BCQ-A	-500.0± 0.00	-500.0± 0.00	-500.0± 0.00	-500.0± 0.00	-500.0± 0.00
Pure-ε-20	PerSim	**-54.20**± 0.56	**-67.80**± 0.48	-111.7± 6.20	**-191.2**± 6.70	**-199.7**± 3.99
	Vanilla CaDM	-56.73± 4.20	-70.70± 1.54	-148.7± 11.6	-463.2± 57.5	-478.9± 35.8
	PE-TS CaDM	-107.6± 36.3	-158.5± 84.3	-464.6± 37.3	-500.0± 0.00	-500.0± 0.00
	BCQ-P	-71.21± 24.4	-72.10± 20.5	**-78.41**± 14.3	-286.6± 196	-328.3± 158
	BCQ-A	-364.5± 180	-435.7± 63.7	-282.7± 308	-260.6± 51.4	-204.5± 68.9
Pure-ε-40	PerSim	-54.60± 0.55	-71.10± 1.89	-115.7± 4.80	**-189.7**± 7.14	**-200.3**± 2.26
	Vanilla CaDM	-55.23± 0.76	-67.90± 7.30	-163.7± 30.8	-481.7± 25.3	-496.2± 4.31
	PE-TS CaDM	-102.3± 20.3	-120.7± 18.8	-476.0± 41.5	-500.0± 0.00	-500.0± 0.00
	BCQ-P	**-50.01**± 7.50	**-57.10**± 10.3	**-66.11**± 3.91	-373.6± 180	-352.0± 211
	BCQ-A	-94.87± 0.88	-80.03± 38.5	-329.6± 242	-358.7± 201	-486.5± 20.6
	True env+MPC	-53.95± 4.10	-72.43±7.80	-110.8± 23.8	-182.9± 22.9	-197.5± 20.7

