Continual Multi-Objective Reinforcement Learning via Reward Model Rehearsal

Lihe Li[1,2] , **Ruotong Chen**[1,2] , **Ziqian Zhang**[1,2] , **Zhichao Wu**[1,2] , **Yi-Chen Li**[1,2,3] ,
Cong Guan[1,2] , **Yang Yu**[1,2,3*] , **Lei Yuan**[1,2,3]
[1]National Key Laboratory for Novel Software Technology, Nanjing University
[2]School of Artificial Intelligence, Nanjing University
[3]Polixir Technologies
{lilh, zhangzq, liyc, guanc, yuanl}@lamda.nju.edu.cn, {chenrt, zcwu}@smail.nju.edu.cn,
yuy@nju.edu.cn

Abstract

Multi-objective reinforcement learning (MORL) approaches address real-world problems with multiple objectives by learning policies maximizing returns weighted by different user preferences. Typical methods assume the objectives remain unchanged throughout the agent's lifetime. However, in some real-world situations, the agent may encounter dynamically changing learning objectives, i.e., different vector-valued reward functions at different learning stages. This issue of evolving objectives has not been considered in problem formulation or algorithm design. To address this issue, we formalize the setting as a continual MORL (CMORL) problem for the first time, accounting for the evolution of objectives throughout the learning process. Subsequently, we propose **C**ontinual Multi-**O**bjective **Re**inforcement Learning via **Re**ward Model **Re**hearsal (CORE3), incorporating a dynamic agent network for rapid adaptation to new objectives. Moreover, we develop a reward model rehearsal technique to recover the reward signals for previous objectives, thus alleviating catastrophic forgetting. Experiments on four CMORL benchmarks showcase that CORE3 effectively learns policies satisfying different preferences on all encountered objectives, and outperforms the best baseline by 171%, highlighting the capability of CORE3 to handle situations with evolving objectives.

1 Introduction

Reinforcement learning (RL) has garnered prominent attention in recent years [Wang *et al.*, 2022], and made exciting progress in various real-world sequential decision-making problems, like robotic control [Singh *et al.*, 2022], autonomous driving [Kiran *et al.*, 2022], and aligning large language models with human values [Kaufmann *et al.*, 2023], etc. These problems could be formalized as completing a specific objective, i.e., maximizing a scalar return. Nevertheless, numerous real-world problems encompass multiple, possibly conflicting objectives. An illustrative example is an autonomous driving agent tasked with multiple objectives, such as maximizing speed and minimizing fuel consumption. To tackle such complex scenarios, multi-objective reinforcement learning (MORL) approaches [Roijers *et al.*, 2013; Liu *et al.*, 2014] have been introduced to concurrently learn these multiple objectives with a vector-valued reward function, obtain a set of policies maximizing returns weighted by different preferences, and employ the appropriate policy given a user preference during testing, like driving fast for a passenger with urgent matters, or driving slowly and smoothly for a leisurely traveler.

Given the extensive and significant applications of MORL, many approaches have been developed [Hayes *et al.*, 2022]. Typical methods primarily concentrate on developing various techniques to enhance the learning efficiency over the multiple objectives, including applying meta-learning techniques [Chen *et al.*, 2019], utilizing prediction-guided evolutionary learning [Xu *et al.*, 2020], developing preference-conditioned network architecture [Abels *et al.*, 2019], proposing novel Bellman update strategies [Basaklar *et al.*, 2023], etc. Beyond that, these methods assume that throughout the agent's entire lifetime, the learning objectives remain unchanged. However, the agent may encounter dynamically changing learning objectives, i.e., altering vector-valued reward functions. On one hand, new objectives and their reward signals may appear at different learning stages, as it is natural and inevitable for human users to propose new requirements. On the other hand, some objectives may be removed from the agent's perspective, as their reward signals become inaccessible after a limited training period, yet the agent must retain the ability to complete them [Chang *et al.*, 2021]. For instance, the reward for the objective "stay in the middle lane" will be unavailable after the corresponding sensors run out of power or malfunction [Shaheen *et al.*, 2022; Wang *et al.*, 2023], while the autonomous driving agent is still obliged to manage it for certain passengers. Without considering these scenarios, the underlying assumption of fixed objectives made by existing methods hinders the further development of MORL and its practical application in scenarios with evolving objectives.

To bridge the gap between existing methods and the mentioned scenarios, and endow MORL agents with continual learning ability, we take the issues into consideration

*Corresponding Author

and formalize the setting as a continual MORL (CMORL) problem for the first time, where the agent encounters a sequence of MORL tasks with objectives altering continually, and should learn to complete all encountered objectives (Figure 1). Furthermore, we propose **C**ontinual Multi-**O**bjective **Re**inforcement Learning via **Re**ward Model **Re**hearsal (CORE3), incorporating a dynamic agent network with a multi-head architecture [Kessler *et al.*, 2022; Zhang *et al.*, 2023a], which enables flexible expansion for new objectives and effective knowledge transfer [Zhu *et al.*, 2023]. To avoid catastrophic forgetting about objectives lacking reward signals, inspired by the concept of rehearsal in continual learning [Parisi *et al.*, 2019], we develop a reward model rehearsal approach to recover these signals and use them to update the policy. This approach not only provides a theoretical guarantee but also effectively alleviates the unique challenge of catastrophic forgetting in CMORL, ensuring the agent's high performance on all encountered objectives.

We conduct experiments within four CMORL benchmarks featuring evolving objective sets. The results showcase that CORE3 effectively learns to accomplish all encountered objectives in accordance with different preferences, outperforming the best baseline by 171% on Hypervolume, a widely-used metric in MORL [Hayes *et al.*, 2022]. More results further provide insight into how CORE3 successfully tackles the CMORL problem with the proposed technologies. These findings highlight the capability of CORE3 to effectively handle real-world situations involving evolving objectives.

2 Related Work

Multi-objective reinforcement learning (MORL) extends the conventional RL framework with a single objective to multi-objective settings [Roijers *et al.*, 2013; Liu *et al.*, 2014], and can be applied to real-world problems like hyperparameter tuning [Chen *et al.*, 2021], canal control [Ren *et al.*, 2021], etc. Among the approaches, the single-policy series [Pan *et al.*, 2020; Siddique *et al.*, 2020; Hwang *et al.*, 2023] predefines objective preferences, converting the problem into a single-objective one which is solvable by traditional RL methods. When the preference cannot be known in advance, multi-policy approaches aim to learn a set of policies that approximates the Pareto front of solutions, and select the optimal one for testing given a preference. One classical work is PG-MORL [Xu *et al.*, 2020], which updates a policy population using an evolutionary algorithm to approximate the Pareto front. Another line of works like Envelope [Yang *et al.*, 2019] and PD-MORL [Basaklar *et al.*, 2023], train a single preference-conditioned network over multiple preferences with different Bellman update strategies. To release the burden of learning multiple policies simultaneously, meta-policy approaches [Chen *et al.*, 2019; Zhang *et al.*, 2023b] first train a meta-policy and finetune it with a small amount of update steps to derive the solution for a given preference. To test the learning efficiency of various MORL methods, researchers also develop different benchmarks [Xu *et al.*, 2020; Felten *et al.*, 2023]. Despite these progress, existing MORL approaches and benchmarks focus on learning a fixed set of objectives, while our work is the first to explore the MORL problem with objectives evolving continually.

Continual reinforcement learning (CRL) [Ring, 1995; Khetarpal *et al.*, 2022] focuses on enabling agents to learn a sequence of different RL tasks and balance the stability-plasticity dilemma, i.e., alleviating catastrophic forgetting of old tasks while adapting to new tasks. One classic method is EWC [Kirkpatrick *et al.*, 2017], which adds an l_2-distance-based regularizer to constrain the update of the agent network. Another task-agnostic approach CLEAR [Rolnick *et al.*, 2019] stores the transitions of every encountered task in the buffer, and uses them to rehearse the agent. Other works [Huang *et al.*, 2021; Kessler *et al.*, 2023] learn a world model to assist the agent's continual learning. OWL [Kessler *et al.*, 2022] and DaCoRL [Zhang *et al.*, 2023a] utilize a multi-head architecture to enhance the agent's continual learning ability. CSP [Gaya *et al.*, 2023] builds a subspace of policies continually. CPPO [Anonymous, 2024] continually learns from human preferences based on PPO [Schulman *et al.*, 2017]. Other researchers also develop benchmarks [Wolczyk *et al.*, 2021; Powers *et al.*, 2022] to test the continual learning ability of different CRL methods. While these methods focus on the standard RL tasks with only one objective, i.e., scalar reward signals, our work presents the first exploration into MORL in the continual learning setting.

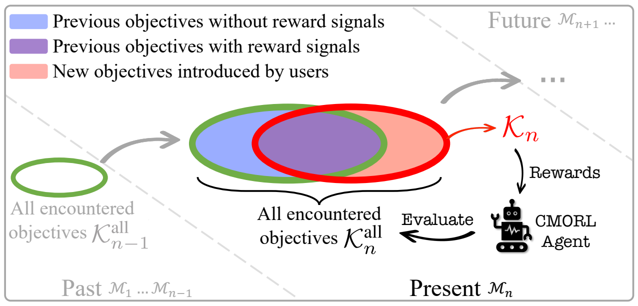

Previous objectives without reward signals
Previous objectives with reward signals
New objectives introduced by users
Future $\mathcal{M}_{n+1}$ ...
...
$\mathcal{K}_n$
Rewards
All encountered objectives $\mathcal{K}_{n-1}^{\text{all}}$
All encountered objectives $\mathcal{K}_n^{\text{all}}$
Evaluate
CMORL Agent
Past $\mathcal{M}_1 \ldots \mathcal{M}_{n-1}$
Present $\mathcal{M}_n$

Figure 1: The overall workflow of continual MORL (CMORL).

3 Problem Formulation

A multi-objective sequential decision making problem can be formulated as a Multi-Objective Markov Decision Process (MOMDP), with tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, \mathcal{K}, \mathbf{R}, \Omega, f_\omega, \gamma \rangle$. Here, $\mathcal{S}$ and $\mathcal{A}$ denote the state space and the action space. $P : \mathcal{S} \times \mathcal{A} \to \Pr(\mathcal{S})$ is the transition function. $\mathcal{K} = \{k_1, \cdots, k_m\}$ is the set of m different objectives, a subset of all possible objectives $\mathbb{K}$. $\mathbf{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^m$ is the vector-valued reward function. Ω is the preference space. $f_\omega : \mathbb{R}^m \to \mathbb{R}$ is the scalarization function under preference $\omega \in \Omega$, and γ is the discount factor. We focus on the linear reward scalarization setting, i.e., $f_\omega(\mathbf{R}(s, a)) = \omega^\top \mathbf{R}(s, a)$, for $s \in \mathcal{S}, a \in \mathcal{A}$, and $\Omega = \Delta^m$ is the unit simplex, aligning with the established practices in the MORL literature [Hayes *et al.*, 2022]. At each time step t, the agent observes state s_t, takes action a_t, and receives a reward vector $\mathbf{r}_t = \mathbf{R}(s_t, a_t) = (R_{k_1}(s_t, a_t), \cdots, R_{k_m}(s_t, a_t))$, where $R_{k_i}(s_t, a_t)$ is the reward signal of objective $k_i (i = 1, \cdots, m)$. The agent knows the objective that each scalar reward corresponds to. Here, we learn a preference-conditioned agent policy π :

$\mathcal{S} \times \Delta^m \rightarrow \Pr(\mathcal{A})$. The goal is to find the optimal policy π^*, such that $\forall \omega \in \Delta^m, \pi^* = \arg\max_\pi \omega^\top J(\pi(\cdot|\cdot;\omega))$, where $J(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(s_t, a_t)]$ is the value vector, and $\tau = (s_0, a_0, s_1, a_1, \cdots)$ is the trajectory.

