# Momba: Network Modernization Improves Multi-Objective Reinforcement Learning

**Adam Štafa, Santeri Heiskanen, Petr Novotný, Joni Pajarinen**

**Keywords:** Reinforcement Learning, Multi-objective Reinforcement Learning, Distributional Reinforcement Learning, Neural Architectures

## Summary

Recent advances in deep reinforcement learning (RL) have shown that improving neural network architectures can yield substantial gains in sample efficiency and asymptotic performance without altering the underlying algorithms. In contrast, work on multi-objective reinforcement learning (MORL), which aims to discover a set of policies that balance trade-offs among conflicting objectives, has predominantly focused on algorithmic innovations, leaving the area of architectures underexplored. While the optimal policies and value functions can differ significantly depending on the trade-offs, MORL algorithms commonly represent them with simple feedforward networks conditioned on the trade-off. This raises the question of whether the performance of the algorithms could be improved with more expressive function approximators. In this paper, we integrate recent advances in neural network design: (i) observation and feature normalization, (ii) weight normalization, and (iii) modeling of distributional returns with an entropy-regularized MORL algorithm. The empirical results across standard continuous control benchmarks demonstrate that these changes substantially improve the quality of the produced solution sets without requiring major changes to the underlying algorithm.

## Contribution(s)

1. We showcase that employing more expressive neural network architectures with a distributional critic in a MORL algorithm substantially improves the performance on continuous control tasks without requiring complex preference selection algorithms or MORL-specific update rules.
   **Context:** Previous works in MORL achieved improvements in sample efficiency and asymptotic performance by employing advanced preference selection algorithms (Alegre et al., 2023) or specialized update rules (Yang et al., 2019; Li et al., 2025). In single-objective RL, in contrast, a range of studies (Nauman et al., 2024; Lee et al., 2025a;b; Palenicek et al., 2025; 2026) has shown that the combination of (i) feature normalization, (ii) weight normalization, and (iii) categorical critic loss can improve the sample efficiency and asymptotic performance of existing reinforcement learning algorithms, without altering them. We apply the same principles to multi-objective reinforcement learning and showcase that improvements in the architecture outperform selected baselines. Additionally, we study these design choices and identify the distributional critic as a major component.
2. We propose a simple adaptation of the categorical critic to the multi-objective domain by directly learning to predict the scalarized return distribution, motivated by the commonplace scalarized expected return objective.
   **Context:** Previous research applying the distributional critic (Bellemare et al., 2017) to multi-valued return functions by modeling the multivariate joint distribution of rewards has been limited to tabular cases (Wiltzer et al., 2024), or required specifying a kernel function for measuring distance between distributions (Zhang et al., 2021). Instead, we specifically target multi-objective reinforcement learning, under the prevalent linear scalarization assumption (Abels et al., 2019; Xu et al., 2020; Kyriakis et al., 2022; Basaklar et al., 2023; Alegre et al., 2023), and propose a simple approach where the Q-network directly predicts the scalarized return distribution. Through extensive empirical studies, we validate the effectiveness of the proposed approach.

# Momba: Network Modernization Improves Multi-Objective Reinforcement Learning

**Adam Štafa**[1], **Santeri Heiskanen**[2], **Petr Novotný**[1], **Joni Pajarinen**[2]
stafa@mail.muni.cz, petr.novotny@fi.muni.cz
{santeri.heiskanen,joni.pajarinen}@aalto.fi

[1]**Faculty of Informatics, Masaryk University**
[2]**Department of Electrical Engineering and Automation, Aalto University**

## Abstract

Recent advances in deep reinforcement learning (RL) have shown that improving neural network architectures can yield substantial gains in sample efficiency and asymptotic performance without altering the underlying algorithms. In contrast, work on multi-objective reinforcement learning (MORL), which aims to discover a set of policies that balance trade-offs among conflicting objectives, has predominantly focused on algorithmic innovations, leaving the area of architectures underexplored. While the optimal policies and value functions can differ significantly depending on the trade-offs, MORL algorithms commonly represent them with simple feedforward networks conditioned on the trade-off. This raises the question of whether the performance of the algorithms could be improved with more expressive function approximators. In this paper, we integrate recent advances in neural network design: (i) observation and feature normalization, (ii) weight normalization, and (iii) modeling of distributional returns with an entropy-regularized MORL algorithm. The empirical results across standard continuous control benchmarks demonstrate that these changes substantially improve the quality of the produced solution sets without requiring major changes to the underlying algorithm.

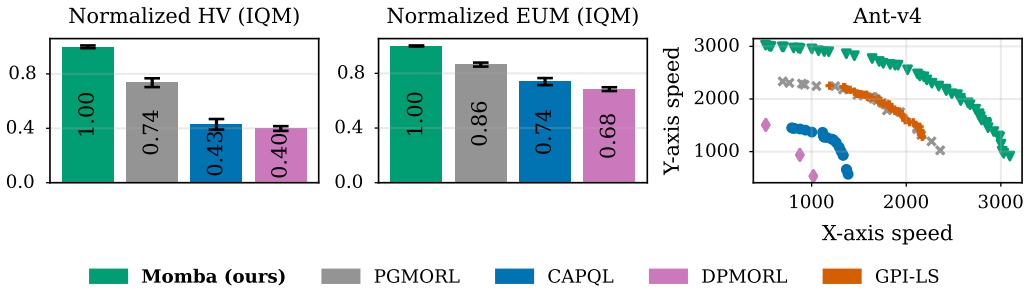


Figure 1: **Momba improves performance on 7 continuous control tasks**. Left and center columns show the normalized hypervolume (HV) and expected utility (EUM) (IQM and 95% SBCIs over 10 seeds), aggregated over 7 tasks. Momba outperforms both the runner-up, PGMORL, and CAPQL, our underlying algorithm. We report GPI results separately in Figure 3 due to a different evaluation protocol. The right column displays the generated solution sets in the challenging Ant environment, demonstrating that Momba can generate solutions with good coverage.

## 1 Introduction

Multi-Objective Reinforcement Learning (MORL) offers a principled approach for identifying a set of policies, considering different trade-offs between multiple, conflicting objectives (Roijers et al., 2013), often seen in real-world problems such as drug design (Olivecrona et al., 2017; Zhou et al., 2019), medical treatment (Jalalimanesh et al., 2017), or even robotics (Haarnoja et al., 2018a; Xie et al., 2019). While single-objective reinforcement learning (SORL) can achieve similar results by encoding the trade-off into the reward function (Skalse et al., 2022; Muslimani et al., 2025), MORL allows one to make an informed decision on the desired trade-off *after* the policies are generated (Hayes et al., 2022). Existing work has developed various learning algorithms for MORL (Abels et al., 2019; Xu et al., 2020; Lu et al., 2023; Cai et al., 2023), yet many of them still suffer from poor sample efficiency (Li et al., 2025).

One approach for improving sample efficiency in MORL is to learn a single policy, conditioned on the desired trade-off (Yang et al., 2019; Basaklar et al., 2023). In theory, efficient information sharing via a single policy should speed up learning, yet in practice, this approach faces significant challenges. As the behaviors between different trade-offs may differ significantly, the policy's ability to generalize is hindered. The previous work on MORL largely addressed this issue by developing new algorithmic frameworks while customarily using simple feedforward networks for the conditioned value function and policy representation (Yang et al., 2019; Lu et al., 2023). At the same time, recent advances in single-objective RL (SORL) have demonstrated that enhanced neural network architectures can significantly improve performance and sample efficiency (Nikishin et al., 2022; Nauman et al., 2024; Lee et al., 2025a;b; Palenicek et al., 2025; 2026). Therefore, we raise a natural question: *Could multi-objective reinforcement learning benefit from recent advances in neural network design for deep RL?*

In this paper, we study this question by building on top of SimbaV2 (Lee et al., 2025b), a recent architecture designed for deep RL that utilizes (i) observation & feature normalization, (ii) weight normalization, and (iii) distributional critic to improve the training dynamics of neural networks. To adapt the distributional critic to the multivariate return distribution, we propose to learn the scalarized distributional returns. We apply the SimbaV2 architecture with the proposed changes on top of an existing entropy-regularized MORL algorithm, CAPQL (Lu et al., 2023), and perform extensive evaluations in 7 continuous control tasks, commonly used in MORL research. Our findings demonstrate that the performance of an existing MORL algorithm can be significantly boosted by adopting the above-mentioned techniques to the neural networks used to represent conditioned value functions and policies. Moreover, we perform comprehensive ablations on our design choices and uncover that the distributional critic is a central component behind the performance improvement.

## 2 Related work

Our work is focused on advancing MORL in continuous control tasks by using techniques from scaling of deep RL and modeling of multivariate distributional returns. Thus, various parts of our work exist in the literature.

**Multi-objective reinforcement learning** can be roughly divided into three fields: 1) single-policy methods, 2) multi-policy methods, and 3) general policy methods. The single-policy methods consider only a single preference, and then transform the problem to a single objective problem, making it difficult to adapt to new, unseen preferences (Roijers et al., 2013; Hayes et al., 2022; Roijers et al., 2018). Multi-policy methods seek to find a set of solutions by training separate policies for different trade-offs, which often results in poor sample efficiency (Roijers et al., 2013; Hayes et al., 2022; Xu et al., 2020). General policy methods sidestep these issues by learning a universal policy, often conditioned on the trade-off, which is used to approximate the whole Pareto front (Abels et al., 2019; Yang et al., 2019; Basaklar et al., 2023). Related to our work, Shu et al. (2024) use a

Code to reproduce the experiments is available at https://github.com/adamstafa/momba
Correspondence to: stafa@mail.muni.cz

trade-off conditioned hypernetwork for generating the weights for the policy, while Li et al. (2025) learn a common latent space using self-consistency loss to avoid redundant learning under different trade-offs. However, these methods also introduce changes to algorithmic components, while we explicitly change only the critic loss and preference sampling frequency.

