Efficient Multi-Objective Reinforcement Learning with Expressive Function Approximators

by

Adam Štafa

Master's Thesis

Masaryk University

Declaration

Hereby I declare that this thesis is my original work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source. I acknowledge the use of artificial intelligence tools in the preparation of this thesis.

Advisor: doc. RNDr. Petr Novotný, Ph.D.

Acknowledgments

I would like to thank Petr Novotný for his guidance and for the opportunities he provided in my studies. I am also thankful to Santeri Heiskanen for making time for me every week and for sharing his knowledge and expertise throughout this work. Writing this thesis would not be nearly as rewarding without the productive and supportive environment of the Formela laboratory. Finally, I would like to express my heartfelt gratitude to my family, friends, and Miriam for their unwavering support.

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 architectural questions 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 their performance could be improved with more expressive function approximators.

In this thesis, we propose two tractable ways of modeling distributional returns in MORL and combine them with recent advances in function approximators for reinforcement learning. Empirical results across standard continuous-control benchmarks demonstrate that these changes substantially improve the quality of the produced solution sets and we identify the distributional value representation as the most important component. Additionally, we show that these improvements in function approximation can be combined with other algorithmic techniques to further enhance performance.

Keywords

multi-objective reinforcement learning, distributional reinforcement learning

Contents

I Introduction

Reinforcement learning (RL) studies how an agent can learn sequential decision-making through interactions with an environment. By receiving rewards as feedback, the agent gradually improves its behavior to maximize long-term return. This paradigm provides a general framework for problems such as robotics, scheduling, and game playing [31, 45, 46].

Classical RL assumes that the quality of a decision can be captured by a single scalar reward. Many real-world problems do not fit this assumption: a robot may need to trade off speed against energy consumption, and an autonomous vehicle may balance travel time and passenger comfort. Multi-objective reinforcement learning (MORL) addresses this by replacing scalar rewards with vector-valued rewards and by seeking policies that realize different trade-offs between conflicting objectives [18, 40].

Although MORL extends the classical RL framework, recent progress in the two areas has emphasized different directions. In single-objective RL, expressive neural architectures and distributional value representations have emerged as ways to strengthen existing algorithms without changing their core learning principles [26, 34, 36]. In MORL, by contrast, most work has focused on new algorithms, while the underlying function approximators have remained comparatively simple [3, 7, 10]. This leaves an important open question: how much of MORL performance is limited by the algorithms themselves, and how much by the function approximators used to represent policies and value functions?

This thesis studies whether recent advances in deep RL function approximators can be transferred to MORL and which components matter in this setting. To this end, we combine CAPQL [28], an entropy-regularized MORL baseline, with the SimbaV2 architecture [26] to obtain a new algorithm, Momba. This requires adapting several design ideas to preference-conditioned vector returns. Notably, we propose two ways of incorporating distributional value representation into MORL: one based on scalarized returns and one based on the marginals of the vector return. We evaluate these designs in continuous-control benchmarks and study the contribution of individual components, identifying the distributional critic as the most important part. The thesis therefore contributes both a new algorithm and an empirical study of which architectural ideas are useful in this setting.

The remainder of this thesis is organized as follows. Chapter 2 introduces the necessary background on RL, MORL, and modern function approximators. Chapter 3 presents CAPQL, our distributional critic adaptations, and the full Momba algorithm. Chapter 4 presents the empirical experiments, and Chapter 5 concludes with a discussion of the main findings and future research directions.

2 Background

This chapter introduces the foundational concepts used throughout the thesis. Section 2.2 introduces Markov decision processes (MDPs) and reinforcement learning (RL). Section 2.3 extends RL to the multi-objective setting. Finally, Section 2.4 discusses why neural architecture design is particularly important in deep reinforcement learning.

2.1 Notation

We denote the set of probability distributions over a set of outcomes X by $\mathscr{D}(X)$. We use bold symbols ($\boldsymbol{R} \in \mathbb{R}^d$, $\boldsymbol{Q} : S \times A \to \mathbb{R}^d$) for vector variables and vector-valued functions, distinguishing them from scalar counterparts with the same symbol when necessary. We denote the differential entropy of a continuous probability distribution p by $H(p)$. For categorical distributions q, r, we denote the cross-entropy of r relative to q by $H(q, r)$.

2.2 Reinforcement Learning

Reinforcement learning is an area of machine learning focused on sequential decision-making. The goal is to train an agent to maximize the rewards it obtains while interacting with an environment. Formally, we model the environment as a Markov decision process [39, 45].

Definition 1 (Markov Decision Process)**.** A Markov decision process is a tuple $M = (S, A, P, R, s_0)$, where S is the state space, A is the action space, $P : S \times A \to \mathscr{D}(S)$ is the transition kernel, $R : S \times A \to \mathbb{R}$ is the reward function, and $s_0 \in S$ is the initial state.

Following the standard definitions of [45], decision-making in an MDP is formalized by a policy $\pi : S \to \mathscr{D}(A)$ from the set of all policies Π. We assume stochastic policies that assign distributions over actions to states. Together with the transition function, a policy induces a probability measure over trajectories $S_0, A_0, R_0, S_1, A_1, R_1 \ldots$, where $A_t \sim \pi(S_t)$, $S_{t+1} \sim P(\cdot \mid S_t, A_t)$, and $R_t = R(S_t, A_t)$. Given a discount factor $\gamma \in [0, 1)$, the discounted return G is defined as the random variable

$$G = \sum_{t=0}^{\infty} \gamma^t R_t. \tag{2.1}$$

The value function V_π of policy π is defined as its expected return when starting in state s,

$$V_\pi(s) = \mathbb{E}_\pi\big[G \mid S_0 = s\big]. \tag{2.2}$$

Similarly, the state-action value function Q_π is defined as the expected return when starting in state s and taking action a,

$$Q_\pi(s,a) = \mathbb{E}_\pi\big[G \mid S_0 = s, A_0 = a\big]. \tag{2.3}$$

The goal is to find a policy π^* that maximizes the value in the initial state,

$$\pi^* = \arg\max_{\pi \in \Pi} V_\pi(s_0).$$

When the transition and reward functions are known, the problem can be solved with dynamic programming [9] or by linear programming formulations [30]. In reinforcement learning, the environment model is usually unknown and the policy must be learned from sampled interactions. We now present an example of such an algorithm — the actor-critic.

2.2.1 Actor-Critic Algorithm

The actor-critic algorithm [25] maintains two components. The *actor* is a parametric policy $\pi_\theta(a \mid s)$ and decides which actions to take. The *critic* is a parametric value estimator, often a state-action value function $Q_\phi(s,a)$, that evaluates actions under the current policy.

We present an off-policy actor-critic variant representative of modern algorithms [15, 16]. During training, the agent collects transitions $(s,a,r,s') \sim (S_t, A_t, R_t, S_{t+1})$ from the environment and stores them in a *replay buffer* $\mathcal{D}$. The critic Q_ϕ is trained to approximate the true value function Q_{π_θ} by minimizing the mean squared error

$$L(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[\big(\hat{Q} - Q_\phi(s,a)\big)^2\Big] \tag{2.4}$$

where $\hat{Q}$ is an estimate of the true value $Q_{\pi_\theta}(s,a)$, for example the temporal-difference estimate

$$\hat{Q} = r + \gamma Q_\phi(s', a'), \qquad a' \sim \pi_\theta. \tag{2.5}$$

The policy is trained to maximize the values estimated by the critic, i.e., to maximize the objective

$$J(\theta) = \mathbb{E}_{s\sim\mathcal{D},\, a\sim\pi_\theta(\cdot\,\mid s)}\Big[Q_\phi(s,a)\Big]. \tag{2.6}$$

Practicalities

The actor-critic algorithm, as presented so far, omits several mechanisms needed in practice. The Q-network can suffer from unstable training dynamics, and the agent still needs to balance optimizing known high-value actions with exploring new behavior. Modern actor-critic algorithms therefore add several stabilizing mechanisms.

Target Networks Estimating the target values for the Q-network Q_ϕ from the network itself may amplify function-approximation errors and destabilize training. Mnih et al. proposed computing the target Q-value estimates using a separate target network [31]. This is achieved by storing a lagged version of the Q-network parameters ϕ as the target network parameters $\overline{\phi}$. During training, the target parameters $\overline{\phi}$ are periodically or gradually updated toward the current Q-network parameters.

Clipped Double Q-learning Fujimoto et al. observed that Q-values tend to overestimate the policy's actual returns, leading to misinformed optimization and unstable training. They proposed to address the overestimation by training two independent Q-value estimators and taking their minimum as the target prediction [15]. Their empirical results showed that the pessimistic estimate stabilizes training, and the technique has since been adopted by other algorithms [16].

Entropy Regularization Even with the Q-network optimization stabilized, the agent still needs to address the exploration-exploitation trade-off. Haarnoja et al. addressed this by including the policy entropy in the objective of the Soft Actor-Critic (SAC) algorithm [16]. Rewarding randomness in action selection improves exploration while still allowing the policy to concentrate on high-value actions. The entropy is scaled by a factor $\alpha > 0$, and the goal is to find a maximizing policy

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t (R_t + \alpha H(\pi(\cdot \mid S_t))) \right]. \tag{2.7}$$

The algorithm achieves this by adding the negative log-likelihood of the action, which is a sample estimate of the entropy, to the Q-values and thus transforming the Q-value target into

$$\hat{Q} = r + \gamma \big(Q_\phi(s', a') - \alpha \log \pi_\theta(a' \mid s') \big), \qquad a' \sim \pi_\theta(\cdot \mid s'). \tag{2.8}$$

The actor then maximizes both the Q-values and the sample entropy in the objective

$$J(\theta) = \mathbb{E}_{s \sim \mathcal{D},\, a \sim \pi_\theta(\cdot \mid s)} \big[Q_\phi(s, a) - \alpha \log \pi_\theta(a \mid s) \big]. \tag{2.9}$$

2.3 Multi-Objective Reinforcement Learning

Real-world decision-making often involves optimizing multiple, possibly conflicting, objectives simultaneously. For example, a robot may need to trade off speed against energy consumption. To capture this formally, we use multi-objective Markov decision processes (MOMDPs) [11], which extend the MDP definition by allowing a vector-valued reward function $\boldsymbol{R} : S \times A \to \mathbb{R}^d$.

With vector-valued rewards, the value of a policy, $\boldsymbol{V}_\pi := \boldsymbol{V}_\pi(s_0) \in \mathbb{R}^d$, is no longer a scalar. Consequently, policies are no longer totally ordered by value. Instead of searching for a single optimal policy as in classical RL, we search for a set of policies whose values form the Pareto front.

Definition 2 (Pareto front)**.** Let $V = \{\boldsymbol{V}_\pi \mid \pi \in \Pi\}$ denote the set of all achievable value vectors. The Pareto front is the set $P(V) = \{\boldsymbol{v} \in V \mid \forall \boldsymbol{v}' \in V : \boldsymbol{v}' \geq \boldsymbol{v} \implies \boldsymbol{v}' = \boldsymbol{v}\}$ of maximal value vectors.

We say that a policy π is Pareto-optimal if and only if $\boldsymbol{V}_\pi \in P(V)$. To search for these policies, we adopt the utility-based approach of [18]. Given a utility (or scalarizing) function $u : \mathbb{R}^d \to \mathbb{R}$, we can compare policies through the scalarized value $u(\boldsymbol{V}_\pi)$. In this work, we focus on linear utility functions, which compute weighted sums of reward components.

Definition 3 (Preferences)**.** Let $W = \{w \in \mathbb{R}^d \mid w \geq 0 \wedge \sum_{i=1}^d w_i = 1\}$ denote the space of preferences. Each preference $w \in W$ gives rise to a scalarization function $u_w(\boldsymbol{v}) = w^\top \boldsymbol{v}$.

We approximate the Pareto front by searching for optimal policies under different scalarizations.

