PARETO-EFFICIENT DECISION AGENTS FOR OFFLINE MULTI-OBJECTIVE REINFORCEMENT LEARNING

Baiting Zhu, Meihua Dang, Aditya Grover
University of California, Los Angeles, CA, USA
baitingzbt@g.ucla.edu, mhdang@cs.ucla.edu, adityag@cs.ucla.edu

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 return-conditioned offline methods including Decision Transformers (Chen et al., 2021) and RvS (Emmons et al., 2022) 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. If the user has a relatively high preference for speed, the agent will move fast regardless of power usage; on the other hand, if the user tries to save energy, the agent will keep a more steady speed. 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 scalarizing the vector-valued rewards of different objectives based on a single preference (Lin [2005]), 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 from multiple agents 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. Our contributions in this work are two-fold in introducing benchmarking datasets and a new family of MORL algorithms, 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). Here, 5 environ-
ments consist of 2 objectives and 1 environment consists of 3 objectives. For each environment in D4MORL, we collect demonstrations from 2 pretrained behavioral agents: \textit{expert} and \textit{amateur}, where the relative expertise is defined in terms of the Pareto-efficiency of the agents and 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. Lack of MORL datasets and large-scale benchmarking has been a major challenge for basic research (Hayes et al., 2021), and we hope that D4MORL can aid future research in the field.

Next, we propose Pareto-Efficient Decision Agents (PEDA), a family of offline MORL algorithms that extends return-conditioned methods including Decision Transformer (DT) (Chen et al., 2021) and RvS (Emmons et al., 2022) to the multi-objective setting. These methods learn a return-conditioned policy via a supervised loss on the predicted actions. In recent work, these methods have successfully scaled to agents that demonstrate broad capabilities in multi-task settings (Lee et al., 2022; Reed et al., 2022). For MORL, we introduce a novel preference and return conditioned policy network and train it via a supervised learning loss. At test time, naively conditioning on the default preferences and maximum possible returns leads to out-of-distribution behavior for the model, as neither has it seen maximum returns for all objectives in the training data nor is it possible to simultaneously maximize all objectives under competition. We address this issue by learning to map preferences to appropriate returns and hence, enabling predictable generalization at test-time.

Empirically, 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 average upper bound on the hypervolume of $5.79 \times 10^6$ achieved by the behavioral policy, PEDA achieves an average hypervolume of $5.77 \times 10^6$ on the Expert and $5.76 \times 10^6$ on the Amateur datasets.

2 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)). Other approaches have also been studied, such as framing MORL as a meta-learning problem (Chen et al., 2018), learning the action distribution for each objective (Abdalmaleki et al., 2020), and learning the relationship between objectives (Zhan & Cao, 2019) among others. 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 decision making with multiple objectives in the offline setting and sidestep any interaction with the environments. Wu et al., 2021 propose a provably efficient offline MORL algorithm for tabular MDPs 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. Their proposed algorithm assumes knowledge of the behavioral policy used to collect the offline data and is demonstrated primarily on tabular MDPs with finite state and action spaces. In contrast, we are interested in developing dataset benchmarks and algorithms for scalable offline policy optimization 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) or Trajectory Transformers (Chen et al., 2021; Janner 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 online learning (Zheng et al., 2022) as well as simpler variants that rely only on multi-layer perceptrons (Emmons et al., 2022). Such agents are generally more stable and robust to optimize due to the simplicity of loss function and easier to scale to more complex settings such as environments with high-dimensional actions or states, as shown in recent works in multi-task RL (Lee et al., 2022; Reed et al., 2022).