**Specializing architectures for RL** is a recent trend that aims to boost the performance and sample efficiency of existing RL algorithms by updating the neural network designs. The approaches have considered biasing the network to use simpler features for predictions (Lee et al., 2025a), pairing strong regularization with optimistic exploration (Nauman et al., 2024), or even designing a network that improves the loss landscape of the Bellman error (Palenicek et al., 2026). Regardless, to our knowledge, the application of these ideas to the multi-objective domain has been limited, with the exception of GPI-LS (Alegre et al., 2023), which used layer normalizations and DroQ networks (Hiraoka et al., 2022). Yet, the authors did not study the effects of these choices.

**Multivariate distributional reinforcement learning** extends distributional RL, a framework for modeling distribution over returns (Bellemare et al., 2017), to multi-valued reward functions. Previously, Zhang et al. (2021) modeled the complex multivariate distribution by minimizing the maximum mean discrepancy over the joint returns, while Wiltzer et al. (2024) proposed a provably convergent algorithm for learning multivariate return distributions in tabular MDPs. Instead, we propose to learn the scalarized distributional returns directly, offering a pragmatic approach specifically targeted for MORL under the common linear scalarization assumption. Cai et al. (2023) also consider the MORL setting without the linearity assumption. They propose to model the distribution of user preferences by learning a set of plausible scalarization functions, yet they train separate policies for each function, resulting in poor sample efficiency.

## 3 Background

**Multi-Objective Reinforcement Learning** tasks are formally defined via Multi-objective Markov Decision Process (MOMDP) (Chatterjee et al., 2006), which is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \Omega, f_\Omega \rangle$, consisting of state-space $\mathcal{S}$, action-space $\mathcal{A}$, transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times S \to [0, 1]$, vector-valued reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^d$, where $d \geq 2$ is the number of objectives, discount factor $\gamma \in [0, 1)$, preference space $\Omega$ and a scalarization function $f_\omega : \mathbb{R}^d \times \Omega \to \mathbb{R}$. In this paper, we limit our discussion to preference space $\Omega = \{\boldsymbol{\omega} \in \mathbb{R}^d | \sum_{i=1}^d \boldsymbol{\omega}_i = 1\}$, and to linear utility functions: $f_\omega(\boldsymbol{\omega}, \boldsymbol{r}) = \boldsymbol{\omega}^T \boldsymbol{r}$, as is commonly done in existing MORL research (Abels et al., 2019; Kyriakis et al., 2022; Lu et al., 2023; Shu et al., 2024). As there is generally no policy that can maximize all objectives simultaneously, one seeks to find a set of *Pareto-optimal* policies (Roijers et al., 2013). We say that a policy $\pi$ dominates another policy $\pi'$ (denoted by $\pi' \prec \pi$) if $\forall i : \boldsymbol{V}_i^{\pi'} \leq \boldsymbol{V}_i^{\pi}$ with at least one strict inequality, where $\boldsymbol{V}^\pi = \mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t \boldsymbol{r}_t\right] \in \mathbb{R}^d$ is vector-valued value function. A policy is considered Pareto-optimal if there is no policy that dominates it. The set of all such policies is called the Pareto-optimal set, and their induced value functions form the Pareto-front (Roijers et al., 2013). In this paper, we focus on *general policy* methods that try to solve the above problem by finding a preference-conditioned policy $\pi(\cdot|\boldsymbol{s}, \boldsymbol{\omega})$ that maximizes the scalarized expected returns (SER) for any preference $\boldsymbol{\omega} \in \Omega$:

$$\pi^*_{\boldsymbol{\omega}} = \arg\max_{\pi \in \Pi} \boldsymbol{\omega}^T \mathbb{E}_{\pi(\cdot|\cdot,\boldsymbol{\omega})} \left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{r}_t\right] \tag{1}$$

The final solution set is generated by conditioning the policy on a range of preferences.

**Stabilizing deep RL**: The non-stationarity of targets in RL can lead to overfitting and unstable learning. The existing literature has addressed this issue using several techniques. Feature normalization and weight normalization prevent plasticity loss and reduce overfitting to early behaviors (Lee et al., 2025a;b). To prevent overfitting on high-variance features, observation normalization and feature normalization are used to bring the magnitudes of hidden values in the network to a comparable scale (Huang et al., 2023; Lee et al., 2025a). Finally, distributional critic bounds gradient norms,

leading to more stable learning dynamics (Lee et al., 2025b; Palenicek et al., 2026). In this work, we will use the architecture from SimbaV2 (Lee et al., 2025b). Namely, SimbaV2 uses (i) $l2$ (hyperspherical) hidden feature normalization and running statistics observation normalization; (ii) the weights are normalized after every gradient update; and (iii) a distributional critic. Additionally, the architecture features scaling layers and learnable residual connections.

**Distributional Critic**: While classical value-based algorithms approximate the expected returns of a policy by a state-action value function $Q(s, a)$, the C51 algorithm (Bellemare et al., 2017) extends on this idea by modeling the distribution of returns instead. The distributions are approximated as categorical distributions over a discrete set of atoms $\{z_i = V_{\min} + i\Delta z : 0 \leq i < N\}, \quad \Delta z := (V_{\max} - V_{\min})/(N-1)$ using a parametric critic $Z_\theta(s, a)$. Given a sample transition $(s, a, r, s')$ and next state action $a'$, the critic is trained by minimizing the cross-entropy loss $\mathcal{L}(\theta) = H(\hat{Z}(r, s'), Z_\theta(s, a))$, where the target returns distribution $\hat{Z}$ is computed by projecting the distribution $r + \gamma Z(s', a')$ onto the common support. Given the output of the critic as probabilities $p_i$ over the atoms $z_i$, the expected return can be computed as $Q_\theta(s, a) = \mathbb{E}[Z_\theta(s, a)] = \sum_{i=0}^{N-1} p_i z_i$. Empirical results have shown that the distributional critic can greatly improve and stabilize learning (Bellemare et al., 2017) and a study of the loss landscape showed that the distributional critic improves the conditioning of the problem, stabilizing the optimization (Palenicek et al., 2026).

## 4 Methodology

In this section, we introduce Momba, a new algorithm that integrates recent advancements in neural network design into the multi-objective setting. We build the algorithm on top of CAPQL (Lu et al., 2023), which is an entropy-regularized algorithm akin to Soft-Actor Critic (SAC) (Haarnoja et al., 2018b). This offers two benefits: First, as shown by Lu et al. (2023), the entropy bonus ensures that the induced solution sets are strictly convex, promoting numerically stable optimization under linear scalarization. Second, SAC has been used as the backbone in many recent works on neural architectures in deep RL (Lee et al., 2025a;b; Palenicek et al., 2026), motivating the choice of using a similar algorithm in the multi-objective case. For the architecture, we adopt SimbaV2 (Lee et al., 2025b), a recent architecture for SORL, that adopts the three main components, (i) feature normalization, (ii) weight normalization, and (iii) distributional critic as mentioned in Section 3. We provide a detailed description of the architectural components and normalizations in Appendix C.

**CAPQL:** Consider a stochastic policy $\pi(a \,|\, s, \boldsymbol{\omega})$ and let $\mathbf{G}^\pi(s_0, a_0) = \sum_{t=0}^{T} \gamma^t \boldsymbol{r}(s_t, a_t) + \sum_{t=1}^{T} \gamma^t \mathbf{1}_d H(\pi(\cdot \mid s_t, \omega))$ denote the entropy-augmented discounted returns of the policy $\pi$ when starting in state $s_0$ and selecting action $a_0$. CAPQL trains a parametric $Q$-network $Q_\theta(s, a, \boldsymbol{\omega})$ and a policy $\pi_\phi(a \mid s, \boldsymbol{\omega})$. The $Q$-network is trained by minimizing the mean squared error (MSE) loss

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{(s,a,\boldsymbol{r},s',\boldsymbol{\omega})\sim\mathcal{D}} \|\hat{\mathbf{Q}} - \mathbf{Q}_\theta(s, a, \boldsymbol{\omega})\|_2^2 \tag{2}$$

where $\hat{\mathbf{Q}} = \boldsymbol{r} + \gamma(\boldsymbol{Q}_{\bar{\theta}}(s', a', \boldsymbol{\omega}) - \alpha \mathbf{1} \log \pi_\phi(a' \mid s', \boldsymbol{\omega}))$, $a' \sim \pi_\phi(\cdot|s', \boldsymbol{\omega})$ denotes the target Q-value estimate, and $\bar{\theta}$ denotes the target network parameters. Intuitively, the $Q$-value approximates the expected vector returns $\mathbb{E}\left[\mathbf{G}^{\pi(\cdot|\cdot,\boldsymbol{\omega})}(s, a)\right]$. The policy is trained for each state $s$ and preference $\boldsymbol{\omega}$ to maximize the scalarized value

$$\mathbb{E}_{a\sim\pi_\phi}[\boldsymbol{\omega}^T \mathbf{Q}_\theta(s, a, \boldsymbol{\omega})] + \alpha H(\pi_\phi(\cdot \mid s, \boldsymbol{\omega})). \tag{3}$$