Definition 4 (Utility-Based Objective)**.** The goal is for each preference $w \in W$ to learn a policy π_w that is optimal under the scalarization function u_w, i.e. $w^\top \boldsymbol{V}_{\pi_w} \geq w^\top \boldsymbol{V}_{\pi'}$ for all policies $\pi' \in \Pi$.

In general, linear scalarization can recover only supported points and may miss non-convex parts of the Pareto front. Lu et al. [28] showed that, in our stochastic-policy setting, entropy-regularized rewards make linear scalarization sufficient for recovering the entire Pareto front. In practice, the assumptions behind this theoretical result are only approximately met because the policy is represented by a finite neural network and trained with imperfect optimization under a finite training budget. Consequently, front quality remains an empirical question even when the scalarized objective is theoretically well motivated.

While current research also studies non-linear utility functions [10], linear scalarization is still attractive because it turns a MOMDP into a scalar-reward MDP with reward $R_w(s, a) = w^\top \boldsymbol{R}(s, a)$, allowing the problem to be solved with an arbitrary single-objective RL algorithm.

2.3.1 Algorithms

With linear scalarization, training a policy for one fixed preference resembles single-objective RL. The harder question is how to approximate the full Pareto front. We divide existing MORL algorithms into three main categories.

Single-policy methods A direct approach is to choose a finite set of diverse preferences and train one policy per preference. This approach is simple and parallelizable, and it is appropriate when only a few preferences are known to be relevant a priori. Its drawback is sample inefficiency: each policy learns independently, leading to each policy rediscovering the problem structure on its own [40].

Multi-policy methods Multi-policy methods maintain a population of policies and share information across them. For example, PGMORL [48] uses an evolutionary approach to grow the population of policies into different parts of the Pareto front. Additionally, it trains a predictive model to decide which policies should be trained under which preferences [48]. Such methods can be robust and can explicitly manage coverage of the front, but dense coverage may require many policies.

General-policy methods General-policy methods learn a single preference-conditioned policy $\pi(a \mid s, w)$ [1, 49]. This representation can interpolate continuously between preferences and can share data and parameters across the front. The difficulty is that one neural network must represent a broad family of behaviors, which can be harder than fitting a single fixed-preference policy. This thesis focuses on general-policy methods, as they may benefit most from improvements in function approximation.

2.4 Function Approximators for Reinforcement Learning

Deep reinforcement learning algorithms commonly use neural networks to represent policies and value functions. In supervised learning, scaling network size has helped solve complex problems in areas such as computer vision and natural language processing [20, 23]. In RL, larger networks can help represent complex policies and value functions, but they can also make learning less stable [4]. The data distribution depends on the policy, the targets are bootstrapped from learned networks, and the critic is optimized on data generated by previous policies. As a result, architecture and optimization details can have a large effect on learning stability and final performance.

Following recent work, we view the relevant difficulties through three connected lenses: overfitting and generalization, non-stationarity and plasticity, and well-behaved optimization. Many techniques have been proposed to tackle these challenges, and together, they can improve training stability and make parameter scaling in reinforcement learning more practical [26, 33, 36].

2.4.1 Overfitting and Generalization

Critics are trained on finite replay-buffer samples. If the critic overfits these samples, the actor may exploit inaccurate high-value predictions rather than improving the true return. Several studies have also connected high critic error on a validation dataset to low policy performance [27, 33].

Different algorithms address this issue through regularization or ensembling. Randomized Ensembled Double Q-learning (REDQ) uses a large ensemble of critics and random subsets of the ensemble in target construction, enabling many critic updates per environment step [12]. Dropout Q-functions (DroQ) combine dropout [44] and layer normalization [6] to obtain similar benefits with lower computational cost [21]. Bigger, Regularized, Optimistic (BRO) uses larger networks together with normalization, weight decay, and residual connections [19] to improve scaling in a SAC-based algorithm [34].

2.4.2 Non-Stationarity and Plasticity

In contrast to supervised learning, the data distribution in RL changes as the agent improves. Early data may be generated by a poor exploratory policy, while later data may concentrate on a narrow part of the state-action space. If a network adapts too strongly to early data, it can lose plasticity, meaning that subsequent learning becomes slow or ineffective. Nikishin et al. describe this phenomenon as primacy bias [35].

Several mechanisms attempt to preserve plasticity. Partial or full network resets can improve final performance, although they may temporarily degrade training [34, 35]. Another approach is to constrain the growth of weight norms by periodically normalizing them, which has been shown to preserve the effective learning rate [29]. SimbaV2 and XQC, for example, combine weight and feature normalization to stabilize learning with large networks [26, 36].

2.4.3 Well-Behaved Optimization

Distributional RL replaces direct prediction of the expected return with prediction of a return distribution. The C51 algorithm approximates the distribution as a categorical distribution over a fixed support and optimizes the cross-entropy loss [8]. Although the policy still optimizes for the expected return, the empirical evaluation demonstrated improvements over its non-distributional counterpart. The improvements can be attributed to the distributional representation providing a richer learning signal, and the cross-entropy loss providing a better-conditioned loss landscape and bounded gradients compared to the usual mean-squared error loss.

Recent high-performing continuous-control architectures combine distributional critics with different forms of feature normalization to bound activations in the network. SimbaV2 uses hyperspherical feature normalization, weight normalization, and a distributional critic to enable stable parameter scaling [26]. Finally, Palenicek et al. showed that a combination of batch normalization [22], weight normalization, and categorical value representation smooths the loss landscape and improves the agent's performance [36].

3 Momba

Section 2.4 discussed function-approximation improvements that have been highly effective in single-objective RL. However, their effect in multi-objective RL remains underexplored. DroQ networks [21] have been used in the GPI-LS algorithm [3], but GPI-LS also introduces additional algorithmic changes and does not isolate the effect of the architecture itself. In our literature review, we did not find other recent architectures integrated into MORL algorithms.

In this chapter, we address this gap by introducing Momba. Momba integrates the RL architecture SimbaV2 [26], including distributional value representation, with the MORL algorithm CAPQL [28]. The goal is to preserve the algorithmic structure of CAPQL as much as possible while replacing its simple function approximators with more expressive alternatives.

We begin by describing CAPQL in Section 3.1. We then propose two adaptations of the distributional critic for multi-objective RL in Section 3.2. Next, we describe the SimbaV2 architecture in Section 3.3. Then, we assemble these ingredients into the full Momba algorithm in Section 3.4. Finally, we discuss how Momba can be extended with additional algorithmic components in Section 3.5.

3.1 Concave-Augmented Pareto Q-Learning

We begin with the *Concave-augmented Pareto Q-learning* (CAPQL) algorithm [28], which serves as the basis for our work. The algorithm follows the actor-critic scheme with entropy regularization described in Section 2.2.1, with both actor and critic conditioned on the preference.

Lu et al. motivate the algorithm by showing that regularizing the rewards by a concave function of the selected action probabilities makes the Pareto front strictly convex. Under their assumptions, this allows linear scalarization to recover policies even in regions that would otherwise be difficult for linear utility functions. By choosing policy entropy as this concave term, CAPQL becomes equivalent to Soft Actor-Critic [16] for a fixed preference.

We now describe the algorithm in detail. Consider a stochastic policy $\pi(a \mid s, w)$ conditioned on preference w, and let

$$\boldsymbol{G}(s, a, w) = \sum_{t=0}^{\infty} \gamma^t \boldsymbol{R}_t + \alpha \sum_{t=1}^{\infty} \gamma^t \mathbf{1}_d H(\pi(\cdot \mid S_t, w)) \tag{3.1}$$

denote the entropy-augmented discounted return vector of policy π after starting in state s and selecting action a. CAPQL trains a parametric vector-valued Q-network $\boldsymbol{Q}_\phi(s, a, w)$ and a policy $\pi_\theta(a \mid s, w)$. The Q-network is trained by minimizing the mean squared error (MSE) loss on the replay buffer $\mathcal{D}$

$$L_Q(\phi) = \mathbb{E}_{(s,a,\boldsymbol{r},s',w)\sim\mathcal{D}} \|\hat{\boldsymbol{Q}} - \boldsymbol{Q}_\phi(s, a, w)\|_2^2 \tag{3.2}$$

where $\hat{\boldsymbol{Q}}$ denotes the target Q-value estimate, computed as

$$\hat{\boldsymbol{Q}} = \boldsymbol{r} + \gamma\Big(\boldsymbol{Q}_\phi(s', a', w) - \alpha \mathbf{1} \log \pi_\theta(a' \mid s', w)\Big), \qquad a' \sim \pi_\theta(\cdot \mid s', w). \tag{3.3}$$

Intuitively, the Q-value approximates the expected vector return $\mathbb{E}[\boldsymbol{G}(s, a, w)]$. For each state s and preference w, the policy is trained to maximize the scalarized Q-value and entropy bonus:

$$J(\theta) = \mathbb{E}_{s,w\sim\mathcal{D},\, a\sim\pi_\theta(\cdot \mid s,w)}\Big[w^\top \boldsymbol{Q}_\phi(s, a, w) - \alpha \log \pi_\theta(a \mid s, w)\Big]. \tag{3.4}$$

3.2 Multi-Objective Distributional Critic

We begin by recalling the single-objective C51 algorithm [8]. Instead of approximating the expected return $\mathbb{E}_\pi[G]$ of a policy with a state-action value function $Q(s, a)$, the algorithm learns the return distribution G using a distributional state-action function $Z(s, a)$. The distribution is represented as a categorical distribution over a discrete set of N atoms evenly spaced on the interval $[V_{\min}, V_{\max}]$, forming the support

$$\operatorname{supp}(Z) = \{z_i \coloneqq V_{\min} + i\Delta z \mid 0 \le i < N\}, \qquad \Delta z = \frac{V_{\max} - V_{\min}}{N - 1}. \tag{3.5}$$

The critic Z_ϕ outputs a categorical distribution over these atoms, and we denote the probability of atom z_i by $Z_\phi(s, a)[i]$. The expected return, i.e., the scalar Q-value, is computed as

$$Q_\phi(s, a) = \mathbb{E}\Big[Z_\phi(s, a)\Big] = \sum_{i=0}^{N-1} Z_\phi(s, a)[i] z_i. \tag{3.6}$$

Given a transition (s, a, r, s') and a next action $a' \sim \pi(\cdot \mid s')$, the distributional temporal difference target $r + \gamma Z_\phi(s', a')$ assigns probability $Z_\phi(s', a')[i]$ to atom $r + \gamma z_i$. This changes the support, so we need to project the target distribution back onto $\operatorname{supp}(Z)$. For any categorical distribution Y with support $\operatorname{supp}(Y) = \{y_i \mid 0 \le i < N\}$, we define the projection operator Φ as

$$(\Phi Y)[i] = \sum_{j=0}^{N-1} Y[j] \max\left(1 - \frac{\left|\operatorname{clip}(y_j, V_{\min}, V_{\max}) - z_i\right|}{\Delta z}, 0\right). \tag{3.7}$$

Intuitively, for atoms $z_i \leq y_j \leq z_{i+1}$ with $z_i, z_{i+1} \in \operatorname{supp}(Z)$ and $y_j \in \operatorname{supp}(Y)$, the projection operator distributes the probability mass of atom y_j to z_i and z_{i+1} proportionally to their distance from y_j. If the atom y_j does not lie between two atoms from $\operatorname{supp}(Z)$, then the closest atom $z_i \in \operatorname{supp}(Z)$ receives the entire probability mass.