### 3 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 \( \langle S, A, P, R, \Omega, f, \gamma \rangle \). At each timestep \( t \), the agent with a current state \( s_t \in S \) takes an action \( a_t \in A \) to transition into a new state \( s_{t+1} \) with probability \( P(s_{t+1} | s_t, a_t) \) and observes a reward vector \( r_t = R(s_t, a_t) \in \mathbb{R}^n \). Here, \( n \) is the number of objectives. The vector-valued expected return \( R \in \mathbb{R}^n \) of an agent is given by the discounted sum of reward vectors over a time horizon, \( R = \sum_{t=1}^{T} \gamma^t r_t \). We also assume that there exists a linear utility function \( f \) and a space of preferences \( \Omega \) that can map the reward vector \( r_t \) and a preference vector \( \omega \in \Omega \) to a scalar reward \( r_t = f(r_t, \omega) = \omega^T r_t \). The expected scalarized return of a policy \( \pi \) is given an \( G^\pi = [G^\pi_1, G^\pi_2, \ldots, G^\pi_n] \) where the expected return of the \( i \)-th objective is given as \( G^\pi_i = \mathbb{E}_{a_{i+1} \sim \pi \mid s_t, a_t} \sum_{t=1}^{T} \gamma^t R(s_t, a_t) \) for some predefined time horizon and preference vector \( \omega \). The goal is to train a multi-objective policy \( \pi(a | s, \omega) \) such that the expected scalarized return \( \omega^T G^\pi = \mathbb{E}[\omega^T \sum_{t=1}^{T} R(s_t, 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 \( \pi(a | s, \omega) \) that is evaluated for \( m \) distinct preferences \( \omega_1, \ldots, \omega_m \), and let the resulting policy set be represented as \( \{\pi_p\}_{p=1, \ldots, m} \), where \( \pi_p = \pi(a | s, \omega = \omega_p) \), and \( G^{\pi_p} \) is the corresponding unweighted expected return. We say the solution \( G^{\pi_p} \) is dominated by \( G^{\pi_q} \) when there is no objective for which \( \pi_q \) is worse than \( \pi_p \), i.e., \( G^{\pi_p}_i < G^{\pi_q}_i \) 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 define metrics for relative comparisons between different algorithms. Specifically, we evaluate a Pareto set \( P \) based on two metrics, hypervolume and sparsity that we describe next.

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

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

where \( H(P) = \{ z \in \mathbb{R}^m : 1 \leq i \leq |P|, r_i \leq z \leq P(i) \} \). \( P(i) \) is the \( i \)-th solution in \( P \), \( \leq \) is the dominance relation operator, and \( 1_{H(P)}(z) \) 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 \( P \):

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

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

See Figure 1 for an illustration and Appendix E for discussion on other possible metrics.

---

See Figure 1 for an illustration and Appendix E for discussion on other possible metrics.

---

**Figure 1:** Illustration of the Hypervolume and Sparsity Metrics. Only undominated solutions (i.e., the Pareto set) are used for calculating the evaluation metrics.
4 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. Even for the online setting, most works in MORL conduct evaluations on toy MDPs (e.g., gridworld) with a few exceptions that include continuous control benchmarks, e.g., Chen et al. (2018); Xu et al. (2020). This calls for a much needed push towards more challenging benchmarks for reliable evaluation of MORL, especially in the offline setting.

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; for instance, the two objectives in MO-Hopper correspond to jumping and running; in MO-HalfCheetah, MO-Swimmer, and MO-Walker2d, they correspond to the speed and energy savings of the agent. See Appendix A for more details on the semantics of the target objectives for each environment. These environments were first introduced in Xu et al. (2020) for online MORL, and as such, we use their pretrained ensemble policies as building blocks for defining new behavioral polices for dataset collection, which we discuss next.

4.1 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. In existing benchmarks for single-objective RL such as D4RL (Fu et al., 2020), the quality of a behavioral policy can be ascertained and varied based on its closeness to a single expert policy, as measured by their scalar-valued returns. For a MOMDP, we do not have the notion of a scalar return and hence, 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. The number of policies in the ensemble can vary significantly; for instance, we have roughly 70 reference policies for MO-Ant and 2445 policies for harder environments such as MO-Hopper-3obj. 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 2, 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.
Figure 2: 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.

Figure 3: 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).

4.2 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 \( \Omega \). First, we restrict our samples to lie within a physically plausible preference space \( \Omega^* \subseteq \Omega \) covered by the behavioral policy \( \pi_{\beta} \). For instance, MO-Hopper has two objectives: jumping and running. Since the agent can never gain running rewards without leaving the floor. Thus, the preference of 100% running and 0% jumping is not achievable and 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 \( \Omega^* \) 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 \( \alpha \)). 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 in Figure 3. Further details on the sampling distributions are deferred to 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.

