Pareto-Efficient Decision Agents for Offline Multi-Objective Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

The goal of multi-objective reinforcement learning (MORL) is to learn policies that simultaneously optimize multiple competing objectives. In practice, an agent's preferences over the objectives may not be known apriori, and hence, we require policies that can generalize to arbitrary preferences at test time. In this work, we propose a new data-driven setup for *offline* MORL, where we wish to learn a preference-agnostic policy agent using only a finite dataset of offline demonstrations of other agents and their preferences. The key contributions of this work are two-fold. First, we introduce D4MORL, (D)atasets for MORL that are specifically designed for offline settings. It contains 1.8 million annotated demonstrations obtained by rolling out reference policies that optimize for randomly sampled preferences on 6 MuJoCo environments with 2-3 objectives each. Second, we propose Pareto-Efficient Decision Agents (PEDA), a family of offline MORL algorithms that builds and extends Decision Transformers (Chen et al., 2021) via a novel preference-and-return conditioned policy. Empirically, we show that PEDA closely approximates the behavioral policy on the D4MORL benchmark and provides an excellent approximation of the Pareto-front with appropriate conditioning, as measured by the hypervolume and sparsity metrics.

1 Introduction

We are interested in learning agents for multi-objective reinforcement learning (MORL) that optimize for one or more competing objectives. This setting is commonly observed in many real-world scenarios. For instance, an autonomous driving car might trade off high speed and energy savings depending on the user's preferences. The car will move fast for users prefer speed, or keep a steady speed for uses who try to save energy. One key challenge with MORL is that different users might have different preferences on the objectives and systematically exploring policies for each preference might be expensive or even impossible. In the online setting, prior work considers several approximations based on learning an ensemble of policies based on enumerating preferences (Mossalam et al., 2016a, Xu et al., 2020), or extensions of single-objective algorithms such as Q-learning to vectorized value functions (Yang et al., 2019).

We introduce the setting of *offline* multi-objective reinforcement learning for high-dimensional state and action spaces, where our goal is to train an MORL policy agent using an offline dataset of demonstrations with known preferences. Similar to the single-task setting, offline MORL can utilize auxiliary logged datasets to minimize interactions when deploying agents, thus improving data efficiency and minimizing interactions when deploying agents in high-risk settings. In addition to its practical utility, offline RL (Levine et al., 2020) has enjoyed major successes in the last few years (Kumar et al., 2020, Kostrikov et al., 2021, Chen et al., 2021) on challenging high-dimensional environments for continuous control and game-playing. Our contributions in this work are two-fold in introducing benchmarking datasets and a new family of MORL, as described below.

We introduce Datasets for Multi-Objective Reinforcement Learning (D4MORL), a collection of 1.8 million trajectories on 6 multi-objective MuJoCo environments (Xu et al., 2020) with two or three objectives each. For each environment in D4MORL, we collect demonstrations from 2 pretrained behavioral agents: *expert* and *amateur*, where the relative expertize is measured empirically via their hypervolumes. Furthermore, we also include 3 kinds of preference distributions with varying entropies to expose additional data-centric aspects for downstream benchmarking.

Next, we propose Pareto-Efficient Decision Agents (PEDA), a family of offline MORL algorithms that extends Decision Transformer (DT) (Chen et al., 2021) to the multi-objective setting. For MORL, we introduce a novel preference and return conditioned policy network and train it via a supervised learning loss. We find PEDA performs exceedingly well on D4MORL and closely approximates the reference Pareto-frontier of the behavioral policy used for data generation. In the multi-objective HalfCheetah environment, compared with an upper bound on the hypervolume of 5.79×10^6 achieved by the behavioral policy, PEDA achieves a hypervolume of 5.77×10^6 on the `Expert` and 5.76×10^6 on the `Amateur` datasets.

2 Preliminaries

Setup and Notation. We operate in the general framework of multi-objective Markov decision process (MOMDP) with linear preferences (Wakuta, 1995). An MOMDP is represented by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, f, \gamma \rangle$. At each timestep t, the agent with a current state $\boldsymbol{s}_t \in \mathcal{S}$ takes an action $\boldsymbol{a}_t \in \mathcal{A}$ to transition into a new state $\boldsymbol{s}_{t+1}$ with probability $\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and observes a reward vector $\boldsymbol{r}_t = \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t) \in \mathbb{R}^n$. Here, n is the number of objectives. The vector-valued return $\mathbf{R} \in \mathbb{R}^n$ of an agent is given by the discounted sum of reward vectors over a time horizon, $\mathbf{R} = \sum_t \gamma^t \boldsymbol{r}_t$. We also assume that there exists a linear utility function f and a space of preferences Ω that can map the reward vector $\boldsymbol{r}_t$ and a preference vector $\boldsymbol{\omega} \in \Omega$ to a scalar reward r_t, i.e., $r_t = f(\boldsymbol{r}_t, \boldsymbol{\omega}) = \boldsymbol{\omega}^\intercal \boldsymbol{r}_t$. The expected vector return of a policy π is given an $\boldsymbol{G}^\pi = [G_1^\pi, G_2^\pi, \ldots, G_n^\pi]^\intercal$ where the expected return of the i^{th} objective is given as $G_i^\pi = \mathbb{E}_{\boldsymbol{a}_{t+1} \sim \pi(\cdot|\boldsymbol{s}_t, \boldsymbol{\omega})}[\sum_t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)_i]$ for some predefined time horizon and preference vector $\boldsymbol{\omega}$. The goal is to train a multi-objective policy $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\omega})$ such that the expected scalarized return $\boldsymbol{\omega}^\intercal\ \boldsymbol{G}^\pi = \mathbb{E}[\boldsymbol{\omega}^\intercal \sum_t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)]$ is maximized.

Pareto Optimality. In MORL, one cannot optimize all objectives simultaneously, so policies are evaluated based on the *Pareto set* of their vector-valued expected returns. Consider a preference-conditioned policy $\pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\omega})$ that is evaluated for m distinct preferences $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_m$, and let the resulting policy set be represented as $\{\pi_p\}_{p=1,\ldots,m}$, where $\pi_p = \pi(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\omega} = \boldsymbol{\omega}_p)$, and $\boldsymbol{G}^{\pi_p}$ is the corresponding unweighted expected return. We say the solution $\boldsymbol{G}^{\pi_p}$ is *dominated* by $\boldsymbol{G}^{\pi_q}$ when there is no objective for which π_q is worse than π_p, i.e., $G_i^{\pi_p} < G_i^{\pi_q}$ for $\forall i \in [1, 2, \ldots, n]$. If a solution is not dominated, it is part of the Pareto set denoted as P. The curve traced by the solutions in a Pareto set is also known as the Pareto front. In MORL, our goal is to define a policy such that its empirical Pareto set is a good approximation of the true Pareto front. While we do not know the true Pareto front for many problems, we can evaluate the Pareto set P based on two metrics, *hypervolume* and *sparsity*. These two metrics are used by Xu et al., 2020. We define and discuss them in Appendix F.

3 D4MORL: Datasets for Offline Multi-Objective Reinforcement Learning

In offline RL, the goal of an RL agent is to learn the optimal policy using a fixed dataset without any interactions with the environment (Levine et al., 2020). This perspective brings RL closer to supervised learning, where the presence of large-scale datasets has been foundational for further progress in the field. Many such data benchmarks exist for offline RL as well; a notable one is the D4RL (Fu et al., 2020) benchmark for continuous control which has led to the development of several state-of-the-art offline RL algorithms (Kostrikov et al., 2021; Kumar et al., 2020; Chen et al., 2021) that can scale favorably even in high dimensions. To the best of our knowledge, there are no such existing benchmarks for offline MORL.