Finally, we can write the critic loss as the categorical cross-entropy loss

$$L(\phi) = H\big(\hat{Z}, Z_\phi(s,a)\big) = -\sum_{i=0}^{N-1} \hat{Z}[i] \log Z_\phi(s,a)[i], \tag{3.8}$$

where the target distribution is computed by projecting the temporal difference estimate

$$\hat{Z} = \Phi(r + \gamma Z_\phi(s', a')), \qquad a' \sim \pi_\vartheta(\cdot \mid s'). \tag{3.9}$$

To adapt the distributional critic to multi-objective RL, we first address how to represent the return distribution. Choosing atoms in a multi-dimensional grid would be intractable. In the single-objective case, C51 used 51 atoms, and performance decreased with fewer atoms due to inaccurate approximation of the return distribution [8]. Matching similar per-dimension resolution in the multi-objective case would require more than 2500 atoms for two objectives and more than 125000 atoms for three objectives. This would increase runtime and could make optimization harder, making a fixed-atom representation of the full multivariate distribution impractical.

For CAPQL, however, the critic predictions affect the policy update only through the scalarized Q-value $w^\top \boldsymbol{Q}(s,a)$. We therefore only need an accurate estimate of the expected scalarized return $\mathbb{E}\big[w^\top \boldsymbol{G}\big]$, rather than the full multivariate return distribution. We propose two ways of avoiding the full multivariate distribution: (i) learning the scalarized return distribution, or (ii) learning the marginals of the return distribution. Figure 3.1 visualizes the two approaches.

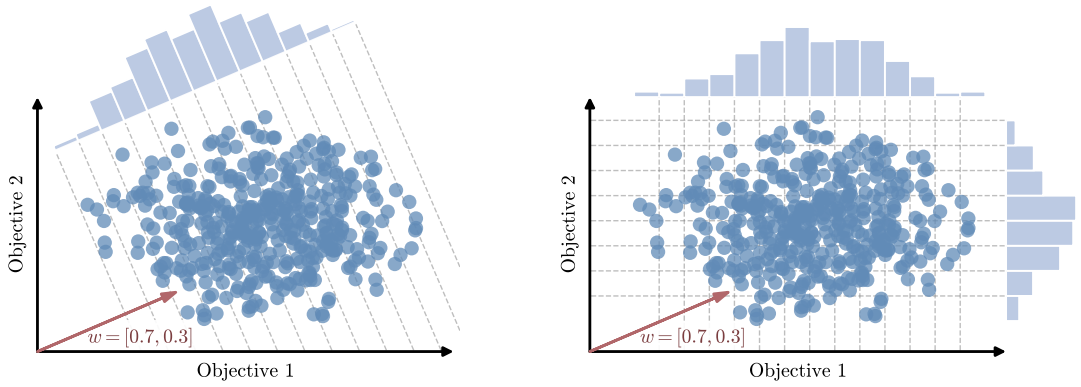

Figure 3.1: **Distributional returns** learned by the scalarized critic (left) and the vector critic (right). The point cloud represents the return distribution, which depends on the preference.

The scalarized critic is conceptually simpler and easier to implement. Its limitation is that it does not directly produce the vector Q-values, which are required by some MORL algorithms, e.g. [3, 7]. The vector critic provides the vector values while still scaling linearly with the number of objectives.

3.2.1 Scalarized Critic

The scalarized critic approximates the univariate distribution of the scalarized return $Z(s, a, w) \approx w^\top \boldsymbol{G}(s, a, w)$. It is trained using the loss

$$L_Z(\phi) = H(\hat{Z}, Z_\phi(s, a, w)) \tag{3.10}$$

where $\hat{Z}$ is computed from the scalarized reward $w^T \boldsymbol{r}$ as

$$\hat{Z} = \Phi(w^\top \boldsymbol{r} + \gamma Z_\phi(s', a', w)), \qquad a' \sim \pi_\theta(\cdot \mid s', w). \tag{3.11}$$

The scalarized Q-value used by the actor is obtained as the expectation

$$w^\top \boldsymbol{Q}(s, a, w) \approx \mathbb{E}\Big[Z_\phi(s, a, w)\Big]. \tag{3.12}$$

3.2.2 Vector Critic

Another option is to approximate the marginals of the return distribution. We train d distributional critics $Z_\phi^1, \dots, Z_\phi^d$, one for each component of the d-dimensional return. The critic loss is the sum of the individual losses:

$$L_Z(\phi) = \sum_{i=1}^{d} H(\hat{Z}_i, Z_\phi^i(s, a, w)) \tag{3.13}$$

with each target $\hat{Z}_i$ computed as

$$\hat{Z}_i = \Phi(\boldsymbol{r}_i + \gamma Z_\phi^i(s', a', w)), \qquad a' \sim \pi_\theta(\cdot \mid s', w). \tag{3.14}$$

The expected values of the critics recover the vector Q-value

$$\boldsymbol{Q}(s, a, w) = \Big(\mathbb{E}\Big[Z_\phi^1(s, a, w)\Big], \dots, \mathbb{E}\Big[Z_\phi^d(s, a, w)\Big]\Big), \tag{3.15}$$

and the scalarized Q-value is computed as

$$w^\top \boldsymbol{Q}(s, a, w) = \sum_{i=1}^{d} w_i \mathbb{E}\Big[Z_\phi^i(s, a, w)\Big]. \tag{3.16}$$

3.3 SimbaV2 Architecture

We now describe the SimbaV2 architecture [26] in detail. SimbaV2 is designed to make parameter scaling more stable by controlling the learning dynamics. It does so by constraining the growth of weight and feature norms with hyperspherical (ℓ_2) normalization, weight normalization, and distributional value representation. Figure 3.2 shows the architecture, and we describe its components in the following paragraphs.

Input embedding The input state is first normalized using the running mean and standard deviation collected throughout training.

$$\tilde{x} = \frac{x - \mu_x}{\sigma_x}$$

A constant c_{shift} is then appended to the normalized state to preserve information about the magnitude of the original state after the ℓ_2 normalization layer. The resulting vector is passed through a linear layer, a component-wise scaling layer, and another normalization layer to produce the state embedding x_{emb}.

$$x_{\text{emb}} = \ell_2\text{-norm}(s_e \odot W_e \ell_2\text{-norm}(\tilde{x} \oplus c_{\text{shift}}))$$

Hidden blocks Each hidden block first applies linear layers and scalers to the current embedding, followed by a ReLU activation and normalization. Let x denote the input to the block, and let the feed-forward output be x_{FFN}.

$$x_{\text{FFN}} = \ell_2\text{-norm}(W_2 \,\text{ReLU}(s_h \odot W_1 x))$$

The LERP layer, with learnable parameter η, then interpolates between its input and x_{FFN}, acting as a learnable residual connection:

$$x_b = \ell_2\text{-norm}(\eta x + (1 - \eta) x_{\text{FFN}}).$$

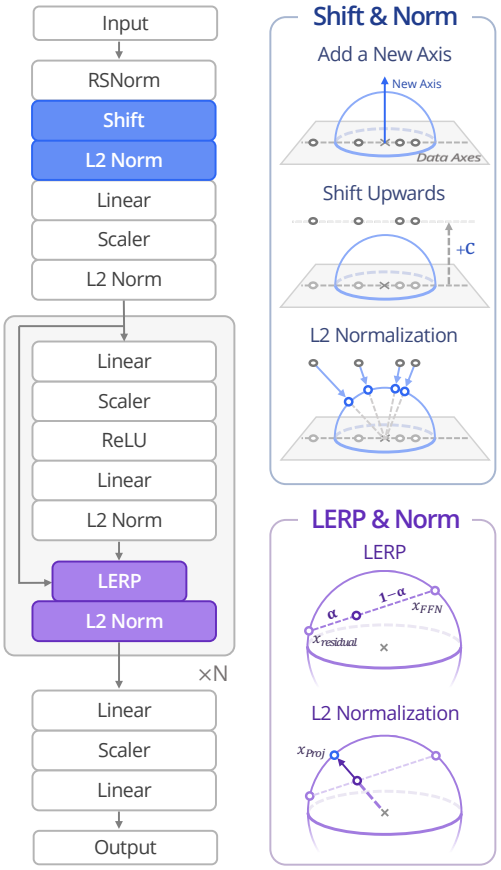

Figure 3.2: **SimbaV2 Architecture**. The diagram is taken from [26].

Output prediction The final hidden embeddings are passed through a linear layer, a scaler, and a final linear layer. In the actor, this produces the parameters of a normal distribution for sampling the action; in the critic, it produces logits for the distributional critic atoms.

Weight normalization Each linear layer multiplies its input by a matrix W without adding a bias term. After every gradient update, we renormalize the rows of the matrix to maintain unit norm:

$$\tilde{W}_{ij} = \frac{W_{ij}}{\|W_{i\cdot}\|_2}.$$

Normalizing the rows helps keep the output activations bounded. Only the linear layers are normalized, while the other learnable parameters (such as scalers) remain unconstrained to preserve the network's expressive power.

3.4 Momba Algorithm

Momba integrates the preference-conditioned SimbaV2 architecture and the multi-objective distributional critic into CAPQL. This section describes the design decisions behind our algorithm, how the components are combined, and the remaining technical details.

We begin by motivating the choice of the main components. Our goal was to address the function-approximation challenges presented in Section 2.4 using an expressive architecture while keeping the algorithmic baseline deliberately lightweight. SimbaV2 systematically addresses these challenges and has demonstrated strong performance in continuous-control tasks. In addition, the authors provide a reference implementation, which helps faithfully adapt its components. As the base algorithm, we selected CAPQL because of its conceptual simplicity. It is based on linear scalarization and closely resembles Soft Actor-Critic, which is also the base algorithm for SimbaV2.

Nevertheless, we do not view this particular pairing as essential: similar results may be obtainable with different architectures, such as XQC [36], or different base algorithms, such as TD3 [15].

3.4.1 Design Decisions

We describe several algorithm-level design choices that set Momba apart from the original CAPQL and SimbaV2 implementations.

Preference Selection We sample the preference w uniformly from the preference simplex at the beginning of every episode. This differs from CAPQL, which samples a new preference at each timestep, but is more in line with other MORL algorithms [1, 3, 48].

Automatic Entropy Coefficient Tuning The parameter α scales the policy's entropy to a reasonable level relative to the rewards. CAPQL uses a fixed value of the parameter throughout the training, which must be selected manually for each environment. However, Soft Actor-Critic also introduced a way to automatically tune this coefficient to achieve a target policy entropy [17]. Following the original SimbaV2 implementation, we adopt this approach and set the target entropy to the negative dimensionality of the action space, i.e. $H_{\text{target}} = -|A|$.

Reward Normalization To allow a fixed support for the distributional critic across all environments, we normalize rewards by dividing them by the maximum discounted return observed during training. Single-objective distributional algorithms usually handle this differently: XQC and SimbaV2 normalize based on the running standard deviation of the measured discounted returns [26, 36]. In our setting, preference conditioning introduces additional variance to the observed returns, so we normalize by the maximum return instead. In preliminary single-objective tests, we did not observe a noticeable difference between these normalization choices.

Distributional Critic We use the scalarized critic from Section 3.2.1. Momba, as presented in this section, does not require vector values, so we use the simpler scalarized critic by default. Nonetheless, our implementation also includes the vector critic as an option.

Preference Conditioning We condition the network by concatenating the preference with the state. This is a common technique in MORL algorithms [1, 3, 28]. While some algorithms explore more involved approaches, such as hypernetworks [43], simple concatenation performed best in our experiments. We follow up on this experimentally in Section 4.3.4.

3.4.2 Training

The training follows the actor-critic scheme from Section 2.2.1. We also follow the common practice of training two critics to implement Clipped Double Q-learning, with target networks to stabilize the training. We denote the target network parameters as $\overline{\phi}$.

We summarize the training loop in Algorithm 1 and describe the process below. Training proceeds in iterations in which interactions with the environment are followed by parameter updates. First, the actor is given the current state s and selects an action a according to the current policy and preference. The environment responds with vector reward $\boldsymbol{r}$, next state s', and termination flag δ. We store this interaction together with the current preference w as a transition $(s, a, \boldsymbol{r}, s', \delta, w)$ in the replay buffer $\mathcal{D}$. The transition is then used to update the running statistics for the input and reward normalization.