Overall Data Generation Pipeline. The pseudocode for generating the dataset is described in Algorithm 1. Given a preference distribution, we first sample a preference \( \omega \) and query the closest behavioral policy in either the amateur/expert ensemble matching \( \omega \). 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:

\[
\tau = \langle \omega, s_1, a_1, r_1, \ldots, s_T, a_T, r_T \rangle
\]
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. We refer the reader to Table 5 in Appendix B for additional statistics on the dataset.

5 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-optimality by extending Decision Transformers (DT) and its variant, Reinf orce ment Learning Via Supervised Learning (RvS), followed by our modifications extending them to the multi-objective setting.

DT casts offline RL as a conditional sequence modeling problem that predicts the next action by conditioning a transformer on past states, actions, and desired returns. The desired returns are defined as returns-to-go (RTG) \( g_t = \sum_{t' = t}^{T} r_{t'} \), the future returns that this action is intended to achieve. Therefore, the trajectory is represented by \( \tau = <s_1, a_1, g_1, \ldots, s_T, a_T, g_T> \). In practice, we use a causally masked transformer architecture such as GPT [Radford et al., 2019] to process this sequence and predict the actions by observing the past \( K \) timesteps consisting of \( 3K \) tokens. DT and its variants have been shown to be more stable and robust to optimize due to the simplicity of loss function; easier to scale to more complex settings such as environments with high-dimensional actions or states, and agents with broad capabilities such as multitask settings [Lee et al., 2022]. Hence, we adopt Decision Transformers [Chen et al., 2021] as the representative base algorithm on which we build our work.

In follow-up work, [Emmons et al., 2022] extend DT and shows that even multi-layer perceptrons conditioned on the average returns-to-go can achieve similar performance without the use of transformers. They call their model as Reinforcement Learning Via Supervised Learning (RvS). However, RvS is generally not very stable when conditioned on very large returns, unlike DT.

5.1 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 \( \omega \) 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 \( \omega \) and the correlation between the preferences and predicted actions is weak. Therefore, we propose to concatenate \( \omega \) to other tokens before any layers 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 \( g_t = \sum_{t'=0}^{T} r_{t'} \). Given a preference vector \( \omega \), we can scalarize the total returns-to-go as \( \hat{g}_t = \omega^T g_t \). In principle, the scalarized RTG \( \hat{g}_t \) can be recovered given the preference vector \( \omega \) and the vector-valued RTG \( g_t \). However, empirically we find that directly feeding MODT/MORvS with the preference-weighted RTG vector \( g_t \odot \omega \) 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. Post this normalization, the values for every objective in the trajectory are on the same scale between 0 and 1. 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.

Training. We follow a simple supervised training procedure where we train the policies on randomly sampled mini-batches with MSE loss (for continuous actions). In MODT and MODT(P), the input states, actions, and returns-to-go (with concatenated preferences) are treated as tokens and embedded through one layer of MLP. We apply a layer of MLP and Tanh on the last hidden state of GPT-2 transformer to predict next action. In MORvS and MORvS(P), we use only information from the current timestep and MLP layers to predict the next action.

6 EXPERIMENTS

In this section, we evaluate the performance of PEDA on D4MORL benchmark. First, we investigate the benefits of preference conditioning by evaluating on decision transformers (DT) and RVs (MORvS) where no preference information is available and we scalarize multi-objective vector returns into weighted sums. We denote our methods with preference conditioning as MODT(P) and MORvS(P). Second, we compare our methods with classic imitation learning and temporal difference learning algorithms with preference conditioning.

Imitation learning. Imitation learning simply 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).

Temporal difference learning. Conservative Q-Learning (CQL) [Kumar et al., 2020] is the state-of-the-art standard offline RL method, which learns a conservative Q-function \( f : S \times A \rightarrow \mathbb{R} \) through neural networks. We modify the network architecture such that it also takes preference vectors as inputs to learn a preference-conditioned Q-function \( f^\Omega : S \times A \times \Omega \rightarrow \mathbb{R} \). We denote this method as CQL(P).