In this paper, we introduce Datasets for Multi-Objective Reinforcement Learning (D4MORL), a large-scale benchmark for offline MORL. Our benchmark consists of offline trajectories from 6 multi-objective MuJoCo environments including 5 environments with 2 objectives each viz. MO-Ant, MO-HalfCheetah, MO-Hopper, MO-Swimmer, MO-Walker2d, and one environment with three objectives: MO-Hopper-3obj. The objectives are conflicting for each environment, please see Appendix A for more details on the semantics of the target objectives. These environments were first introduced in Xu et al. (2020) for online MORL.

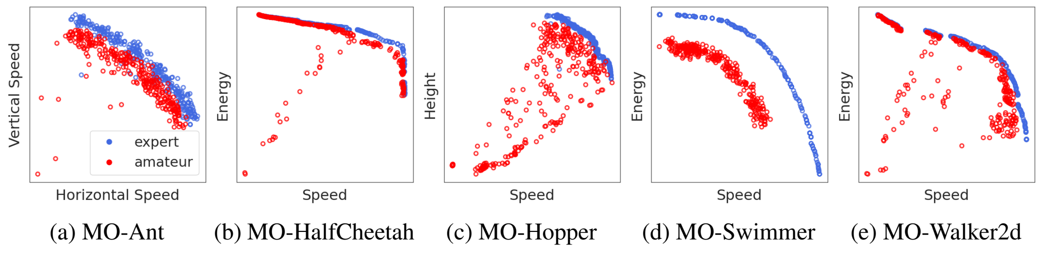


Figure 1: Empirical returns for expert and amateur trajectory datasets for the two-objective environments in D4MORL. For each environment and dataset, we randomly plot returns for 300 trajectories.

3.1 Trajectory Sampling

The quality of the behavioral policy used for sampling trajectories in the offline dataset is a key factor for benchmarking downstream offline RL algorithms. For a MOMDP, a reference expert policy (or set of policies) should reflect the optimal returns for all possible preferences in the preference space.

We use Prediction-Guided Multi-Objective Reinforcement Learning (PGMORL), a state-of-the-art MORL algorithm for defining reference expert policies. PGMORL (Xu et al., 2020) uses evolutionary algorithms to train an ensemble of policies to approximate the Pareto set. Each reference policy in the ensemble is associated with a unique preference; for any new preference, it is mapped to the closest preference in the reference set. Given a desired preference, we define two sets of behavioral policies:

1. **`Expert` Dataset:** We find the best reference policy in the policy ensemble, and always follow the action taken by the reference policy.
2. **`Amateur` Dataset:** As before, we first find the best reference policy in the policy ensemble. With a fixed probability p, we randomly perturb the actions of the reference policies. Otherwise, with probability $1 - p$, we take the same action as the reference policy. In D4MORL, we set $p = 0.65$.

Further details are described in Appendix C. In Figure 1, we show the returns of the trajectories rolled out from the expert and amateur policies for the 2 objective environments evaluated for a uniform sampling of preferences. We can see that the expert trajectories typically dominate the amateur trajectories, as desired. For the amateur trajectories, we see more diversity in the empirical returns for both objectives under consideration. The return patterns for the amateur trajectories vary across different environments providing a diverse suite of datasets in our benchmark.

3.2 Preference Sampling

The coverage of any offline dataset is an important factor in dictating the performance of downstream offline RL algorithms (Levine et al., 2020). For MORL, the coverage depends on both the behavioral MORL policy as well as the distribution of preferences over which this policy is evaluated. We use the following protocols for sampling from the preference space $\mathbf{\Omega}$. First, we restrict our samples to lie within a physically plausible preference space $\mathbf{\Omega}^* \subseteq \mathbf{\Omega}$ covered by the behavioral policy π_β. For instance, the MO-Hopper agent can never gain running rewards without leaving the floor. Thus, the preference of 100% running and 0% jumping is excluded from our preference sampling distribution.

Second, we are primarily interested in offline trajectories that emphasize competition between multiple objectives rather than focusing on a singular objective. To enforce this criteria, we define 3 sampling distributions concentrated around the centroid of the preference simplex. The largest spread distribution samples uniformly from $\mathbf{\Omega}^*$ and is denoted as **High-Entropy** (`High-H`). Next, we have a **Medium-Entropy** (`Med-H`) distribution specified via samples of Dirichlet distributions with large values of their concentration hyperparameters (aka α). Finally, we have a **Low-Entropy** (`Low-H`) distribution that is again specified via samples of Dirichlet distributions, but with low values of their concentration hyperparameters. We illustrate the samples for each of the preference distributions along with their empirical entropies and other details in Figure 2 from Appendix B. By ensuring different levels of coverage, we can test the generalizability of an MORL policy to preferences unseen during training. In general, we expect `Low-H` to be the hardest of the three distributions due to its restricted coverage, followed by `Med-H` and `High-H`.

4 Pareto-Efficient Decision Agents (PEDA)

In this section, we propose *Pareto-Efficient Decision Agents (PEDA)*, a family of offline multi-objective RL agents that aims to achieve Pareto-efficiency by extending Decision Transformers (Chen et al., 2021) into multi-objective setting. Our models extend from the architecture of Decision Transformers (DT) and its variant, Reinforcement Learning Via Supervised Learning (RvS).

4.1 Multi-Objective Reinforcement Learning Via Supervised Learning

In PEDA, our goal is to train a single preference-conditioned agent for offline MORL. By including preference conditioning, we enable the policy to be trained on arbitrary offline data, including trajectories collected from behavioral policies that are associated with alternate preferences. To parameterize our policy agents, we extend the DT and RvS architectures to include preference tokens and vector-valued returns. We refer to such preference-conditioned extensions of these architectures as MODT(P) and MORvS(P) respectively, which we describe next.

Preference Conditioning. Naively, we can easily incorporate the preference $\boldsymbol{\omega}$ into DT by adding this token for each timestep and feeding it a separate embedding layer. However, empirically we find that such a model design tends to ignore $\boldsymbol{\omega}$ and the correlation between the preferences and predicted actions is weak. Therefore, we propose to concatenate ω to other tokens before any layers in MODT(P). Concretely, we define $\boldsymbol{s}^* = \boldsymbol{s} \bigoplus \boldsymbol{\omega}$, $\boldsymbol{a}^* = \boldsymbol{a} \bigoplus \boldsymbol{\omega}$, and $\boldsymbol{g}^* = g \bigoplus \boldsymbol{\omega}$ where $\bigoplus$ denotes the concatenation operator. Hence, triples of $\boldsymbol{s}^*$, $\boldsymbol{a}^*$, $\boldsymbol{g}^*$ form the new trajectory. As for MORvS(P), we concatenate the preference with the states and the average RTGs by default and the network interprets everything as one single input.

Multi-Objective Returns-to-Go. Similar to RTG for the single objective case, we can define vector-valued RTG as $\boldsymbol{g}_t = \sum_{t'=t}^{T} \boldsymbol{r}_{t'}$ Given a preference vector $\boldsymbol{\omega}$, we can scalarize the total returns-to-go as $\hat{g}_t = \boldsymbol{\omega}^T \boldsymbol{g}_t$. In principle, the scalarized RTG $\hat{g}_t$ and preference vector $\boldsymbol{\omega}$ can recover the vector-valued RTG $\boldsymbol{g}_t$. However, empirically we find that directly feeding MODT/MORvS with the preference-weighted RTG vector $\boldsymbol{g}_t \odot \boldsymbol{\omega}$ is slightly preferable for stable training, where $\odot$ denotes the elementwise product operator.