In this work, we aim to solve the CMORL problem where the agent encounters a sequence of multi-objective tasks: $(\mathcal{M}_1, \cdots, \mathcal{M}_n, \cdots)$, as shown in Figure 1. Each task $\mathcal{M}_n$ is a MOMDP with specific objectives $\mathcal{K}_n = \{k_1^n, \ldots, k_{m_n}^n\} \subset \mathbb{K}$, reward function $\mathbf{R}_n = (R_1^n, \cdots, R_{m_n}^n)$, and preference space $\Omega_n = \Delta^{m_n}$. Here, R_i^n denotes $R_{k_i^n}(i = 1, \cdots, m_n)$ for simplicity. The agent knows if the task has changed. However, it can only receive the reward signals of the current objective set $\mathcal{K}_n$, and is evaluated on all encountered ones, i.e., $\mathcal{K}_n^{\text{all}} = \cup_{i=1}^n \mathcal{K}_i = \{k_1^1, \cdots, k_{m_1}^1\} \cup \cdots \cup \{k_1^n, \cdots, k_{m_n}^n\} = \{k_1, \cdots, k_{M_n}\}$, where M_n represents the total number of objectives introduced up to and including task $\mathcal{M}_n$. Note that $M_n \leq \sum_{i=1}^n m_n$ because some objectives may appear repeatedly in different tasks (e.g., $k_1^1 = k_1^2 = k_1$). These settings of CMORL necessitate the agent's rapid adaptation to new objectives while maintaining the ability to accomplish the previously encountered ones.

4 Method

In this section, we present the detailed design of our proposed method CORE3. First, we introduce a novel dynamic agent network architecture that adaptively expands as new objectives arrive, facilitating rapid adaptation and circumventing the need to learn the entire network from scratch. Next, we utilize a multi-objective reward model to recover reward signals of previously encountered objectives, thereby alleviating the phenomenon of catastrophic forgetting. Through the incorporation of these techniques, CORE3 effectively learns the sequence of multi-objective tasks in a continual manner.

4.1 Dynamic Network for Evolving Objectives

Under the CMORL setting, the learning objectives alter continually along with the task sequence, leading to an increase in the number of all encountered objectives. As a result, the conventional network architecture with fixed input and output shapes will no longer work. To address this problem, we design a dynamic and expandable network architecture.

Concretely, for discrete action spaces, we learn a vectorized Q network $\mathbf{Q}(s, a, \omega; \theta)$ to approximate the action-value function and use it as the policy. We design the Q network architecture as a combination of a feature extractor $\mathcal{E}(s, a, \omega; \xi)$ and distinct heads $\{h(e; \psi_i)\}_{i=1}^{M_n}$ for all encountered objectives. The feature extractor first transforms the varying-length preference ω into a fixed-length embedding z_ω using a GRU [Chung *et al.*, 2014]. Then, a multilayer perceptron (MLP) takes the state, action, and embedding z_ω as input and outputs feature $e = \mathcal{E}(s, a, \omega; \xi)$. Then, each MLP head $h(e; \psi_i)$ takes e as input, and outputs the Q value of objective k_i, forming the overall output Q vector $\big(h(e; \psi_1), \cdots, h(e; \psi_{M_n})\big) = \mathbf{Q}(s, a, \omega; \theta)$ for all objectives. With the Q vector, the CMORL agent is able to select the optimal action $a = \arg\max_{\tilde{a}\in\mathcal{A}} \omega^\top \mathbf{Q}(s, \tilde{a}, \omega; \theta)$ w.r.t. the given preference ω. During training task $\mathcal{M}_n$, the Q network is optimized to minimize the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,\mathbf{r},s',\omega)\sim\mathcal{D}}\left[\|\mathbf{y} - \mathbf{Q}(s, a, \omega; \theta)\|_2\right], \\ \text{where } \mathbf{Q}(s, a, \omega; \theta) = \big(h(e; \psi_1), \cdots, h(e; \psi_{M_n})\big), \tag{1}$$

and $\mathcal{D}$ is the replay buffer, $\theta = (\xi, \{\psi_i\}_{i=1}^{M_n})$ are the updated parameters of the networks, $\mathbf{y} = \mathbf{r} + \gamma \mathbf{Q}(s', a', \omega; \theta^-)$ denotes the target value vector which is obtained using target Q network's parameters θ^- and the action $a' = \arg\max_{\tilde{a}\in\mathcal{A}} \omega^\top \mathbf{Q}(s', \tilde{a}, \omega; \theta^-)$. When new objectives arrive, we dynamically create new objective heads and expand the network. For continuous action spaces, we apply $\mathbf{Q}(s, a, \omega; \theta)$ as the critic and learn an actor $\pi(a|s, \omega; \phi)$. When computing the target value vector $\mathbf{y}$ to optimize $\mathbf{Q}$, the action a' is obtained by the actor: $a' \sim \pi(\cdot|s', \omega; \phi)$. The actor $\pi(\cdot|s, \omega; \phi)$ is also equipped with a GRU to process the varying-length preference, and is optimized to maximize $\omega^\top \mathbf{Q}$. Such network architecture not only handles the increasing objective number, but also facilitates efficient knowledge transfer between tasks with different objectives. More details about the network architecture and the optimization process are provided in Appendix C.1.

4.2 Multi-Objective Reward Model Rehearsal

With the designed network architecture and loss function, the agent can learn to complete all encountered objectives effectively. Nevertheless, the computation of the loss function $\mathcal{L}(\theta)$ defined in Equation 1 requires access to the reward signals of all previous objectives, i.e., $\mathcal{K}_n^{\text{all}} \backslash \mathcal{K}_n$. Unfortunately, in the CMORL setting, these reward signals are unavailable from the environment. Simply neglecting these objectives would lead to the undesirable consequence of the agent losing its capability to accomplish them, i.e., catastrophic forgetting phenomenon. These issues pose a great challenge to the agent's learning process.

To solve these issues, we propose learning a multi-objective reward model $\hat{\mathbf{R}}$ to recover the reward signals for all previous objectives $\mathcal{K}_n^{\text{all}} \backslash \mathcal{K}_n$. Concretely, when learning task $\mathcal{M}_n$, the environment provides reward signals $\mathbf{r}_t = (R_1^n(s_t, a_t), \cdots, R_{m_n}^n(s_t, a_t))$ corresponding to objectives $\mathcal{K}_n = \{k_1^n, \ldots, k_{m_n}^n\}$. Next, we predict reward vector $\hat{\mathbf{r}}_t = \hat{\mathbf{R}}(s_t, a_t) = (\hat{R}_1(s_t, a_t), \cdots, \hat{R}_{M_n}(s_t, a_t))$. Note that if objective $k_i \in \mathcal{K}_n (i = 1, \cdots, M_n)$, we directly set $\hat{R}_i(s_t, a_t)$ as the ground-truth reward in $\mathbf{r}_t$. Then, $\hat{\mathbf{r}}_t$ is employed to compute the target value vector in Equation 1, to rehearse the agent about previous objectives. From the agent's perspective, the reward signals for all encountered objectives persist throughout the entire learning process. Consequently, the agent's performance significantly depends on the quality of the multi-objective reward model. Here, we theoretically analyze their relationship.

Theorem 1 (Bounded Performance Gap). *Consider two MOMDPs, $\mathcal{M}$ and $\hat{\mathcal{M}}$, differing only in their reward functions: $\mathbf{R} = (R_1, \cdots, R_m)$ and $\hat{\mathbf{R}} = (\hat{R}_1, \cdots, \hat{R}_m)$. The value vectors of a policy π in $\mathcal{M}$ and $\hat{\mathcal{M}}$ are $J_{\mathcal{M}}(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(s_t, a_t)]$ and $J_{\hat{\mathcal{M}}}(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t \hat{\mathbf{R}}(s_t, a_t)]$, respectively. The optimal policies for $\mathcal{M}$ and $\hat{\mathcal{M}}$ are denoted as π^* and $\hat{\pi}^*$, satisfy-*

ing $\forall\omega \in \Delta^m, \pi^* = \arg\max_\pi \omega^\top J_{\mathcal{M}}(\pi)$, *and* $\hat{\pi}^* = \arg\max_\pi \omega^\top J_{\hat{\mathcal{M}}}(\pi)$. *Define the difference between these reward functions as* $\delta = \sup_{s\in\mathcal{S},a\in\mathcal{A}} ||\mathbf{R}(s,a) - \hat{\mathbf{R}}(s,a)||_\infty$. *Based on these definitions, the following inequality holds,*

$$\forall\omega \in \Delta^m, \omega^\top J_{\mathcal{M}}(\pi^*) - \omega^\top J_{\mathcal{M}}(\hat{\pi}^*) \leq \frac{2\delta}{1-\gamma}. \quad (2)$$

The proof is provided in Appendix A. As indicated by Theorem 1, the performance gap between the policy learned from the recovered rewards $\hat{\mathbf{r}}_t$ and the policy learned from the ground-truth rewards $\mathbf{r}_t$ is bounded by $\frac{2\delta}{1-\gamma}$, with δ representing the prediction error of the multi-objective reward model in our context. So, we can minimize the performance gap by improving the reward model's prediction accuracy. To achieve this goal and circumvent the necessity of training a separate reward model for each objective, we also design a multi-head architecture for the reward model network. A dynamic encoder $\mathcal{F}$ takes the current state and action as input, and outputs the dynamic feature $z = \mathcal{F}(s,a;\vartheta)$. For each objective $k_i (i = 1, \cdots, M_n)$, a reward prediction head $g(\cdot;\varphi_i)$ takes z as input, and predicts reward $\hat{R}_i(s,a) = g(z;\varphi_i)$. New heads will be created when novel objectives appear. We also learn a state prediction head to predict the next state $\hat{s}' = g(z;\varphi_s)$, thereby stabilizing training and enhancing the dynamic encoder's ability to capture the fundamental information of the environment. Let $\Phi = (\vartheta, \{\varphi_i^n\}_{i=1}^{m_n}, \varphi_s)$ denote the network parameters. The model is optimized in an end-to-end manner to minimize the following prediction loss term:

$$\begin{aligned}\mathcal{L}_{\text{pred}}(\Phi) = \mathbb{E}_{(s,a,\mathbf{r}_{\text{env}},s')\sim\mathcal{D}}\Big[&\|s' - g(z;\varphi_s)\|_2 \\ &+ \big\|\mathbf{r}_{\text{env}} - (g(z;\varphi_1^n), \cdots, g(z;\varphi_{m_n}^n))\big\|_2\Big],\end{aligned} \quad (3)$$

where $\mathcal{D}$ is the replay buffer, $\mathbf{r}_{\text{env}} = \mathbf{R}_n(s,a)$ is the ground-truth reward vector received from the environment. Specifically, only the reward heads $\{\varphi_i^n\}_{i=1}^{m_n}$ corresponding to the objectives $\{k_1^n, \cdots, k_{m_n}^n\}$ of task $\mathcal{M}_n$ are updated when $n > 1$. Empirical results showcase that it does not weaken the expressiveness and accuracy of the model. We also apply the ensemble technique to improve the robustness of the model. More details about model learning are provided in Appendix C.2. With the accurate learned multi-objective reward model, we can recover the lost reward signals to rehearse the agent about previously encountered objectives, alleviating catastrophic forgetting and ensuring the performance on all objectives encountered so far.

4.3 Overall Learning Procedure

Through the incorporation of the technologies mentioned in Section 4.1 and 4.2, we formulate our overall method, **C**ontinual Multi-**O**bjective **Re**inforcement Learning via **Re**ward Model **Re**hearsal (CORE3), which can effectively learn the sequence of multi-objective tasks in a continual manner. In this section, we outline the overall learning procedure of CORE3 in Algorithm 1.