6.1 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)

<table>
<thead>
<tr>
<th>Environments</th>
<th>B</th>
<th>MODT(P)</th>
<th>MORvS(P) BC(P)</th>
<th>CQL(P)</th>
<th>MODT</th>
<th>MORvS</th>
<th>BC</th>
<th>CQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO-Ant (10^5)</td>
<td>6.32</td>
<td>6.02±0.02</td>
<td>6.24±0.07</td>
<td>4.49±0.30</td>
<td>5.76±0.10</td>
<td>5.10±0.16</td>
<td>5.05±0.07</td>
<td>0.78±0.57</td>
</tr>
<tr>
<td>MO-HalfCheetah (10^6)</td>
<td>5.79</td>
<td>5.69±0.01</td>
<td>5.77±0.00</td>
<td>5.52±0.04</td>
<td>5.63±0.04</td>
<td>5.59±0.05</td>
<td>4.56±0.56</td>
<td>1.45±0.04</td>
</tr>
<tr>
<td>MO-Hopper (10^7)</td>
<td>2.09</td>
<td>1.92±0.01</td>
<td>1.98±0.02</td>
<td>1.40±0.03</td>
<td>0.33±0.39</td>
<td>1.60±0.04</td>
<td>1.72±0.04</td>
<td>0.82±0.42</td>
</tr>
<tr>
<td>MO-Hopper-3obj (10^10)</td>
<td>3.73</td>
<td>3.24±0.09</td>
<td>3.27±0.10</td>
<td>2.29±0.29</td>
<td>0.78±0.24</td>
<td>1.22±0.27</td>
<td>2.33±0.14</td>
<td>0.03±0.01</td>
</tr>
<tr>
<td>MO-Swimmer (10^6)</td>
<td>3.25</td>
<td>3.16±0.01</td>
<td>3.22±0.01</td>
<td>3.21±0.01</td>
<td>3.22±0.08</td>
<td>2.49±0.28</td>
<td>3.19±0.01</td>
<td>1.81±0.03</td>
</tr>
<tr>
<td>MO-Walker2d (10^5)</td>
<td>5.21</td>
<td>4.84±0.10</td>
<td>5.15±0.01</td>
<td>3.47±0.13</td>
<td>3.21±0.32</td>
<td>0.61±0.43</td>
<td>4.95±0.06</td>
<td>0.07±0.01</td>
</tr>
</tbody>
</table>

Hypervolume. We compare hypervolume of our methods with all baselines on expert datasets in Table 1 as well as amateur dataset in Table 2. For the two-objective environments, we evaluate the
Table 2: 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)

<table>
<thead>
<tr>
<th>Environments</th>
<th>B</th>
<th>MODT(P)</th>
<th>MORvS(P) BC(P)</th>
<th>CQL(P)</th>
<th>MODT</th>
<th>MORvS</th>
<th>BC</th>
<th>CQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO-Ant (10^4)</td>
<td>5.61</td>
<td>5.60±0.3</td>
<td>5.94±0.04</td>
<td>4.35±0.07</td>
<td>5.62±0.23</td>
<td>3.34±0.71</td>
<td>4.91±0.06</td>
<td>2.34±0.09</td>
</tr>
<tr>
<td>MO-HalfCheetah (10^6)</td>
<td>5.64</td>
<td>5.77±0.00</td>
<td>5.26±0.06</td>
<td>5.64±0.04</td>
<td>5.51±0.01</td>
<td>5.28±0.35</td>
<td>3.14±0.38</td>
<td>4.41±0.08</td>
</tr>
<tr>
<td>MO-Hopper (10^4)</td>
<td>1.97</td>
<td>1.82±0.02</td>
<td>1.56±0.01</td>
<td>1.15±0.24</td>
<td>1.54±0.08</td>
<td>1.24±0.04</td>
<td>0.48±0.48</td>
<td>0.00±0.06</td>
</tr>
<tr>
<td>MO-Hopper-3obj (10^4)</td>
<td>3.09</td>
<td>2.03±0.17</td>
<td>2.98±0.06</td>
<td>2.31±0.16</td>
<td>0.59±0.42</td>
<td>1.72±0.12</td>
<td>1.57±0.08</td>
<td>0.13±0.04</td>
</tr>
<tr>
<td>MO-Swimmer (1)</td>
<td>2.11</td>
<td>2.83±0.04</td>
<td>2.76±0.02</td>
<td>2.81±0.04</td>
<td>1.69±0.93</td>
<td>0.67±0.11</td>
<td>2.78±0.01</td>
<td>0.46±0.14</td>
</tr>
<tr>
<td>MO-Walker2d (10^4)</td>
<td>4.99</td>
<td>3.17±1.14</td>
<td>4.96±0.02</td>
<td>2.93±0.75</td>
<td>1.78±0.33</td>
<td>2.62±0.81</td>
<td>4.21±0.27</td>
<td>1.19±0.21</td>
</tr>
</tbody>
</table>