Another unique challenge in the MORL setting concerns the scale of different objectives. Since different objectives can signify different physical quantities (e.g., energy and speed), the choice of scaling can influence policy optimization. We adopt a simple normalization scheme, where the returns for each objective are normalized by subtracting the minimum observed value for that objective and dividing it by the range of values (max-min). Note that the maximum and minimum are computed based on the offline dataset and hence, they are not necessarily the true min/max objective values. For evaluating the hypervolume and sparsity, we use the unnormalized values so that we can make comparisons across different datasets that may have different min/max boundaries.

5 Experiments

In this section, we evaluate the performance of PEDA on D4MORL benchmark. First, we investigate the benefits of preference conditioning by evaluating on MODT and MORvS where no preference information is available while still using multi-objective return-to-go. We denote our methods with preference conditioning as MODT(P) and MORvS(P). Then, we compare our methods with classic imitation learning that uses supervised loss to train a mapping from states (w/ or w/o concatenating preferences) to actions. We use behavioral cloning (BC) here and train multi-layer MLPs as models named BC (w/o preference) and BC(P) (w/ preference). Finally, we modify the network architecture of Conservative Q-Learning (CQL) (Kumar et al., 2020), a state-of-the-art standard offline RL method, such that it also takes preference vectors as inputs to learn a preference-conditioned Q-function $f^* : \mathcal{S} \times \mathcal{A} \times \Omega \rightarrow \mathbb{R}$. We denote this method as CQL(P).

5.1 Multi-objective Offline Benchmark

Table 1: Hypervolume performance on `High-H-Expert` dataset. PEDA variants MODT(P) and MORvS(P) always approach the expert behavioral policy. (B: Behavioral policy)

Environments	B	MODT(P)	MORvS(P)	BC(P)	CQL(P)	MODT	MORvS	BC	CQL
MO-Ant (10^6)	6.32	6.02±.02	**6.24±.07**	4.49±.30	5.76±.10	5.10±.16	5.05±.07	0.78±.57	3.52±.45
MO-HalfCheetah (10^6)	5.79	5.69±.01	**5.77±.00**	5.52±.04	5.63±.04	5.59±.05	4.56±.56	1.45±.04	3.78±.46
MO-Hopper (10^7)	2.09	1.92±.01	**1.98±.02**	1.40±.03	0.33±.39	1.60±.04	1.72±.04	0.82±.42	0.02±.02
MO-Hopper-3obj (10^{10})	3.73	**3.24±.09**	**3.27±.10**	2.29±.29	0.78±.24	1.22±.27	2.33±.14	0.03±.01	0.00±.00
MO-Swimmer (10^4)	3.25	3.16±.01	**3.22±.01**	**3.21±.01**	**3.22±.08**	2.49±.28	**3.19±.01**	1.81±.03	2.08±.08
MO-Walker2d (10^6)	5.21	4.84±.10	**5.15±.01**	3.47±.13	3.21±.32	0.61±.43	4.95±.06	0.07±.01	0.82±.62

Hypervolume. We compare hypervolume of our methods with all baselines on `Expert` datasets in Table 1. For the two-objective environments, we evaluate the models on 501 and 325 equally spaced preference points for two-objective and three-objective environments respectively. Each point is evaluated 5 times with random environment re-initialization, and the median value is recorded. Finally, all the results are based on average of 3 random seeds along with the standard error. In Table 1, we can see that MODT(P) and MORvS(P) outperform other baselines with higher mean, lower standard errors. They also approach the behavioral policy upper-bound.

Sparsity. We also evaluate sparsity performance. We only show results for preference-conditioned models since sparsity is only meaningful when models are sensitive to preference and are similar in hypervolume performance. Overall, MORvS(P) has the lowest sparsity in most environments, while at the same time featuring an outstanding hypervolume.

We include the the results and analysis on the `Amateur` datasets in Table 4 and Table 3 from Appendix G. These tables lead to the same conclusion.

6 Conclusion

In this paper, we introduced a novel dataset benchmark and algorithms for offline Multi-Objective Reinforcement Learning. We first proposed D4MORL, a dataset benchmark consisting of offline datasets generated from behavioral policies of different fidelities (expert/amateur) and rolled out under preference distributions with varying entropies (high/medium/low). Then, we propose PEDA, a family of offline MORL policy optimization algorithms based on decision transformers and show that by concatenating and embedding preference together with other inputs, our policies can effectively approximate the Pareto front of the underlying behavioral policy as measured by the hypervolume and sparsity metrics. Our proposed family includes MLP and transformer based variants, viz. the MORvS(P) and MODT(P), with MORvS(P) performing the best overall. In some scenarios, the learned policies can also generalize to higher target rewards that exceeds the data distribution. To our knowledge, the PEDA variants are the first *offline* MORL policies that supports both the continuous action control and continuous preference space.

Table 2: Sparsity (↓) performance on `High-H-Expert` dataset. MODT(P) and MORvS(P) have a lower density. BC(P) also has a competitive sparsity in smaller environments such as Swimmer.

Environments	MODT(P)	MORvS(P)	BC(P)	CQL(P)
MO-Ant ($\times 10^4$)	7.06±1.29	4.55±.59	32.1±12.2	**0.58±.10**
MO-HalfCheetah ($\times 10^4$)	1.44±.28	0.74±.03	1.69±.70	**0.10±0.00**
MO-Hopper ($\times 10^5$)	8.89±1.84	**1.91±.51**	23.7±20.1	2.84±2.46
MO-Hopper-3obj ($\times 10^5$)	1.42±.42	**1.11±.22**	**0.75±.15**	2.60±3.14
MO-Swimmer ($\times 1$)	12.1±6.49	5.00±.42	**3.79±1.03**	13.61±5.31
MO-Walker2d ($\times 10^4$)	5.28±.99	**1.91±.25**	46.9±23.2	6.23±10.71

REFERENCES

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline RL via supervised learning? In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=S874XAIpkR-`.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

Sandy Huang, Abbas Abdolmaleki, Giulia Vezzani, Philemon Brakel, Daniel J. Mankowitz, Michael Neunert, Steven Bohez, Yuval Tassa, Nicolas Heess, Martin Riedmiller, and Raia Hadsell. A constrained multi-objective reinforcement learning framework. In Aleksandra Faust, David Hsu, and Gerhard Neumann (eds.), *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 883–893. PMLR, 08–11 Nov 2022. URL `https://proceedings.mlr.press/v164/huang22a.html`.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning, 2021. URL `https://arxiv.org/abs/2110.06169`.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Romain Laroche, Paul Trichelair, and Rémi Tachet des Combes. Safe policy improvement with baseline bootstrapping, 2017. URL `https://arxiv.org/abs/1712.06924`.

Alessandro Lazaric and Mohammad Ghavamzadeh. Bayesian multi-task reinforcement learning. In *ICML-27th International Conference on Machine Learning*, pp. 599–606. Omnipress, 2010.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020. URL `https://arxiv.org/abs/2005.01643`.

Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. Multi-objective deep reinforcement learning, 2016a. URL `https://arxiv.org/abs/1610.02707`.

Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. Multi-objective deep reinforcement learning, 2016b. URL `https://arxiv.org/abs/1610.02707`.

Simone Parisi, Matteo Pirotta, and Marcello Restelli. Multi-objective reinforcement learning through continuous pareto manifold approximation. *Journal of Artificial Intelligence Research*, 57:187–227, 10 2016. doi: 10.1613/jair.4961.

Diederik M. Roijers, Shimon Whiteson, and Frans A. Oliehoek. Linear support for multi-objective coordination graphs. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '14, pp. 1297–1304, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450327381.