Table B.5: Average Reward: CartPole

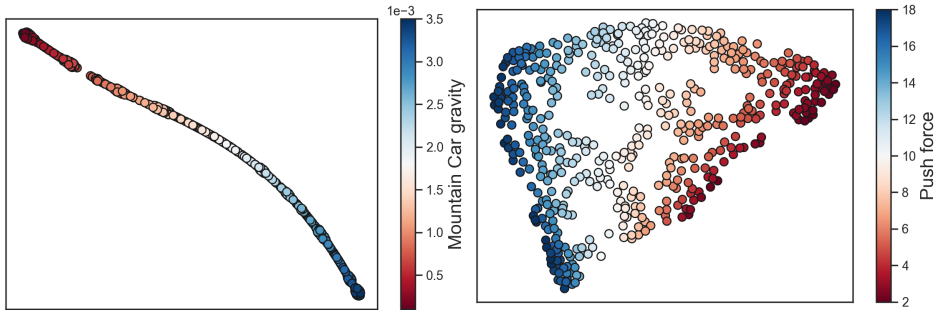
Data	Method	Agent 1 (2/0.5)	Agent 2 (10.0/0.5)	Agent 3 (18.0/0.5)	Agent 4 (10/0.85)	Agent 5 (10/0.15)
Pure	PerSim	**199.7**± 0.58	198.7± 0.86	198.5± 1.16	**193.8**± 4.28	**192.0**± 2.28
	Vanilla CaDM	168.0± 19.7	197.7± 1.50	173.6± 6.10	190.8± 6.80	58.10± 10.7
	PE-TS CaDM	92.30± 44.8	**200.0**± 0.00	**200.0**± 0.00	193.6± 8.30	127.5± 9.50
	BCQ-P	166.2± 39.3	187.4± 14.7	187.2± 15.0	181.2± 13.5	182.8± 15.0
	BCQ-A	65.40± 67.5	138.0± 80.3	79.20± 79.9	79.20± 69.5	132.1± 85.0
Random	PerSim	**197.7**± 7.82	189.5± 4.28	190.3± 4.74	193.0± 6.60	**185.7**± 3.49
	Vanilla CaDM	150.5± 15.7	158.7± 17.1	161.4± 15.8	175.6± 5.80	65.10± 16.3
	PE-TS CaDM	88.60± 18.5	**194.0**± 2.00	**197.6**± 2.20	**196.1**± 3.00	171.0± 21.7
	BCQ-P	44.80± 34.0	58.21± 58.0	56.92± 56.0	57.91± 53.0	36.40± 36.0
	BCQ-A	43.90± 16.4	18.70± 13.1	7.200± 0.84	21.10± 5.81	39.50± 12.1
Pure-ε-20	PerSim	**199.8**± 0.24	**200.0**± 0.00	**200.0**± 0.00	**199.1**± 1.3	**197.8**± 1.68
	Vanilla CaDM	171.1± 38.1	195.3± 3.00	180.7± 4.30	193.4± 2.10	64.20± 10.0
	PE-TS CaDM	98.30± 42.9	199.6± 0.50	199.0± 1.40	198.6± 0.40	141.1± 12.0
	BCQ-P	98.90± 30.2	170.9± 19.0	163.0± 36.5	162.1± 15.5	86.10± 72.1
	BCQ-A	67.30± 62.2	130.0± 76.1	33.40± 0.62	65.60± 51.8	140.0± 80.6
Pure-ε-40	PerSim	**199.9**± 0.18	**199.8**± 0.20	199.3± 1.34	**198.0**± 1.21	**197.4**± 1.72
	Vanilla CaDM	160.6± 46.6	197.3± 1.50	194.9± 3.70	191.9± 6.40	79.60± 31.6
	PE-TS CaDM	91.90± 67.6	**199.8**± 0.20	**200.0**± 0.00	197.0± 1.40	143.5± 17.5
	BCQ-P	28.90± 6.80	24.97± 12.8	27.90± 25.9	31.80± 25.9	18.50± 11.1
	BCQ-A	34.60± 1.55	23.20± 17.8	7.180± 0.76	47.71± 48.7	23.20± 9.44
	True env+MPC	200.0± 0.00	200.0± 0.00	200.0± 0.00	198.4± 7.20	200.0± 0.00

Table B.6: Average Reward: HalfCheetah

Data	Method	Agent 1 (0.3/1.7)	Agent 2 (1.7/0.3)	Agent 3 (0.3/0.3)	Agent 4 (1.7/1.7)	Agent 5 (1.0/1.0)
Pure	PerSim	**1984** ± 763	997.0± 403	**714.7**± 314	113.5± 289	1459 ± 398
	Vanilla CaDM	50.31± 71.7	-134.0± 81.1	11.39± 171	-169.8± 67.5	331.3± 201
	PE-TS CaDM	481.1± 252	503.7± 181	553.0± 127	246.0± 261	840.1± 383
	BCQ-P	549.8± 322	**2006** ± 153	-65.18± 92.8	**2564** ± 70.2	**2469** ± 67.2
	BCQ-A	-262.7± 96.6	-139.0± 236	165.6± 83.1	1649 ± 622	937.2± 221
Random	PerSim	**2124** ± 518	**2060** ± 900	472.0± 56.9	**565.2**± 377	474.8± 344
	Vanilla CaDM	288.4± 32.4	362.9± 55.4	351.8± 34.9	358.4± 205	475.0± 102
	PE-TS CaDM	754.6± 242	744.5± 281	**767.4**± 214	555.4± 73.1	**2486** ± 1488
	BCQ-P	-1.460± 0.16	-1.750± 0.22	-1.690± 0.19	-1.790± 0.21	-1.720± 0.20
	BCQ-A	279.6± 160	-44.05± 24.9	-75.03± 44.0	-109.8± 11.1	-217.5± 60.0
Pure-ε-20	PerSim	**3186** ± 604	1032.4±232	**1120.9**± 243	971.2± 916	1666 ± 930
	Vanilla CaDM	412.0± 152	31.92± 109	460.2± 159	60.33± 139	166.6± 71.8
	PE-TS CaDM	1082 ± 126	**1125** ± 132	1067 ± 64.3	1098 ± 344	**2843** ± 204
	BCQ-P	254.6± 352	406.7± 71.1	385.9± 57.1	-95.34± 65.81	738.0± 512
	BCQ-A	376.8± 102	84.66± 53.3	230.1± 10.0	**1180** ± 87.3	617.5± 32.6
Pure-ε-40	PerSim	**2590** ± 813	1016 ± 283	**1365** ± 582	803.9± 912	724.9± 236
	Vanilla CaDM	465.6± 49.2	452.7± 130	720.0± 74.9	176.7± 359	952.8± 591
	PE-TS CaDM	1500 ± 246	**1218** ± 221	1339 ± 54.8	**1569** ± 306	**3094** ± 825
	BCQ-P	78.25± 200	173.8± 189	417.1± 155	-56.12± 64.4	55.46± 128
	BCQ-A	269.2± 60.7	-181.5± 57.4	193.0± 31.8	636.4± 137	207.0± 106
	True env+MPC	7459 ± 171	42893± 6959	66675± 9364	1746± 624	36344± 7924