For each learning task $\mathcal{M}_n$, we use a replay buffer to store and only store its transition data (Line 3), obviating the explicit storage of transition data of each encountered task, and ensuring the scalability of our approach. If novel objectives appear in the current task $\mathcal{K}_n$, we dynamically expand the policy network and the multi-objective reward model network (Lines 5 $\sim$ 9). Then, the agent explores $\mathcal{M}_n$, and collect transitions into the replay buffer $\mathcal{D}$. With the collected data, we finetune the reward model so that it can predict the reward of novel objectives (Line 13). Leveraging this reward model, we recover the reward signals of previously encountered objectives, and utilize them to train and rehearse the agent (Line 14). Finally, the policy is optimized based on the reward signals for all encountered objectives, alleviating the catastrophic forgetting phenomenon (Lines 15 $\sim$ 16). To assess the efficacy of CORE3 in addressing the CMORL problem, we evaluate the learned policy on all encountered objectives (Line 18), as elaborated in Section 5.

Algorithm 1 CORE3

Input: Multi-objective task $(\mathcal{M}_1, \mathcal{M}_2, \cdots)$, policy π, multi-objective reward model $\hat{\mathbf{R}}$, and replay buffer $\mathcal{D}$.
Parameter: Q function $\theta = (\xi)$, actor ϕ (for continuous action spaces), and multi-objective reward model $\Phi = (\vartheta, \varphi_s)$.

```
1:  $\mathcal{K}_0^{\text{all}} \leftarrow \emptyset, M_n \leftarrow 0$.
2:  for $n = 1, 2, \cdots$ do
3:      Clear replay buffer $\mathcal{D}$.
4:      $\mathcal{K}_n^{\text{all}} \leftarrow \mathcal{K}_{n-1}^{\text{all}} \cup \mathcal{K}_n, m_n \leftarrow |\mathcal{K}_n|, M_n \leftarrow |\mathcal{K}_n^{\text{all}}|$.
5:      if $\mathcal{K}_n^{\text{all}} \backslash \mathcal{K}_{n-1}^{\text{all}} \neq \emptyset$ then
6:          // Dynamic Network Expansion
7:          Create objective heads $\{\psi_i\}_{i=M_{n-1}+1}^{M_n}$, add to $\theta$.
8:          Create reward heads $\{\varphi_i\}_{i=M_{n-1}+1}^{M_n}$, add to $\Phi$.
9:      end if
10:     while task $\mathcal{M}_n$ has not ended do
11:         Uniformly sample preference $\omega \sim \Delta^{M_n}$.
12:         Execute $\pi(\cdot|\cdot,\omega)$ in task $\mathcal{M}_n$ to collect transitions
            $\{(s,a,\mathbf{r}_{\text{env}},s',\omega)\}$ and append to $\mathcal{D}$.
13:         Update $\Phi = (\vartheta, \{\varphi_i^n\}_{i=1}^{m_n}, \varphi_s)$ of the multi-
            objective reward model $\hat{\mathbf{R}}$ to minimize $\mathcal{L}_{\text{pred}}$ in
            Equation 2. // Reward Model Learning
14:         Sample transitions $\{(s,a,\mathbf{r}_{\text{env}},s',\omega)\} \sim \mathcal{D}$, replace
            $\mathbf{r}_{\text{env}}$ with prediction $\hat{\mathbf{r}}$, as mentioned in Section 4.2.
            // Reward Model Rehearsal
15:         Update $\theta = (\xi, \{\psi_i\}_{i=1}^{M_n})$ of the Q function to mini-
            mize $\mathcal{L}$ in Equation 1. // Policy Learning
16:         (For continuous action spaces) Update actor $\phi$.
17:     end while
18:     Evaluate $\pi$ on all encountered objectives $\mathcal{K}_n^{\text{all}}$.
19: end for
```

5 Experiments

In this section, we conduct experiments to answer the following questions: (1) Can CORE3 better accomplish all encountered objectives compared with the baselines, and alleviate forgetting about previous objectives lacking reward signals (Section 5.2) ? (2) How does CORE3 adapt to new objectives, and learn previous objectives from the perspective of Pareto fronts (Section 5.3) ? (3) How does the hyperparameters influence the performance of CORE3 (Section 5.4) ?

Envs	Metrics	Finetune	CLEAR	EWC	CORE3 (Oracle)	CORE3 (Ours)
FTN	HV (↑)	1.00 ± 0.07	0.85 ± 0.07	1.04 ± 0.07	1.10 ± 0.04	**1.06 ± 0.00**
	SP (↓)	1.00 ± 0.14	0.73 ± 0.02	**0.58 ± 0.06**	0.72 ± 0.01	0.82 ± 0.02
	BWT (↑)	-0.12 ± 0.02	0.00 ± 0.01	-0.14 ± 0.01	0.03 ± 0.00	**0.02 ± 0.01**
Grid	HV (↑)	1.00 ± 0.82	4.21 ± 0.66	2.26 ± 1.34	4.60 ± 0.26	**4.54 ± 0.60**
	SP (↓)	1.00 ± 0.58	**0.42 ± 0.11**	0.47 ± 0.10	0.37 ± 0.02	0.44 ± 0.10
	BWT (↑)	-0.44 ± 0.07	**-0.14 ± 0.17**	-0.58 ± 0.05	-0.07 ± 0.12	-0.17 ± 0.15
Ant	HV (↑)	1.00 ± 0.42	1.78 ± 0.37	1.03 ± 0.60	2.30 ± 0.18	**1.89 ± 0.35**
	SP (↓)	1.00 ± 0.56	**0.33 ± 0.06**	1.43 ± 1.78	1.06 ± 0.57	0.73 ± 0.72
	BWT (↑)	-0.27 ± 0.02	0.01 ± 0.18	-0.35 ± 0.1	0.34 ± 0.18	**0.11 ± 0.16**
Hopper	HV (↑)	1.00 ± 1.22	2.03 ± 3.49	1.01 ± 1.43	7.84 ± 0.29	**8.23 ± 1.42**
	SP (↓)	1.00 ± 0.76	7.59 ± 2.98	4.76 ± 3.54	0.35 ± 0.11	**0.21 ± 0.10**
	BWT (↑)	-0.40 ± 0.07	-0.38 ± 0.22	-0.37 ± 0.07	0.11 ± 0.09	**0.06 ± 0.05**
API (%) (↑)	HV	\	121.72	33.61	296.03	**293.18**
	SP	\	−126.81	−80.85	37.61	**45.37**
	BWT	\	69.09	−17.87	145.36	**109.09**

Table 1: Metric values are presented as mean ± std for different methods across four CMORL benchmarks. For simplicity, we re-scale HV and SP values by taking the Finetune's results as an anchor and present the average performance improvement (API, %) relative to it. The best results except the Oracle baseline in each row is highlighted in **bold**. ↑ indicates bett er performance with higher values, and ↓ the opposite.

5.1 Benchmarks, Baselines, and Metrics

To provide a testbed for CORE3, we first design four CMORL benchmarks with both discrete and continuous action spaces, by extending widely-used MORL benchmarks into continual learning settings. The first benchmark is **Fruit Tree Navigation (FTN)** [Yang *et al.*, 2019], a task with a full binary tree, and an agent navigating from the root node to one of the leaf node to receive a 6-dimension reward, representing the amount of six different nutrients. Next, inspired by the four-room [Alegre *et al.*, 2022] environment, we design a more complicated **Grid** world benchmark with five goal positions. The agent is rewarded when navigating closer to the goals, and should navigate to the optimal position given a preference on the goals. Both FTN and Grid are environments with discrete action spaces, and we design two continuous control CMORL benchmarks based on MuJoCo physics engine [Todorov *et al.*, 2012; Xu *et al.*, 2020]. In the **Ant** benchmark, the agent controls the ant robot to complete five objectives, including saving energy and moving in $\{\pm x, \pm y\}$-axes. Another MuJoCo benchmark is **Hopper**, where the five objectives of the hopper robot include saving energy, moving forward, moving backward, jumping high, and staying low. In the CMORL setting, the agent learns a sequence of five MOMDPs. Each MOMDP is trained for 100k steps in FTN and 500k steps otherwise, with two objectives drawn from the objectives mentioned above. More details about the benchmarks are in Appendix B.1.

Next, we compare CORE3 with multiple strong baselines. To evaluate the impact of reward model rehearsal on the agent's learning process, we compare CORE3 with **Finetune**, which can be seen as an ablation of CORE3 without reward model rehearsal, and directly tunes the policy on the current task, ignoring previous objectives lacking reward signals. Additionally, we extend representative works in single-objective CRL into multi-objective settings as baselines. **CLEAR** [Rolnick *et al.*, 2019] is a task-agnostic method which stores transitions of all previous tasks and use them to rehearse the agent. **EWC** [Kirkpatrick *et al.*, 2017] adds an l_2-distance-based regularizer to constrain the update of the agent network. We further investigate **CORE3 (Oracle)**, representing a performance upper bound of CORE3, as it has access to the rewards of previous objectives, i.e., CORE3 with ground-truth multi-objective reward model. We run each method for five distinct seeds. More details about the baselines are in Appendix B.2.

To evaluate the methods' performance on completing all encountered objectives, we randomly sample multiple preferences to test the agent, deriving a set V of all non-dominated return vectors, i.e., the approximated Pareto front. To measure the quality of V, we adopt two metrics widely used in the MORL literature [Hayes *et al.*, 2022]. The first one is **Hypervolume (HV)** $:= \int_{H(V)} \mathbb{I}\{z \in H(V)\}dz$, where $H(V) := \{z \in \mathbb{R}^m : \exists \boldsymbol{v} \in V, \boldsymbol{v} \succ z \succ \boldsymbol{v}_0\}$, m is the number of objectives, $\boldsymbol{v}_0$ is a predefined reference point, $\succ$ is the Pareto dominance relation, and $\mathbb{I}$ is the indicator function. A larger HV indicates a better approximation of the optimal Pareto front. The second one **Sparsity (SP)** $:= \frac{1}{|V|-1}\sum_{j=1}^{m}\sum_{i=1}^{|V|-1}(\tilde{V}_j(i) - \tilde{V}_j(i+1))^2$ measures the density of V, where $\tilde{V}_j(i)$ is the i-th value in the sorted list for the j-th objective values in V. Given two Pareto fronts with close HV, the one with smaller SP is denser and considered better. As we study a continual learning problem, it is necessary to measure the forgetting phenomenon. We therefore calculate **Backward Transfer (BWT)** [Wang *et al.*, 2023] $:= \frac{1}{N-1}(\sum_{n=2}^{N}\frac{1}{n-1}\sum_{j=1}^{n-1}[(\alpha_n^j - \alpha_j^j)/\alpha_j^j]$, where $N = 5$ is the number of tasks, α_i^j is the HV tested on objectives $\mathcal{K}_j$ after training on task $\mathcal{M}_i$ with objectives $\mathcal{K}_i$. BWT evaluates the influence of learning new objectives on completing the previous ones. A larger BWT indicates a better anti-forgetting.

5.2 Competitive Results

In this section, we compare CORE3 with the baselines on the four CMORL benchmarks. The overall results on all three metrics are shown in Table 1, and the learning curves on the primary metric, HV of all objectives encountered so

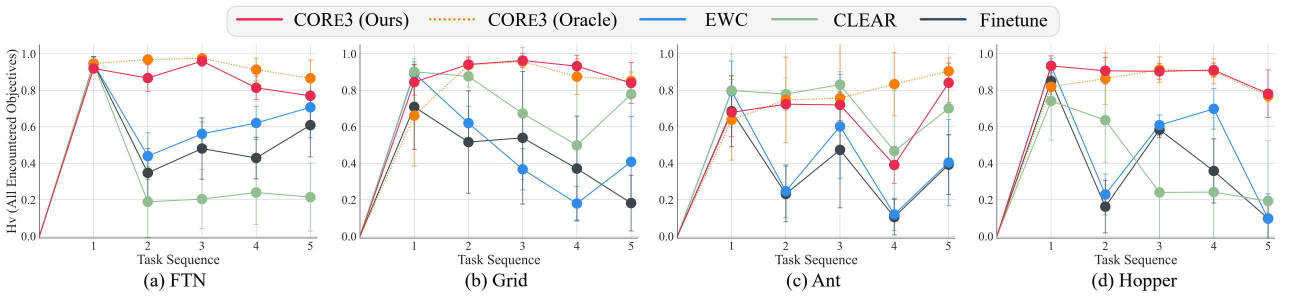


Figure 2: Performance comparison during the learning process on metric HV in four benchmarks. Each benchmark has a sequence of five MORL tasks, and each plot denotes the HV value on all objectives encountered so far. As the numerical range of HV varies greatly with the number of objectives, we re-scale the HV values in each task to the range of $[0, 1]$ for a comprehensive visualization.