Harsh Satija, Philip S. Thomas, Joelle Pineau, and Romain Laroche. Multi-objective spibb: Seldonian offline policy improvement with safety constraints in finite mdps. In *Advances in Neural Information Processing Systems*, volume 34, pp. 2004–2017, 2021. URL `https://proceedings.neurips.cc/paper/2021/file/0f65caf0a7d00afd2b87c028e88fe931-Paper.pdf`.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Oldrich Vasicek. A test for normality based on sample entropy. *Journal of the Royal Statistical Society: Series B (Methodological)*, 38(1):54–59, 1976. doi: https://doi.org/10.1111/j.2517-6161.1976.tb01566.x. URL `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1976.tb01566.x`.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Kazuyoshi Wakuta. Vector-valued markov decision processes and the systems of linear inequalities. *Stochastic Processes and their Applications*, 56(1):159–169, 1995. ISSN 0304-4149. doi: https://doi.org/10.1016/0304-4149(94)00064-Z. URL `https://www.sciencedirect.com/science/article/pii/030441499400064Z`.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 1015–1022, 2007.

Runzhe Wu, Yufeng Zhang, Zhuoran Yang, and Zhaoran Wang. Offline constrained multi-objective reinforcement learning via pessimistic dual value iteration. In *Advances in Neural Information Processing Systems*, volume 34, pp. 25439–25451, 2021. URL `https://proceedings.neurips.cc/paper/2021/file/d5c8e1ab6fc0bfeb5f29aafa999cdb29-Paper.pdf`.

Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *Advances in Neural Information Processing Systems 32*. 2019.

Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007. doi: 10.1109/TEVC.2007.892759.

A Environment Description

All environments are the same as in Xu et al., 2020, except for when resetting the environment, each parameter are uniformly sampled from the $[x - 10^{-3}, x + 10^{-3}]$ with x being the default value. The only exception is that MO-Hopper and MO-Hopper-3objalways reset *height* the same way as 1.25, since this parameter directly relates to the reward function. All environments have a max episode length of 500 steps per trajectory, but the agent may also die before reaching the maximum length.

A.1 MO-Ant

The two objectives in MO-Ant are achieved distance in x and y axes respectively, denoted as $\boldsymbol{r} = [r_t^{vx}, r_t^{vy}]^\intercal$.

Consider the position of the agent is represented as (x_t, y_t) at time t and takes the action a_t. The agent has a fixed survival reward $r^s = 1.0$, $dt = 0.05$, and an action cost of $r^a = \frac{1}{2} \sum_k a_k^2$. The rewards are calculated as:

$$\begin{aligned} r_t^{vx} &= (x_t - x_{t-1}) \,/\, dt + r^s - r^a \\ r_t^{vy} &= (y_t - y_{t-1}) \,/\, dt + r^s - r^a \end{aligned} \tag{1}$$

A.2 MO-HalfCheetah

The two objectives in MO-HalfCheetah are running speed, and energy saving, denoted as $\boldsymbol{r} = [r_t^v, r_t^e]^\intercal$.

Consider the position of the agent is represented as (x_t, y_t) at time t and takes the action a_t. The agent has a fixed survival reward $r^s = 1.0$, fixed $dt = 0.05$, and an action cost of $r^a = \sum_k a_k^2$. The rewards are calculated as:

$$\begin{aligned} r_t^v &= \min\{4.0,\ (x_t - x_{t-1}) \,/\, dt\} + r^s \\ r_t^e &= 4.0 - r^a + r^s \end{aligned} \tag{2}$$

A.3 MO-Hopper

The two objectives in MO-Hopper are running and jumping, denoted as $\boldsymbol{r} = [r^r, r^j]^\intercal$.

Consider the position of the agent is represented as (x_t, h_t) at time t and takes the action a_t. The agent has a fixed survival reward $r^s = 1.0$, a fixed initial height as $h_{init} = 1.25$, a fixed $dt = 0.01$, and an action cost of $r^a = 2 \times 10^{-4} \ \sum_k a_k^2$. The rewards are calculated as:

$$\begin{aligned} r_t^r &= 1.5 \times (x_t - x_{t-1}) \,/\, dt + r^s - r^a \\ r_t^j &= 12 \times (h_t - h_{init}) \,/\, dt + r^s - r^a \end{aligned} \tag{3}$$

A.4 MO-Hopper-3obj

The physical dynamics are the same in MO-Hopper and MO-Hopper-3obj, while this environment has 3 objectives: running, jumping, and energy saving. The rewards are denoted as $\boldsymbol{r} = [r^r, r^j, r^e]^\intercal$.

Consider the position of the agent is represented as (x_t, h_t) at time t and takes the action a_t. The agent has a fixed survival reward $r^s = 1.0$, a fixed initial height as $h_{init} = 1.25$, a fixed $dt = 0.01$, and an action cost of $r^a = \sum_k a_k^2$. The rewards are calculated as:

$$\begin{aligned} r_t^r &= 1.5 \times (x_t - x_{t-1}) \,/\, dt + r^s \\ r_t^j &= 12 \times (h_t - h_{init}) \,/\, dt + r^s \\ r_t^e &= 4.0 - r^a + r^s \end{aligned} \tag{4}$$

A.5 MO-SWIMMER

The two objectives inMO-Swimmer are speed and energy saving, denoted as $\boldsymbol{r} = [r^v, r^e]^\intercal$.

Consider the position of the agent is represented as (x_t, y_t) at time t and takes the action a_t. The agent has a fixed $dt = 0.05$, and an action cost of $r^a = \sum_k a_k^2$. The rewards are calculated as:

$$\begin{aligned} r_t^v &= (x_t - x_{t-1}) \,/\, dt \\ r_t^e &= 0.3 - 0.15 \times r^a \end{aligned} \tag{5}$$

A.6 MO-WALKER2D

The objectives in MO-Walker2d are speed and energy saving, denoted as $\boldsymbol{r} = [r^v, r^e]^\intercal$.

Consider the position of the agent is represented as (x_t, y_t) at time t and takes the action a_t. The agent has a fixed survival reward $r^s = 1.0$, a fixed $dt = 0.008$, and an action cost of $r^a = \sum_k a_k^2$. The rewards are calculated as:

$$\begin{aligned} r_t^v &= (x_t - x_{t-1}) \,/\, dt + r^s \\ r_t^e &= 4.0 - r^a + r^s \end{aligned} \tag{6}$$

To uniformly sample the `High-H` data from the entire preference space, the problem is equivalent to sampling from a n-dimensional simplex, where n is the number of objectives. The resulting sampling is:

$$\boldsymbol{\omega}^{\text{high}} \sim ||\boldsymbol{f}_{\text{exp}}(\,\cdot\,, \lambda = 1)||_1 \tag{7}$$

We take the 1-norm following the exponential distribution to make sure each preference add up to 1. When $\boldsymbol{\Omega}^* \neq \boldsymbol{\Omega}$, we perform rejection sampling to restrict the range.