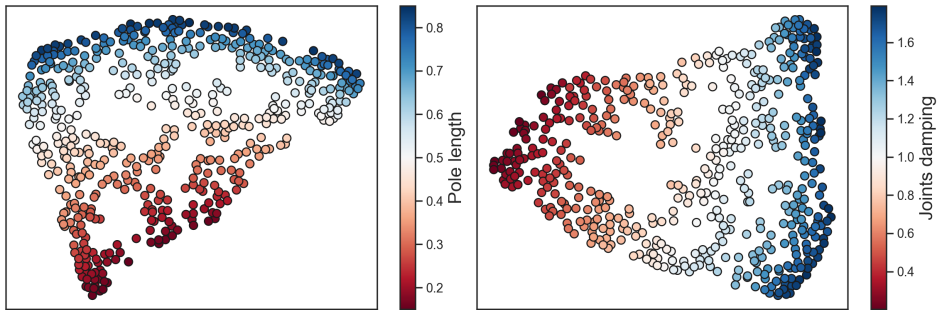
B.3 Visualization of Agent Latent Factors

In this section, we visualize the learned agent latent factors associated with the 500 heterogeneous agents in each of the three benchmark environments. These figures demonstrate that the learned latent factors indeed capture the relevant information about the agents heterogeneity in all environments. The covariates varied in each environment are gravity for MountainCar; pole length and push force for CartPole; and cheetah mass and joint damping for HalfCheetah.



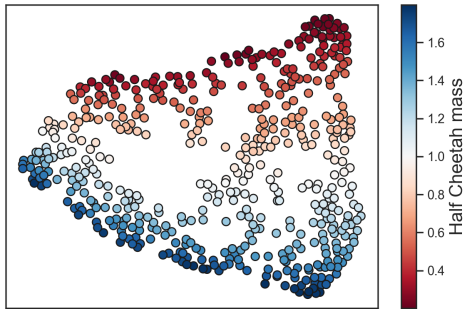
(a) MountainCar: gravity

(b) CartPole: push force



(c) CartPole: pole length

(d) HalfCheetah: joints' damping



(e) HalfCheetah: links' mass

Figure B-2: t-SNE Van der Maaten and Hinton (2008) visualization of the agent latent factors for the 500 heterogeneous agents in MountainCar, CartPole, and HalfCheetah. Colors indicate the value of the modified parameter in each environment (e.g., gravity in MountainCar).

B.4 Ablation Study

In this section, we show the full prediction error and average reward results for experiments conducted in the ablation study. The experiment setup and evaluation is identical to Appendix B.1 and B.2, but the size of the datasets are varied to 250, 100, 50, and 25 units. We show results for PerSim, PerSim Unfactorized, Vanilla CaDM, and PE-TS CaDM.