**Multi-objective distributional critic**: C51-style (Bellemare et al., 2017) distributional critic has been a crucial component in improving the performance of existing deep RL algorithms (Lee et al., 2025b; Palenicek et al., 2026). However, the categorical critic cannot be directly applied to model the multivariate returns distribution in MORL. Instead, we notice that the critic predictions only affect the policy optimization via the scalarized Q-value $\boldsymbol{\omega}^T \boldsymbol{Q}(s, a, \boldsymbol{\omega})$. Thus, we can avoid learning the multivariate return distribution, and propose to use a conditioned distributional critic $Z_\theta(s, a, \boldsymbol{\omega})$, which learns the univariate distribution of the scalarized returns $\boldsymbol{\omega}^T \mathbf{G}^{\pi(\cdot|\cdot,\boldsymbol{\omega})}(s, a)$. This is done by

minimizing the cross-entropy (CE) loss

$$\mathcal{L}_Z(\theta) = \mathbb{E}_{(s,a,\boldsymbol{r},s',\boldsymbol{\omega})\sim\mathcal{D}} H(\hat{Z}(\boldsymbol{\omega}^T\boldsymbol{r}, s'), Z_\theta(s,a,\boldsymbol{\omega})) \tag{4}$$

where $\hat{Z}$ is the TD(0) bootstrap estimate of the scalarized returns distribution computed using the scalarized reward $\boldsymbol{\omega}^T\boldsymbol{r}$. The actor then maximizes the value

$$\mathbb{E}_{a\sim\pi_\phi}\left[\mathbb{E}[Z_\theta(s,a,\boldsymbol{\omega})]\right] + \alpha H(\pi_\phi(\cdot \mid s,\boldsymbol{\omega})). \tag{5}$$

In order to bound the Q-values to the support of the distributional critic, SORL algorithms commonly normalize the rewards by the running standard deviation of the returns (Lee et al., 2025b; Palenicek et al., 2026). However, in MORL, the magnitude of returns can change significantly based on the preference, and thus, we propose to normalize the rewards component-wise by the maximum returns encountered throughout the training. Finally, in Appendix B we describe how the distributional critic can be modeled while retaining the non-scalarized expected returns, which are required, for instance, by envelope style algorithms (Yang et al., 2019; Li et al., 2025).

**Preference selection**: One crucial question in MORL algorithms is how the preference $\boldsymbol{\omega}$, used as conditioning variable, is selected during training (Xu et al., 2020; Hayes et al., 2022; Alegre et al., 2023). Indeed, identifying promising preferences can improve the convergence speed of the algorithms (Hayes et al., 2022; Alegre et al., 2023). As our goal is to study the effect of the architectural changes, we choose a fairly unsophisticated approach: We sample a new preference at the beginning of each episode from a static uniform distribution over the preference space. While this differs from CAPQL (Lu et al., 2023), where the authors sample a new preference at each timestep, the proposed approach is more in line with other work in MORL (Abels et al., 2019; Alegre et al., 2023; Xu et al., 2020; Li et al., 2025), thus supporting our goal of studying the effect of architectural changes. We provide the complete pseudocode, with our changes highlighted in Appendix E.

## 5 Experiments

This section is structured as follows: Firstly, Section 5.1 describes the experimental setup, including the baselines and benchmarks. Then Section 5.2 evaluates whether the proposed method results in improvements in asymptotic performance and sample efficiency. Lastly, Section 5.3 performs ablations over the three key components: (i) observation and feature normalization, (ii) weight normalization, and (iii) distributional critic, validating our proposed strategies.

### 5.1 Experiment setup

We compare the proposed approach to 4 baselines. CAPQL (Lu et al., 2023) is the underlying algorithm behind Momba, without any architectural improvements. PGMORL (Xu et al., 2020) and DPMORL (Cai et al., 2023) are multi-policy methods that train separate policies for each scalarization function, thus leading to a discrete approximation of the Pareto-front. GPI-LS (Alegre et al., 2023) obtains best-in-class sample efficiency by using General Policy Improvement (GPI) for selecting preferences during training. However, this comes with the downside of high computational cost (300k timesteps taking more than 120 hours in our experiments). We provide a more detailed discussion of the baselines and details on how they were run in Appendix F.

We use 7 continuous control tasks from Xu et al. (2020) for evaluating our methods. These tasks consist of 6 environments, implemented in the MuJoCo physics engine (Todorov et al., 2012): Ant, Swimmer, HalfCheetah, Humanoid, Walker2D, and Hopper. For Hopper, we consider a two- and three-objective variant (referred to as Hopper-v4 and Hopper-v3, respectively). Since SimbaV2 targets continuous control domains, we omit evaluation on discrete environments. Additionally, existing discrete benchmarks for MORL consist of small state or action spaces (Vamplew et al., 2011; Michailidis et al., 2026); thus, function approximation is unlikely to be the limiting factor. A detailed description of the environments, including the reference points used to compute the results, is provided in Appendix G.

For measuring the quality of the solution sets, we use two common metrics in MORL research, Hypervolume (HV) and Expected Utility Metric (EUM):

- **Hypervolume** HV (⇑) (Zitzler & Thiele, 1998) measures the space or volume enclosed by the solutions in the set P: $\text{HV}(P) = \int_{\mathbb{R}^n} \mathbb{1}_{H(P)}(z)dz$ where $H(P) = \{z \in Z | \exists i : 0 \leq i \leq |P|, r_0 \preceq z \preceq P(i)\}$. Here $P(i)$ is the $i^{\text{th}}$ solution, $r_0$ is the reference point and $\mathbb{1}_{H(P)}$ is an indicator function.
- **Expected Utility Metric** EUM (⇑) (Zintgraf et al., 2015) captures the expected utility for a user from a given solution set, defined as $\text{EUM}(P) = \mathbb{E}_{f \sim P_f} [\max_{\pi \in P} f(\boldsymbol{V}^{\pi})]$, where $P$ is the solution set and $P_f$ is distribution over scalarization functions, which simplifies to distribution over the preferences, as we are limited to linear scalarization functions.

As noted by Zintgraf et al. (2015), hypervolume can detect improvements in uniformity, spread, and convergence of the solution set, making it a good indicator of the overall quality, and thus, we use HV as our main metric. When reporting aggregate metrics, we normalize the HV and EUM values by the mean values Momba obtained after being trained for 1M steps. Regardless, we report both HV and EUM when comparing against the baselines, while for ablations, we only report HV, as we notice that both metrics behave similarly under linear scalarization. Notably, both metrics fail to explicitly capture the diversity of the generated Pareto front. Ideally, a high-quality solution set exhibits both high density and uniform dispersion across the objective space, thereby facilitating fine-grained trade-off exploration (Hayes et al., 2022). However, standard metrics, such as sparsity, rely on Euclidean distances between solutions, rendering them ill-suited for comparing fronts with varying convergence rates; Indeed, optimal sparsity can be trivially achieved by clustering solutions (Zhu et al., 2023; Liu et al., 2025). Consequently, we resort to visualizing the obtained solution sets for qualitative assessment.

To measure performance, we first approximate the Pareto front by averaging returns across 5 episodes for different preferences. Following Alegre et al. (2023), we use 100 equally spaced preferences for general policy methods, while PGMORL and DPMORL are evaluated based on their current (policy, utility function) pairs. Finally, we compute HV and EUM from the approximated Pareto front. When reporting results, we use interquartile mean (IQM) and stratified bootstrapped confidence intervals (SBCIs) as recommended by Agarwal et al. (2021).

### 5.2 Experimental validation

We begin by investigating whether the proposed method can improve in i) asymptotic performance and ii) sample efficiency over the baselines. Figure 1, displayed on the first page, answers the former question, showcasing the performance of the final solution set, aggregated over all environments. The proposed approach *outperforms baselines*, achieving $\sim 35\%$ improvement in HV and $\sim 16\%$ improvement in EUM over the runner-up, PGMORL. When compared to CAPQL, the underlying algorithm behind Momba, we achieve $\sim 132\%$ and $\sim 35\%$ improvements in aggregate HV and EUM, respectively. However, we wish to acknowledge that the performance we report for DPMORL (Cai et al., 2023) is worse than expected based on the original paper. We discuss the possible reasons for the discrepancy in DPMORL performance in Appendix F.

To examine if the proposed approach improves the sample efficiency of the underlying algorithm, Figure 2 displays the training curves of CAPQL and Momba in 3 environments. In addition to improving on CAPQL, in Ant and Humanoid, Momba reaches the final performance of PGMORL after training only for 100k and 200k steps respectively. We highlight that while CAPQL could not match the performance of PGMORL in any of the shown environments, Momba outperforms PGMORL in 2 of the 3 environments, and matches it in the challenging 3-objective Hopper environment, while requiring a fraction of the training steps.

To further study the sample efficiency of our method, we compare it against GPI-LS, a method with best-in-class sample efficiency. Figure 3 displays the training curves for normalized HV and EUM with 95% SBCIs aggregated over all environments. To remove the effect of update-to-data (UTD)

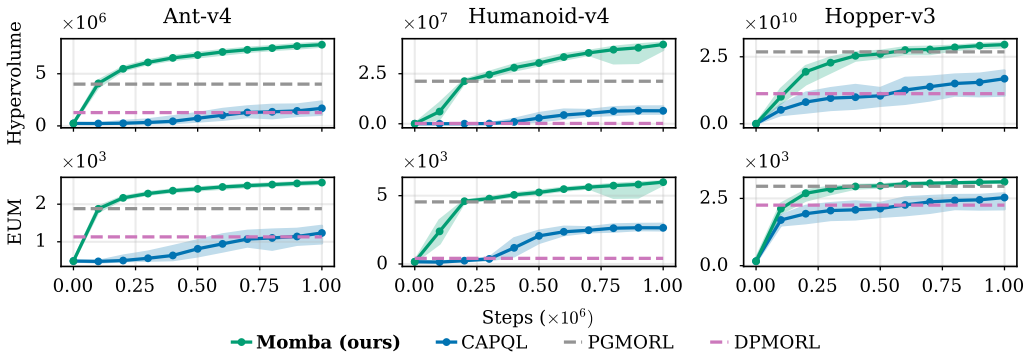


Figure 2: **Momba substantially improves sample efficiency against CAPQL**. The figures show HV (top) and EUM (bottom) (IQM and 95% SBCIs over 10 seeds) during training. Horizontal lines indicate final performance of PGMORL at $4.8 \times 10^7$ timesteps for Ant and $12 \times 10^7$ timesteps for Humanoid and Hopper, and final performance of DPMORL at $1 \times 10^7$ timesteps.