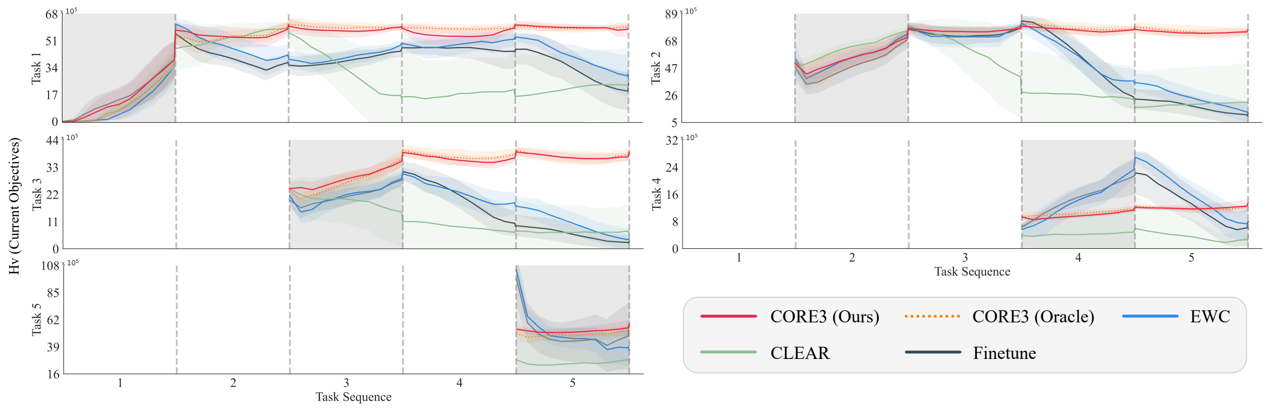


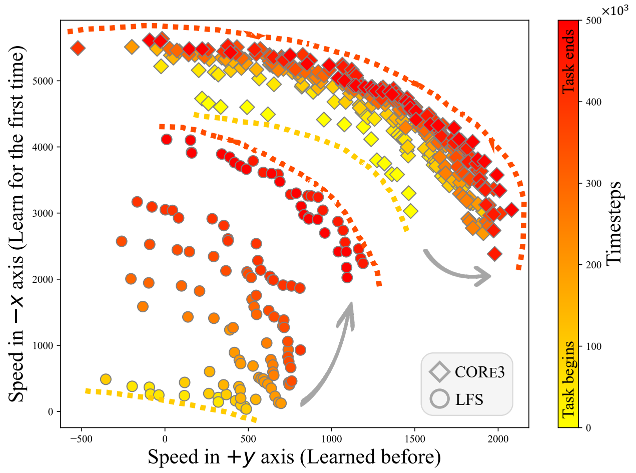
Figure 3: The learning curves of HV on *each* encountered task in the Hopper benchmark. Each of the five plots corresponds to a specific task, and a gray background within each plot signifies the period during which the agent is training on that particular task. There is blank space in the plots for task $n > 1$ because the task has not yet appeared. The results on other benchmarks are provided in Appendix D.1.

far, are displayed in Figure 2. We find that Finetune has the most inferior overall performance on HV and suffers from forgetting according to a low BWT, showing that ignoring the lost reward signals cannot perform well in CMORL settings, even with a well-designed multi-head policy network. To alleviate forgetting, CLEAR uses transitions of previous tasks to rehearse the agent, but still suffers from performance degradation, because the transitions trading off objectives in different tasks are still lacking. Another widely-used approach for single-objective CRL, EWC, also cannot perform well, demonstrating the necessity of specific considerations for multi-objective scenarios. Having access to ground-truth reward signals for all encountered objectives, CORE3 (Oracle) can be seen as an upper bound of performance, and achieves the best overall results, demonstrating the reward rehearsal strategy can solve the CMORL problem while conventional CRL approaches fail. Our approach CORE3, achieves comparable performance to Oracle on all four benchmarks and all three metrics, and outperforms the best baseline CLEAR on the average HV improvement by $293.18\% - 121.72\% = 171.46\%$, indicating the effectiveness of reward model rehearsal.

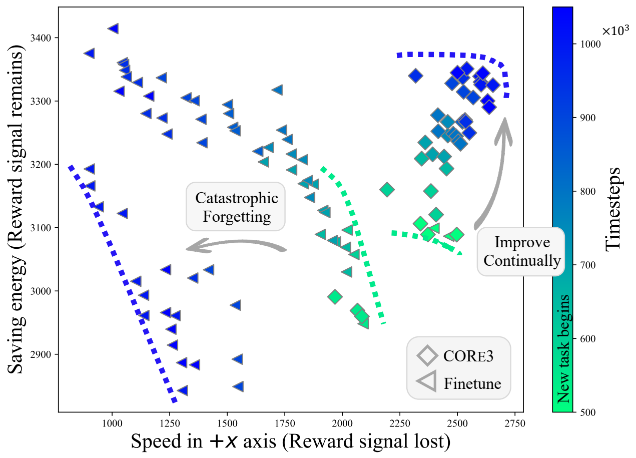
We further display the learning curves of HV on *each* encountered task in the Hopper benchmark in Figure 3. We observe the catastrophic forgetting phenomenon, where baselines suffer from performance degradation on tasks after training on them. Take task 1 as an example, all methods display similar learning curves when training on it, but baselines Finetune, EWC, and CLEAR suffer from performance degradation after starting to train on task 2 or task 3. On the contrary, CORE3 shows non-decreasing learning curves, which are comparable to CORE3 (Oracle) on every task, indicating that our approach successfully alleviates the catastrophic forgetting phenomenon, and maintains the agent's ability to complete previous objectives. The results on other benchmarks are provided in Appendix D.1.

5.3 Pareto Front Analysis

To provide a more comprehensive illustration of how CORE3 adapts to new tasks and alleviates catastrophic forgetting, we further analyze the learning process by visualizing the Pareto fronts in Figure 4. As shown in Figure 4(a), after learning previous tasks, the CMORL agent is now confronted with a new task with two objectives. One objective (moving in $+y$ axis) has been learned before, and the other one (moving in $-x$ axis) is new. To provide reference and comparison, we introduce Learning-From-Scratch (LFS), which trains a randomly initialized policy network to complete these two objectives from scratch, and expands the Pareto front stating from a position near $(0, 0)$. The markers with different colors in the figure (● $\rightarrow$ ●) demonstrate the progression of the learning process. We then display the learning process of CORE3 in the same manner (◆ $\rightarrow$ ◆). At the early beginning of the task, CORE3 already achieves a Pareto front that surpasses



(a) CORE3 v.s. LFS on learning a new task.



(b) CORE3 v.s. Finetune on an ended task.

Figure 4: Comparison of Pareto fronts on the Ant benchmark. The x, y-axes represent the returns of two different objectives. Each marker is a found solution, and markers in the same color form an approximation of the optimal Pareto front. Different colors represent different training periods. (a) CORE3 v.s. Learning-From-Scratch (LFS) on learning a new task. LFS trains a random initialized policy network from scratch. (b) CORE3 v.s. Finetune on a task whose training period has ended. The displayed training period begins at 500k steps (lower right corner) instead of 0, indicating the solutions are induced by policies learning subsequent tasks with objectives different from the ones in x, y-axes.

the one obtained by LFS at the task's conclusion. It indicates that, by learning from previous objectives, CORE3 has acquired fundamental and transferable task knowledge, like controlling the joints of the Ant robot in a reasonable way. Armed with this knowledge and the continual learning of the new task, CORE3 further effectively expands the Pareto front to a broader and denser curve, better approximating the optimal Pareto front of the task.

Next, we display the Pareto fronts of a task after training on it. As shown in Figure 4(b), the reward signal of one objective (moving in $+x$ axis) is lost in the subsequent task while the other remains. Without specific consideration, Finetune suffers from a significant performance degradation about 50% of the objective lacking reward signal (◀ $\rightarrow$ ◀), i.e., catastrophic forgetting. On the contrary, with the reward model rehearsal technique, CORE3 retains and continually improves the ability of completing this objective (◆ $\rightarrow$ ◆), explaining how CORE3 successfully alleviates catastrophic forgetting.

5.4 Sensitivity Studies

The performance of CORE3 significantly depends on the quality of the multi-objective reward model, which is learned through established world model learning techniques [Luo *et al.*, 2022]. As the learning process includes multiple hyperparameters, we here conduct experiments on the Hopper benchmark to investigate how each one influences the performance of CORE3. One important hyperparameter is the ensemble size, as we learn multiple ensemble models to improve robustness. On one hand, a small ensemble size is insufficient for robustness. On the other hand, learning an excessively large number of ensemble models will reduce the overall efficiency. As shown in Figure 5, we find that CORE3 achieves the highest Hv and lowest sparsity when the ensemble size is equal to 5. CORE3 with 10 ensemble models performs similar to the former, meaning that an ensemble size of 5 is enough to learn a robust multi-objective reward model. When the ensemble size is less than 5, the reward model is not stable enough, resulting in worse performance. More detailed analysis of other important hyperparameters like the batch size of model learning is provided in Appendix D.2.

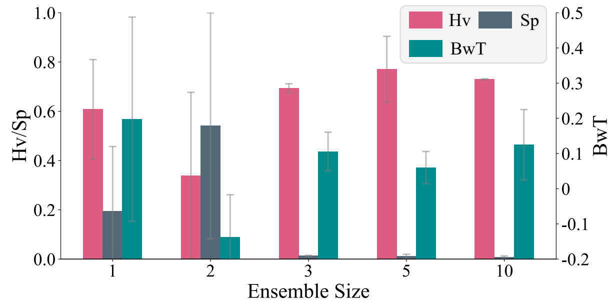


Figure 5: Sensitivity studies on model ensemble size in Hopper.

6 Final Remarks

Recognizing the significance of continual learning, this work takes a further step towards continual multi-objective RL (CMORL). We formulate this problem for the first time, where learning objectives may evolve throughout the agent's lifetime. Then, we propose **C**ontinual Multi-**O**bjective **Re**inforcement Learning via **Re**ward Model **Re**hearsal (CORE3), which utilizes a dynamic agent network for adaptation to new objectives, and a multi-objective reward model to rehearse the agents about objectives lacking reward signals. Experiments demonstrate the effectiveness of our approach. Furthermore, an intriguing avenue for future exploration is extending multi-objective RL to the domain of multi-agent reinforcement learning (MARL) [Yuan *et al.*, 2023], thereby fostering progress in both these vital fields.

Acknowledgments

This work is supported by National Science Foundation of China (61921006). We would like to thank all reviewers for their valuable suggestions.

References

[Abels *et al.*, 2019] Axel Abels, Diederik M. Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *ICML*, pages 11–20, 2019.

[Alegre *et al.*, 2022] Lucas N. Alegre, Ana L. C. Bazzan, and Bruno C. da Silva. Optimistic linear support and successor features as a basis for optimal policy transfer. In *ICML*, pages 394–413, 2022.

[Anonymous, 2024] Anonymous. CPPO: Continual learning for reinforcement learning with human feedback. In *ICLR*, 2024.

[Basaklar *et al.*, 2023] Toygun Basaklar, Suat Gumussoy, and Ümit Y. Ogras. PD-MORL: preference-driven multi-objective reinforcement learning algorithm. In *ICLR*, 2023.

[Chang *et al.*, 2021] Yifan Chang, Wenbo Li, Jian Peng, Bo Tang, Yu Kang, Yinjie Lei, Yuanmiao Gui, Qing Zhu, Yu Liu, and Haifeng Li. Reviewing continual learning from the perspective of human-level intelligence. *arXiv preprint arXiv:2111.11964*, 2021.