Table B.7: Prediction Error: MountainCar (Ablation Study)

Data	N	Method	0.0001	0.0005	0.001	0.0025	0.0035
Random	250	PerSim	**0.005 (1.000)**	**0.005 (1.000)**	**0.007 (0.999)**	**0.001 (1.000)**	**0.002 (1.000)**
		Unfactorized	0.014 (0.999)	0.034 (0.985)	0.029 (0.994)	0.013 (0.995)	0.007 (0.998)
		Vanilla	0.314 (-2.886)	0.254 (-3.865)	0.175 (-1.414)	0.037 (0.947)	0.099 (0.659)
		PETS	0.321 (-1.375)	0.242 (-0.784)	0.175(-1.140)	0.052 (0.893)	0.103 (0.750)
Random	100	PerSim	**0.007 (1.000)**	**0.011 (0.999)**	**0.009 (0.998)**	**0.002 (1.000)**	**0.002 (1.000)**
		Unfactorized	0.051 (0.950)	0.027 (0.984)	0.062 (0.829)	0.021 (0.972)	0.017 (0.988)
		Vanilla	0.305 (-2.406)	0.228 (-2.013)	0.16 (-1.235)	0.077 (0.783)	0.212 (-0.6)
		PETS	0.327 (-2.458)	0.261 (-2.056)	0.183 (-1.678)	0.026 (0.957)	0.054 (0.943)
Random	50	PerSim	**0.008 (0.999)**	**0.016 (0.999)**	**0.006 (0.999)**	**0.002 (1.000)**	**0.002 (1.000)**
		Unfactorized	0.034 (0.991)	0.140 (0.482)	0.056 (0.919)	0.021 (0.990)	0.015 (0.996)
		Vanilla	0.251 (-0.501)	0.187 (0.302)	0.129 (-0.138)	0.11 (0.871)	0.174 (0.013)
		PETS	0.271 (-0.609)	0.214 (-0.932)	0.126 (0.523)	0.075 (0.676)	0.142 (-0.183)
Random	25	PerSim	**0.052 (0.895)**	**0.053 (0.918)**	**0.036 (0.944)**	**0.027 (0.928)**	**0.032 (0.942)**
		Unfactorized	0.055 (0.950)	0.312 (-0.270)	0.125 (0.502)	**0.027 (0.951)**	0.037 (0.981)
		Vanilla	0.238 (0.474)	0.223 (-2.009)	0.144 (-1.72)	0.101 (0.913)	0.185 (0.153)
		PETS	0.335 (-3.500)	0.241 (-3.196)	0.165 (-2.815)	0.084 (0.984)	0.123 (0.864)

Table B.8: Prediction Error: CartPole (Ablation Study)