Table 3: 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.

<table>
<thead>
<tr>
<th>Environments</th>
<th>MODT(P)</th>
<th>MORvS(P)</th>
<th>BC(P)</th>
<th>CQL(P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO-Ant (×10^4)</td>
<td>7.06±1.29</td>
<td>4.55±0.59</td>
<td>32.1±12.2</td>
<td>0.58±0.10</td>
</tr>
<tr>
<td>MO-HalfCheetah (×10^4)</td>
<td>1.44±0.28</td>
<td>0.74±0.03</td>
<td>1.69±0.70</td>
<td>0.10±0.00</td>
</tr>
<tr>
<td>MO-Hopper (×10^3)</td>
<td>8.89±1.48</td>
<td>1.91±0.51</td>
<td>23.7±20.1</td>
<td>0.24±0.26</td>
</tr>
<tr>
<td>MO-Hopper-3obj (×10^4)</td>
<td>1.42±0.42</td>
<td>1.11±0.22</td>
<td>0.75±0.15</td>
<td>0.26±0.14</td>
</tr>
<tr>
<td>MO-Swimmer (×1)</td>
<td>12.1±6.49</td>
<td>5.00±4.2</td>
<td>3.79±1.03</td>
<td>13.61±5.31</td>
</tr>
<tr>
<td>MO-Walker2d (×10^4)</td>
<td>5.28±9.99</td>
<td>1.91±0.25</td>
<td>46.9±23.2</td>
<td>6.23±0.71</td>
</tr>
</tbody>
</table>

models on 501 equally spaced preference points in the range [0, 1]; on the three-objective environment MO-Hopper-3obj, models are evaluated on 325 equally spaced points. Each point is evaluated 5 times with random environment re-initialization, and the median value is recorded. Finally, all the results are based on 3 random seeds and we report the mean performance along with the standard error. In Table 1 and Table 2, we can see that MODT(P) and MORvS(P) outperform other baselines and has a relatively very low standard error. Also, PEDA variants including MODT(P) and MORvS(P) approaches the behavioral policy upper-bound.

Sparsity. We also evaluate sparsity performance. Since sparsity comparison is only meaningful between models that are sensitive to preference and have a relatively similar hypervolume performance, we only show results for models that concatenate preference. Overall, MORvS(P) has the lowest sparsity in most environments, while at the same time featuring an outstanding hypervolume.

6.2 Ablation Study

Pareto front approximation. We ablate how well the MODT(P) and MORvS(P) can approximate the Pareto front through conditioning on different preference points. We show the results in Figure 3, where we can see that the models 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, and the full plot can be found in Appendix F.

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.