[Chen *et al.*, 2019] Xi Chen, Ali Ghadirzadeh, M**r**arten Björkman, and Patric Jensfelt. Meta-learning for multi-objective reinforcement learning. In *IROS*, pages 977–983, 2019.

[Chen *et al.*, 2021] SenPeng Chen, Jia Wu, and XiYuan Liu. Emorl: Effective multi-objective reinforcement learning method for hyperparameter optimization. *Engineering Applications of Artificial Intelligence*, 104:104315, 2021.

[Chung *et al.*, 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[Felten *et al.*, 2023] 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 *NeurIPS*, 2023.

[Gaya *et al.*, 2023] Jean-Baptiste Gaya, Thang Doan, Lucas Caccia, Laure Soulier, Ludovic Denoyer, and Roberta Raileanu. Building a subspace of policies for scalable continual learning. In *ICLR*, 2023.

[Hayes *et al.*, 2022] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022.

[Huang *et al.*, 2021] Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *ICRA*, pages 799–805, 2021.

[Hwang *et al.*, 2023] Minyoung Hwang, Luca Weihs, Chanwoo Park, Kimin Lee, Aniruddha Kembhavi, and Kiana Ehsani. Promptable behaviors: Personalizing multi-objective rewards from human preferences. *arXiv preprint arXiv:2312.09337*, 2023.

[Kaufmann *et al.*, 2023] Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. A survey of reinforcement learning from human feedback. *arXiv preprint arXiv:2312.14925*, 2023.

[Kessler *et al.*, 2022] Samuel Kessler, Jack Parker-Holder, Philip J. Ball, Stefan Zohren, and Stephen J. Roberts. Same state, different task: Continual reinforcement learning without interference. In *AAAI*, pages 7143–7151, 2022.

[Kessler *et al.*, 2023] Samuel Kessler, Mateusz Ostaszewski, Michał Bortkiewicz, Mateusz Żarski, Maciej Wołczyk, Jack Parker-Holder, Stephen J. Roberts, and Piotr Miłoś. The effectiveness of world models for continual reinforcement learning. In *CoLLAs*, pages 184–204, 2023.

[Khetarpal *et al.*, 2022] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

[Kiran *et al.*, 2022] B. Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Kumar Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2022.

[Kirkpatrick *et al.*, 2017] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[Liu *et al.*, 2014] Chunming Liu, Xin Xu, and Dewen Hu. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, 2014.

[Luo *et al.*, 2022] Fan-Ming Luo, Tian Xu, Hang Lai, Xiong-Hui Chen, Weinan Zhang, and Yang Yu. A survey on model-based reinforcement learning. *SCIENCE CHINA Information Sciences*, 2022.

[Pan *et al.*, 2020] Anqi Pan, Wenjun Xu, Lei Wang, and Hongliang Ren. Additional planning with multiple objectives for reinforcement learning. *Knowledge-Based Systems*, 193:105392, 2020.

[Parisi *et al.*, 2019] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter.

Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[Powers *et al.*, 2022] Sam Powers, Eliot Xing, Eric Kolve, Roozbeh Mottaghi, and Abhinav Gupta. Cora: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. In *CoLLAs*, pages 705–743, 2022.

[Ren *et al.*, 2021] Tao Ren, Jianwei Niu, Jiahe Cui, Zhenchao Ouyang, and Xuefeng Liu. An application of multi-objective reinforcement learning for efficient model-free control of canals deployed with iot networks. *Journal of Network and Computer Applications*, 182:103049, 2021.

[Ring, 1995] Mark B. Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin, TX, USA, 1995.

[Roijers *et al.*, 2013] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[Rolnick *et al.*, 2019] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *NeurIPS*, pages 350–360, 2019.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Shaheen *et al.*, 2022] Khadija Shaheen, Muhammad Abdullah Hanif, Osman Hasan, and Muhammad Shafique. Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks. *Journal of Intelligent & Robotic Systems*, 105(1):9, 2022.

[Siddique *et al.*, 2020] Umer Siddique, Paul Weng, and Matthieu Zimmer. Learning fair policies in multiobjective (deep) reinforcement learning with average and discounted rewards. In *ICML*, pages 8905–8915, 2020.

[Singh *et al.*, 2022] Bharat Singh, Rajesh Kumar, and Vinay Pratap Singh. Reinforcement learning in robotic applications: a comprehensive survey. *Artificial Intelligence Review*, 55(2):945–990, 2022.

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033, 2012.

[Wang *et al.*, 2022] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[Wang *et al.*, 2023] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*, 2023.

[Wolczyk *et al.*, 2021] Maciej Wolczyk, Michal Zajac, Razvan Pascanu, Lukasz Kucinski, and Piotr Milos. Continual world: A robotic benchmark for continual reinforcement learning. In *NeurIPS*, pages 28496–28510, 2021.

[Xu *et al.*, 2020] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *ICML*, pages 10607–10616, 2020.

[Yang *et al.*, 2019] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *NeurIPS*, page 14636–14647, 2019.

[Yuan *et al.*, 2023] Lei Yuan, Ziqian Zhang, Lihe Li, Cong Guan, and Yang Yu. A survey of progress on cooperative multi-agent reinforcement learning in open environment. *arXiv preprint arXiv:2312.01058*, 2023.

[Zhang *et al.*, 2023a] Tiantian Zhang, Zichuan Lin, Yuxing Wang, Deheng Ye, Qiang Fu, Wei Yang, Xueqian Wang, Bin Liang, Bo Yuan, and Xiu Li. Dynamics-adaptive continual reinforcement learning via progressive contextualization. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[Zhang *et al.*, 2023b] Zizhen Zhang, Zhiyuan Wu, Hang Zhang, and Jiahai Wang. Meta-learning-based deep reinforcement learning for multiobjective optimization problems. *IEEE Transactions on Neural Networks and Learning Systems*, 34(10):7978–7991, 2023.

[Zhu *et al.*, 2023] Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):13344–13362, 2023.

Continual Multi-Objective Reinforcement Learning via Reward Model Rehearsal: Appendix

A Proof of Theorem 1

Definition 1 (Occupancy Measure). *We first define the occupancy measure $\rho^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of a policy π as follows:*

$$\rho^\pi(s,a) = (1-\gamma)\pi(a|s,\omega)\sum_{t=0}^{\infty}\gamma^t \Pr(s_t = s|\pi). \tag{1}$$

Lemma 1. *Let $\tau = (s_0, a_0, s_1, a_1, \cdots)$ denote a trajectory. Given a policy π and a MOMDP $\mathcal{M}$, the value vector of π w.r.t. $\mathcal{M}$, denoted as $J_\mathcal{M}(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^t \mathbf{R}(s_t,a_t)]$, can be rewritten as follows:*

$$J_\mathcal{M}(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim\rho^\pi}[\mathbf{R}(s,a)]. \tag{2}$$

Proof.

$$\begin{aligned}
J_\mathcal{M}(\pi) &= \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^t \mathbf{R}(s_t,a_t)] \\
&= \sum_{\tau}\Pr(\tau)\sum_{t=0}^{\infty}\gamma^t \mathbf{R}(s_t,a_t) \\
&= \sum_{(s,a)}\sum_{t=0}^{\infty}\gamma^t\left(\sum_{\tau:(s_t,a_t)=(s,a)}\Pr(\tau)\right)\mathbf{R}(s_t,a_t) \\
&= \sum_{(s,a)}\left[\sum_{t=0}^{\infty}\gamma^t \Pr(s_t=s|\pi)\pi(a|s)\right]\mathbf{R}(s_t,a_t) \\
&= \sum_{(s,a)}\frac{1}{1-\gamma}\rho^\pi(s,a)\mathbf{R}(s_t,a_t) \\
&= \frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim\rho^\pi}[\mathbf{R}(s,a)].
\end{aligned}$$

□

Theorem 1 (Bounded Performance Gap). *Consider two MOMDPs, $\mathcal{M}$ and $\hat{\mathcal{M}}$, differing only in their reward functions: $\mathbf{R} = (R_1, \cdots, R_m)$ and $\hat{\mathbf{R}} = (\hat{R}_1, \cdots, \hat{R}_m)$. The value vectors of a policy π in $\mathcal{M}$ and $\hat{\mathcal{M}}$ are $J_\mathcal{M}(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^t\mathbf{R}(s_t,a_t)]$ and $J_{\hat{\mathcal{M}}}(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty}\gamma^t\hat{\mathbf{R}}(s_t,a_t)]$, respectively. The optimal policies for $\mathcal{M}$ and $\hat{\mathcal{M}}$ are denoted as π^* and $\hat{\pi}^*$, satisfying $\forall\omega \in \Delta^m, \pi^* = \arg\max_\pi \omega^\top J_\mathcal{M}(\pi)$, and $\hat{\pi}^* = \arg\max_\pi \omega^\top J_{\hat{\mathcal{M}}}(\pi)$. Define the difference between these reward functions as $\delta = \sup_{s\in\mathcal{S},a\in\mathcal{A}}||\mathbf{R}(s,a) - \hat{\mathbf{R}}(s,a)||_\infty$. Based on these definitions, the following inequality holds,*

$$\forall\omega \in \Delta^m, \omega^\top J_\mathcal{M}(\pi^*) - \omega^\top J_\mathcal{M}(\hat{\pi}^*) \leq \frac{2\delta}{1-\gamma}. \tag{3}$$

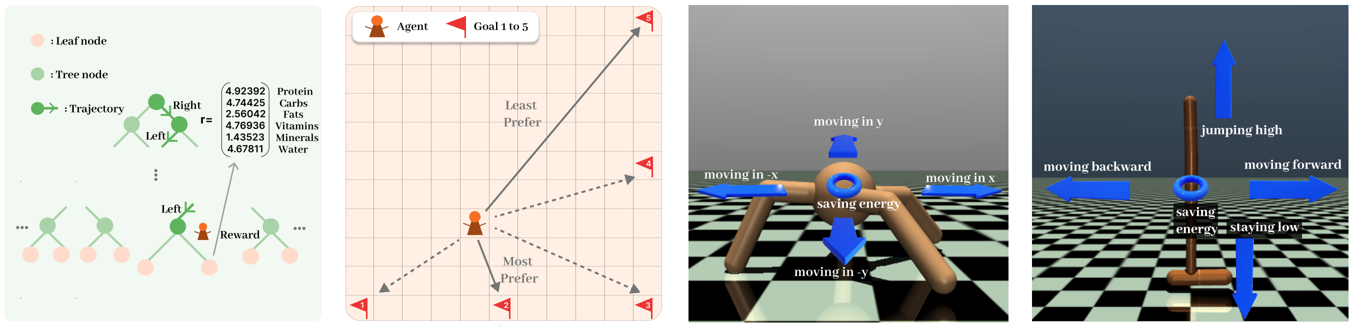
Proof. First, for a given policy π, and $\forall\omega$,

$$\begin{aligned}
&|\omega^\top J_\mathcal{M}(\pi) - \omega^\top J_{\hat{\mathcal{M}}}(\pi)| \\
=&|\omega^\top\frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim\rho^\pi}[\mathbf{R}(s,a)] - \omega^\top\frac{1}{1-\gamma}\mathbb{E}_{(s,a)\sim\rho^\pi}[\hat{\mathbf{R}}(s,a)]| \\
=&\frac{1}{1-\gamma}|\mathbb{E}_{(s,a)\sim\rho^\pi}[\omega^\top\mathbf{R}(s,a)] - \mathbb{E}_{(s,a)\sim\rho^\pi}[\omega^\top\hat{\mathbf{R}}(s,a)]| \\
=&\frac{1}{1-\gamma}|\sum_{(s,a)}\rho^\pi(s,a)\omega^\top\mathbf{R}(s,a) - \sum_{(s,a)}\rho^\pi(s,a)\omega^\top\hat{\mathbf{R}}(s,a)| \\
\leq&\frac{1}{1-\gamma}\sum_{(s,a)}\rho^\pi(s,a)|\omega^\top(\mathbf{R}(s,a) - \hat{\mathbf{R}}(s,a))| \\
\leq&\frac{1}{1-\gamma}\sum_{(s,a)}\rho^\pi(s,a)\delta \\
=&\frac{\delta}{1-\gamma}.
\end{aligned}$$