After interaction, a minibatch is sampled from the replay buffer, and the actor and critics are updated. Below, we write the losses for a single transition; in practice, each loss is averaged over the batch.

Critic Loss

For each critic $i = 1, 2$, we first compute the temporal difference target estimate $\hat{Z}_i$ projected onto the common support,

$$\hat{Z}_i = \Phi\Big(w^\top \boldsymbol{r} + \gamma(1-\delta)\Big(Z_{\bar{\phi}_i}(s', a', w) - \alpha \log \pi_\theta(a' \mid s', w)\Big)\Big), \qquad a' \sim \pi_\theta(\cdot \mid s', w).$$

We select the projected target with the lower expected value for training[1]:

$$\hat{Z} = \hat{Z}_i, \qquad i = \arg\min_{i=1,2} \mathbb{E}\big[\hat{Z}_i\big]$$

Finally, both critics are trained toward the selected target using the loss

$$L_{\text{critic}}(\phi_1, \phi_2) = H(\hat{Z}, Z_{\phi_1}(s, a, w)) + H(\hat{Z}, Z_{\phi_2}(s, a, w)) \tag{3.17}$$

Actor Loss

The actor maximizes the scalarized Q-values and the sample entropy bonus by minimizing

$$L_{\text{actor}}(\theta) = -\min_{i=1,2} Q_{\phi_i}(s, a', w) + \alpha \log \pi_\theta(a' \mid s, w), \qquad a' \sim \pi_\theta(\cdot \mid s, w). \tag{3.18}$$

Alpha Loss

The coefficient α acts as a Lagrange multiplier for the constraint $\mathbb{E}_{\pi_\theta} H(\pi_\theta(\cdot \mid s, w)) \geq H_{\text{target}}$. We tune the coefficient by taking gradient steps on the loss

$$L_{\text{alpha}}(\alpha) = -\alpha(\log \pi_\theta(a \mid s, w) + H_{\text{target}}). \tag{3.19}$$

To maintain $\alpha > 0$, we reparametrize $\alpha = \exp(\beta)$ and optimize the parameter β instead.

3.4.3 Implementation Notes

We implemented Momba in PyTorch [37]. Our SimbaV2 implementation is derived from the original implementation in the JAX framework[2]. We use the Adam optimizer [24] to minimize the losses. The training loop is based on the LeanRL implementation of SAC[3]. To ensure reasonable runtime with the SimbaV2 architecture, we use just-in-time compilation. Our code unifies the implementation for distributional and non-distributional critic losses, vector and scalarized value predictions, and SimbaV2 and plain MLP architectures, providing a consistent platform for comparing design decisions.

[1] For the vector critic, this would be based on the scalarized expected value.
[2] https://github.com/DAVIAN-Robotics/SimbaV2
[3] https://github.com/meta-pytorch/LeanRL

Algorithm 1: Momba Training Loop

Input: Environment, interaction budget T, hyperparameters $\tau \approx 0.005$, initial $\alpha \approx 0.2$
Initialize replay buffer $\mathcal{D}$
Initialize critics Z_{ϕ_1}, Z_{ϕ_2}, target critics $Z_{\overline{\phi}_1}, Z_{\overline{\phi}_2}$, policy π_θ, and entropy coefficient α
Copy target parameters $\overline{\phi}_i \leftarrow \phi_i$ for $i \in \{1, 2\}$
Initialize normalization statistics $\mu_{\text{obs}}, \sigma_{\text{obs}}, \max_{\text{rew}}$
Receive initial state s_0; sample initial preference $w \sim \text{Dirichlet}(\mathbf{1}_d)$
for $t = 1, \ldots, T$ **do**
    Sample $a_t \sim \pi_\theta(\cdot \mid s_t, w)$
    Execute a_t, observe reward $\boldsymbol{r}_t$, next state s_{t+1}, and termination flag δ_t
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, \boldsymbol{r}_t, s_{t+1}, \delta_t, w)\}$
    Update normalization statistics using state s_t and reward $\boldsymbol{r}_t$
    if *Episode terminated* **then**
        Reset the environment and receive the next initial state s_{t+1}
        Sample preference $w \sim \text{Dirichlet}(\mathbf{1}_d)$
    foreach *gradient step* **do**
        Sample minibatch B from $\mathcal{D}$
        Normalize minibatch with running statistics $\mu_{\text{obs}}, \sigma_{\text{obs}}, \max_{\text{rew}}$
        Take gradient step on CriticLoss(B)
        Take gradient step on ActorLoss(B)
        Take gradient step on AlphaLoss(B)
        Normalize linear layers in ϕ_1, ϕ_2, and θ
        Update target parameters $\overline{\phi}_i \leftarrow (1 - \tau)\overline{\phi}_i + \tau\phi_i$ for $i \in \{1, 2\}$

3.5 Further Algorithmic Improvements

Momba can be combined with additional algorithmic ideas from existing MORL algorithms. Specifically, we consider a preference-relabeling variant of hindsight experience replay [42] and a PD-MORL-style preference alignment loss [7]. In this section, we consider Momba with the vector distributional critic, as the preference alignment loss requires access to the vector values.

3.5.1 Hindsight Experience Replay

Our algorithm saves the preference used to sample a transition to the replay buffer. This preference is then reused when updating the Q-networks and the policy. Since the environment dynamics and the vector reward in the transition do not depend on the preference, the transition can also be relabeled with another preference to provide additional training signal. This technique is commonly known as *hindsight experience replay* (HER) [5].

In the context of MORL, Alegre et al. [3] proposed relabeling preferences toward regions with high potential for improvement. In combination with prioritized preference selection, they showed im-

provements in sample efficiency. Shianifar et al. later studied preference relabeling independently of preference selection and reported improved Pareto front coverage with CAPQL [42].

We implement HER by modifying the preferences in the sampled minibatches. For each transition $(s, a, \boldsymbol{r}, s', \partial, w)$, we replace the preference w with a preference w' sampled as

$$w' \sim \begin{cases} w & \text{with probability } p_{\text{keep}}, \\ \text{Dirichlet}(kw) & \text{with probability } 1 - p_{\text{keep}}. \end{cases} \tag{3.20}$$

We use parameters $p_{\text{keep}} = 0.2$ (i.e., 80% of transitions are relabeled) and $k = 5$, which biases resampling toward preferences close to the original preference w.

3.5.2 Preference Alignment Regularization

Linear scalarization might struggle with recovering solutions in the flat parts of the Pareto front. Depending on the preference, the interior points of the flat regions are either suboptimal or share the same scalarized value, and thus the algorithm cannot control which solution it learns. PD-MORL [7] augments the scalarized Q-value with the cosine similarity between the Q-value and the preference. [4] This helps the value of the policy remain aligned with the direction of the preference, resulting in a denser coverage of the Pareto front.

We implement this technique by modifying the actor loss from Equation 3.18 by multiplying the scalarized Q-values by the cosine similarity term (highlighted in green):

$$L_{\text{actor}}^{\text{PAR}}(\theta) = -\min_{i=1,2}\Big(w^\top \boldsymbol{Q}_{\phi_i}(s, a', w) \cdot \cos\Big(w, \boldsymbol{Q}_{\phi_i}(s, a', w)\Big)\Big) + \alpha \log \pi_\theta(a' \mid s, w), \qquad a' \sim \pi_\theta(\cdot \mid s, w). \tag{3.21}$$

The critic loss stays unmodified, and the critic only performs policy evaluation.

We assume nonnegative rewards, as the cosine similarity term could otherwise steer the solutions in an undesired direction. PD-MORL also learns an interpolation between the preferences and the target directions for the Q-values; however, this was not necessary in our benchmarks. Nonetheless, we acknowledge that a robust implementation of this technique might require additional changes.

[4] The authors motivate this approach differently. They argue that objectives with large values relative to the others might dominate the scalarized objective. Nonetheless, the results demonstrate better Pareto front uniformity.

4 Evaluation

This chapter evaluates Momba, the algorithm introduced in Chapter 3. The evaluation has two goals: to compare Momba against established MORL baselines, and to assess which function-approximation choices matter in the preference-conditioned setting. Specifically, we address the following questions:

- Is Momba competitive against selected MORL baselines?
- What is the effect of the SimbaV2 architecture and the distributional value representation?
- Does Momba benefit from parameter scaling?
- Can Momba be improved with additional algorithmic changes?

The rest of the chapter is structured as follows. Section 4.1 describes the experimental setup and benchmarks. Section 4.2 compares Momba against selected baselines. Section 4.3 studies the algorithm's components through ablations. Section 4.4 evaluates the effects of hindsight experience replay and preference alignment regularization. Finally, Section 4.5 summarizes the main findings and discusses the results.

4.1 Experimental Setup

We describe the benchmarks, metrics, and baseline algorithms used throughout the evaluation.

4.1.1 Environments

We evaluate the algorithms on seven continuous-control tasks from PGMORL [48]. These tasks come from six environments implemented in the MuJoCo physics engine [47]: 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). We summarize the environments, their action and state spaces, and objectives in Table 4.1.

4.1.2 Metrics

To measure solution-set quality, we use two common metrics from MORL research: hypervolume (HV) and the expected utility metric (EUM).

Table 4.1: Summary of environments, state/action space dimensionality, and objectives.

Environment	|S|	|A|	Objectives
Ant-v4	27	8	X-axis speed, Y-axis speed
HalfCheetah-v4	17	6	Forward speed, energy efficiency
Hopper-v3	11	3	Forward speed, jumping height, efficiency
Hopper-v4	11	3	Forward speed, jumping height
Humanoid-v4	376	17	Forward speed, energy efficiency
Swimmer-v4	8	2	Forward speed, energy efficiency
Walker2d-v4	17	6	Forward speed, energy efficiency

- **Hypervolume** HV (⇑) [51] measures the volume dominated by the solutions in the set S:

$$\mathrm{HV}(S) = \int_{\mathbb{R}^d} \mathbb{1}_{D(S)}(z)\,dz \tag{4.1}$$

  where $D(S) = \{z \in \mathbb{R}^d \mid \exists \pi \in S : r_0 \leq z \leq \boldsymbol{V}_\pi\}$. Here $r_0 = \mathbf{0}_d$ is the reference point, and $\mathbb{1}_{D(S)}$ is the indicator function of the set $D(S)$.

- **Expected Utility Metric** EUM (⇑) [50] captures the expected utility of the solution set for a random user preference:

$$\mathrm{EUM}(S) = \mathbb{E}_{w \sim P_w}\left[\max_{\pi \in S} w^\top \boldsymbol{V}_\pi\right], \tag{4.2}$$

  where P_w is a distribution over the preferences. We use the uniform distribution.

Hypervolume captures changes in uniformity, spread, and convergence of the solution set, making it a useful indicator of overall quality [50]; we therefore use HV as our main metric. EUM, in contrast, closely matches the training objective because the algorithms optimize scalarized rewards. We report both HV and EUM when comparing against the baselines, while for ablations, we only report HV, since both metrics behave similarly on our benchmarks.

We do not use a separate metric for the uniformity of the solution sets. While sparsity [48] is often used in MORL literature, it is not suitable for comparing solution sets with different hypervolumes and coverage [18]. Moreover, general-policy methods represent infinitely many policies, which makes direct comparison to finite multi-policy sets difficult. Instead, we visualize the learned solution sets to illustrate the main strengths and shortcomings of the evaluated algorithms.

We approximate each solution set by averaging returns over five episodes for each evaluated preference. Following [14], we use 100 equally spaced preferences on the preference simplex for general-policy methods, while PGMORL and DPMORL are evaluated based on their current (policy, utility function) pairs. We then compute HV and EUM from the estimated value vectors.