To sample the `Med-H` and `Low-H` data, we first sample $\boldsymbol{\alpha}$ from a non-negative uniform distribution, then sample the corresponding Dirichlet preference. Here, we sample a *different* alpha to make sure the center of the Dirichlet changes and thus allows more variation.

$$\begin{aligned} \boldsymbol{\omega}^{\text{med}} &\sim \boldsymbol{f}_{\text{Dirichlet}}(\boldsymbol{\alpha}) \text{ ; where } \boldsymbol{\alpha} \sim \text{Unif}(0, 10^6) \\ \boldsymbol{\omega}^{\text{low}} &\sim \boldsymbol{f}_{\text{Dirichlet}}(\boldsymbol{\alpha}) \text{ ; where } \boldsymbol{\alpha} \sim \text{Unif}(1/3 \times 10^6, 2/3 \times 10^6) \end{aligned} \tag{8}$$

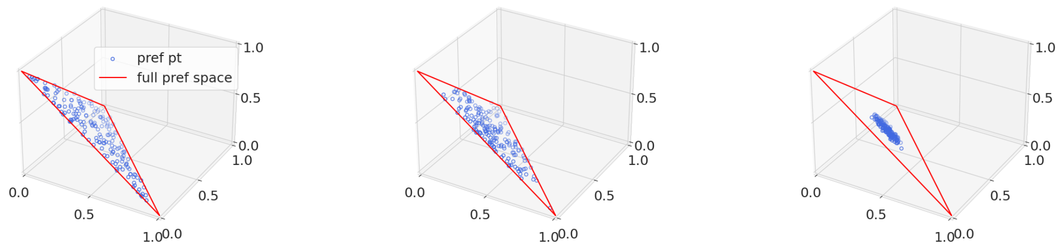
For sampling from behavioral policy consists of a group of single-objective policies $\pi_\beta = \{\pi_1, \ldots, \pi_B\}$ with B being the total number of candidate policies, we recommend first find the expected unweighted raw rewards $\boldsymbol{G}^{\pi_1}, \ldots, \boldsymbol{G}^{\pi_B}$. Then, find the estimated $\hat{\boldsymbol{\omega}}^{\pi_1}, \ldots, \hat{\boldsymbol{\omega}}^{\pi_B}$ by letting $\hat{\omega}_i^{\pi_b} = \frac{G_i^{\pi_b}}{\sum_{j=1}^n G_j^{\pi_b}}$, which represents the estimated preference on ith objective of bth candidate policy. For a sampled preference $\boldsymbol{\omega} \sim \boldsymbol{\Omega}^*$, use the policy that provides the smallest euclidean distance $d(\boldsymbol{\omega}, \hat{\boldsymbol{\omega}}^{\pi_b})$. Empirically, this means picking the candidate policy that has the expected reward ratio closest to $\boldsymbol{\omega}$.

B PREFERENCE SAMPLING DETAILS

To uniformly sample the `High-H` data from the entire preference space, the problem is equivalent to sampling from a n-dimensional simplex, where n is the number of objectives. The resulting sampling is:

$$\boldsymbol{\omega}^{\text{high}} \sim ||\boldsymbol{f}_{\text{exp}}(\,\cdot\,, \lambda = 1)||_1 \tag{9}$$

We take the 1-norm following the exponential distribution to make sure each preference add up to 1. When $\boldsymbol{\Omega}^* \neq \boldsymbol{\Omega}$, we perform rejection sampling to restrict the range.



(a) High Entropy (H=-0.2) (b) Med Entropy (H=-0.35) (c) Low Entropy (H=-1.54)

Figure 2: Illustration of the preference distributions for 3 objectives. Entropy is estimated on 50K preference samples using the Vasicek estimator in Scipy (Vasicek, 1976, Virtanen et al., 2020).

To sample the `Med-H` and `Low-H` data, we first sample $\boldsymbol{\alpha}$ from a non-negative uniform distribution, then sample the corresponding Dirichlet preference. Here, we sample a *different* alpha to make sure the center of the Dirichlet changes and thus allows more variation.

$$\begin{aligned} \boldsymbol{\omega}^{\text{med}} &\sim \boldsymbol{f}_{\text{Dirichlet}}(\boldsymbol{\alpha}) \text{ ; where } \boldsymbol{\alpha} \sim \text{Unif}(0, 10^6) \\ \boldsymbol{\omega}^{\text{low}} &\sim \boldsymbol{f}_{\text{Dirichlet}}(\boldsymbol{\alpha}) \text{ ; where } \boldsymbol{\alpha} \sim \text{Unif}(1/3 \times 10^6, 2/3 \times 10^6) \end{aligned} \tag{10}$$

For sampling from behavioral policy consists of a group of single-objective policies $\pi_\beta = \{\pi_1, \ldots, \pi_B\}$ with B being the total number of candidate policies, we recommend first find the expected unweighted raw rewards $\boldsymbol{G}^{\pi_1}, \ldots, \boldsymbol{G}^{\pi_B}$. Then, find the estimated $\hat{\boldsymbol{\omega}}^{\pi_1}, \ldots, \hat{\boldsymbol{\omega}}^{\pi_B}$ by letting $\hat{\omega}_i^{\pi_b} = \frac{G_i^{\pi_b}}{\sum_{j=1}^n G_j^{\pi_b}}$, which represents the estimated preference on i^{th} objective of b^{th} candidate policy. For a sampled preference $\boldsymbol{\omega} \sim \boldsymbol{\Omega}^*$, use the policy that provides the smallest euclidean distance $d(\boldsymbol{\omega}, \hat{\boldsymbol{\omega}}^{\pi_b})$. Empirically, this means picking the candidate policy that has the expected reward ratio closest to $\boldsymbol{\omega}$.

C Expert & Amateur Details

In `Expert` collection, we sample trajectories using the fully-trained behavioral policy π_β. In this paper, we use PGMORL by Xu et al., 2020 as π_β

$$\boldsymbol{a}_{\boldsymbol{t+1}}^{\text{expert}} = \pi_\beta(\boldsymbol{a}|\boldsymbol{s} = \boldsymbol{s_t}, \boldsymbol{\omega} = \boldsymbol{\omega_t}) \tag{11}$$

In the `Amateur` collection, the policies has a 35% chance being stochastic on top of the expert collection. Actions has a chance being stochastic, during which it is scaled from the expert action, as following:

$$\boldsymbol{a}_{\boldsymbol{t+1}}^{\text{amateur}} = \begin{cases} \boldsymbol{a}_{\boldsymbol{t+1}}^{\text{expert}} & 35\% \\ \boldsymbol{a}_{\boldsymbol{t+1}}^{\text{expert}} \times \text{Unif}(0.35, 1.65) & 65\% \end{cases} \tag{12}$$

In the MO-Swimmer environment only, we let actions has a 35% chance to be a uniform random sample from the entire action space rather than being the same as expert to increase variance and achieve a performance similar to amateur. The resulting strategy for MO-Swimmer is:

$$\boldsymbol{a}_{\boldsymbol{t+1}}^{\text{amateur}} = \begin{cases} \text{Unif}(\mathcal{A}) & 35\% \\ \boldsymbol{a}_{\boldsymbol{t+1}}^{\text{expert}} \times \text{Unif}(0.35, 1.65) & 65\% \end{cases} \tag{13}$$

D Related Work

Multi-Objective Reinforcement Learning Predominant works in MORL focus on the online setting where the goal is to train agents that can generalize to arbitrary preferences. This can be achieved by training a single preference-conditioned policy (Yang et al., 2019; Parisi et al., 2016), or an ensemble of single-objective policies for a finite set of preferences (Mossalam et al., 2016b; Xu et al., 2020; Zhang & Li, 2007). Many of these algorithms consider vectorized variants of standard algorithms such as Q-learning (Mossalam et al., 2016b; Yang et al., 2019), often augmented with strategies to guide the policy ensemble towards the Pareto front using evolutionary or incrementally updated algorithms (Xu et al., 2020; Zhang & Li, 2007; Mossalam et al., 2016b; Roijers et al., 2014; Huang et al., 2022). In contrast to these online MORL works, our focus is on learning a single policy that works for all preferences using only offline datasets.