Based on this inequality, and the optimality of $\hat{\pi}^*$ w.r.t. MOMDP $\hat{\mathcal{M}}$: $\forall\pi, \forall\omega \in \Delta^m, \omega^\top J_{\hat{\mathcal{M}}}(\hat{\pi}^*) \geq \omega^\top J_{\hat{\mathcal{M}}}(\pi)$, the following inequality holds: $\forall\omega \in \Delta^m$,

$$\begin{aligned}
&\omega^\top J_\mathcal{M}(\pi^*) - \omega^\top J_\mathcal{M}(\hat{\pi}^*) \\
=&\omega^\top J_\mathcal{M}(\pi^*) - \omega^\top J_{\hat{\mathcal{M}}}(\pi^*) + \omega^\top J_{\hat{\mathcal{M}}}(\pi^*) - \omega^\top J_{\hat{\mathcal{M}}}(\hat{\pi}^*) + \\
&\omega^\top J_{\hat{\mathcal{M}}}(\hat{\pi}^*) - \omega^\top J_\mathcal{M}(\hat{\pi}^*) \\
\leq&|\omega^\top J_\mathcal{M}(\pi^*) - \omega^\top J_{\hat{\mathcal{M}}}(\pi^*)| - (\omega^\top J_{\hat{\mathcal{M}}}(\hat{\pi}^*) - \omega^\top J_{\hat{\mathcal{M}}}(\pi^*)) + \\
&|\omega^\top J_{\hat{\mathcal{M}}}(\hat{\pi}^*) - \omega^\top J_\mathcal{M}(\hat{\pi}^*)| \\
\leq&\frac{\delta}{1-\gamma} - 0 + \frac{\delta}{1-\gamma} \\
=&\frac{2\delta}{1-\gamma}.
\end{aligned}$$

□



(a) FTN (b) Grid (c) Ant (d) Hopper

Figure 1: Benchmarks used in this paper.

B Details about Benchmarks and Baselines

B.1 Benchmarks

Below, we provide a description of each benchmark used in our experiments. Figure 1 presents a visualization of each benchmark. For discrete action spaces, we provide **Fruit Tree Navigation (FTN)** [Yang *et al.*, 2019] and **Grid** for performance comparison. For continuous control settings, we extend **Ant** and **Hopper** benchmarks to five objectives based on MuJoCo physics engine [Todorov *et al.*, 2012; Xu *et al.*, 2020] to satisfy the CMORL setting.

FTN

The first benchmark is a full binary tree of depth $d = 7$ with randomly assigned reward vectors $\mathbf{r} \in \mathbb{R}^6$ on the leaf nodes. Each reward signal in $\mathbf{r}$ represents one of the six nutrients $\{$Protein, Carbs, Fats, Vitamins, Minerals, Water$\}$. We use the same nutrients' value as defined by [Yang *et al.*, 2019]. The reward function of this benchmark is defined as:

$$r_i(s, a, s') = \text{value of nutrient } i \text{ in } s', \text{ for } i = 1 \ldots 6.$$

The agent should travel from the root node to one of the leaf nodes in a correct path, picking a desired fruit based on the given preference on the nutrients. At each time step, the agent observes its current state (row, column), indicating its current location on the tree. It then takes action among $\mathcal{A} = \{\text{left}, \text{right}\}$ to move to the left or right subtree at every non-leaf node. For the CMORL setting, we assign the six objectives into five tasks in a continual manner:

$$\{\text{Protein}, \text{Carbs}\} \to \{\text{Carbs}, \text{Fats}\} \to \{\text{Fats}, \text{Vitamins}\} \to \{\text{Vitamins}, \text{Minerals}\} \to \{\text{Minerals}, \text{Water}\},$$

and each task is trained for 100k steps.

Grid

The second benchmark is a 11×11 grid world with the state space defined as $\mathcal{S} = \{0, \cdots, 10\}^2$, indicating the agent's current position. There are five goals distributed in this environment, representing five different objectives and their coordinates are $(10, 0), (10, 5), (10, 10), (5, 10), (0, 10)$. The agent begins at position $(0, 0)$ and chooses action from $\{$None, North, South, West, East$\}$ to stay still or move towards the corresponding direction for one cell. The reward for each objective will increase if the Manhattan distance D between the agent and the related goal decreases. The multi-objective reward function of this benchmark is defined as:

$$r_i(s, a, s') = 20 - D(\text{agent's position}, \text{goal}_i\text{'s position}).$$

Using the coordinates to denote objectives, the task sequence can be shown as below:

$$\{(10, 0), (10, 5)\} \to \{(10, 5), (10, 10)\} \to \{(10, 10), (5, 10)\} \to \{(5, 10), (0, 10)\} \to \{(0, 10), (10, 0)\},$$

and each task is trained for 500k steps.

Ant

The third benchmark is a multi-objective version of the MuJoCo's Ant-v2 environment. The state space $\mathcal{S} \subset \mathbb{R}^{27}$ consists of the positional values and velocities of each part of the ant. An action represents the torques applied at the hinge joints, $\mathcal{A} = [-1, 1]^8$. In the multi-objective version in [Xu *et al.*, 2020], the agent should balance the speed towards the positive direction of x-axis and y-axis. In our benchmark, we extend the objectives to keeping a large speed towards positive/negative direction of x-axis/y-axis and saving energy. The multi-objective reward function of this benchmark is:

$$\begin{aligned}
r_1(s, a, s') &= \text{velocity of the agent in the } +x\text{-axis direction},\\
r_2(s, a, s') &= \text{velocity of the agent in the } +y\text{-axis direction},\\
r_3(s, a, s') &= \text{velocity of the agent in the } -x\text{-axis direction},\\
r_4(s, a, s') &= \text{velocity of the agent in the } -y\text{-axis direction},\\
r_5(s, a, s') &= 5 - \|a\|_2^2.
\end{aligned}$$

The task sequence is:

$$\{+x\text{-axis}, \text{saving energy}\} \to \{\text{saving energy}, +y\text{-axis}\} \to \{+y\text{-axis}, -x\text{-axis}\} \to \{-x\text{-axis}, -y\text{-axis}\} \to \{-y\text{-axis}, +x\text{-axis}\},$$

and each task is trained for 500k steps.

Hopper

The fourth benchmark is a multi-objective version of the MuJoCo's Hopper-v2 environment. The state space $\mathcal{S} \subset \mathbb{R}^{11}$ encodes the positional values and velocities of different body parts of the hopper. An action contains the torques applied at the thigh joint, leg joint and foot joint and the action space is $\mathcal{A} = [-1, 1]^3$. There are two objectives of the multi-objective version in [Xu *et al.*, 2020]: keeping a large speed towards the positive direction of x-axis and jumping as high as possible. We extend the objectives to keeping a large speed towards the positive/negative direction of x-axis, saving energy, jumping high and staying low. The multi-objective reward function of this benchmark is define as:

$$\begin{aligned}
r_1(s,a,s') &= \text{velocity of the agent in the } x\text{-axis direction},\\
r_2(s,a,s') &= \text{height of the agent over the } z\text{-axis direction},\\
r_3(s,a,s') &= \text{velocity of the agent in the } -x\text{-axis direction},\\
r_5(s,a,s') &= -(\text{height of the agent over the } z\text{-axis direction}),\\
r_4(s,a,s') &= 5 - \|a\|_2^2.
\end{aligned}$$

The task sequence is shown below:

$$\{+x\text{-axis, saving energy}\} \rightarrow \{\text{saving energy, jumping high}\} \rightarrow \{\text{jumping high}, -x\text{-axis}\} \rightarrow \{-x\text{-axis, staying low}\} \rightarrow \{\text{staying low}, +x\text{-axis}\},$$

and each task is trained for 500k steps.

B.2 Baselines

Finetune can be seen as an ablation of CORE3 without reward model rehearsal. It also has the same network architecture as CORE3. However, without reward model predicting the lost rewards, the agent can only observe the reward signals of the objectives in the current learning task (MOMDP). The feature extractor of the Q network and the actor for continuous action space would change dramatically to acquire good performance on the current task, leading to the phenomenon of catastrophic forgetting.

EWC [Kirkpatrick *et al.*, 2017] is a regularization-based approach to avoid catastrophic forgetting during the continual learning process. It tries to evaluate the importance of the network parameters after learning task $\mathcal{M}_{n-1}$ and slow down the learning on weights that are important for task $\mathcal{M}_{n-1}$. For CMORL setting, we implement it with the same network architecture as CORE3, and restrict the updating of the feature extractor of Q network for discrete action spaces while slowing down the learning on the actor network for continuous action spaces. The loss function of the policy network for discrete action spaces when learning the current task $\mathcal{M}_n$ is:

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathcal{M}_n}(\theta) + \frac{\lambda}{2}\sum_j F_j(\xi_j - \xi_{\mathcal{M}_{n-1},j})^2. \qquad (4)$$

For continuous action spaces, the loss function is:

$$\mathcal{L}(\phi) = \mathcal{L}_{\mathcal{M}_n}(\phi) + \frac{\lambda}{2}\sum_j F_j(\phi_j - \phi_{\mathcal{M}_{n-1},j})^2. \qquad (5)$$

Here, we use $\mathcal{L}_{\mathcal{M}_n}(\theta)$ or $\mathcal{L}_{\mathcal{M}_n}(\phi)$ to denote the original loss of policy network for task $\mathcal{M}_n$, where θ is the parameters of Q network for discrete action spaces and ϕ represents the parameters of the actor network for continuous action spaces. F_j is the j^{th} diagonal element of the Fisher information matrix. $\xi_{\mathcal{M}_{n-1}}$ or $\phi_{\mathcal{M}_{n-1}}$ is the saved snapshot of ξ or ϕ after training task $\mathcal{M}_{n-1}$, and j labels each parameter. λ is an adjustable coefficient to control the trade-off between the current task and previous ones. In this paper, we set $\lambda = 2$.

Clear [Rolnick *et al.*, 2019] is one of the replay-based methods to address the catastrophic forgetting by saving the transitions of all previous tasks and use them to rehearse the agent. When learning the current task, the stored data is sampled for training to keep the knowledge of previous tasks. In this paper, we set the replay buffer to uniformly store transitions of all encountered tasks, including the current one.

CORE3 (Oracle) represents a performance upper bound of CORE3, as it has access to the rewards of previous objectives, i.e., CORE3 with ground-truth multi-objective reward model. The only difference between CORE3 (Oracle) and CORE3 is that we replace the predicted reward signals with the ground-truth ones when training the agent.

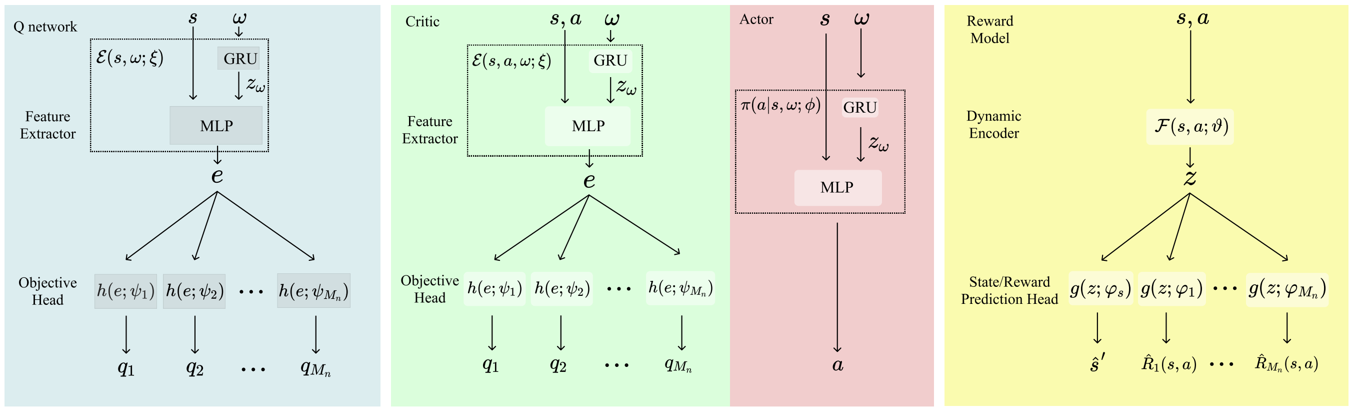
C The Training Details and Hyperparameter Choices of CORE3

C.1 Training Details about MORL

For the purpose of handling the various number of objectives during the whole learning process, facilitating rapid adaptation and circumventing the need to learn the entire network form scratch, we design a dynamic and expandable network architecture as follows.