When reporting results, we follow [2] and use the interquartile mean (IQM) and 95% stratified bootstrapped confidence intervals (SBCIs). Unless stated otherwise, we evaluate algorithms on 10 different seeds for statistical robustness. For aggregate metrics, we report the IQM after normalizing each metric by an appropriate baseline: PGMORL for the main comparison and Momba for the ablations.

4.1.3 Baselines

We compare against four baselines. Together, they cover both multi-policy and general-policy methods and represent several design choices in MORL. We list the algorithms, including the number of trainable parameters in Table 4.2, and describe them in the following paragraphs.

CAPQL

Concave-Augmented Pareto Q-Learning (CAPQL) [28] is a general-policy method used as the base for Momba. Compared with Momba, CAPQL uses smaller feedforward networks for the policy and the Q-network. It also uses a fixed entropy coefficient α. Unlike the other evaluated algorithms, it selects a new preference at every timestep. For evaluation, we use the implementation from MORL-baselines [14] and run the algorithm for 1 million steps.

PGMORL

Prediction-Guided Multi-Objective Reinforcement Learning (PGMORL) [48] is a PPO-based multi-policy method [41]. Throughout the training, it maintains a population of policies specialized for different preferences. The algorithm also trains a predictive model that estimates how the value of a policy will move when trained on a given preference. The predictive model is used to select preferences for the policy population with the goal of improving hypervolume and uniformity. The algorithm trains 6–15 policies, each trained for 2M–20M timesteps, depending on the environment.

DPMORL

Distributional Pareto-Optimal Multi-Objective Reinforcement Learning (DPMORL) [10] is a multi-policy method that first generates a diverse set of non-linear utility functions and then optimizes policies for those utilities. To optimize the non-linear utility functions, the algorithm learns the multivariate distribution of returns. It therefore serves as a useful comparison to our scalarized adaptation of the distributional critic. The algorithm trains 10 policies, each for 1 million steps.

GPI-LS

Generalized Policy Improvement with Linear Support (GPI-LS) [3] is a general-policy method based on the TD3 algorithm [15]. It uses General Policy Improvement (GPI) to identify promising prefer-

ences where performance improvement is likely. These selected preferences are used for environment interaction and for prioritizing transitions in the Q-function update. The preference selection, however, requires additional environment samples that are not counted toward the training budget. The algorithm also uses the DroQ architecture [21] for the Q-networks, enabling it to perform 20 updates per environment step and making it highly sample-efficient. Due to excessive runtime, we capped GPI-LS at 100 hours, limiting it to 200,000 timesteps.

Table 4.2: **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

4.2 Main Comparison

We begin by comparing Momba against existing MORL algorithms. For this comparison, Momba uses the scalarized distributional critic and does not include the additional algorithmic improvements from Section 3.5. For the baselines, we use the default hyperparameters from their respective implementations. For Momba, the hyperparameters are reported in Appendix B.

4.2.1 Final Performance

We first compare the final solution-set quality of Momba and the baselines. Figure 4.1 summarizes aggregate hypervolume and EUM over all environments.

Momba obtains the highest aggregate scores, with a 29% improvement in HV and a 13% improvement in EUM over the runner-up, PGMORL. Notably, PGMORL, as a multi-policy algorithm, used 10–100 times more timesteps, depending on the environment. Compared with CAPQL, the underlying algorithm, Momba obtains 130% and 40% higher aggregate HV and EUM, respectively. Table 4.3 reports the per-environment results. Momba achieves the best score in all tasks except Swimmer, where PGMORL performs slightly better.

This comparison should be read with one caveat: GPI-LS is capped at 2×10^5 timesteps, because its runtime exceeded 100 hours. For reference, CAPQL and Momba generally finished within two hours. We further compare Momba and GPI-LS under the same timestep budget in Section 4.2.3.

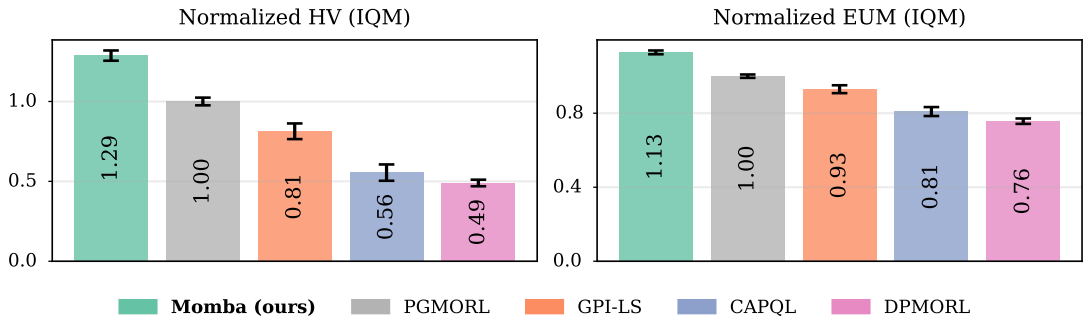

Figure 4.1: **Momba outperforms the selected baselines in aggregate across seven continuous-control tasks.** The plots show normalized hypervolume (HV) and expected utility (EUM) (IQM and 95% SBCIs over 10 seeds), aggregated over 7 tasks. Momba obtains the highest aggregate scores among the evaluated algorithms.

Table 4.3: **Per-environment evaluation results**. We report hypervolume and EUM (IQM with 95% SBCIs over 10 seeds), with the best result in each environment bolded.

Environment	Metric	Momba	PGMORL	GPI-LS	CAPQL	DPMORL
Ant-v4	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	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	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	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	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	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	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]

4.2.2 Pareto Front Quality

We next inspect the generated solution sets qualitatively in Figure 4.2. We select the solution set with the median hypervolume as the representative for each algorithm.

Momba tends to produce solution sets with broad coverage and often dominates the policies discovered by the baseline algorithms. In complex high-dimensional environments such as Humanoid and Walker2d, Momba learns high-forward-speed solutions that the other algorithms were unable to find in our runs.

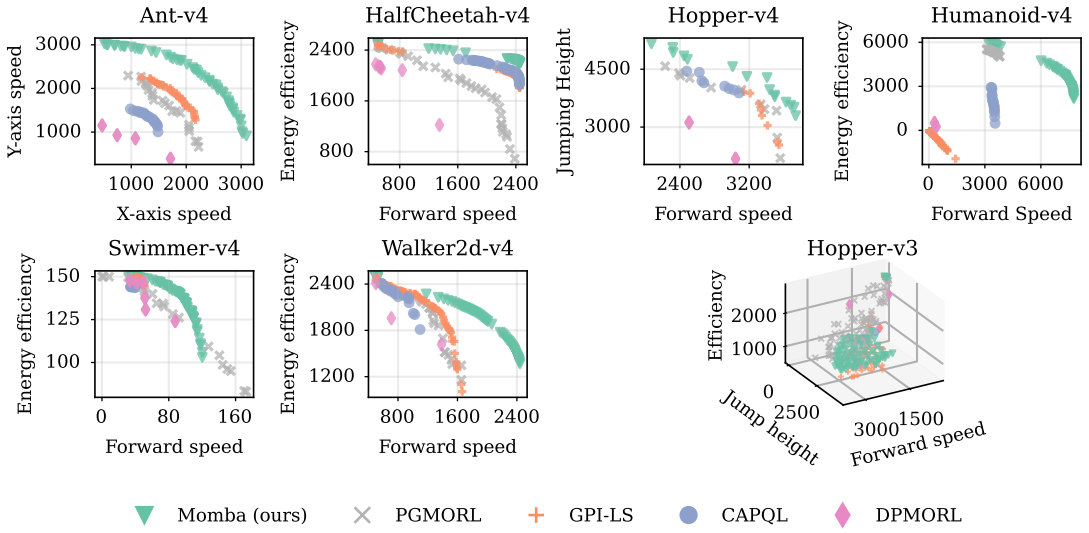

Figure 4.2: **Momba produces solution sets with broad coverage**. We visualize the generated solution sets in all seven environments. Momba produces strong solution sets, although some gaps remain in difficult regions of the fronts.

In some environments, however, Momba struggles to produce a dense solution set. For instance, we observe gaps in the upper-left corner of the front in the Humanoid and HalfCheetah environments. One possible cause is the use of linear scalarization. There is a similar effect for CAPQL in HalfCheetah, suggesting that entropy regularization alone is insufficient to reliably find solutions in this region. Another interpretation is that in those regions, the policy switches the agent's gait from "walking" to "running", making interpolation between the two behaviors difficult.

We do not explicitly optimize for the uniformity of the front, so these failure cases are not surprising. Later, in Section 4.4, we add preference alignment regularization and study its effect on the uniformity of the solution set.

4.2.3 Sample Efficiency

We next examine sample efficiency. Figure 4.3 displays the training curves of hypervolume and EUM for CAPQL and Momba. For PGMORL and DPMORL, we show only final performance because these multi-policy methods require substantially more environment steps.

Momba converges quickly to high-quality solution sets and outperforms CAPQL throughout the training. It reaches the PGMORL and DPMORL reference levels using fewer timesteps across all environments except Swimmer, where Momba is slightly behind PGMORL.

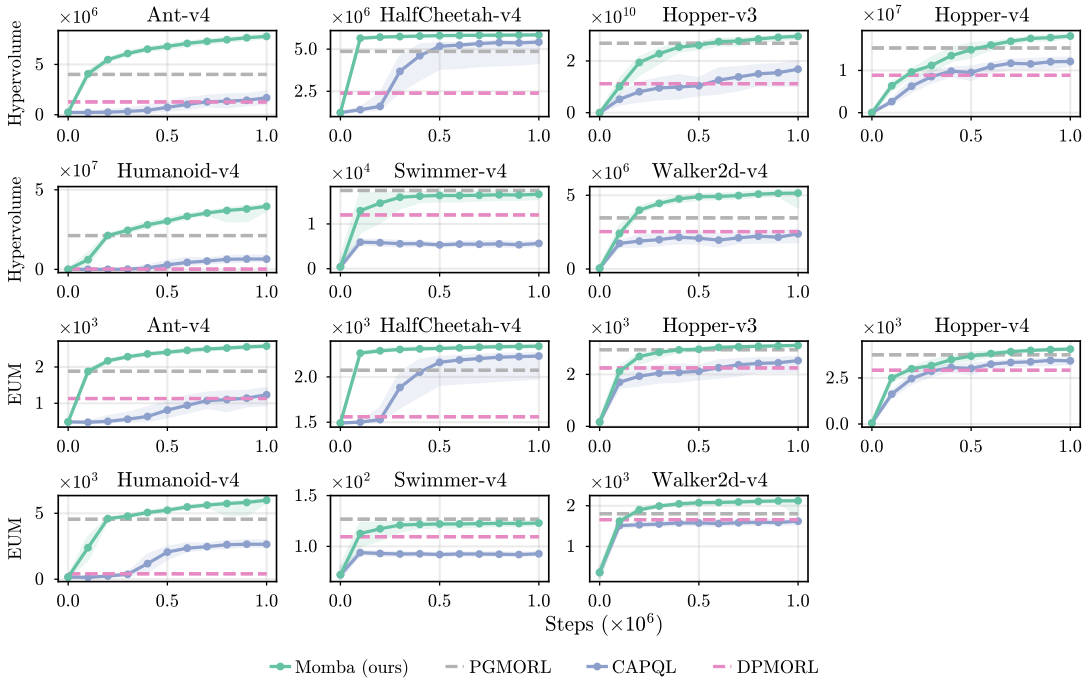

Figure 4.3: **Sample efficiency curves of Momba and CAPQL**. The figures show HV (top) and EUM (bottom) (IQM and 95% SBCIs over 10 seeds) during training. Horizontal lines indicate the final performance of PGMORL (at 1.2×10^7–1.2×10^8 timesteps) and DPMORL (at 1×10^7 timesteps).

We also compare Momba and GPI-LS under a limited timestep budget. GPI-LS uses a UTD ratio of 20, meaning it performs 20 updates to the policy and value networks before collecting a transition from the environment. To compare the methods at similar update budgets, we evaluated GPI-LS at UTD 1 and 20, and Momba at UTD 1 and 8.