<table>
<thead>
<tr>
<th>Environments</th>
<th>MODT(P)</th>
<th>MORvS(P)</th>
<th>BC(P)</th>
<th>CQL(P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO-Ant (×10^4)</td>
<td>6.56±4.5</td>
<td>4.96±1.1</td>
<td>11.7±6.45</td>
<td>1.06±0.28</td>
</tr>
<tr>
<td>MO-HalfCheetah (×10^4)</td>
<td>0.81±0.52</td>
<td>0.31±0.01</td>
<td>2.18±0.49</td>
<td>0.45±0.27</td>
</tr>
<tr>
<td>MO-Hopper (×10^4)</td>
<td>1.54±0.69</td>
<td>1.75±0.52</td>
<td>9.57±8.64</td>
<td>3.30±5.25</td>
</tr>
<tr>
<td>MO-Hopper-3obj (×10^4)</td>
<td>4.59±1.51</td>
<td>1.04±0.28</td>
<td>0.79±0.19</td>
<td>2.00±1.72</td>
</tr>
<tr>
<td>MO-Swimmer (×1)</td>
<td>2.74±1.51</td>
<td>1.53±0.10</td>
<td>1.21±0.97</td>
<td>8.87±6.24</td>
</tr>
<tr>
<td>MO-Walker2d (×10^4)</td>
<td>30.7±13.6</td>
<td>2.10±0.02</td>
<td>0.14±0.04</td>
<td>7.33±5.89</td>
</tr>
</tbody>
</table>
Figure 4: We show that MODT(P) and MORvS(P) can be good approximator to the Pareto front. There are relatively more dominated points in MO-Hopper and MO-Walker2d colored in red.

Figure 5: 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.

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 5 for MORvS(P) under High-H-Expert datasets and refer to Appendix G for full settings. 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.

7 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.
REPRODUCIBILITY STATEMENT

In this paper, we make sure the experiment results are reproducible through training the models on multiple seeds and evaluating each preference point for multiple times. We report the mean value with standard error in all of our experiment result tables. Furthermore, we upload our model code along with our submission, and sample data can be found in this anonymous Google Drive folder: [https://drive.google.com/drive/folders/1FiF5xmCSj2vLfrLYmeZNC_nUrRfUXC?usp=sharing](https://drive.google.com/drive/folders/1FiF5xmCSj2vLfrLYmeZNC_nUrRfUXC?usp=sharing). Due to storage limits, we currently release only a subset of our data that can be used directly for training; we will release our full datasets in the final version.

REFERENCES


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-3obj always 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 $r = [r^x_t, r^y_t]^T$. 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:

$$
\begin{align*}
    r^x_t &= (x_t - x_{t-1}) / dt + r^s - r^a \\
    r^y_t &= (y_t - y_{t-1}) / dt + r^s - r^a
\end{align*}
$$

A.2 MO-HalfCheetah

The two objectives in MO-HalfCheetah are running speed, and energy saving, denoted as $r = [r^v_t, r^e_t]^T$. 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:

$$
\begin{align*}
    r^v_t &= \min\{4.0, (x_t - x_{t-1}) / dt\} + r^s \\
    r^e_t &= 4.0 - r^a + r^s
\end{align*}
$$

A.3 MO-Hopper

The two objectives in MO-Hopper are running and jumping, denoted as $r = [r^r_t, r^j_t]^T$. 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:

$$
\begin{align*}
    r^r_t &= 1.5 \times (x_t - x_{t-1}) / dt + r^s - r^a \\
    r^j_t &= 12 \times (h_t - h_{init}) / dt + r^s - r^a
\end{align*}
$$

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 $r = [r^r_t, r^j_t, r^e_t]^T$. 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:

$$
\begin{align*}
    r^r_t &= 1.5 \times (x_t - x_{t-1}) / dt + r^s \\
    r^j_t &= 12 \times (h_t - h_{init}) / dt + r^s \\
    r^e_t &= 4.0 - r^a + r^s
\end{align*}
$$
A.5 MO-Swimmer

The two objectives in MO-Swimmer are speed and energy saving, denoted as \( r = [r^v, r^e]^\top \).

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:

\[
\begin{align*}
    r^v_t &= (x_t - x_{t-1}) / dt \\
    r^e_t &= 0.3 - 0.15 \times r^a
\end{align*}
\]  

(5)

A.6 MO-Walker2d

The objectives in MO-Walker2d are speed and energy saving, denoted as \( r = [r^v, r^e]^\top \).