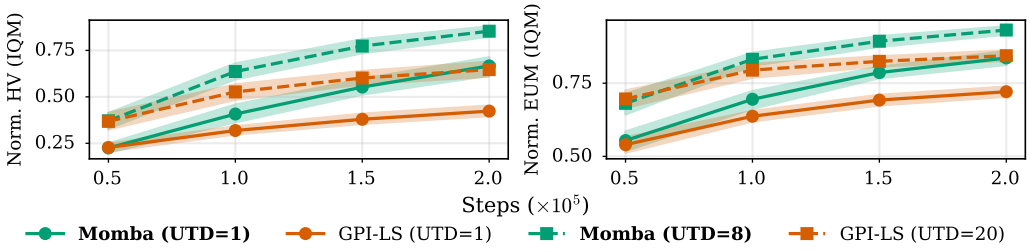


Figure 3: **Momba outperforms GPI-LS at similar UTD after 200k steps**. We compare the normalized HV and EUM (IQM and 95% SBCIs over 7 seeds) of Momba and GPI-LS with different UTD-ratios during the first 200k steps.

ratio, we show the results at $\text{UTD} = \{\mathbf{1}, 8\}$ for Momba and at $\text{UTD} = \{1, \mathbf{20}\}$ for GPI-LS, where the default value is bolded. The results demonstrate that when paired with a similar UTD ratio, Momba can beat the sample efficiency of GPI-LS. We want to emphasize that this result is achieved by training Momba *using preferences sampled from a static distribution*, whereas GPI-LS uses a sophisticated strategy for preference selection, highlighting the importance of an expressive neural network.

Lastly, we display the generated solution sets in 3 environments in Figure 4 for qualitative analysis. In Walker2d and Hopper-v4, Momba produces solution sets with good coverage, outperforming other methods. In HalfCheetah, Momba has a more limited coverage, while still producing a solution set that dominates most of the baselines. We note that in Walker2d and HalfCheetah, the upper left corner contains gaps, which we attribute to our use of linear scalarization. While entropy regularization turns the set of induced value functions strictly convex (Lu et al., 2023), it is possible that a stronger regularization would be required to reliably obtain policies in these regions.

### 5.3 Validating design choices

In this section, we explore the effectiveness of our three key components: (i) distributional critic, (ii) feature and observation normalization, and (iii) weight normalization. We begin by comparing the distributional critic with the cross-entropy loss (CE) to the non-distributional critic with standard mean squared error loss (MSE). To ensure that any performance improvements are not due to architectural changes introduced in Simba, such as normalizations and residual connec-

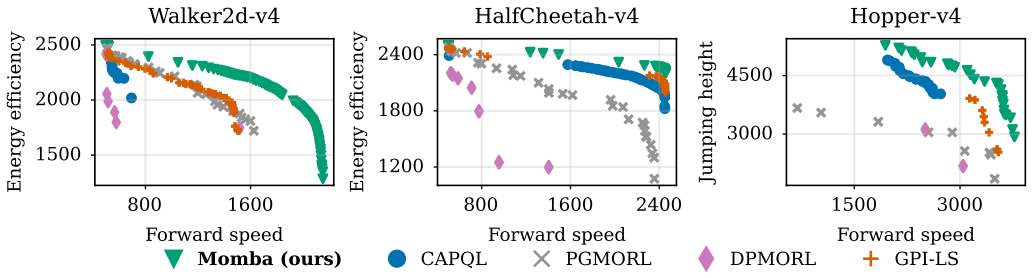


Figure 4: **Momba produces solution sets with good coverage**. We visualize the generated solution sets in Walker2d (left), HalfCheetah (middle), and Hopper (right). Momba produces superb solution sets in Walker2d and Hopper. Gaps in the upper left corner of Walker2d and HalfCheetah are likely a result of poor numerical properties caused by linear scalarization.

tions, we test the performance of the proposed distributional critic with the standard 2-layer MLP, commonly used to represent the Q-network in MORL algorithms. In order to eliminate the effect of the size of the critic, we vary the hidden dimension from 256 to 2048 and from 64 to 512 for MLP and Simba, respectively, resulting in networks with approximately similar parameter sizes. Figure 5 showcases that the proposed approach (Momba=Simba+CE) clearly outperforms the Simba+MSE variant across all critic sizes. Perhaps surprisingly, the standard MLP with categorical loss (MLP+CE) *matches or even outperforms Simba+MSE*, showcasing the effectiveness of the proposed categorical critic adaptation to the multi-objective domain. The default MLP with MSE loss corresponds to a parameter-scaled version of CAPQL, the base algorithm behind Momba. As expected, this combination retains poor performance, regardless of the critic size, implying that the performance gains are *not due to the increase in the parameter counts*, but rather stem from the improvements to function approximation and critic loss.

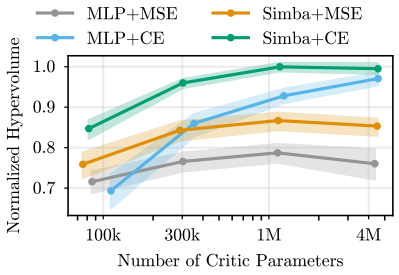


Figure 5: **The categorical critic adaptation is highly effective**. The figure displays normalized HV (IQM and 95% SBCIs over 10 seeds) as a function of the critic size. Momba is Simba+CE.

To validate the effect of the (i) weight normalization (WN), (ii) feature normalization (FN), and (iii) observation normalization (ON), we evaluate Momba under all combinations of these normalization methods. Figure 6 displays the normalized HV and 95% SBCIs over 10 seeds, aggregated over all tasks, and the Shapley values of the three different normalization techniques.

The results demonstrate that observation normalization was the most effective technique on our benchmarks, accounting for $\approx$ 55% of the improvements according to the Shapley values. While feature and weight normalization provide more modest improvements, all normalization schemes synergistically improve the performance.

## 6 Conclusions & Discussions

In this paper, we investigated whether the recent advances in neural network design, such as i) feature & observation normalization, ii) weight normalization, and iii) distributional critic, can improve the performance of an existing MORL algorithm. We applied and adapted a recent deep RL architecture, SimbaV2, to an existing entropy-regularized MORL algorithm, which we kept close to the original, only changing the preference sampling rate and critic loss. To deal with the multivariate return distribution, we proposed a simple yet effective tactic of modeling scalarized return distri-

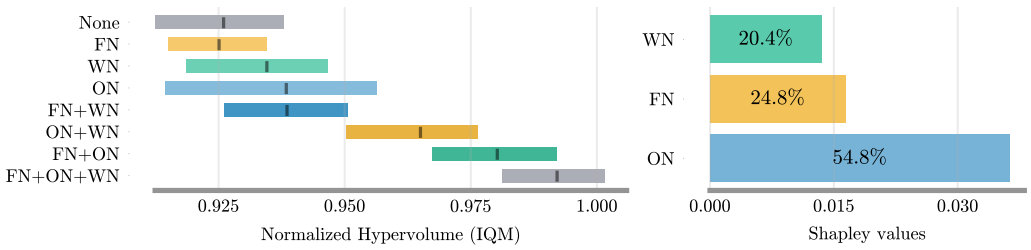


Figure 6: **Observation normalization is the most effective component**. Left: We investigate the effect of i) Feature Normalization (FN), ii) Observation normalization (ON), and iii) Weight Normalization (WN) to the final HV (IQM and 95% SBCIs over 10 seeds). Momba is FN+ON+WN. Right: Observation normalization is associated with the highest Shapley value, while weight and feature normalization are given similar importance.

bution, motivated by the prevalent scalarized expected return objective. Our extensive experiments demonstrated that one can achieve competitive performance while matching or improving the sample efficiency of the base algorithm in multi-objective continuous control tasks. We demonstrated the effectiveness of our adaptations via ablations.

We acknowledge that our study has several limitations. First, we restrict attention to linear scalarization. While common in MORL, nonlinear or learned scalarization may be preferable in some settings (Hayes et al., 2022; Cai et al., 2023). Notably, our distributional critic does not assume a specific scalarization form; the constraint arises from TD learning via the Bellman error, which may not hold under nonlinear scalarization (Roijers et al., 2018; Hayes et al., 2022). Second, our benchmarks are limited to continuous control tasks, with 2 objectives, with only one with 3 objectives. This makes it difficult to assess how applicable the proposed changes would be in discrete state and action spaces, and how well the approach would scale with increasing number of objectives. Finally, our more expressive function approximators can increase wall-clock time, though recent systems advances (e.g., JIT compilation) can mitigate this (Frostig et al., 2018; Ansel et al., 2024), and replacing SimbaV2 with the XQC architecture may offer similar gains with simpler designs (Palenicek et al., 2026).