Figure 4.4 displays the training curves for normalized HV and EUM aggregated over all environments. The results indicate that, when paired with a similar UTD ratio in our setup, Momba outperforms GPI-LS in terms of sample efficiency. This comparison suggests that the combination of

expressive architectures and distributional value representations can provide large sample-efficiency gains even without the preference-prioritization strategy used in GPI-LS.

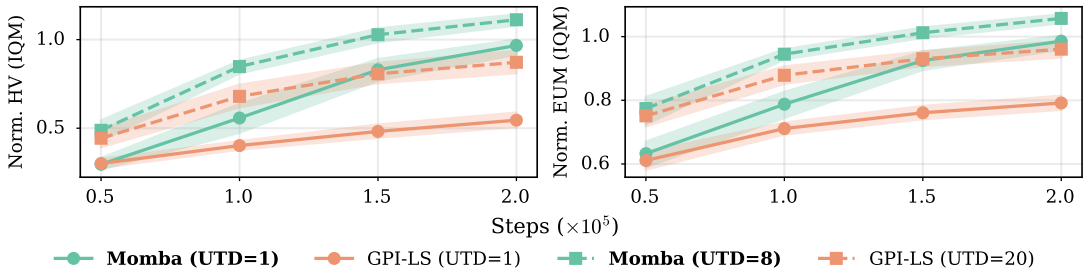

Figure 4.4: **Momba outperforms GPI-LS at similar UTD after 200k steps**. We compare the normalized HV and EUM (IQM and 95% SBCIs over 7 seeds) of Momba and GPI-LS with different UTD ratios during the first 200k steps.

4.2.4 Understanding the Performance

The previous experiments show that Momba performs strongly as a MORL algorithm, producing high-quality solution sets across preferences. For practical use, however, this should not come at the cost of substantially worse performance than a single-objective RL agent trained for a known preference. We therefore compare Momba with fixed-preference runs of the same algorithm to measure how much performance is lost by approximating the full front at once.

We construct this reference from 20 uniformly spaced preferences for the two-objective environments and 50 preferences for Hopper-v3, training Momba for one million timesteps for each selected preference. The resulting solution sets are shown in Figure 4.5.

The fixed-preference runs are slightly stronger, as expected, but the gap is modest. This suggests that the preference-conditioned policy retains much of the performance of specialized training while covering the front in a single run. This is notable because Momba obtains the full solution set from one run of one million timesteps, whereas the fixed-preference reference requires retraining for every selected preference. The fixed-preference reference front is also less uniform: because each preference is trained independently, seed variance and local optimization failures appear as irregular jumps along the front. Momba shares parameters across preferences, so these fluctuations are smoothed into a more coherent solution set.

We also visualize the discovered policies in the challenging Humanoid environment in Figure 4.6. The rollouts show that the agent learns a range of meaningful behaviors and adapts them to the selected preference. This illustrates the practical appeal of Momba in settings where the desired trade-off is not known in advance and the user wants to adjust the agent's behavior after training.

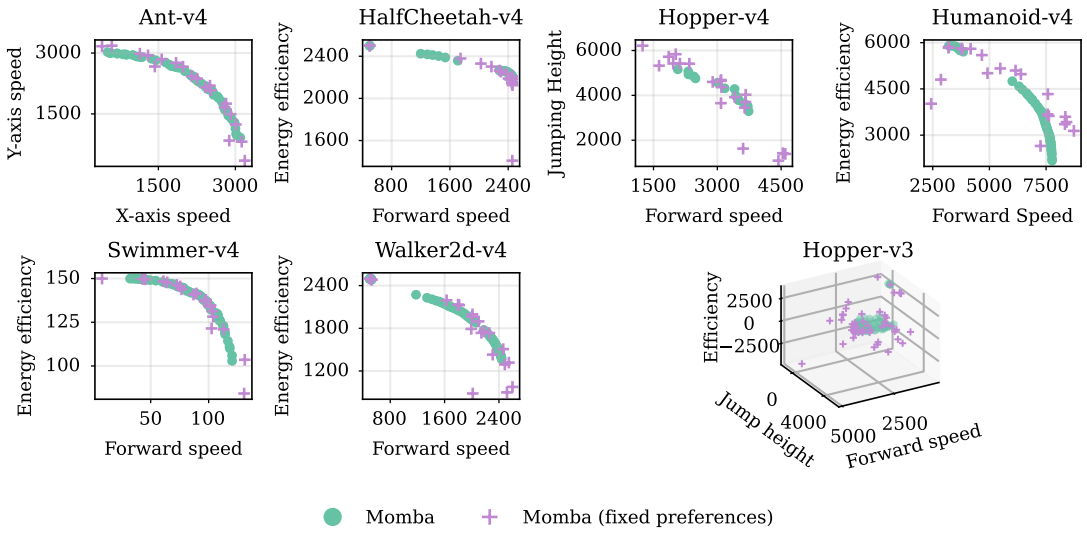

Figure 4.5: **Momba approximates the Pareto front in a single run**. We compare the learned solution sets to reference values obtained by training Momba several times with a fixed preference.

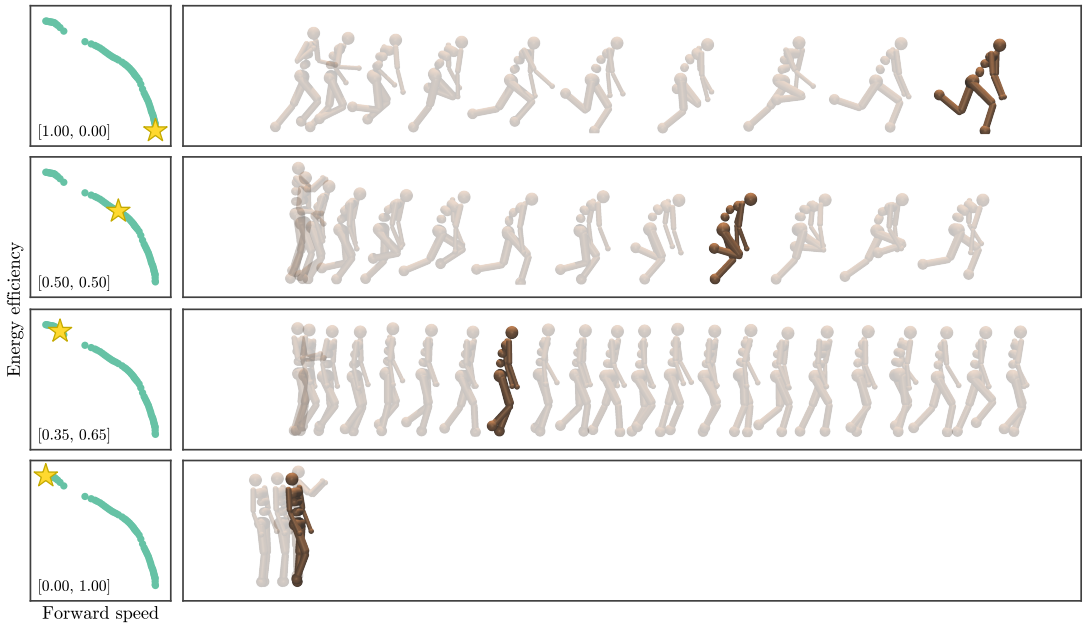

Figure 4.6: **Momba learns several behaviors in a single agent**. We show rollouts in the challenging high-dimensional Humanoid environment with four different preferences. The humanoid can stand and balance, walk slowly, and run fast, depending on the user's preference. The opaque pose shows the position after 200 timesteps, illustrating the difference in the forward speed of the policies.

4.3 Ablations

In this section, we isolate which design choices are responsible for Momba's performance gains. The first two ablations study value representation and critic scaling. For these experiments, we vary two main axes: the network architecture, choosing between SimbaV2 and a standard MLP with two hidden layers of 256 neurons, and the critic objective, choosing between a distributional critic trained with cross-entropy (CE) loss and an expected-value critic trained with mean-squared error (MSE) loss. The remaining ablations use the default Momba configuration, i.e. SimbaV2 with the distributional critic, and study the normalization components, preference conditioning, and update-to-data (UTD) ratio.

4.3.1 Value Representation

We first study the effects of the scalarized and the vector value representation. Figure 4.7 compares them across the four architecture and critic-loss configurations. The two approaches perform similarly overall, although the scalarized critic is slightly stronger with the MLP architecture. Importantly, Momba (Simba+CE) works equally well with both representations. This supports using the scalarized critic as the default when only scalarized values are needed, while retaining the vector critic for methods that require vector Q-values.

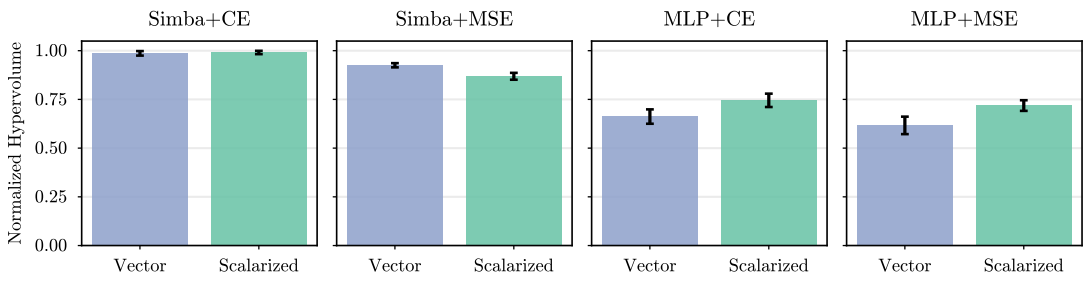

Figure 4.7: **Vector and scalarized value representation**. We show the normalized hypervolume (IQM with 95% SBCIs) for the vector and scalarized value representations, while varying the network architecture and critic loss.

4.3.2 Critic Scaling

One motivation for SimbaV2 is that it stabilizes learning and enables parameter scaling. We therefore study how performance changes as the critic size increases. We compare SimbaV2 against the MLP architecture with the distributional and non-distributional variants. We focus on scaling the critic, as scaling the actor typically provides limited benefits [32].

Since the default MLP and SimbaV2 architectures have very different numbers of trainable parameters, we vary the hidden dimension from 256 to 2048 for MLP and from 64 to 512 for Simba, resulting

in approximately matched critic sizes. Figure 4.8 shows that Momba (Simba+CE) outperforms the evaluated variants across all critic sizes and benefits from increasing the number of parameters.

The most important observation is that the distributional critic is highly effective even with the standard MLP: at large critic sizes, MLP+CE approaches the performance of Momba. This indicates that our distributional critic adaptation is a general improvement to value learning in MORL, rather than a benefit specific to SimbaV2.

At the same time, SimbaV2 remains beneficial across the evaluated critic sizes. By contrast, the MLP with MSE loss performs poorly regardless of critic size, and SimbaV2 with MSE loss does not benefit from scaling. This suggests that larger critics are useful only when paired with a well-behaved value representation and loss function.

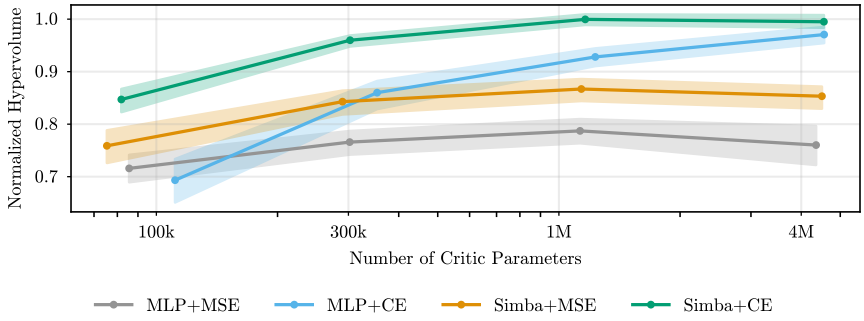

Figure 4.8: **The distributional critic adaptation is effective**. The figure displays normalized HV (IQM and 95% SBCIs over 10 seeds) as a function of the critic size. Here, Momba corresponds to Simba+CE.