Data	N	Method	0.0001	0.0005	0.001	0.0025	0.0035
Random	250	PerSim	**0.038 (0.836)**	**0.040 (0.937)**	**0.028 (0.992)**	**0.012 (0.998)**	**2.602 (-33.2)**
		Unfactorized	0.067 (0.576)	0.133 (0.349)	1.306 (-32.0)	0.095 (0.923)	3.317 (-66.0)
		Vanilla	0.201 (-0.414)	0.26 (-0.252)	0.468 (-0.358)	0.089 (0.69)	3.722 (-0.898)
		PETS	0.465 (-1.26)	0.254 (0.361)	0.381 (0.225)	0.146 (0.72)	3.08 (-0.679)
Random	100	PerSim	**0.007 (1.000)**	**0.011 (0.999)**	**0.009 (0.998)**	**0.002 (1.000)**	**0.002 (1.000)**
		Unfactorized	0.051 (0.950)	0.027 (0.984)	0.062 (0.829)	0.021 (0.972)	0.017 (0.988)
		Vanilla	0.22 (-0.757)	0.443 (-0.685)	1.168 (-0.667)	0.212 (-0.167)	3.979 (-1.112)
		PETS	0.319 (-0.961)	0.222 (0.233)	0.539 (-0.114)	0.156 (0.509)	3.409 (-0.878)
Random	50	PerSim	0.124 (-0.976)	**0.188 (-0.202)**	0.905 (-16.56)	**0.100 (0.87)**	2.76 (-39.60)
		Unfactorized	**0.116 (-0.362)**	0.385 (-3.88)	1.184 (-27.4)	0.223 (0.421)	**2.480 (-27.6)**
		Vanilla	0.18 (0.116)	0.346 (-0.358)	**0.373 (0.153)**	0.173 (0.352)	2.702 (-0.575)
		PETS	0.657 (-1.273)	0.577 (-0.34)	0.862 (-0.355)	0.272 (0.228)	3.716 (-1.044)
Random	25	PerSim	0.121 (-0.593)	**0.103 (0.642)**	0.941 (-14.6)	**0.080 (0.899)**	7.99 (-390)
		Unfactorized	**0.078 (0.476)**	0.229 (-0.518)	0.808 (-10.0)	0.170 (0.651)	**2.860 (-44.8)**
		Vanilla	0.263 (-0.531)	0.262 (-0.32)	**0.443 (0.019)**	0.231 (0.28)	3.279 (-0.791)
		PETS	0.848 (-1.307)	0.72 (-0.416)	1.455 (-0.739)	0.361 (0.088)	5.703 (-1.373)

Table B.9: Prediction Error: HalfCheetah (Ablation Study)

Data	N	Method	0.0001	0.0005	0.001	0.0025	0.0035
Random	250	PerSim	1.387 (0.891)	4.685 (0.598)	4.986 (0.751)	**1.433 (0.831)**	2.528 (0.791)
		Unfactorized	1.387 (0.891)	4.684 (0.598)	4.986 (0.751)	**1.433 (0.831)**	2.528 (0.791)
		Vanilla	4.811 (-0.51)	6.439 (0.255)	5.145 (0.717)	4.456 (-0.703)	4.571 (0.116)
		PETS	**1.063 (0.915)**	**3.98 (0.542)**	**4.432 (0.752)**	2.19 (0.568)	**2.113 (0.759)**
Random	100	PerSim	2.882 (0.498)	5.076 (0.530)	6.99 (0.467)	**2.000 (0.710)**	2.590 (0.766)
		Unfactorized	3.121 (0.441)	4.727 (0.586)	6.583 (0.531)	2.321 (0.546)	2.686 (0.766)
		Vanilla	4.955 (-0.488)	5.125 (0.465)	5.246 (0.694)	5.075 (-1.19)	4.605 (0.18))
		PETS	**1.451 (0.839)**	**3.996 (0.545)**	**4.399 (0.77)**	3.054 (0.224)	**2.24 (0.719)**
Random	50	PerSim	1.607 (0.869)	5.203 (0.503)	6.214 (0.614)	**1.509 (0.812)**	2.889 (0.754)
		Unfactorized	**1.539 (0.866)**	**4.614 (0.604)**	5.305 (0.700)	1.628 (0.782)	2.724 (0.758)
		Vanilla	5.612 (-1.195)	6.93 (-0.053)	**4.828 (0.72)**	5.917 (-2.568)	6.129 (-0.636)
		PETS	1.628 (0.825)	4.783 (0.46)	5.152 (0.73)	2.876 (0.464)	**2.486 (0.742)**
Random	25	PerSim	4.045 (0.532)	5.623 (0.461)	9.364 (0.395)	1.529 (0.808)	5.237 (0.524)
		Unfactorized	1.640 (0.844)	4.663 (0.593)	5.590 (0.665)	1.817 (0.728)	2.763 (0.753)
		Vanilla	**0.238 (0.474)**	**0.223 (-2.009)**	**0.144 (-1.72)**	**0.101 (0.913)**	**0.185 (0.153)**
		PETS	5.011 (-0.712)	5.109 (0.421)	7.426 (0.398)	4.317 (-1.08)	4.417 (0.127)