There are also a few works that study offline MORL. Wu et al., 2021 propose a provably efficient algorithm based on dual gradient ascent. Satija et al., 2021 study learning of safe policies by extending the approach of Laroche et al., 2017 to the offline MORL setting. However, these algorithms assume the environments are primarily tabular MDPs with finite state and action spaces, while we are interested in high-dimensional MDPs with continuous states and actions.

Multi-Task Reinforcement Learning MORL is also closely related to multi-task reinforcement learning, where every task can be interpreted as a distinct objective. There is an extensive body of work in learning multi-task policies both in the online and offline setups (Wilson et al., 2007; Lazaric & Ghavamzadeh, 2010; Teh et al., 2017) inter alia. However, the key difference is that typical MTRL benchmarks and algorithms do not consider solving multiple tasks that involve inherent trade-offs. Consequently, there is no notion of Pareto efficiency and an agent can simultaneously excel in all the tasks without accounting for user preferences.

Reinforcement Learning Via Supervised Learning A body of recent works have formulated offline reinforcement learning as an autoregressive sequence modeling problem using Decision Transformers (DT) (Chen et al., 2021). The key idea in DT is to learn a transformer-based policy that conditions on the past history and a dynamic estimate of the returns (a.k.a. returns-to-go). Follow-up works consider simpler variants that rely only on multi-layer perceptrons (Emmons et al., 2022).

E TRANING DETAILS

Common hyper-parameters have the same values across all models, except for the learning rate scheduler and warmup steps. In MODT family, inputs are embedded by a 1-layer MLP into Hidden Size, and n_layer represents the number of transformer blocks; in BC family, n_layer represents the number of MLP layers to embedding each input; in MORvS and MORvS(P) family, the default embedding strategy are used, please check Emmons et al., 2022. Here, we consider MORvS and MORvS(P) both have context length of 1 because they only use the current state to predict the next action, whereas MODT and BC use the past 20.

E.1 PARAMETERS

Hyperparameter	MODT	MORvS	BC
Context Length - K	20	1	20
Batch Size		64	
Hidden Size		512	
Learning Rate		1e-4	
Weight Decay		1e-3	
Dropout		0.1	
n_layer		3	
Optimizer		AdamW	
Loss Function		MSE	
LR Scheduler	lambda	None	lambda
Warmup Steps	10000	N/A	4000
Activation		ReLU	

E.2 Training Steps

Dataset Name	MODT Steps	RvS/BC Steps
MO-Ant	20K	200K
MO-HalfCheetah	80K	200K
MO-Hopper	400K	200K
MO-Hopper-3obj	400K	200K
MO-Swimmer	260K	200K
MO-Walker2d	360K	200K

The exception is that we train MO-Swimmer and MO-Hopper-3obj under `Low-H-Amateur` for 40K and 120K steps respectively using MODT.

E.3 Attempted MODT and MODT(P) Architectures

Here are all the MODT architectures we tried in our experiments. We followed *Case 4* in all of the experiment results as we find that it makes the model follows closely to the given preference.

1. Consider $\boldsymbol{\omega}$ as an independent token of the causal transformer.
2. Train a separate embedding for $\boldsymbol{\omega}$, concatenate the embeddings to get $f_{\phi_{\boldsymbol{s}}}(\boldsymbol{s}) \bigoplus f_{\phi_{\boldsymbol{\omega}}}(\boldsymbol{\omega})$, $f_{\phi_{\boldsymbol{a}}}(\boldsymbol{a}) \bigoplus f_{\phi_{\boldsymbol{\omega}}}(\boldsymbol{\omega})$, and $f_{\phi_{\boldsymbol{g}}}(\boldsymbol{g}) \bigoplus f_{\phi_{\boldsymbol{\omega}}}(\boldsymbol{\omega})$ then pass into the transformer.
3. Add another MLP layer on top of the *Case 2* after concatenation, then pass output into the transformer.
4. Concatenate $\boldsymbol{\omega}$ to other tokens before any layers. This means we have $\boldsymbol{s}^* = \boldsymbol{s} \bigoplus \boldsymbol{\omega}$, $\boldsymbol{a}^* = \boldsymbol{a} \bigoplus \boldsymbol{\omega}$, and $\boldsymbol{g}^* = \boldsymbol{g} \bigoplus \boldsymbol{\omega}$.

F Evaluation Metrics

Hypervolums & Sparsity We introduce the hypervolume and sparsity metric in this section with a visualization.

Definition 1 (Hypervolume)**.** Hypervolume $\mathcal{H}(P)$ measures the space or volume enclosed by the solutions in the Pareto set P:

$$\mathcal{H}(P) = \int_{\mathbb{R}^m} \mathbb{1}_{H(P)}(z)\, dz,$$

where $H(P) = \{z \in Z | \exists i : 1 \leqslant i \leqslant |P|, \boldsymbol{r} \preceq \boldsymbol{z} \preceq P(i)\}$. $P(i)$ is the i^{th} solution in P, $\preceq$ is the dominance relation operator, and $\mathbb{1}_{H(P)}(z)$ equals 1 if $z \in H(P)$ and 0 otherwise. Higher hypervolumes are better.

Definition 2 (Sparsity)**.** Sparsity $\mathcal{S}(P)$ measures the density of the Pareto front covered by a Pareto set P:

$$\mathcal{S}(P) = \frac{1}{|P|-1} \sum_{i=1}^{n} \sum_{k=1}^{|P|-1} (\tilde{P}_i(k) - \tilde{P}_i(k+1))^2,$$

where $\tilde{P}_i$ represents a list sorted as per the values of the i^{th} objective in P and $\tilde{P}_i(k)$ is the k^{th} value in the sorted list. Lower sparsity is better.

See Figure 3 for an illustration and Appendix F for discussion on other possible metrics.

Other Metrics. Among a variety of metrics for MORL, we use Hypervolume and Sparsity (SP) to benchmark models in this paper for several reasons. First, metrics such as the ϵ-metric require prior knowledge of the *true* Pareto Fronts, which are not available for our MuJoCo Environments. Second, due to limited resources, we only assume linear reward function and cannot collect real-time user feedback, thus utility-based metrics such as *expected utility metric* (EUM) are not applicable. Finally, it is easier to benchmark both our offline agents and behavioral policy on these metrics, allowing for informative comparisons.

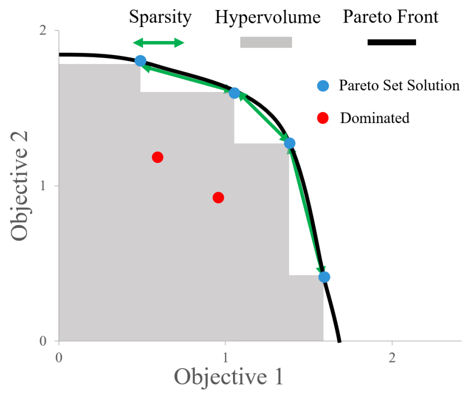


Figure 3: Illustration of the Hypervolume and Sparsity Metrics. Only undominated solutions (i.e., the Pareto set) are used for calculating the evaluation metrics.

G Amateur Results

Due to space limit, we present the results for `Amateur` datasets with `High-H` entropy for all environments here in Table 3 and Table 4. We can achieve the same conclusion that PEDA variants including MODT(P) and MORvS(P) have the best hypervolume and sparsity performance against other baselines. They also can approach or exceed the `Amateur` behavioral policy, which shows that our algorithms are able to learn strong policies from considerable amount of suboptimal data rather than merely imitation learning agents.