4.3.3 Normalization Components

SimbaV2 combines several normalization mechanisms to stabilize training. To estimate the effect of weight normalization (WN), feature normalization (FN), and observation normalization (ON), we evaluate Momba under all combinations of these methods. Figure 4.9 displays the aggregate hypervolume of these combinations. We also compute Shapley values for the three normalization techniques to quantify their contributions to final performance.

The results indicate that observation normalization is the most effective component on our benchmarks, accounting for approximately 55% of the improvement according to the Shapley values. Feature and weight normalization provide more modest gains individually, but the full combination performs best, suggesting that the normalization schemes are complementary.

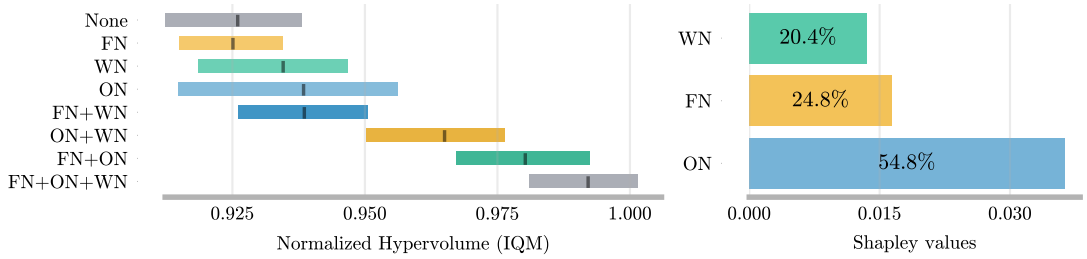

Figure 4.9: **Observation normalization is the most effective component**. Left: We investigate the effect of feature normalization (FN), observation normalization (ON), and weight normalization (WN) on the final HV (IQM and 95% SBCIs over 10 seeds). Here, Momba corresponds to FN+ON+WN. Right: Observation normalization is associated with the highest Shapley value, while weight and feature normalization are given similar, smaller importance.

4.3.4 Conditioning

Our default implementation conditions both the actor and the critic by concatenating the preference to the state. Because SimbaV2 is relatively deep, one might expect stronger conditioning to help by exposing the preference to later layers as well. We therefore also concatenate the preference to the hidden blocks and test FiLM layers [38], which apply preference-dependent feature-wise affine transformations at the end of each hidden block.

Table 4.4 compares the conditioning styles and their combinations. Contrary to the intuition, the input-concatenation baseline achieves the highest aggregate HV, while adding stronger conditioning tends to reduce performance. We suspect that the simpler conditioning encourages more parameter sharing and more stable training dynamics.

Table 4.4: **Comparison of conditioning approaches**. We report normalized hypervolume (IQM and 95% SBCIs over 5 seeds). The simplest concatenation-based conditioning works best in our setting.

Conditioning style	Normalized HV
Concatenate input	1.007 [0.995, 1.019]
Concatenate all	0.994 [0.980, 1.007]
FiLM	0.966 [0.946, 0.981]
FiLM + Concatenate all	0.940 [0.923, 0.956]

4.3.5 UTD Scaling

A more stable critic may allow more aggressive optimization between environment interactions. We therefore vary the update-to-data ratio and report the results in Figure 4.10.

Momba remains stable as UTD increases. In some environments, such as Humanoid and Hopper, higher UTD also improves early convergence. However, final performance is mostly unaffected, and the benefits are environment-dependent. These results suggest that value approximation is unlikely to be the main bottleneck in Momba, and that more efficient exploration or preference-selection schemes may be needed to further improve sample efficiency.

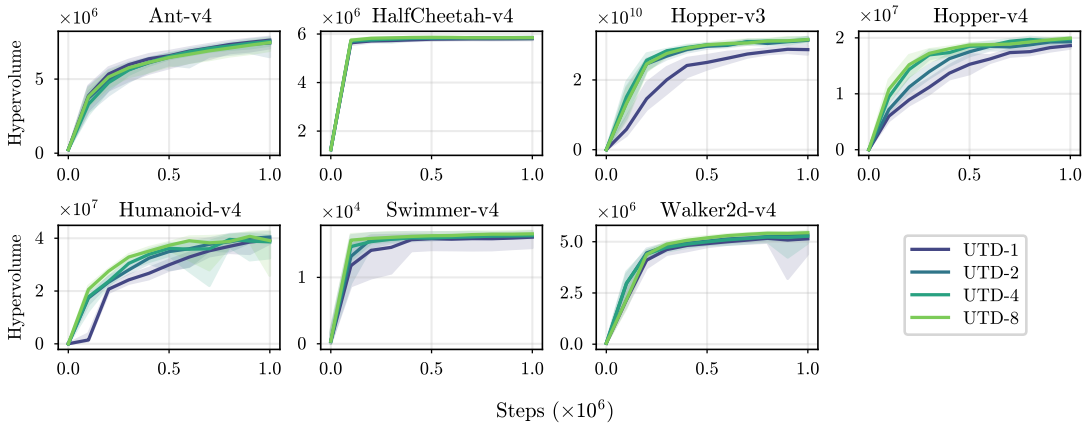

Figure 4.10: **UTD scaling of Momba**. We investigate the effect of the UTD ratio on performance. Momba remains stable at all UTD ratios. Depending on the environment, increasing the UTD ratio may lead to faster convergence. We show hypervolume (IQM with 95% SBCIs over 10 seeds).

4.4 Algorithmic Improvements

We now evaluate the algorithmic improvements presented in Section 3.5. Specifically, we integrate Momba with hindsight experience replay (HER) and preference alignment regularization (PAR). We evaluate these changes with the vector distributional critic, as the vector value predictions are needed for the preference alignment.

Figure 4.11 shows the resulting solution sets. The visualizations suggest that HER and PAR improve uniformity. The modified algorithm also finds policies at the edges of the learned front in Ant and Swimmer, improving coverage.

We quantify this effect using the inverted generational distance (IGD) [13]. Given a reference front R and a learned solution set S, IGD measures the average distance from each point on R to its nearest point in S. Formally,

$$\mathrm{IGD}(S, R) = \frac{1}{|R|} \sum_{r \in R} \min_{s \in S} \|r - s\|_2. \tag{4.3}$$

Lower IGD values indicate that the learned solution set covers the reference front more accurately. Because the true Pareto front is unknown, we construct an empirical reference front for each environ-

ment by taking the convex envelope of all solutions observed across every run and method considered in our experiments. This is not the true Pareto front, but it yields an optimistic and realizable approximation for the given environment. We compute IGD with both fronts normalized into the [0, 1] range to account for different reward magnitudes across environments.

Table 4.5 summarizes the quantitative results, reporting normalized HV and IGD aggregated across the seven environments. Both HER and PAR slightly improve HV. The larger effect appears in IGD: PAR substantially improves coverage, and HER provides complementary gains.

These results show that Momba can benefit from additional MORL-specific algorithmic components. They also support the usefulness of the vector distributional critic: it provides access to vector Q-values while retaining strong performance. Momba remains stable even when the training process and losses are modified.

Table 4.5: **Comparison of algorithmic improvements**. We evaluate the effect of hindsight experience replay and preference alignment regularization. Base denotes Momba with the vector distributional critic and no HER or PAR. We report normalized HV and IGD (IQM with 95% SBCIs over 10 seeds).

Configuration	Normalized HV ↑	IGD ↓
Base	0.986 [0.975, 0.998]	0.141 [0.131, 0.152]
HER	1.001 [0.989, 1.011]	0.126 [0.112, 0.134]
PAR	1.005 [0.995, 1.014]	0.094 [0.086, 0.101]
HER + PAR	1.019 [1.008, 1.029]	0.079 [0.073, 0.086]

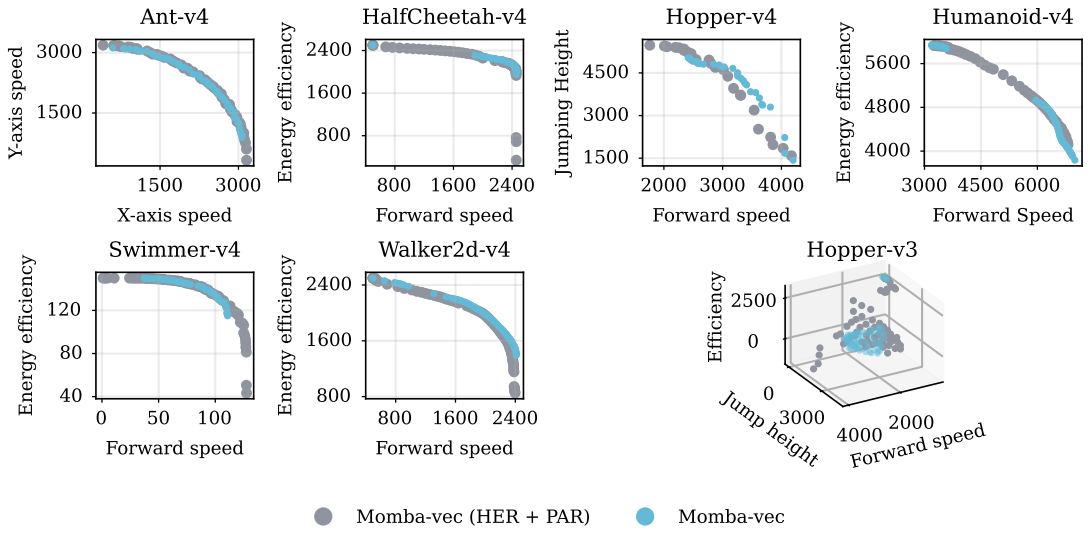

Figure 4.11: **Algorithmic improvements help with the uniformity of the solution sets**. HER and PAR were effective in filling the gaps in the front. They also increased coverage near the corners of the Pareto front in Ant and Swimmer.

4.5 Discussion

The results support the main hypothesis of this thesis: improving function approximators can substantially strengthen a MORL algorithm without changing its core learning principles. We demonstrate this by extending CAPQL with the SimbaV2 architecture and distributional value representation to create Momba. Momba substantially outperformed CAPQL and the selected MORL baselines in solution-set quality measured by hypervolume and expected utility. The ablations further suggest that the distributional critic is the largest contributor among the tested components, while architectural elements, including observation, feature, and weight normalization, provide complementary gains.

At the same time, improvements in function approximation alone do not resolve all challenges in MORL. Some learned fronts still contain gaps, particularly in regions where the policy transitions between qualitatively different behaviors. Preference alignment regularization and hindsight experience replay both improve the uniformity of the solution sets, demonstrating that advances in function approximation and MORL-specific algorithmic techniques are complementary rather than competing approaches.

Finally, Figure 4.5 suggests that Momba approaches the fixed-preference performance for this algorithm family on the evaluated benchmarks. These tasks may therefore represent a relatively simple regime for MORL. Future work would benefit from evaluating on more challenging environments with higher-dimensional observations, task complexity, and greater numbers of objectives.

5 Conclusions

In this work, we studied the effect of using expressive function approximators in multi-objective reinforcement learning. Our findings demonstrate that distributional value representation and a modern neural architecture can substantially improve the performance of a preference-conditioned MORL algorithm. Additionally, the improved function approximators can be successfully integrated with MORL-specific algorithmic ideas.

We devised two tractable approaches to adapting distributional value representation to the multi-objective setting: learning the scalarized return distribution or learning the marginals of the vector return distribution. We combined these distributional critic adaptations with the SimbaV2 architecture and integrated them with CAPQL to obtain Momba. Both value representations were effective in our experiments, and Momba outperformed the evaluated MORL baselines. We also provide a unified implementation that supports controlled experimentation with architectures, value representations, and architectural normalization schemes.