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:

\[
\begin{align*}
    r^v_t &= (x_t - x_{t-1}) / dt + r^s \\
    r^e_t &= 4.0 - r^a + r^s
\end{align*}
\]  

(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:

\[
\omega^{\text{high}} \sim ||f_{\exp}(\cdot, \lambda = 1)||_1
\]  

(7)

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

To sample the Med-H and Low-H data, we first sample \( \alpha \) 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.

\[
\begin{align*}
    \omega^{\text{med}} &\sim f_{\text{Dirichlet}}(\alpha) ; \text{ where } \alpha \sim \text{Unif}(0, 10^6) \\
    \omega^{\text{low}} &\sim f_{\text{Dirichlet}}(\alpha) ; \text{ where } \alpha \sim \text{Unif}(1/3 \times 10^6, 2/3 \times 10^6)
\end{align*}
\]  

(8)

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

B Dataset 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:

\[
\omega^{\text{high}} \sim ||f_{\exp}(\cdot, \lambda = 1)||_1
\]  

(9)

We take the 1-norm following the exponential distribution to make sure each preference add up to 1. When \( \Omega^* \neq \Omega \), we perform rejection sampling to restrict the range.
Table 5: A comprehensive view of the dataset. All datasets have a 500 maximum step per trajectory, 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>max step per traj</th>
<th>expert avg. step per traj</th>
<th>amateur avg. step per traj</th>
<th>trajectories per dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO-Ant</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>50K</td>
</tr>
<tr>
<td>MO-HalfCheetah</td>
<td>500</td>
<td>499.91</td>
<td>482.11</td>
<td>50K</td>
</tr>
<tr>
<td>MO-Hopper</td>
<td>500</td>
<td>499.94</td>
<td>387.72</td>
<td>50K</td>
</tr>
<tr>
<td>MO-Hopper-3obj</td>
<td>500</td>
<td>499.99</td>
<td>442.87</td>
<td>50K</td>
</tr>
<tr>
<td>MO-Swimmer</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>50K</td>
</tr>
<tr>
<td>MO-Walker2d</td>
<td>500</td>
<td>500</td>
<td>466.18</td>
<td>50K</td>
</tr>
</tbody>
</table>

To sample the Med-H and Low-H data, we first sample $\alpha$ 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.

$$
\omega^{\text{med}} \sim f_{\text{Dirichlet}}(\alpha) ; \text{where } \alpha \sim \text{Unif}(0, 10^6)
$$

$$
\omega^{\text{low}} \sim f_{\text{Dirichlet}}(\alpha) ; \text{where } \alpha \sim \text{Unif}(1/3 \times 10^6, 2/3 \times 10^6)
$$

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 $G^{\pi_1}, \ldots, G^{\pi_B}$. Then, find the estimated $\hat{\omega}^{\pi_1}, \ldots, \hat{\omega}^{\pi_B}$ by letting

$$
\hat{\omega}^\pi_i = \frac{G^\pi_i}{\sum_{j=1}^B G^\pi_j},
$$

which represents the estimated preference on $i^{th}$ objective of $j^{th}$ candidate policy. For a sampled preference $\omega \sim \Omega^\pi$, use the policy that provides the smallest euclidean distance $d(\omega, \hat{\omega}^\pi_i)$. Empirically, this means picking the candidate policy that has the expected reward ratio closest to $\omega$.

C Expert & Amateur Datasets

In Expert collection, we sample trajectories using the fully-trained behavioral policy $\pi_\beta$. In this paper, we use PGMORL by [Xu et al., 2020] as $\pi_\beta$

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

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:

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

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:

$$
a_{t+1}^\text{amateur} = \begin{cases} \text{Unif}(A) & \text{35\%} \\ a_{t+1}^\text{expert} \times \text{Unif}(0.35, 1.65) & \text{65\%} \end{cases}
$$

D Training 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 $n_{\text{layer}}$ represents the number of transformer blocks; in BC family, $n_{\text{layer}}$ represents the number of MLP layers to embedding each input; in MOR$\text{V}$ and MOR$\text{V}(P)$ family, the default embedding strategy are used, please check [Emmons et al., 2022]. Here, we consider MOR$\text{V}$ and MOR$\text{V}(P)$ 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.
D.1 PARAMETERS

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

D.2 TRAINING STEPS

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>MODT Steps</th>
<th>Rs/BC Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO-Ant</td>
<td>20K</td>
<td>200K</td>
</tr>
<tr>
<td>MO-HalfCheetah</td>
<td>80K</td>
<td>200K</td>
</tr>
<tr>
<td>MO-Hopper</td>
<td>400K</td>
<td>200K</td>
</tr>
<tr>
<td>MO-Hopper-3obj</td>
<td>400K</td>
<td>200K</td>
</tr>
<tr>
<td>MO-Swimmer</td>
<td>260K</td>
<td>200K</td>
</tr>
<tr>
<td>MO-Walker2d</td>
<td>360K</td>
<td>200K</td>
</tr>
</tbody>
</table>

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

D.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 \( \omega \) as an independent token of the causal transformer.