Table 3: Hypervolume performance on `High-H-Amateur` dataset. PEDA variants still approach or even exceed the behavioral policy even when a considerable portion of data is suboptimal. MODT(P) and MORvS(P) still present to be the strongest models and outperform other baselines. (B: Behavioral policy)

Environments	B	MODT(P)	MORvS(P)	BC(P)	CQL(P)	MODT	MORvS	BC	CQL
MO-Ant (10^6)	5.61	5.60±.03	**5.94±.04**	4.35±.07	5.62±.23	3.34±.71	4.91±.06	2.34±.09	2.80±.68
MO-HalfCheetah (10^6)	5.68	5.64±.01	**5.77±.00**	5.26±.06	5.64±.04	5.51±.01	5.28±.35	3.14±.38	4.41±.08
MO-Hopper (10^7)	1.97	**1.82±.02**	1.80±.03	1.56±.01	1.15±.24	1.54±.08	1.42±.04	0.48±.48	0.00±.06
MO-Hopper-3obj (10^{10})	3.09	2.63±.17	**2.98±.06**	2.31±.16	0.59±.42	1.72±.12	1.57±.08	0.13±.04	0.10±.16
MO-Swimmer (1)	2.11	**2.83±.04**	2.76±.02	2.81±.04	1.69±.93	0.67±0.1	2.78±.01	0.46±.14	0.74±.47
MO-Walker2d (10^4)	4.99	3.17±.14	**4.96±.02**	2.93±.75	1.78±.33	2.62±.81	4.21±.27	1.19±.21	0.76±.81

H Pareto Front & Return Distribution

Pareto front approximation. We ablate how well the PEDA variants and other baselines can approximate the Pareto front through conditioning on different preference points. We show the

Table 4: Sparsity (↓) performance on `High-H-Amateur` dataset. We can see that all models still have a similar or stronger sparsity performance when trained on amateur datasets. Furthermore, MORvS(P) still presents the strongest performance. While BC(P) has strong performance in MO-Hopper-3obj and MO-Swimmer, it also fails to give a dense solution in other environments and has a higher standard error.

Environments	MODT(P)	MORvS(P)	BC(P)	CQL(P)
MO-Ant ($\times 10^4$)	6.56±.45	4.96± 1.1	11.7±6.45	**1.06±.28**
MO-HalfCheetah ($\times 10^4$)	**0.81±.52**	**0.31±.01**	2.18±.49	0.45±.27
MO-Hopper ($\times 10^5$)	**1.54±.69**	**1.75±.52**	9.57±8.64	3.30±5.25
MO-Hopper-3obj ($\times 10^5$)	4.59±1.51	**1.04±.28**	**0.79±.19**	2.00±1.72
MO-Swimmer ($\times 1$)	**2.74±1.51**	**1.53±.10**	**1.21±.07**	8.87±6.24
MO-Walker2d ($\times 10^4$)	30.7±13.6	**2.10±.02**	154±144	7.33±5.89

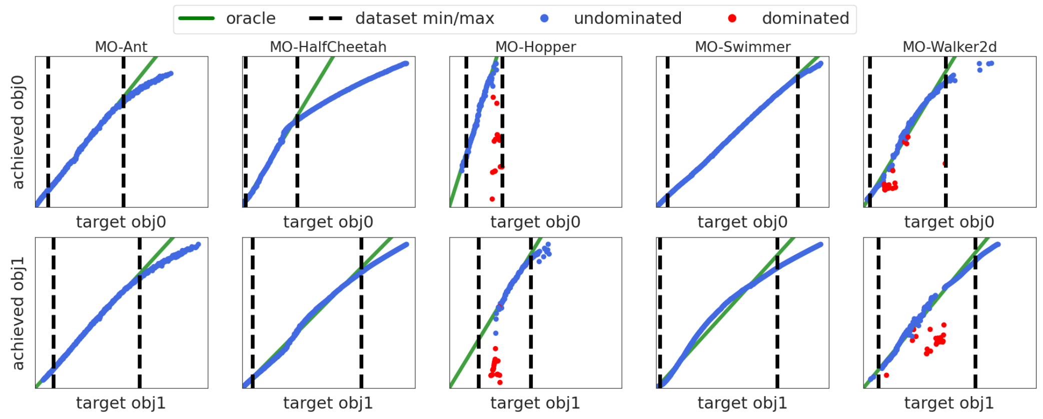


Figure 4: We show that MORvS(P) Model can follow the given target reward that are within the dataset's minimum and maximum record. The plots are for all of the two-objective environments. In addition, MO-Hopper and MO-Walker2d present to be the most challenging environments for PEDA variants, featuring more dominated solutions than other environments.

results in Figure 5, where we can see that the MODT(P) and MORvS(P) can approximate the Pareto front, while having some dominated points colored in pink mostly in the MO-Hopper and MO-Walker2d environments. The results are based on average of 3 seeds.

The distribution of returns. We ablate how well MODT(P) and MORvS(P) follow their given target return, based on a normalized and weighted value. We present the results in Figure 4 for MORvS(P) under `High-H-Expert` datasets. Here, we see that the models follow the oracle line nicely when conditioned on target within the dataset distribution, and generalize to targets outside of the dataset distribution as well. We present the Pareto Set visualizations for all of our models trained under each `High-H` dataset in Figure 5. All subplots are based on 1 seed, in which we evaluate the model using 501 equally spaced preference points in 2 objective environment and 351 equally spaced preference in the 3 objective environment from the *full* preference space. Since the environments are stochastically initialized, we evaluate 5 times at each preference point and take the median value. We here allow a small tolerance on coloring the dominated points.

If a preference point is within the achievable preference Ω^* but the solution is dominated, we color it in red. Since our models condition on continuous preference points and environments are initializaed stochastically, we give a small tolerance (3%-8%) for points to be colored in blue. The hypervolume and sparsity metric, on the other hand, can based on strictly undominated solutions without tolerance.

I Medium & Low Entropy Dataset Training

We train on the Medium-Entropy and Low-Entropy datasets for the MO-HalfCheetah environment. Overall, models have a similar performance under `Med-H` and `High-H` datasets, but suffers when only trained on `Low-H`. We present the results in Table 5, in which we illustrate that the `Low-H` dataset has a worse expert and amateur performance due to reduced variability on preference. However, MODT(P) and MORvS(P) are still able to get close or exceed in hypervolume on all datasets, which showcase the effectiveness of PEDA as an efficient MORL policy. Results are based on average of 3 seeds with the standard error given.

J Training with 1-dim RTG

We attempted to train MODT and RvS with 1-dim return-to-go rather than a separate rtg for each objective. According to results on MO-HalfCheetah and the `High-H` datasets in 6, using multi-dimensional rtg enhances the performance of MODT(P), and are about the same for MORvS(P) when preference are concatenated to states. However, it reduces standard error significantly in both