Table B.10: Average Reward: MountainCar (Ablation Study)

Data	N	Method	0.0001	0.0005	0.001	0.0025	0.0035
Random	250	PerSim	**-53.70** ±0.41	**-66.50** ±1.21	-116.6 ±3.18	-192.3 ±1.23	**-199.6** ±3.40
		Unfactorized	-54.00 ±0.22	-65.90 ±3.23	**-104.2** ±2.10	**-189.4** ±12.1	-199.7 ±10.4
		Unpersonalized	-65.00 ±8.14	-66.60 ±4.65	-162.4 ±25.5	-320.2 ±5.84	-371.7 ±6.72
		Vanilla	-59.90 ±1.61	-78.13 ±7.11	-332.6 ±41.3	-467.8 ±16.2	-500.0 ±0.00
		PETS	-73.66 ±3.15	-106.8 ±7.15	-473.8 ±37.0	-500.0 ±0.00	-500.0 ±0.00
		BCQ-P	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00
Random	100	PerSim	-55.00 ±0.70	**-67.02** ±1.76	**-110.3** ±3.46	-193.1 ±5.21	**-197.4** ±3.05
		Unfactorized	**-53.60** ±1.25	-69.60 ±0.50	-160.3 ±59.0	**-188.6** ±1.70	-198.1 ±3.50
		Unpersonalized	-67.50 ±10.2	-72.40 ±12.4	-149.4 ±27.4	-310.2 ±6.56	-321.1 ±11.4
		Vanilla	-66.00 ±5.06	-83.1 ±8.64	-307.13 ±96.17	-486.33 ±23.67	-500.0 ±0.00
		PETS	-79.33 ±4.36	-106.5 ±19.3	-418.4 ±27.3	-492.7 ±10.3	-499.9 ±0.14
		BCQ-P	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00
Random	50	PerSim	-54.20 ±0.90	-66.40 ±0.17	**-110.2** ±8.65	**-188.7** ±5.25	**-199.6** ±3.23
		Unfactorized	**-53.70** ±1.13	-66.70 ±4.55	-157.3 ±46.8	-382.7 ±54.5	-329.6 ±125.0
		Unpersonalized	-66.50 ±9.78	-68.10 ±4.93	-152.0 ±10.6	-286.3 ±8.64	-337.1 ±5.00
		Vanilla	-58.80 ±1.40	**-66.37** ±0.35	-131.8 ±17.6	-497.2 ±4.85	-500.0 ±0.00
		PETS	-67.86 ±7.15	-79.73 ±6.37	-290.5 ±76.9	-458.4 ±26.8	-498.5 ±2.12
		BCQ-P	-214.3 ±202	-348.6 ±186	-428.1 ±103	-500.0 ±0.00	-500.0 ±0.00
Random	25	PerSim	-56.80 ±1.81	**-67.50** ±2.81	**-139.9** ±29.5	**-246.8** ±39.1	**-243.8** ±47.1
		Unfactorized	**-54.50** ±0.62	-264.6 ±181	-194.7 ±30.6	-406.0 ±126	-343.1 ±114
		Unpersonalized	-64.10 ±5.46	-75.00 ±5.96	-184.6 ±57.7	-362.6 ±94.7	-412.9 ±62.4
		Vanilla	-62.33 ±3.34	-76.80 ±10.9	-275.7 ±63.9	-473.0 ±24.1	-496.5 ±4.90
		PETS	-67.26 ±6.95	-86.96 ±10.9	-331.0 ±121	-497.9 ±2.92	-500.0 ±0.00
		BCQ-P	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00	-500.0 ±0.00

Table B.11: Average Reward: HalfCheetah (Ablation Study)