### Acknowledgments

Adam Štafa and Petr Novotný were supported by U.S. Army Research Office under contract number W911NF261A180. Santeri Heiskanen was supported by the Ministry of Education and Culture's Doctoral Education Pilot in Finland (Decision No. VN/3137/2024-OKM-6; Finnish Doctoral Program Network in Artificial Intelligence, AI-DOC). We acknowledge the computational resources provided by the Aalto Science-IT project.

## References

Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic Weights in Multi-Objective Deep Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, 2021.

Lucas N. Alegre, Ana L. C. Bazzan, Diederik M. Roijers, Ann Nowé, and Bruno C. da Silva. Sample-Efficient Multi-Objective Learning via Generalized Policy Improvement Prioritization. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2023.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM international conference on architectural support for programming languages and operating systems, volume 2*, 2024.

Toygun Basaklar, Suat Gumussoy, and Umit Ogras. PD-MORL: Preference-driven multi-objective reinforcement learning algorithm. In *The Eleventh International Conference on Learning Representations*, 2023.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Xin-Qiang Cai, Pushi Zhang, Li Zhao, Jiang Bian, Masashi Sugiyama, and Ashley Juan Llorens. Distributional Pareto-Optimal Multi-Objective Reinforcement Learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Krishnendu Chatterjee, Rupak Majumdar, and Thomas A Henzinger. Markov decision processes with multiple objectives. In *STACS 2006*, 2006.

Florian Felten, Lucas Nunes Alegre, Ann Nowe, Ana L. C. Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno Castro da Silva. A Toolkit for Reliable Benchmarking and Research in Multi-Objective Reinforcement Learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *SysML conference*, 2018.

Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International conference on machine learning*, 2018b.

Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. A Practical Guide to Multi-Objective Reinforcement Learning and Planning. *Autonomous Agents and Multi-Agent Systems*, 36(1), 2022.

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning. In *International Conference on Learning Representations*, 2022.

Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(8), 2023.

Ammar Jalalimanesh, Hamidreza Shahabi Haghighi, Abbas Ahmadi, Hossein Hejazian, and Madjid Soltani. Multi-objective optimization of radiotherapy: distributed Q-learning and agent-based simulation. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(5), 2017.

Panagiotis Kyriakis, Jyotirmoy Deshmukh, and Paul Bogdan. Pareto policy adaptation. In *International Conference on Learning Representations*, 2022.

Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R. Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. SimBa: Simplicity Bias for Scaling Up Parameters in Deep Reinforcement Learning. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical Normalization for Scalable Deep Reinforcement Learning. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025b.

Pengyi Li, Hongyao Tang, Yifu Yuan, Jianye Hao, Zibin Dong, and Yan Zheng. Cola: Towards efficient multi-objective reinforcement learning with conflict objective regularization in latent space. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ruohong Liu, Yuxin Pan, Linjie Xu, Lei Song, Pengcheng You, Yize Chen, and Jiang Bian. Efficient discovery of pareto front for multi-objective reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Haoye Lu, Daniel Herman, and Yaoliang Yu. Multi-Objective Reinforcement Learning: Convexity, Stationarity and Pareto Optimality. In *The Eleventh International Conference on Learning Representations*, 2023.

Dimitris Michailidis, Willem Röpke, Diederik M. Roijers, Sennay Ghebreab, and Fernando P. Santos. Scalable multi-objective reinforcement learning with fairness guarantees using lorenz dominance. *Journal of Artificial Intelligence Research*, 85, 2026.

Calarina Muslimani, Kerrick Johnstonbaugh, Suyog Chandramouli, Serena Booth, W. Bradley Knox, and Matthew E. Taylor. Towards improving reward design in RL: A reward alignment metric for RL practitioners. *Reinforcement Learning Journal*, 6, 2025.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample efficient continuous control. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The Primacy Bias in Deep Reinforcement Learning. In *Proceedings of the 39th International Conference on Machine Learning*. PMLR, 2022.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, 9(1), 2017.

Daniel Palenicek, Florian Vogt, Joe Watson, and Jan Peters. Scaling off-policy reinforcement learning with batch and weight normalization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Daniel Palenicek, Florian Vogt, Joe Watson, Ingmar Posner, and Jan Peters. XQC: Well-conditioned optimization accelerates deep reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research*, 48, 2013.

Diederik M. Roijers, Denis Steckelmacher, and Ann Nowé. Multi-objective reinforcement learning for the expected utility of the return. In *Proceedings of the Adaptive and Learning Agents workshop at FAIM*, 2018.

Tianye Shu, Ke Shang, Cheng Gong, Yang Nan, and Hisao Ishibuchi. Learning pareto set for multi-objective continuous robot control. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, 2024.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems*, 2022.

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

Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84: 51–80, July 2011.

Harley Wiltzer, Jesse Farebrother, Arthur Gretton, and Mark Rowland. Foundations of Multivariate Distributional Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2024.

Jiexin Xie, Zhenzhou Shao, Yue Li, Yong Guan, and Jindong Tan. Deep reinforcement learning with optimized reward functions for robotic trajectory planning. *IEEE Access*, 7, 2019.

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*, 2019.

Pushi Zhang, Xiaoyu Chen, Li Zhao, Wei Xiong, Tao Qin, and Tie-Yan Liu. Distributional Reinforcement Learning for Multi-Dimensional Reward Functions. In *Advances in Neural Information Processing Systems*, 2021.

Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N. Zare, and Patrick Riley. Optimization of Molecules via Deep Reinforcement Learning. *Scientific Reports*, 9(1), 2019.

Baiting Zhu, Meihua Dang, and Aditya Grover. Scaling pareto-efficient decision making via offline multi-objective RL. In *The Eleventh International Conference on Learning Representations*, 2023.

Luisa M Zintgraf, Timon V Kanters, Diederik M Roijers, Frans A Oliehoek, and Philipp Beau. Quality assessment of MORL algorithms: A utility-based approach. In *Benelearn 2015: Proceedings of the 24th Annual Machine Learning Conference of Belgium and the Netherlands*, 2015.

Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms — a comparative case study. In *Parallel Problem Solving from Nature — PPSN V*, 1998.

# Supplementary Materials

*The following content was not necessarily subject to peer review.*

## A Full Evaluation Results

To facilitate more detailed analysis of the proposed method, we report the performance metrics of Momba and baselines across all environments in Table 1. The results showcase that Momba can outperform all the baselines in most of the continuous control tasks, with the exception of Swimmer, where PGMORL outperforms Momba. However, this environment is clearly difficult for general policy methods, likely due to high imbalance between the objective difficulties. It should be noted that in the rest of the environments, Momba's and the runner-up methods 95% SBCIs for Hypervolume don't overlap. While GPI-LS is trained for considerably fewer timesteps, this matches the default training protocol used in the original paper. Moreover, as is mentioned in Section 5.1, training GPI-LS for more timesteps is extremely time-consuming. The hyperparameters for Momba and a detailed description of the baselines are given in Appendix D and Appendix F respectively.

Table 1: Full results of our evaluation. We report the IQM and 95% SBCIs of hypervolume (⇑) and EUM (⇑) across 10 seeds for each algorithm and environment. The best and runner-up methods for each environment are **bolded** and underlined respectively.

| Environment | Metric | MOMBA | PGMORL | GPI-LS | CAPQL | DPMORL |
|---|---|---|---|---|---|---|
| ANT-V4 | Steps ($\times 10^6$) | 1 | 48 | 0.20 | 1 | 10 |
| | HV ($\times 10^6$) | **7.78** [7.61, 7.97] | 4.01 [3.16, 4.57] | 4.23 [3.73, 4.49] | 1.69 [1.08, 2.33] | 1.28 [1.17, 1.40] |
| | EUM ($\times 10^3$) | **2.58** [2.55, 2.61] | 1.88 [1.66, 2.00] | 1.93 [1.81, 1.99] | 1.23 [0.96, 1.42] | 1.13 [1.07, 1.19] |
| HALFCHEETAH-V4 | Steps ($\times 10^6$) | 1 | 30 | 0.20 | 1 | 10 |
| | HV ($\times 10^6$) | **5.84** [5.79, 5.87] | 4.86 [4.58, 5.01] | 5.53 [4.70, 5.61] | 5.42 [4.20, 5.52] | 2.39 [2.31, 2.46] |
| | EUM ($\times 10^3$) | **2.34** [2.33, 2.35] | 2.07 [2.02, 2.11] | 2.30 [2.10, 2.33] | 2.23 [1.99, 2.26] | 1.56 [1.54, 1.58] |
| HOPPER-V3 | Steps ($\times 10^6$) | 1 | 120 | 0.20 | 1 | 10 |
| | HV ($\times 10^{10}$) | **2.95** [2.85, 3.02] | 2.68 [2.62, 2.80] | 2.07 [1.72, 2.23] | 1.68 [1.05, 1.99] | 1.12 [0.96, 1.30] |
| | EUM ($\times 10^3$) | **3.12** [3.09, 3.13] | 2.95 [2.92, 3.00] | 2.81 [2.65, 2.84] | 2.53 [2.10, 2.69] | 2.25 [2.12, 2.38] |
| HOPPER-V4 | Steps ($\times 10^6$) | 1 | 48 | 0.20 | 1 | 10 |
| | HV ($\times 10^7$) | **1.81** [1.78, 1.85] | 1.53 [1.40, 1.64] | 1.35 [1.29, 1.42] | 1.21 [1.11, 1.30] | 0.88 [0.75, 1.00] |
| | EUM ($\times 10^3$) | **4.07** [4.03, 4.10] | 3.76 [3.60, 3.90] | 3.55 [3.48, 3.63] | 3.43 [3.27, 3.54] | 2.92 [2.70, 3.10] |
| HUMANOID-V4 | Steps ($\times 10^6$) | 1 | 120 | 0.20 | 1 | 10 |
| | HV ($\times 10^7$) | **3.96** [3.69, 4.05] | 2.12 [2.02, 2.27] | 0.00 [0.00, 0.00] | 0.65 [0.51, 0.86] | 0.02 [0.01, 0.02] |
| | EUM ($\times 10^3$) | **5.99** [5.81, 6.05] | 4.54 [4.48, 4.63] | -0.01 [-0.02, 0.16] | 2.65 [2.47, 2.94] | 0.41 [0.39, 0.46] |
| SWIMMER-V4 | Steps ($\times 10^6$) | 1 | 12 | 0.20 | 1 | 10 |
| | HV ($\times 10^4$) | 1.66 [1.57, 1.74] | **1.75** [1.10, 2.29] | 0.75 [0.73, 0.98] | 0.56 [0.52, 0.61] | 1.20 [1.17, 1.31] |
| | EUM ($\times 10^2$) | 1.23 [1.21, 1.25] | **1.27** [1.07, 1.45] | 0.99 [0.98, 1.04] | 0.93 [0.92, 0.94] | 1.10 [1.09, 1.13] |
| WALKER2D-V4 | Steps ($\times 10^6$) | 1 | 30 | 0.20 | 1 | 10 |
| | HV ($\times 10^6$) | **5.15** [4.17, 5.32] | 3.47 [3.23, 3.64] | 3.33 [3.03, 3.67] | 2.38 [1.82, 2.66] | 2.54 [2.33, 2.70] |
| | EUM ($\times 10^3$) | **2.12** [1.76, 2.15] | 1.80 [1.77, 1.83] | 1.78 [1.72, 1.84] | 1.62 [1.54, 1.67] | 1.66 [1.61, 1.70] |