For discrete action spaces, as shown in Figure 2(a), we learn a Q network $\mathbf{Q}(s,\omega;\theta)$ with an output shape of $(|\mathcal{A}|, M_n)$ to approximate the action-value function and use it as the policy, where $|\mathcal{A}|$ and M_n represent the size of the discrete action spaces and the number of all encountered objectives, respectively. Each row of the output matrix represents a Q value vector of all encountered objectives corresponding to a specific action $a \in \mathcal{A}$ and we use $\mathbf{Q}(s,a,\omega;\theta)$ to denote it for simplicity. The Q network is initialized as a feature extractor $\mathcal{E}(s,\omega;\xi)$ parameterized with ξ, which is a multilayer perceptron with four hidden layers, and each layer contains 512 neurons and the activation function is the Rectified Linear Unit (ReLU) [Agarap, 2018]. When new objectives arrive, we dynamically create an objective head $h(e;\psi_i)$ for each novel objective, which is a linear layer and much smaller than the feature extractor. Due to the increase in the number of all encountered objectives, the length of preference vectors ω also varies along with the task sequence. To handle this problem, we use a GRU cell [Chung *et al.*, 2014] to encode ω into an embedding z_ω with a fixed-length of 8, and concatenate it with the state s as the input of the feature extractor in practical implementation. After that, the feature extractor outputs feature $e = \mathcal{E}(s,\omega;\xi)$ and then each head $h(e;\psi_i)$ takes e as input, outputting the Q value vectors of all possible actions corresponding to objective k_i, which forms the Q matrix $\big(h(e;\psi_1),\cdots,h(e;\psi_{M_n})\big) = \big(\mathbf{Q}(s,a_1,\omega;\theta);\cdots;\mathbf{Q}(s,a_{|\mathcal{A}|},\omega;\theta)\big) = \mathbf{Q}(s,\omega;\theta) \in \mathbb{R}^{|\mathcal{A}|\times M_n}$ for all possible actions and encountered objectives. With the Q matrix, the agent is able to select the optimal action $a = \arg\max_{\tilde{a}\in\mathcal{A}} \omega^\top \mathbf{Q}(s,\tilde{a},\omega;\theta)$ w.r.t. the given preference ω. In the main body of the paper, we omit the matrix part and incorporate the action a as part of the feature extractor's input to provide a more comprehensive illustration.

At the beginning of our algorithm, we first initialize the transition buffer $\mathcal{D}$, the Q network $\mathbf{Q}(s,\omega;\theta)$, the target Q


Q network
s
ω
$\mathcal{E}(s,\omega;\xi)$
GRU
z_ω
Feature Extractor
MLP
e
Objective Head
$h(e;\psi_1)$ $h(e;\psi_2)$ $\cdots$ $h(e;\psi_{M_n})$
q_1 q_2 $\cdots$ q_{M_n}
Critic
s, a
ω
$\mathcal{E}(s,a,\omega;\xi)$
GRU
z_ω
Feature Extractor
MLP
e
Objective Head
$h(e;\psi_1)$ $h(e;\psi_2)$ $\cdots$ $h(e;\psi_{M_n})$
q_1 q_2 $\cdots$ q_{M_n}
Actor
s
ω
$\pi(a|s,\omega;\phi)$
GRU
z_ω
MLP
a
Reward Model
s, a
Dynamic Encoder
$\mathcal{F}(s,a;\vartheta)$
z
State/Reward Prediction Head
$g(z;\varphi_s)$ $g(z;\varphi_1)$ $\cdots$ $g(z;\varphi_{M_n})$
$\hat{s}'$ $\hat{R}_1(s,a)$ $\cdots$ $\hat{R}_{M_n}(s,a)$

(a) Discrete

(b) Continuous

(c) Reward Model

Figure 2: The detailed network architecture of CORE3.

network $\mathbf{Q}(s,\omega;\theta^-)$, and the reward model $\hat{\mathbf{R}}$. During task $\mathcal{M}_n$, we first expand our Q network, target Q network and reward model if novel objectives arrive. Then, for each episode, we randomly sample a preference vector ω from the preference space $\Omega = \Delta^{M_n}$, where M_n denotes the number of all encountered objectives and Ω is a unit simplex. Then the agent interacts with the environment using the ϵ-greedy exploration strategy and collect transitions $(s, a, \mathbf{r}, s', \omega, \text{done})$. It is worth noting that the reward vector $\mathbf{r} \in \mathbb{R}^{m_n}$ only contains the reward signals of the objectives in the current learning task. For each transition $(s, a, \mathbf{r}, s', \omega, \text{done})$, we also store $N_\omega = 3$ additional transitions $(s, a, \mathbf{r}, s', \omega', \text{done})$ with different preferences randomly sampled from the preference space in transition buffer $\mathcal{D}$. During the updating phase, we sample a minibatch of transitions from $\mathcal{D}$ and for each transition $(s, a, \mathbf{r}, s', \omega, \text{done})$, we use the reward model to predict the full reward vector $\hat{\mathbf{r}} \in \mathbb{R}^{M_n}$ of all encountered objectives. For the objectives learned in the current task $\mathcal{M}_n$, we replace the predicted rewards with the ground-truth ones recorded in the reward vector $\mathbf{r}$. We use $\mathbf{Q}(s,a,\omega;\theta)$ to denote the Q value vector from the Q matrix $\mathbf{Q}(s,\omega;\theta)$ of all encountered objectives w.r.t. the action a. Then, the Q network is optimized to minimize the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,\hat{\mathbf{r}},s',\omega)\sim\mathcal{D}}\left[\|\mathbf{y} - \mathbf{Q}(s,a,\omega;\theta)\|_2\right], \tag{6}$$

where $\mathbf{y} = \hat{\mathbf{r}} + \gamma\mathbf{Q}(s', a', \omega; \theta^-)$ denotes the target value vector which is obtained using target Q network's parameters θ^- and the action $a' = \arg\max_{\tilde{a}\in\mathcal{A}} \omega^\top \mathbf{Q}(s', \tilde{a}, \omega; \theta^-)$. The parameters θ^- of the target network are updated as bellow:

$$\theta^- = (1-\tau)\theta^- + \tau\theta, \tag{7}$$

where $\tau = 0.005$ is the coefficient to control the trade-off between the updated parameters θ and θ^-.

For continuous action spaces, we align with the multi-objective version of TD3 algorithm [Fujimoto *et al.*, 2018; Basaklar *et al.*, 2023]. Specifically, as shown in Figure 2(b), we apply $\mathbf{Q}(s,a,\omega;\theta)$ as the critic and learn an actor $\pi(a|s,\omega;\phi)$. For the critic $\mathbf{Q}(s,a,\omega;\theta)$, we implement a two-hidden-layer MLP feature extractor $\mathcal{E}(s,a,\omega;\xi)$ with 400 neurons per layer for initialization. An MLP objective head $h(e;\psi_i)$ with two hidden layers will be created similar to the discrete action space situation. We also use a GRU cell to deal with the preference ω. Different from the discrete action space, the feature extractor takes the state s, action a and preference embedding z_ω as input and outputs the feature e corresponding to the specific action a. Then, each head $h(e;\psi_i)$ takes e as input and outputs the Q value of objective k_i, which forms the Q vector $\big(h(e;\psi_1), \cdots, h(e;\psi_{M_n})\big) = \mathbf{Q}(s,a,\omega;\theta)$ w.r.t. action a. For the actor $\pi(a|s,\omega;\phi)$, which is also a MLP with two hidden layers, it takes the state s and the preference embedding z_ω as input and outputs the continuous action a directly.

At the beginning of our algorithm, we first initialize the transition buffer $\mathcal{D}$, the actor network $\pi(a|s,\omega;\phi)$, the target actor network $\pi(a|s,\omega;\phi^-)$, the reward model $\hat{\mathbf{R}}$, two critic networks $\mathbf{Q}(s,a,\omega;\theta_1)$, $\mathbf{Q}(s,a,\omega;\theta_2)$, and two target critic networks $\mathbf{Q}(s,a,\omega;\theta_1^-)$, $\mathbf{Q}(s,a,\omega;\theta_2^-)$, following the TD3 algorithm. During task $\mathcal{M}_n$, we expand our critic network, target critic networks, and reward model if novel objectives arrive. Then, for each episode, we randomly sample a preference $\omega \in \Omega$ as the discrete action space situation. The agent interacts with the environment to collect transitions $(s, a, \mathbf{r}, s', \omega, \text{done})$ by selecting action form the actor $\pi(a|s,\omega;\phi)$ with an exploration noise term ϵ. We also substitute ω with some randomly sampled preferences in a similar way and store them in the transition buffer $\mathcal{D}$. During the updating phase, we sample a minibatch of transitions and use the reward model to process the reward vectors in the transitions. The algorithm computes actions for the next state s' using the target actor network plus a smooth policy noise bounded by a noise clip term. In the multi-objective setting, both the two target critic networks output vectorized Q value instead of scalar return. So that we calculated the target Q value vector using whichever of the two critics gives a smaller $\omega^\top\mathbf{Q}$. Then, the critic is optimized to minimize the loss:

$$\mathcal{L}(\theta_1,\theta_2) = \mathbb{E}_{(s,a,\hat{\mathbf{r}},s',\omega)\sim\mathcal{D}}\left[\|\mathbf{y}_1, \mathbf{Q}_1\|_1^{\text{smooth}} + \|\mathbf{y}_2, \mathbf{Q}_2\|_1^{\text{smooth}}\right],$$

where $\mathbf{Q}_i = \mathbf{Q}(s,a,\omega;\theta_i)$, and $\mathbf{y}_i = \hat{\mathbf{r}} + \gamma\mathbf{Q}(s', a', \omega; \theta_i^-)$ denotes the target value vector which is obtained using target critic network's parameters θ_i^-. $\|\cdot\|_1^{\text{smooth}}$ represents the smooth ℓ_1 loss. The actor is optimized to minimize the loss:

$$\mathcal{L}(\psi) = \mathbb{E}_{(s,a,\hat{\mathbf{r}},s',\omega)\sim\mathcal{D}}\left[-\omega^\top \mathbf{Q}(s,a,\omega;\theta_1)\right]. \tag{8}$$

For stability, we delay the update of $\pi(a|s,\omega;\phi)$, and update it once every 10 steps for updating the critic network. The parameters ϕ^-, θ_1^-, θ_2^- of the target actor and critic networks are updated as bellow:

$$\begin{aligned}\phi^- &= (1-\tau)\phi^- + \tau\phi,\\ \theta_1^- &= (1-\tau)\theta_1^- + \tau\theta_1,\\ \theta_2^- &= (1-\tau)\theta_2^- + \tau\theta_2,\end{aligned} \tag{9}$$

where $\tau = 0.005$ is the coefficient to control the trade-off between the updated parameters ϕ, θ_1, θ_2 and the target networks parameters ϕ^-, θ_1^-, θ_2^-.