Data	N	Method	0.0001	0.0005	0.001	0.0025	0.0035
Random	250	PerSim	1164.52 ±0.41	693.96 ±564	508.07 ±580	115.27 ±162	290.25 ±349
		Unfactorized	**2602.99** ±871	**2437.01** ±649	648.96 ±148	446.12 ±84.3	560.48 ±72.2
		Vanilla	277.55 ±62.6	335.03±278	240.44 ±531	278.9 ±98.0	362.2 ±124
		PETS	1005.57 ±557	1833.08 ±667	**986.15** ±210	**659.15** ±76.3	**2303.07** ±571
		BCQ-P	-1.78 ±0.36	-1.41 ±0.19	-1.83 ±0.21	-1.88 ±0.25	-1.83 ±0.26
Random	100	PerSim	**2046.39** ±46.0	813.44 ±95.3	**1115.01** ±376	**850.93** ±410	935.48 ±133
		Unfactorized	1062.57 ±14.2	**2108.38** ±24.3	485.72 ±83.0	203.77 ±93.76	1154.24 ±93.3
		Vanilla	168.9 ±154	131.11 ±111	93.2 ±182	176.54 ±131	415.9 ±151
		PETS	802.95 ±335	657.46 ±126	675.95 ±244	586.78 ±43.2	**1484.29** ±934
		BCQ-P	-1.63 ±0.08	-1.43 ±0.08	-1.77 ±0.12	-1.83 ±0.10	-1.86 ±0.10
Random	50	PerSim	776.98 ±529	**2072.3** ±59.9	-80.30 ±122	68.29 ±13.3	500 ±39.2
		Unfactorized	**2389.97** ±8.40	1668.04 ±662	**122.73** ±8.72	55.18 ±22.7	910.75 ±57.7
		Vanilla	103.28 ±113	388.75 ±137	-1.29 ±349	210.58 ±26.2	514.81 ±273
		PETS	957.3 ±351	667.4 ±180	100.37 ±124	**858.57** ±81.9	**1653.32** ±764
		BCQ-P	-1.71 ±0.28	-1.51 ±0.22	-1.80 ±0.21	-1.77 ±0.26	-1.91 ±0.17
Random	25	PerSim	1468.16 ±405	1339.04 ±36.0	**386.71** ±372	34.99 ±9.55	668.61 ±244
		Unfactorized	**2340.45** ±102	**1607.63** ±104	337.80 ±526	-7.12 ±69.4	440.39 ±80.7
		Vanilla	371.87 ±268	193.18 ±114	-2.84 ±112	**481.71** ±292	881.93 ±446
		PETS	940.62 ±18.2	629.25 ±568	6.22 ±41.1	263.1 ±237	**1187.63** ±653
		BCQ-P	-134.8 ±8.4	-210.49 ±53.4	-169.97 ±14.9	-178.52 ±22.9	-87.75 ±19.0

Table B.12: Average Reward: CartPole (Ablation Study)