## B Vectorized versus scalarized returns

In Section 4, we propose to directly learn the scalarized distributional returns. However, in MORL literature, it is customary to predict the vector returns, which allows algorithms to take advantage of this information, for e.g., envelope-style updates (Yang et al., 2019; Li et al., 2025). We show that it is possible to learn the vector returns with the distributional critic by approximating the marginals of

the returns distributions, from which one can recover the vector Q-values by taking component-wise expectation of the critic output.

To achieve this, we implemented the vector distributional critic as $d$ distributional critics $Z_\theta^1, \ldots, Z_\theta^d$, each predicting the distribution of one component of the $d$-dimensional returns. The critic loss is

$$\mathcal{L}_Z(\theta) = \mathbb{E}_{(s,a,\boldsymbol{r},s',\boldsymbol{\omega})\sim\mathcal{D}} \sum_{i=1}^{d} H(\hat{Z}_i, Z_\theta^i(s,a,\boldsymbol{\omega})). \tag{6}$$

where $\hat{Z}_i$ is computed from the $i$-th component of the reward $\boldsymbol{r}_i$. The actor then maximizes the value

$$\mathbb{E}_{a\sim\pi_\phi}\left[\sum_{i=1}^{d} \boldsymbol{\omega}_i \mathbb{E}\left[Z_\theta^i(s,a,\boldsymbol{\omega})\right]\right] + \alpha H(\pi_\phi(\cdot \mid s,\boldsymbol{\omega})). \tag{7}$$

In practice, the vector critic is implemented as separate output layers, thus sharing most of their parameters. Analogously to CAPQL, we implement clipped double Q-learning by taking argmin of the scalarized (non-distributional) value.

We compared the vector critic to the scalarized critic, used in the main text, in Figure 7. Similarly to Section 5.3, the comparison is performed with Simba and MLP architecture. For completeness, we also report the performance of the non-distributional vector critic. As can be seen in the Figure 7, the performance of the two approaches is nearly identical, demonstrating that our proposed adaptation of the distributional critic could also be considered in cases where the vector-values are required.

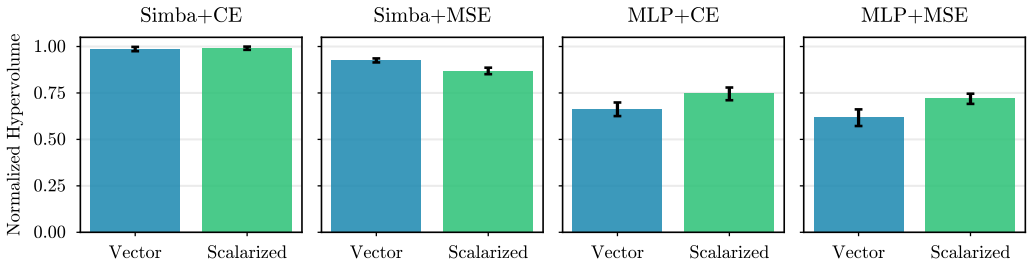


Figure 7: Normalized HV (IQM and 95% SBCIs from 10 seeds), aggregated over all environments, with all proposed components. Learning the distribution of each reward component separately matches the performance of learning the scalarized return distribution. Momba is Simba+CE.

## C Detailed description of architectural improvements

As mentioned in Section 3, SimbaV2 architecture (Lee et al., 2025b) consists of three main components, which we describe in detail here.

**Observation and feature normalization**: We follow Lee et al. (2025b), and apply Running Statistic Normalization (RSNorm) to normalize each dimension of the observations $\boldsymbol{o}$ to zero mean and unit variance. More specifically, at each timestep $t$, the running mean $\boldsymbol{\mu}_t \in \mathbb{R}^{|O|}$ and variance $\boldsymbol{\sigma}^2 \in \mathbb{R}^{|O|}$ are updated with the standard online algorithm

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + \frac{1}{t}\boldsymbol{\delta}_t, \quad \boldsymbol{\sigma}_t^2 = \boldsymbol{\sigma}_{t-1}^2 + \frac{1}{t}(\boldsymbol{\delta}_t^2 - \boldsymbol{\sigma}_{t-1}^2) \tag{8}$$

where $\boldsymbol{\delta}_t = \boldsymbol{o}_t - \boldsymbol{\mu}_{t-1}$. Using these running statistics, a given observation is then normalized as

$$\overline{\boldsymbol{o}}_t = \mathrm{RSNorm}(\boldsymbol{o}_t) = \frac{\boldsymbol{o}_t - \boldsymbol{\mu}_t}{\sqrt{\boldsymbol{\sigma}_t^2 + \epsilon}} \tag{9}$$

The normalized observation $\overline{\boldsymbol{o}}$ is then mapped onto a unit hypersphere to stabilize learning. To ensure that the magnitude information is not lost during the mapping, Lee et al. (2025b) propose to concatenate a positive constant $c_{\text{shift}}$ to the observation vector before mapping the vector. Then, the final normalized observation becomes

$$\tilde{\boldsymbol{o}}_t = \text{L2norm}([\overline{\boldsymbol{o}}_t, c_{\text{shift}}]) \tag{10}$$

where $\text{L2norm}$ divides each component of the vector by the square root of sum of squares of the vector, and $[\cdot, \cdot]$ denotes the concatenation operation.

**Architecture**: SimbaV2 introduces two new formulations for the learnable layers. Firstly, instead of using standard linear layers, Lee et al. (2025b) decompose the standard linear layer into a linear layer (without a bias) with weights that are constrained to the unit hypersphere, and a learnable scaling vector that is applied element-wise. To ensure that the weights remain in the hypersphere, $\text{L2Norm}$ is applied after each gradient update. Secondly, instead of using traditional skip-connections, the authors propose the use of learnable interpolation layers (LERP), which can interpolate between the original input $\boldsymbol{h}^l$ and non-linearly transformed input $\tilde{\boldsymbol{h}}^l$:

$$\boldsymbol{h}^{l+1} = \text{L2Norm}((1-\alpha) \odot \boldsymbol{h}_t^l + \alpha \odot \tilde{\boldsymbol{h}}_t^l) \tag{11}$$

where $\alpha$ is a learnable parameter.

**Reward scaling**: as noted in Section 4, the C51-style categorical critic requires one to map the returns to interval $[G_{\min}, G_{\max}]$. We follow SimbaV2 (Lee et al., 2025b), and achieve this by applying the normalization to rewards directly. In addition to limiting the returns to a given interval, this also i) provides stable gradient for both actor and critic, and ii) ensures that the reward components have similar magnitudes, which is crucial when applying reward scalarization. In detail, we normalize each reward component based on the observed maximum returns so far:

$$\tilde{R}_t^{\,i} = v_{\max} \cdot \frac{R_t^i}{G_{t,\max}^i}, \quad i = 1, \dots d \tag{12}$$

where $v_{\max} \in \mathbb{R}$ is the desired normalized return, $R_t^i$ is the $i\text{th}$ reward component and $G_{t,\max}^i$ is the maximum observed return for the $i\text{th}$ reward component. Intuitively, this normalization simply ensures that the normalized returns are in the symmetric range $[-v_{\max}, v_{\max}]$. SimbaV2 also included an additional term to ensure that the variance of the rewards is close to 1, but we found that to bring no benefits, and thus excluded it.

## D Training details

We report the default hyperparameters used in our experiments in Table 2. We did not perform extensive hyperparameter optimization, and thus, it is possible that better performance could be extracted via large-scale hyperparameter search. Best return scale denotes the coefficient used to compute the scaling factor for rewards. Notably, we automatically tune the entropy regularization coefficient, while the original CAPQL implementation used fixed entropy coefficients, tuned specifically for each environment.