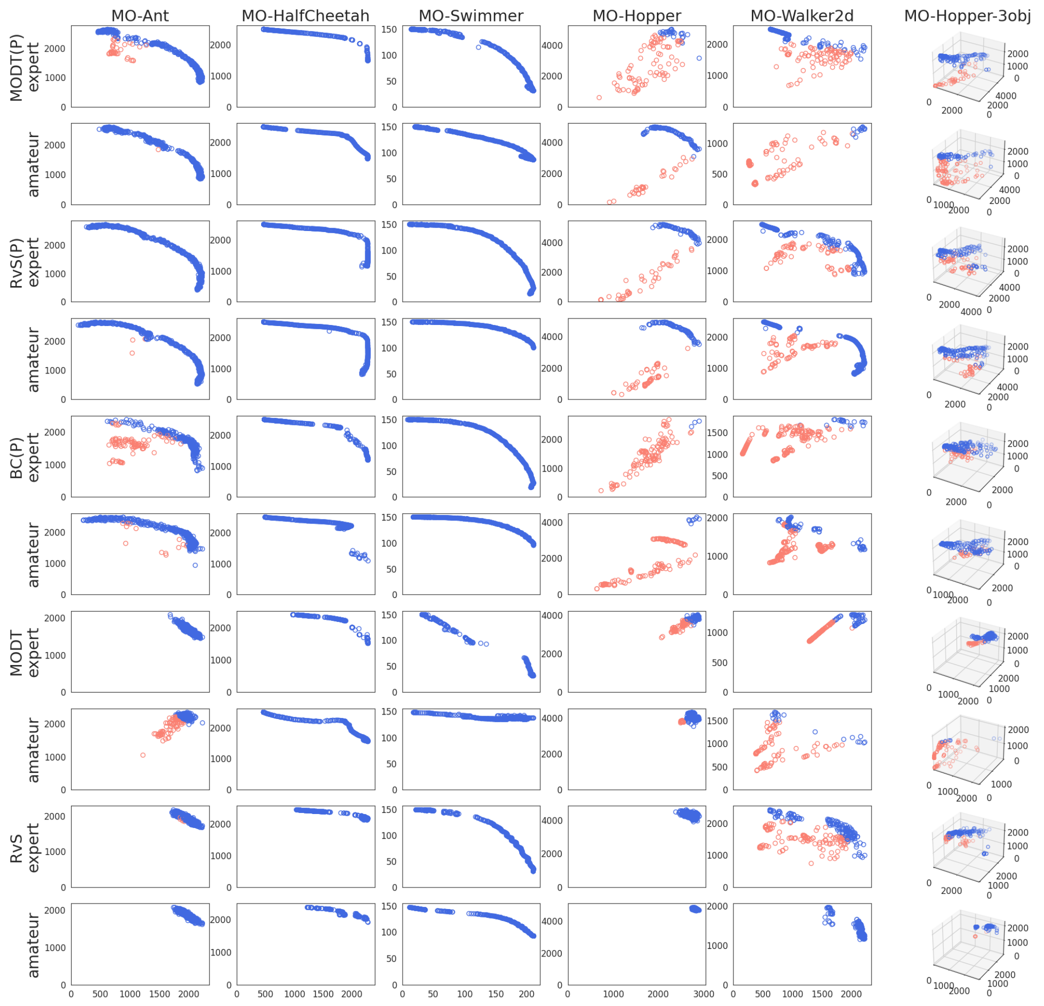


Figure 5: Full Pareto Visulization for all PEDA variants and baselines on `High-H` datasets. We notice that MO-Hopper and MO-Walker2d to be the harder environments, having significant of dominated points for even the preference conditioned models. On other environments, however, the PEDA variants produce good results for preference conditioned models, while other baselines fail at a higher chances.

Table 5: We run MO-HalfCheetah environment on all the of PEDA variants on the full datasets including `Med-H` and `Low-H`. (B: Behavioral Policy)

Dataset ($\times 10^6$)	B	MODT(P)	MORvS(P)	BC(P)	MODT	MORvS	BC
`Med-H-Expert`	5.79	5.69±.02	**5.77±.01**	3.60±.04	4.66±.05	4.09±.10	1.61±.12
`Med-H-Amateur`	5.69	5.61±.01	**5.77±.00**	1.63±.26	3.84±.44	4.61±.03	3.63±.12
`Low-H-Expert`	4.68	**4.73±.01**	**4.83±.01**	4.51±.02	4.09±.23	3.94±.05	3.18±.33
`Low-H-Amateur`	4.21	**4.74±.02**	**4.83±.02**	4.67±.07	2.54±.05	4.04±.18	4.09±.09
`High-H-Expert`	5.79	5.69±.01	**5.77±.00**	5.52±.04	5.59±.05	4.56±.56	1.45±.04
`High-H-Amateur`	5.68	5.64±.01	**5.77±.00**	5.36±.06	5.51±.01	5.28±.35	3.14±.38

Table 6: We explore the importance of using multi-dimensional rtg instead of a one-dimensional rtg by taking the weighted sum of objectives. (B: Behavioral Policy)

Setting	Dataset ($\times 10^6$)	B	MODT(P)	MORvS(P)	MODT	MORvS
1-dim rtg	`High-H-Expert`	5.79	5.54±.09	**5.78**±.03	4.34±.13	3.06±.42
	`High-H-Amateur`	5.68	**5.69±.01**	**5.77**±.01	2.81±1.2	2.52±.19
mo rtg	`High-H-Expert`	5.79	**5.69±.01**	**5.77±.00**	5.59±.05	4.56±.56
	`High-H-Amateur`	5.68	**5.64±.01**	**5.77±.00**	5.51±.01	5.28±.35

MODT(P) and MORvS(P). In the naive models when preference are not concatenated to states, using a multi-dimensional rtg helps to achieve a much more competitive hypervolume. We thus believe multi-dimensional rtg conveys important preference information when the model doesn't directly take preference as an input. Results are based on average of 3 seeds with the standard error given.

K Overall Data Generation Pipeline

The pseudocode for generating the dataset is described in Algorithm 1. Given a preference distribution, we first sample a preference $\boldsymbol{\omega}$ and query the closest behavioral policy in either the amateur/expert ensemble matching $\boldsymbol{\omega}$. We rollout this policy for T time steps (or until the end of an episode if sooner) and record the state, action, and reward information. Each trajectory in our dataset is represented as:

$$\tau =< \boldsymbol{\omega}, \boldsymbol{s}_1, \boldsymbol{a}_1, \boldsymbol{r}_1, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T, \boldsymbol{r}_T >$$

For every environment in D4MORL, we collect 50K trajectories of length $T = 500$ for both expert and amateur trajectory distributions under each of the 3 preference distributions. Overall, this results in a total of 1.8M trajectories over all 6 environments, which corresponds to roughly 867M time steps. Please refer to Table 7 in for additional statistics on the dataset.

Algorithm 1 Data Collection in D4MORL

```
procedure COLLECT(prefDist, nTraj, env, pretrainedAgents, T)
    agents = pretrainedAgents
    prefs = prefDist(nTraj)
    all_trajs = []
    for $\boldsymbol{\omega}$ in prefs do
        agent = closestAgent(agents, $\boldsymbol{\omega}$)
        $\boldsymbol{s}$ = env.reset()
        done = False
        $\tau$ = [$\boldsymbol{\omega}$]
        t = 0
        while (NOT done) AND (t < T) do
            $\boldsymbol{a}$ = agent.get_action($\boldsymbol{s}$)
            $\boldsymbol{s}'$, done, $\boldsymbol{r}$ = env.step($\boldsymbol{a}$)
            append $\boldsymbol{s}$, $\boldsymbol{a}$, $\boldsymbol{s}'$, $\boldsymbol{r}$ to $\tau$
            $\boldsymbol{s} = \boldsymbol{s}'$
            t = t + 1
        append $\tau$ to all_trajs
    return all_trajs
```

Table 7: A comprehensive view of the dataset. All datasets have a 500 maximum step per trajectorie, and 50K trajectories are collection under each setting. We also how the average step per trajectory for expert and amateur respectively, where we see amateur's step are always shorter or same as expert, making the return to be lower.

	max step per traj	expert avg. step per traj	amateur avg. step per traj	trajectories per dataset
MO-Ant	500	500	500	50K
MO-HalfCheetah	500	499.91	482.11	50K
MO-Hopper	500	499.94	387.72	50K
MO-Hopper-3obj	500	499.99	442.87	50K
MO-Swimmer	500	500	500	50K
MO-Walker2d	500	500	466.18	50K