Data	N	Method	0.0001	0.0005	0.001	0.0025	0.0035
Random	250	PerSim	189.37 ±15.1	174.53 ±23.3	184.1 ±3.48	170.1 ±12.8	174.17 ±23.0
		Unfactorized	**195.93** ±4.56	**200.0** ±0.0	**200.0** ±0.00	**198.37** ±2.82	188.70 ±9.30
		Vanilla	168.53 ±15.3	164.92 ±5.99	156.92 ±8.53	170.94 ±2.65	84.33 ±4.71
		PETS	131.03 ±11.5	**200.0** ±0.00	199.17 ±1.18	197.7 ±2.06	**189.83** ±7.19
		BCQ-P	18.09 ±0.18	4.61 ±0.01	8.34 ±0.03	10.99 ±0.04	6.33 ±0.02
Random	100	PerSim	**182.90** ±2.40	145.60 ±11.6	198.1 ±0.71	173.7 ±1.56	118.10 ±20.8
		Unfactorized	164.3 ±25.9	**186.4** ±19.2	194.7 ±7.49	**192.4** ±1.13	177.4 ±32.0
		Vanilla	146.67 ±5.31	165.16 ±3.85	126.43 ±5.04	157.12 ±4.10	64.43 ±16.2
		PETS	73.0 ±17.2	185.1 ±19.0	**199.0** ±1.41	174.7 ±17.0	**192.23** ±3.19
		BCQ-P	18.02 ±0.09	5.37 ±1.07	8.38 ±0.03	10.91 ±0.03	6.35 ±0.01
Random	50	PerSim	**199.75** ±0.35	**199.25** ±1.06	181.55 ±26.09	**195.35** ±5.59	**167** ±46.7
		Unfactorized	198.75 ±1.77	177.05 ±25.1	190.55 ±13.4	174.10 ±10.9	122.05 ±41.8
		Vanilla	165.9 ±13.73	174.02 ±8.94	152.66 ±24.1	175.37 ±10.8	74.43±15.2
		PETS	102.83 ±1.84	196.6 ±3.49	**193.33** ±2.05	193.5 ±2.03	149.97 ±31.1
		BCQ-P	17.99 ±0.16	5.84 ±1.52	9.34 ±1.38	11.68 ±1.04	7.77 ±2.00
Random	25	PerSim	133.0 ±29.0	195.8 ±0.14	**200.0** ±0.00	**200.0** ±0.00	**181.9** ±9.90
		Unfactorized	**197.5** ±2.12	**200.0** ±0.00	192.9 ±5.44	**200.0** ±0.00	147.85 ±14.4
		Vanilla	183.57 ±18.1	156.52 ±10.6	133.23 ±2.66	162.33 ±4.92	66.57 ±11.8
		PETS	89.57 ±11.2	189.7 ±9.13	194.8 ±6.66	163.33 ±17.44	160.4 ±20.4
		BCQ-P	24.44 ±9.47	13.3 ±4.27	16.01 ±8.36	20.92 ±9.51	12.58 ±5.18

Bibliography

Anish Agarwal, Devavrat Shah, and Dennis Shen. Synthetic interventions, 2020a.

Anish Agarwal, Abdullah Alomar, Varkey Alumootil, Devavrat Shah, Dennis Shen, Zhi Xu, and Cindy Yang. Persim: Data-efficient offline reinforcement learning with heterogeneous agents via personalized simulators, 2021.

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020b.

David J Aldous. Representations for partially exchangeable arrays of random variables. *Journal of Multivariate Analysis*, 11(4):581–598, 1981.

Boaz Barak and Ankur Moitra. Noisy tensor completion via the sum-of-squares hierarchy. In *Conference on Learning Theory*, pages 417–445. PMLR, 2016.

Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier, 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pages 617–629. PMLR, 2018.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

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

Douglas N Hoover. Relations on probability spaces and arrays of random variables. *Preprint, Institute for Advanced Study, Princeton, NJ*, 2, 1979.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *arXiv preprint arXiv:1906.08253*, 2019.

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

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11784–11794, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661. PMLR, 2019.

Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. Context-aware dynamics model for generalization in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 5757–5766. PMLR, 2020.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Provably good batch reinforcement learning without great exploration. *arXiv preprint arXiv:2007.08202*, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Andrea Montanari and Nike Sun. Spectral algorithms for tensor completion. *Communications on Pure and Applied Mathematics*, 71(11):2381–2425, 2018.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

Devavrat Shah and Christina Lee Yu. Iterative collaborative filtering for sparse noisy tensor estimation. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 41–45. IEEE, 2019.

Devavrat Shah, Dogyoon Song, Zhi Xu, and Yuzhe Yang. Sample efficient reinforcement learning via low-rank matrix estimation. *arXiv preprint arXiv:2006.06135*, 2020.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017b.

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

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.