## E Algorithmic modifications

As noted in the main text, while our goal was to avoid making algorithmic changes, we ended up doing *three* changes to the original CAPQL (Lu et al., 2023) to make our results more broadly applicable. Firstly, instead of using normal distribution as our preference sampling distribution, as done by Lu et al. (2023), we opt to use Dirichlet distribution $\text{Dir}(\mathbf{1}_k)$ instead. This choice was made mostly due to convenience, as the Dirichlet distribution has exactly the same support as our preference space $\Omega$. Secondly, we sample a new preference once per episode (Line 4, Algorithm 1),

Table 2: Default hyperparameters for Momba, used in our main experiments

| Parameter | Value(s) | Parameter | Value(s) |
|---|---|---|---|
| Training steps | 1M | Actor blocks | 2 |
| Replay buffer size | 1M | Actor hidden dim | 128 |
| Initial training steps | 500 | Critic blocks | 2 |
| $\gamma$ | 0.99 | Critic hidden dim | 256 |
| Target critic momentum | 0.05 | Number of critics | 2 |
| Optimizer | Adam | Number of atoms | 101 |
| Policy LR | $1 \times 10^{-4}$ | Best return scale | 3.0 |
| Critic LR | $1 \times 10^{-4}$ | Categorical support | $[-5, 5]$ |
| Initial $\alpha$ | 0.2 | UTD | 1 |
| Target $\alpha$ | $-\lvert A \rvert$ | Batch size | 256 |

as opposed to sampling a new preference for each timestep. This change was done, since this is a far more common choice in MORL algorithms (Abels et al., 2019; Xu et al., 2020; Alegre et al., 2023; Li et al., 2025). Lastly, we update the critic loss function (Line 11 in Algorithm 1) to use the categorical loss, similar to Bellemare et al. (2017). As discussed in the main text, using categorical loss for the critic is one of the main components driving the improved performance of the proposed architecture. The complete algorithm, with the proposed changes highlighted, is displayed in Algorithm 1.

**Algorithm 1:** CAPQL algorithm Lu et al. (2023) with the modifications highlighted in blue

**Input :** preference sampling distribution $D_\psi = \mathrm{Dir}(\mathbf{1}_d)$

1 Initialize parameter vectors $\theta_1, \theta_2, \overline{\theta}_1, \overline{\theta}_2, \phi$

2 $\overline{\theta}_i \leftarrow \theta_i$ for $i \in \{1, 2\}$

3 **foreach** *iteration* **do**

4 Sample $\boldsymbol{\omega} \sim D_\psi$ `// Sample preference once per episode`

5 **foreach** *environment step* **do**

6 $a_t \sim \pi_\phi(s_t, \boldsymbol{\omega})$

7 $s_{t+1} \sim \mathcal{P}(a_t, s_t)$

8 $\mathcal{D} \sim \mathcal{D} \cup \{(s_t, a_t, \mathcal{R}(a_t, s_t), s_{t+1}, \boldsymbol{\omega})\}$

9 **foreach** *training step* **do**

10 $\mathcal{S} \leftarrow$ sample $N$ transitions from $\mathcal{D}$

`/* Update Critic networks using categorical loss */`

11 $\theta_i \leftarrow \theta_i - \lambda_\theta \nabla_{\theta_i} (\mathbb{E}_{\mathcal{S}} H(\hat{Z}(s_t, a_t, \boldsymbol{\omega}), Z_{\theta_i}(s_t, a_t, \boldsymbol{\omega}))$ for $i \in \{1, 2\}$

12 Where $\hat{Z}(s_t, a_t, \boldsymbol{\omega}) = \boldsymbol{\omega}^T \mathcal{R}(a_t, s_t) + \gamma(\min_{i \in \{1,2\}} Z_{\overline{\theta}_i}(s_{t+1}, a_{t+1}, \boldsymbol{\omega}) - \alpha \log\ \pi_\phi(a_{t+1}, s_{t+1}, \boldsymbol{\omega})\mathbf{1})$ and $a_{t+1} \sim \pi_\phi(s_{t+1}, \boldsymbol{\omega})$

`/* Update the policy parameters */`

13 $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi \mathbb{E}_{\mathcal{S}} \left( D_{\mathrm{KL}}(\pi_\phi(\cdot, s_j, \boldsymbol{\omega}) \| \frac{\exp(\boldsymbol{\omega}^T \min_{i \in \{1,2\}} Q_{\theta_i}(s_j, \boldsymbol{\omega})/\alpha)}{\Delta(s_j, \boldsymbol{\omega})} \right)$

14 with $\Delta(s_j, \boldsymbol{\omega}) = \int_{\mathcal{A}} \exp(\boldsymbol{\omega}^T \min_{\{1,2\}} Q_{\theta_i}(s_j, a, \boldsymbol{\omega})/\alpha)\, da$

15 $\overline{\theta}_i \leftarrow \tau\theta_i + (1 - \tau)\theta_i$ for $i \in \{1, 2\}$

## F Baselines

In this section, we give a brief description of the baselines used in this work, and how the results showcased in the main text were obtained.

**CAPQL** (Lu et al., 2023) is a general policy method that also acts as a backbone of our experiments. It utilizes an entropy regularized formulation, similar to Soft-Actor Critic (SAC) (Haarnoja et al.,

2018b) for learning an approximation of the solution set. However, the authors' motivation differs from SAC: They showcase that by utilizing entropy regularization, one can ensure that the solution set becomes strictly convex, making it possible to retrieve flat portions of the Pareto front even with linear scalarization. In our experiments, we used an implementation from MORL-baselines (Felten et al., 2023), utilizing a 2 layer MLP for representing the policy and value-functions and the algorithm was trained for $1 \times 10^6$ timesteps.

**GPI-LS** (Alegre et al., 2023) is a recent MORL method that, similarly to CAPQL, learns a single policy that is conditioned on the desired preference. The algorithm prioritizes preferences during training based on the magnitude of achievable improvement via General Policy Improvement (GPI). The novel GPI-based algorithm leads to best-in-class sample efficiency, yet the runtime of the method is quite high, as the process for evaluating the GPI policy is time-consuming. In this paper, we use the original implementation from MORL-baselines (Felten et al., 2023). Notably, this implementation uses layer normalizations and DroQ networks (Hiraoka et al., 2022), making it the only baseline to utilize more recent architectural improvements. We train the algorithms for 200k timesteps, as this is close to the default setting used in the original paper. We also tried training the algorithms for 300k timesteps, but in this case, the experiments could not finish in 5 days (120 hours). We also note that GPI-LS performs *additional environment steps* to evaluate the policy at corner weights (refer to Algorithm 1 in (Alegre et al., 2023)) during the GPI update. To the best of our knowledge, these additional steps ***are not*** counted towards the timesteps reported in the training curves.

**PGMORL** (Xu et al., 2020) represents a multi-policy method that trains separate neural networks for different preferences. The authors evolve a population of policies using an evolutionary algorithm, which is paired with a prediction model that tries to detect which preferences will improve the current solution set the most. The individual policies are trained using multi-objective policy gradient. We utilize the original PGMORL implementation in our experiments, using the original hyperparameters and two-layer MLPs for representing policies. With these parameters, the method uses $1.2 \times 10^7$ to $1.2 \times 10^8$ timesteps, depending on the environment.

**DPMORL** (Cai et al., 2023) is a multi-policy method that firstly learns a set of plausible utility functions using a diversity-based objective, and then trains separate policies for each learned utility function. The authors also modify the objective for the policy optimization to take into account distributed returns, thus offering a direct comparison with our simpler approach for modeling the distributional returns. In our experiments, we utilize the original implementation by the authors. We use the default hyperparameters from the original paper, and thus, in each environment, we train at most 10 policies, each represented using a 3 layer MLP. In total, these policies are trained for $1 \times 10^7$ timesteps.

We would like to note that in our evaluations (presented in Figures 1, 2 and 4), DPMORL performs worse than expected based on the results in the original paper. We hypothesize possible causes for this discrepancy. Firstly, the author-provided code seems to differ from the description in the paper, as it doesn't seem to augment the state space with the cumulative multivariate returns (refer to Algorithm 1 in Cai et al. (2023)). While we tried enabling the state-space augmentation, it resulted in even further performance deterioration. Secondly, the original code utilizes environment-specific precomputed values for normalizing the utility functions, which we do not have access to, as our environments differ from the original setup. To the best of our knowledge, the method for computing these normalization values is not mentioned in the original paper (this normalization seems to differ from the one mentioned in Appendix B in Cai et al. (2023)), making it challenging for us to reproduce the results.

For all evaluated algorithms, we summarize the number of trainable parameters and the number of training steps in Table 3. While Momba uses more trainable parameters than the baselines, its superior performance cannot be attributed solely to the number of parameters. In Figure 5, we showed that distributional value representation and the architectural components are necessary to enable parameter scaling.

Table 3: **Summary of evaluated algorithms.** We report the number of trainable parameters for the policy and value networks and the number of training steps. In multi-policy methods, the parameters are for a single policy, and the training steps are shared between the policies.

| Algorithm | Classification | Policies | Trainable parameters | Training steps |
|---|---|---|---|---|
| Momba | General policy | 1 | 2,500,805 | 1M |
| CAPQL | General policy | 1 | 258,723 | 1M |
| GPI-LS | General policy | 1 | 259,119 | 200k |
| DPMORL | Multi-policy | 10 | 17,477 | 10M |
| PGMORL | Multi-policy | 6–15 | 17,551 | 12M–120M |

## G Environments

In this section, we provide a brief overview of the environments used in this work. They are originally from (Xu et al., 2020), and they are implemented using Mujoco physics engine (Todorov et al., 2012). We use $r_i$ to denote the ith reward component. When computing Hypervolume, we use the vector of zeros $\mathbf{0}_d$ as our reference point in all environments.