The ablations identify the distributional critic as the largest contributor among the tested components. Among the normalization schemes used in SimbaV2, observation normalization contributes the largest improvement, while feature normalization and weight normalization provide smaller complementary gains. This provides actionable insights for designing new function approximators in multi-objective RL.

While Momba performed well in continuous-control environments, several research questions remain open. First, we studied a specific combination of CAPQL as the base algorithm and SimbaV2 as the architecture. Future work should test whether the results extend beyond this pairing, for example by replacing CAPQL with a TD3-based algorithm [15] or SimbaV2 with another recent architecture such as XQC [36]. It also remains unclear whether these architectures help outside continuous control or improve performance in environments with discrete states or actions. Finally, the remaining gaps in some learned fronts suggest that better function approximators are not always sufficient on their own. Additional preference-sampling, relabeling, or alignment techniques may still be required for reliable coverage of the Pareto front.

Bibliography

1. A. Abels, D. Roijers, T. Lenaerts, A. Nowé, and D. Steckelmacher. "Dynamic Weights in Multi-Objective Deep Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 11–20. URL: https://proceedings.mlr.press/v97/abels19a.html.

2. R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. "Deep reinforcement learning at the edge of the statistical precipice". *Advances in Neural Information Processing Systems* 34, 2021.

3. L. N. Alegre, A. L. C. Bazzan, D. M. Roijers, A. Nowé, and B. 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*. AAMAS '23. International Foundation for Autonomous Agents and Multiagent Systems, London, United Kingdom, 2023, pp. 2003–2012. ISBN: 9781450394321.

4. M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem. *What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study*. 2020. arXiv: 2006.05990 [cs.LG]. URL: https://arxiv.org/abs/2006.05990.

5. M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. *Hindsight Experience Replay*. 2018. arXiv: 1707.01495 [cs.LG]. URL: https://arxiv.org/abs/1707.01495.

6. J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML]. URL: https://arxiv.org/abs/1607.06450.

7. T. Basaklar, S. Gumussoy, and U. Y. Ogras. *PD-MORL: Preference-Driven Multi-Objective Reinforcement Learning Algorithm*. 2023. arXiv: 2208.07914 [cs.LG]. URL: https://arxiv.org/abs/2208.07914.

8. M. G. Bellemare, W. Dabney, and R. Munos. *A Distributional Perspective on Reinforcement Learning*. 2017. arXiv: 1707.06887 [cs.LG]. URL: https://arxiv.org/abs/1707.06887.

9. R. Bellman, R. Corporation, and K. M. R. Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. ISBN: 9780691079516.

10. X.-Q. Cai, P. Zhang, L. Zhao, J. Bian, M. Sugiyama, and A. Llorens. "Distributional Pareto-Optimal Multi-Objective Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., 2023, pp. 15593–15613.

11. K. Chatterjee, R. Majumdar, and T. A. Henzinger. "Markov decision processes with multiple objectives". In: *Proceedings of the 23rd Annual Conference on Theoretical Aspects of Computer Science*. STACS'06. Springer-Verlag, Marseille, France, 2006, pp. 325–336. ISBN: 3540323015. DOI: 10.1007/11672142_26. URL: https://doi.org/10.1007/11672142_26.

12. X. Chen, C. Wang, Z. Zhou, and K. Ross. *Randomized Ensembled Double Q-Learning: Learning Fast Without a Model*. 2021. arXiv: 2101.05982 [cs.LG]. URL: https://arxiv.org/abs/2101.05982.

13. C. A. Coello Coello and M. Reyes Sierra. "A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm". In: *MICAI 2004: Advances in Artificial Intelligence*. Ed. by R. Monroy, G. Arroyo-Figueroa, L. E. Sucar, and H. Sossa. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 688–697. ISBN: 978-3-540-24694-7.

14. F. Felten, L. N. Alegre, A. Nowe, A. Bazzan, E. G. Talbi, G. Danoy, and B. C. da Silva. "A Toolkit for Reliable Benchmarking and Research in Multi-Objective Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., 2023, pp. 23671–23700.

15. S. Fujimoto, H. van Hoof, and D. Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI]. URL: https://arxiv.org/abs/1802.09477.

16. T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG]. URL: https://arxiv.org/abs/1801.01290.

17. T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. *Soft Actor-Critic Algorithms and Applications*. 2019. arXiv: 1812.05905 [cs.LG]. URL: https://arxiv.org/abs/1812.05905.

18. C. F. Hayes, R. Rădulescu, E. Bargiacchi, J. Källström, M. Macfarlane, M. Reymond, T. Verstraeten, L. M. Zintgraf, R. Dazeley, F. Heintz, E. Howley, A. A. Irissappane, P. Mannion, A. Nowé, G. Ramos, M. Restelli, P. Vamplew, and D. M. Roijers. "A practical guide to

multi-objective reinforcement learning and planning". en. *Auton. Agent. Multi. Agent. Syst.* 36:1, 2022.

19. K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

20. J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou. *Deep Learning Scaling is Predictable, Empirically*. 2017. arXiv: 1712.00409 [cs.LG]. URL: https://arxiv.org/abs/1712.00409.

21. T. Hiraoka, T. Imagawa, T. Hashimoto, T. Onishi, and Y. Tsuruoka. *Dropout Q-Functions for Doubly Efficient Reinforcement Learning*. 2022. arXiv: 2110.02034 [cs.LG]. URL: https://arxiv.org/abs/2110.02034.

22. S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG]. URL: https://arxiv.org/abs/1502.03167.

23. J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

24. D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

25. V. Konda and J. Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999.

26. H. Lee, Y. Lee, T. Seno, D. Kim, P. Stone, and J. Choo. *Hyperspherical Normalization for Scalable Deep Reinforcement Learning*. 2025. arXiv: 2502.15280 [cs.LG]. URL: https://arxiv.org/abs/2502.15280.

27. Q. Li, A. Kumar, I. Kostrikov, and S. Levine. *Efficient Deep Reinforcement Learning Requires Regulating Overfitting*. 2023. arXiv: 2304.10466 [cs.LG]. URL: https://arxiv.org/abs/2304.10466.

28. H. Lu, D. Herman, and Y. Yu. "Multi-Objective Reinforcement Learning: Convexity, Stationarity and Pareto Optimality". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=TjEzIsyEsQ6.

29. C. Lyle, Z. Zheng, K. Khetarpal, J. Martens, H. van Hasselt, R. Pascanu, and W. Dabney. *Normalization and effective learning rates in reinforcement learning*. 2024. arXiv: 2407.01800 [cs.LG]. URL: https://arxiv.org/abs/2407.01800.

30. A. S. Manne. “Linear Programming and Sequential Decisions”. *Management Science* 6:3, 1960, pp. 259–267. ISSN: 00251909, 15265501. URL: http://www.jstor.org/stable/2627340 (visited on 04/13/2026).

31. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. “Human-level control through deep reinforcement learning”. *Nature* 518:7540, 2015, pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: http://dx.doi.org/10.1038/nature14236.

32. S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko. *Honey, I Shrunk The Actor: A Case Study on Preserving Performance with Smaller Actors in Actor-Critic RL*. 2021. arXiv: 2102.11893 [cs.LG]. URL: https://arxiv.org/abs/2102.11893.

33. M. Nauman, M. Bortkiewicz, P. Miłoś, T. Trzciński, M. Ostaszewski, and M. Cygan. *Overestimation, Overfitting, and Plasticity in Actor-Critic: the Bitter Lesson of Reinforcement Learning*. 2024. arXiv: 2403.00514 [cs.LG]. URL: https://arxiv.org/abs/2403.00514.

34. M. Nauman, M. Ostaszewski, K. Jankowski, P. Miłoś, and M. Cygan. *Bigger, Regularized, Optimistic: scaling for compute and sample-efficient continuous control*. 2024. arXiv: 2405.16158 [cs.LG]. URL: https://arxiv.org/abs/2405.16158.

35. E. Nikishin, M. Schwarzer, P. D’Oro, P.-L. Bacon, and A. Courville. *The Primacy Bias in Deep Reinforcement Learning*. 2022. arXiv: 2205.07802 [cs.LG]. URL: https://arxiv.org/abs/2205.07802.

36. D. Palenicek, F. Vogt, J. Watson, I. Posner, and J. Peters. *XQC: Well-conditioned Optimization Accelerates Deep Reinforcement Learning*. 2026. arXiv: 2509.25174 [cs.LG]. URL: https://arxiv.org/abs/2509.25174.

37. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703.

38. E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville. *FiLM: Visual Reasoning with a General Conditioning Layer*. 2017. arXiv: 1709.07871 [cs.CV]. URL: https://arxiv.org/abs/1709.07871.

39. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. John Wiley & Sons, Inc., USA, 1994. ISBN: 0471619779.

40. D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. "A survey of multi-objective sequential decision-making". *J. Artif. Int. Res.* 48:1, 2013, pp. 67–113. ISSN: 1076-9757.

41. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

42. J. Shianifar, M. Schukat, and K. Mason. *Hindsight Preference Replay Improves Preference-Conditioned Multi-Objective Reinforcement Learning*. 2026. arXiv: 2601.11604 [cs.LG]. URL: https://arxiv.org/abs/2601.11604.

43. T. Shu, K. Shang, C. Gong, Y. Nan, and H. Ishibuchi. "Learning Pareto Set for Multi-Objective Continuous Robot Control". In: *International Joint Conference on Artificial Intelligence*. 2024.

44. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research* 15:56, 2014, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

45. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN: 0262039249.

46. J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. *Sim-to-Real: Learning Agile Locomotion For Quadruped Robots*. 2018. arXiv: 1804.10332 [cs.RO]. URL: https://arxiv.org/abs/1804.10332.

47. E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

48. J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik. "Prediction-Guided Multi-Objective Reinforcement Learning for Continuous Robot Control". In: *Proceedings of the 37th International Conference on Machine Learning*. 2020.

49. R. Yang, X. Sun, and K. Narasimhan. "A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation". *CoRR* abs/1908.08342, 2019. arXiv: 1908.08342. URL: http://arxiv.org/abs/1908.08342.

50. L. M. Zintgraf, T. V. Kanters, D. M. Roijers, F. Oliehoek, and P. 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.

51. E. Zitzler and L. Thiele. "Multiobjective optimization using evolutionary algorithms — A comparative case study". In: *Parallel Problem Solving from Nature — PPSN V*. 1998.

A Implementation Archive

The attached archive `momba.zip` contains the implementation of Momba used in this thesis. The code is structured as a Python module, with the training launched through the `momba.train` module. The options for reproducing the experiments discussed in the text are listed in Table A.1.

Table A.1: Options for the algorithm variants evaluated in the thesis.

Option	**Values**
`--architecture`	`simba`, `mlp`
`--critic-loss`	`ce`, `mse`
`--vector-values`	boolean
`--her`	boolean
`--preference-alignment-regularization`	boolean

B Hyperparameters

We report the default hyperparameters used in our experiments in Table B.1. We did not perform extensive hyperparameter optimization, so better performance might be achieved through a large-scale hyperparameter search. The best return scale controls the reward rescaling used with the fixed categorical support, so the Q-values span the available support. We use automatic tuning of the entropy regularization coefficient α so the actor matches the target entropy, whereas the original CAPQL implementation used fixed entropy coefficients manually tuned for each environment.

Table B.1: 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	5000	Critic blocks	2
γ	0.99	Critic hidden dim	256
Target critic momentum	0.005	Number of critics	2
Optimizer	Adam	Number of atoms	101
Policy LR	1×10^{-4}	Best return scale	3.0
Critic LR	1×10^{-4}	Categorical support	$[-5, 5]$
Initial α	0.2	UTD	1
Target entropy	$-\lvert A \rvert$	Batch size	256