2. Train a separate embedding for \( \omega \), concatenate the embeddings to get \( f_{\phi_s}(s) \oplus f_{\phi_\omega}(\omega) \), \( f_{\phi_a}(a) \oplus f_{\phi_\omega}(\omega) \), and \( f_{\phi_g}(g) \oplus f_{\phi_\omega}(\omega) \) 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 \( \omega \) to other tokens before any layers. This means we have \( s^* = s \oplus \omega, \ a^* = a \oplus \omega, \) and \( g^* = g \oplus \omega. \)

E OTHER EVALUATION 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 \( \epsilon \)-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 6: Full Pareto Visualization 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 environment, however, the PEDA variants produce good results for preference conditioned models, while other baselines fail at a higher chances.

F  Pareto Set Visualizations

We present the Pareto Set visualizations for all of our models trained under each High-H dataset in Figure 6. 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 $\Omega^*$ but the solution is dominated, we color it in red. Since our models condition on continuous preference points and environments are initialized 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.
Table 6: We ablate the study of how well PEDA variants perform under a different preference distribution and how well it generalizes to unseen preferences for MO-HalfCheetah environment. We can see that PEDA can perform well when trained under partially-clustered Med-H dataset but performance drops when it is trained under entirely clustered Low-H dataset. (B: Behavioral Policy)

<table>
<thead>
<tr>
<th>Dataset (×10⁶)</th>
<th>B</th>
<th>MODT(P)</th>
<th>MORvS(P)</th>
<th>BC(P)</th>
<th>MODT</th>
<th>MORvS</th>
<th>BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Med-H-Amateur</td>
<td>5.69</td>
<td>5.61±0.01</td>
<td>5.77±0.00</td>
<td>1.63±0.26</td>
<td>3.84±0.44</td>
<td>4.61±0.03</td>
<td>3.63±0.12</td>
</tr>
<tr>
<td>Low-H-Amateur</td>
<td>4.21</td>
<td>4.74±0.02</td>
<td>4.83±0.02</td>
<td>4.67±0.07</td>
<td>2.54±0.05</td>
<td>4.04±0.18</td>
<td>4.09±0.09</td>
</tr>
<tr>
<td>High-H-Amateur</td>
<td>5.68</td>
<td>5.64±0.01</td>
<td>5.77±0.00</td>
<td>5.36±0.06</td>
<td>5.51±0.01</td>
<td>5.28±0.35</td>
<td>3.14±0.38</td>
</tr>
</tbody>
</table>

Table 7: We ablate the importance of using multi-dimensional rtg instead of a one-dimensional rtg by taking the weighted sum of objectives on the MO-HalfCheetah environment. We see using multi-objective rtg provides a lower variance for MORvS(P) or performance improvement in other models. (B: Behavioral Policy)

<table>
<thead>
<tr>
<th>Setting</th>
<th>Dataset (×10⁶)</th>
<th>B</th>
<th>MODT(P)</th>
<th>MORvS(P)</th>
<th>MODT</th>
<th>MORvS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-dim rtg</td>
<td>High-H-Expert</td>
<td>5.79</td>
<td>5.54±0.09</td>
<td>5.78±0.03</td>
<td>4.34±0.13</td>
<td>3.06±0.42</td>
</tr>
<tr>
<td>no rtg</td>
<td>High-H-Expert</td>
<td>5.79</td>
<td>5.69±0.01</td>
<td>5.77±0.00</td>
<td>5.59±0.05</td>
<td>4.56±0.56</td>
</tr>
</tbody>
</table>

G 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 6, 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.

H 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 7 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 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.