**Ant-v4** has an observation space $\mathcal{S} \in \mathbb{R}^{27}$, and action-space $\mathcal{A} \in \mathbb{R}^8$. The agent is limited to 500 timesteps in this environment. The environment has two objectives, x-axis speed and y-axis speed:

$$r_1 = v_x + C$$
$$r_2 = v_y + C$$

where $v_x$ is the speed in $x$ direction, $v_y$ is the speed in $y$ direction and $C = 1 - 0.5 \sum_i a_i^2$ is combination of alive bonus and energy efficiency, defined as the squared sum of actions for each actuator.

**HalfCheetah-v4** has an observation-space $\mathcal{S} \in \mathbb{R}^{17}$ and action-space $\mathcal{A} \in \mathbb{R}^6$. The agent is limited to 500 timesteps in this environment. The environment has two objectives, forward speed and energy efficiency:

$$r_1 = \min(v_x, 4) + C$$

$$r_2 = 4 - \sum_i a_i^2 + C$$

where $v_x$ is the speed in $x$ direction, $C = 1$ is alive bonus and $a_i$ is the action of each actuator.

**Hopper-v4** has an observation-space $\mathcal{S} \in \mathbb{R}^{11}$ and action-space $\mathcal{A} \in \mathbb{R}^3$. The agent is limited to 500 timesteps in this environment. In the *two objective* cases, the rewards are forward speed and jumping height:

$$r_1 = 1.5 v_x + C$$
$$r_2 = 12(h - h_{\text{init}}) + C$$

where $v_x$ is the speed in $x$ direction, $C = 1 - 0.0002 \sum_i a_i^2$ combines alive bonus and energy efficiency, while $h$ is the current height and $h_{\text{init}}$ is the initial height.

In the *three objective* version, the rewards are forward speed, jumping height and energy efficiency

$$r_1 = 1.5 v_x + C$$
$$r_2 = 12(h - h_{\text{init}}) + C$$
$$r_3 = 4 - \sum_i a_i^2 + C$$

where $C = 1$ is the alive bonus, and the rest of the symbols have same meaning as in the two-objective configuration.

**Swimmer-v4** has an observation-space $\mathcal{S} \in \mathbb{R}^8$ and action-space $\mathcal{A} \in \mathbb{R}^2$. The agent is limited to 500 timesteps in this environment. The environment has two objectives, forward speed and energy efficiency:

$$r_1 = v_x$$
$$r_2 = 0.3 - 0.15 \sum_i a_i^2$$

where $v_x$ is the speed in $x$ direction and $a_i$ is the action of each actuator.

**Walker2d-v4** has an observation-space $\mathcal{S} \in \mathbb{R}^{17}$ and action-space $\mathcal{A} \in \mathbb{R}^6$. The agent is limited to 500 timesteps in this environment. The environment has two objectives, forward speed and energy efficiency:

$$r_1 = v_x + C$$
$$r_2 = 4 - \sum_i a_i^2 + C$$

where $C = 1$ is the alive bonus, $v_x$ is the speed in $x$ direction and $a_i$ is the action of each actuator.

**Humanoid-v4** has an observation-space $\mathcal{S} \in \mathbb{R}^{376}$ and action-space $\mathcal{A} \in \mathbb{R}^{17}$. The agent is limited to 500 timesteps in this environment. The environment has two objectives, forward speed and energy efficiency:

$$r_1 = 1.25 v_x + C$$
$$r_2 = 3 - 4 \sum_i a_i^2 + C$$

where $C = 3$ is the alive bonus, $v_x$ is the speed in $x$ direction and $a_i$ is the action of each actuator.

## H Further empirical evaluations

In this section, we showcase the training curves (Figure 8), UTD-scaling results (Figure 9) and obtained solution sets (Figure 10) in the environments not shown in the main text.

Figure 8 showcases that Momba outperforms its base algorithm, CAPQL, in all environments. We also outperform PGMORL, while requiring a fraction of the training steps in all environments except Swimmer, where Momba matches the performance of PGMORL. Interestingly, CAPQL converges very quickly to a sub-optimal policy in Swimmer, again highlighting the effectiveness of the proposed approach.

Figure 9 displays the UTD-scaling in different environments. While we see that the Hopper and Humanoid environments benefit from higher UTD, the performance improvements in other environments are limited. We note that in HalfCheetah, most algorithms start to approach the theoretical upper bound of $6.25 \times 10^6$ for hypervolume (see Figure 8), indicating that the environment may be too easy for the current methods.

Lastly, Figure 10 displays the generated solution sets in Humanoid, Swimmer, and Hopper-v3. Humanoid remains a difficult environment, as none of the algorithms can obtain a good coverage of the solution set. On the other hand, both CAPQL and GPI-LS, representing the general policy methods, struggle to find policies that can move the actor forward, instead collapsing and generating a set of policies that all optimize for the energy efficiency. On the contrary, Momba recovers a solution set that is on par with PGMORL, being the only general policy method to do so.

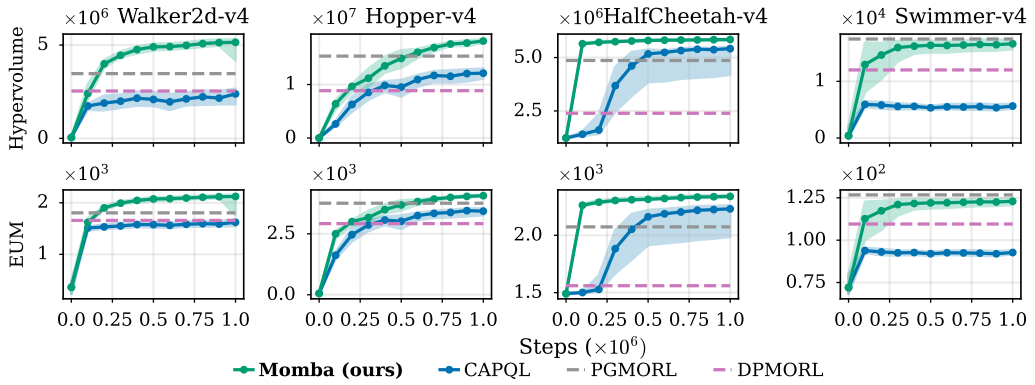


Figure 8: HV (top) and EUM (bottom) (IQM and 95% SBCIs over 10 seeds) during training. Horizontal lines indicate the final performance of PGMORL and DPMORL.

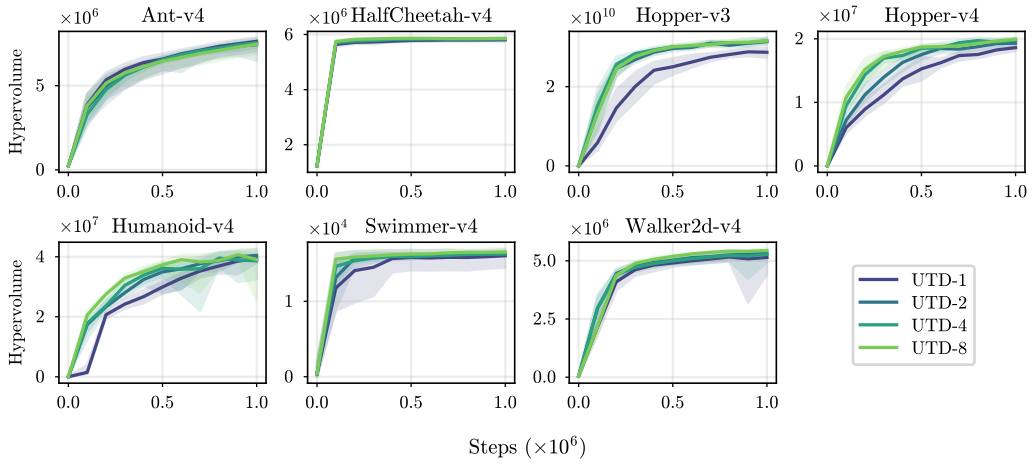


Figure 9: The figure shows the HV of Momba during training on all environments (IQM and 95% SBCIs over 10 seeds) with various UTD-ratios.

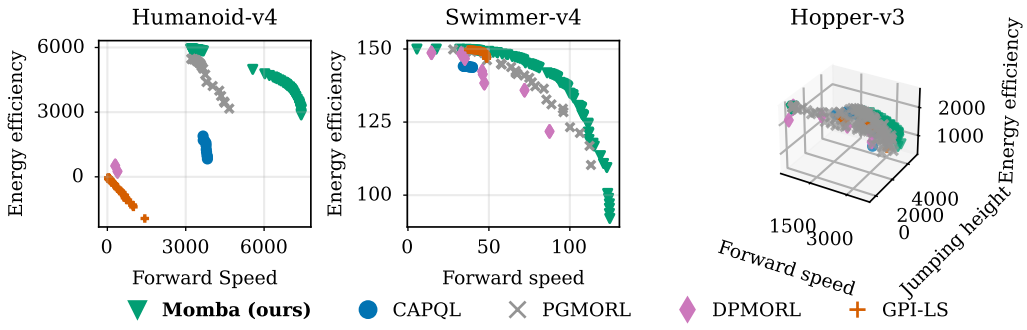


Figure 10: The figure displays the generated solution sets in Humanoid (left), Swimmer (center), and Hopper-v3 (right). Humanoid and Swimmer remain difficult environments for most of the algorithms.