C.2 Training Details about Multi-objective Reward Model

To prevent the agent from losing the ability to complete previous tasks and to enable it to balance all encountered objectives, we propose a multi-objective reward model to rehearse the agent. Considering the CMORL setting, we also design a multi-head architecture for the reward model network (Figure 2(c)). Concretely, the reward model is initialized with a dynamic encoder $\mathcal{F}(s,a;\vartheta)$ and a state prediction head $g(z;\varphi_s)$. The dynamic encoder $\mathcal{F}(s,a;\vartheta)$ is a three-hidden-layer MLP with 1024 neurons per layer and is responsible for encoding the state and action information. When new objectives appear, we dynamically create a reward prediction head $g(z;\varphi_i)$ for each novel objective k_i. Both the reward heads and state prediction head consist of a mean value output head and a variance value output head, all with one hidden layer. The mean value head outputs the mean value $\mu(z)$ while the variance value head outputs the logarithmic variance value $\log\sigma^2(z)$ of the reward or state. During the agent's training phase, the transitions $(s,a,\mathbf{r},s',\omega,\text{done})$ for agent's training are also provided to the reward model. The encoder takes state s and action a as input, and then outputs the dynamic feature $z = \mathcal{F}(s,a;\vartheta)$. After that each reward head $g(z;\varphi_i)$ takes z as input, and uses the mean value head and variance value head to predict reward $\hat{R}_i(s,a)$ for objective k_i:

$$\hat{R}_i(s,a) \sim \mathcal{N}\left(\mu_i(z), \sigma_i^2(z)\right), \tag{10}$$

where $\mathcal{N}\left(\mu_i(z), \sigma_i^2(z)\right)$ represents a normal distribution with $\mu_i(z)$ as the mean and $\sigma_i^2(z)$ as the variance, forming the predicted reward vector $\hat{\mathbf{r}} = \hat{\mathbf{R}}(s,a) = (\hat{R}_1(s,a),\cdots,\hat{R}_{M_n}(s,a))$. When objective k_i appears in the current learning task, we directly set $\hat{R}_i(s,a)$ as the ground-truth reward in $\mathbf{r}$. The predicted reward vector will replace the environment reward signals $\mathbf{r}$ and be used for agent's updating. In practice, we also apply the ensemble technique to improve the robustness of the model. Specifically, we simultaneously train 5 reward models and for each transition, we randomly choose one for prediction.

For the training process of the reward model, we update the reward model for 500 epochs with an interval of 50k timesteps, and for each epoch, we sample a minibatch of 5000 transitions $(s,a,\mathbf{r},s',\omega,\text{done})$. The dynamic encoder $\mathcal{F}(s,a;\vartheta)$ first takes the state and action as input and outputs the feature z. Then, the reward heads corresponding to the learning objectives of the current task $\mathcal{M}_n$ will take z as input and output the mean values and logarithmic variance values. For instance, if task $\mathcal{M}_n$ contains objectives $\mathcal{K}_n = \{k_1^n,\cdots,k_{m_n}^n\}$, then the reward heads of the these objectives will predict means and logarithmic variances of their rewards, forming the reward mean vector $\hat{\mathbf{r}}_{\text{mean}} = (\mu_1^n(z),\cdots,\mu_{m_n}^n(z)) \in \mathbb{R}^{m_n}$, and reward variance vector $\hat{\mathbf{r}}_{\text{variance}} = (var_1^n(z),\cdots,var_{m_n}^n(z)) \in \mathbb{R}^{m_n}$, where $var_i^n = \exp\left(-\log\sigma_i^{2,n}(z)\right)$. With the purpose of improving the accuracy of the reward model and lowering the variance, we design two loss terms as bellows:

$$\begin{aligned}\mathcal{L}_1(\vartheta,\varphi_1^n,\cdots,\varphi_{m_n}^n) &= \frac{1}{m_n}\sum_{i=1}^{m_n}(\mu_i^n(z) - r_i^n)^2 \cdot var_i^n(z),\\ \mathcal{L}_2(\vartheta,\varphi_1^n,\cdots,\varphi_{m_n}^n) &= \frac{1}{m_n}\sum_{i=1}^{m_n} var_i^n(z).\end{aligned} \tag{11}$$

The dynamic encoder $\mathcal{F}(s,a;\vartheta)$ and reward heads corresponding to the objectives in task $\mathcal{M}_n$ is optimized to minimize the following loss:

$$\mathcal{L}(\vartheta,\varphi_1^n,\cdots,\varphi_{m_n}^n) = \mathcal{L}_1 + \mathcal{L}_2. \tag{12}$$

Besides, in order to stabilize the training process and enhance the encoder's ability to capture the fundamental information of the environment, the state prediction head $g(z;\varphi_s)$ will also take the feature z as input and then predict the next state based on feature z. The encoder $\mathcal{F}(s,a;\vartheta)$ and state prediction head $g(z;\varphi_s)$ is optimized to minimize the loss:

$$\mathcal{L}(\vartheta,\varphi_s) = \|(\hat{s'}_{\text{mean}} - s')^2 \cdot \hat{s'}_{\text{variance}}\|_1 + \|\hat{s'}_{\text{variance}}\|_1. \tag{13}$$

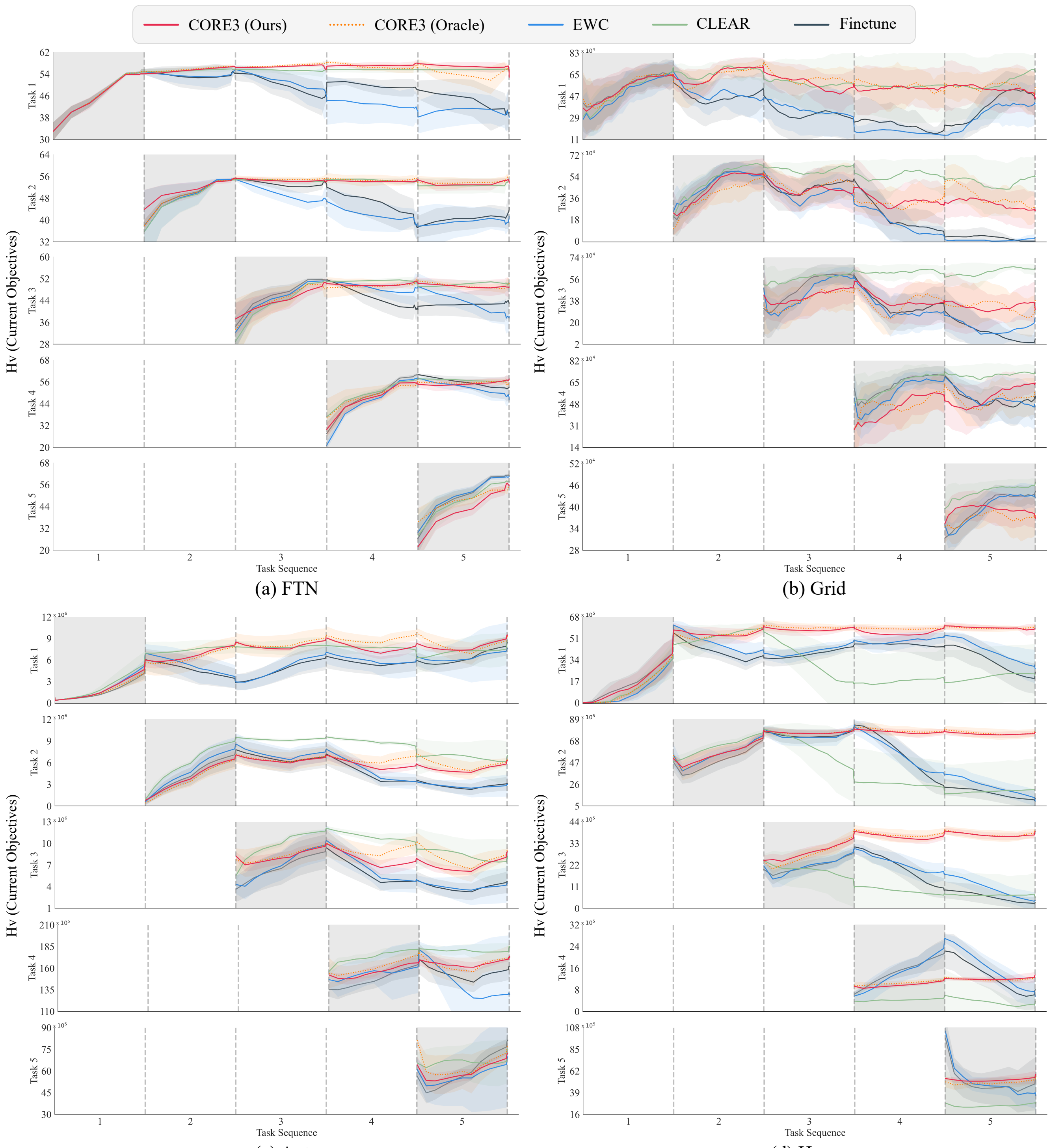
Finally, the overall loss function for learning the multi-objective reward model is defined as follows:

$$\mathcal{L}(\vartheta,\varphi_s,\varphi_1^n,\cdots,\varphi_{m_n}^n) = \mathcal{L}(\vartheta,\varphi_1^n,\cdots,\varphi_{m_n}^n) + \mathcal{L}(\vartheta,\varphi_s). \tag{14}$$

To avoid catastrophic forgetting about the previous objectives, we update the encoder in the first task only and keep its parameters fixed during the following ones. Due to the small state and action spaces in benchmark FTN, we use a table as the reward model for simplicity.

C.3 The Infrastructure of CORE3

We adopt Adam [Kingma and Ba, 2014] as the optimizer with learning rate 3×10^{-4} for agent networks and 1×10^{-4} for the reward models. The whole framework of CORE3 is trained on NVIDIA GeForce RTX 2080 Ti GPUs with a time cost of about 3 hours in FTN scenario, 27 hours in Grid World scenario and 72 hours in Ant/Hopper scenarios. It is worth noting that the time cost includes both training and testing, and the latter requires running a large amount of episodes, accounting for a significant portion of the time cost.



(a) FTN (b) Grid

(c) Ant (d) Hopper

Figure 3: The complete results of the Hv learning curves on all encountered tasks in the four benchmarks.

C.4 Hyperparameter Choices

We list the selection of the hyperparameters introduced in our approach for both discrete and continuous action spaces, e.g., the buffer size, in Table 1.

Hyperparameter	Value
Discrete Action Spaces	
buffer size	10000 (FTN), 500000 (Grid)
discount factor γ	0.99
epsilon start ϵ_{start}	0.8
epsilon finish ϵ_{finish}	0.05
policy update batch size	32 (FTN), 256 (Grid)
Continuous Action Spaces	
noise clip	0.5
buffer size	2000000
policy noise	0.2
discount factor γ	0.995
exploration noise ϵ	0.1
policy update batch size	256
critic network learning rate	3×10^{-4}
policy network update interval	10
Both Discrete & Continuous Action Spaces	
target update τ	0.005
policy network learning rate	3×10^{-4}
reward model learning rate	1×10^{-4}
reward model update epoch	500
reward model ensemble size	5
reward model update interval	50000
reward model update batch size	5000
additional preference number N_ω	3

Table 1: Hyperparameter choices of CORE3.

D The Complete Learning Results

In this section, we provide the complete results of the Hv learning curves on all encountered tasks in the four benchmarks, and sensitivity studies on multiple hyperparameters.

D.1 Hv Learning Curves of Each Task

The complete results of the Hv learning curves on all encountered tasks in the four benchmarks is shown in Figure 3. We observe that baseline CLEAR [Rolnick *et al.*, 2019] achieves comparable performance to CORE3 in some benchmarks. The reason is that CLEAR stores old data to rehearse the agent, alleviating catastrophic forgetting on the encountered individual tasks. However, as mentioned in the main body of the paper, it has no access to data balancing objectives in different tasks, limiting its performance on all encountered objectives. Meanwhile, baselines EWC [Kirkpatrick *et al.*, 2017] and Finetune suffer from performance degradation on some tasks after training on them, i.e., catastrophic forgetting. In contrast, CORE3 exhibits non-decreasing learning curves, comparable to CORE3 (Oracle) on every task. This suggests that our approach effectively mitigates the issue of catastrophic forgetting, preserving the agent's competence in completing previous objectives.

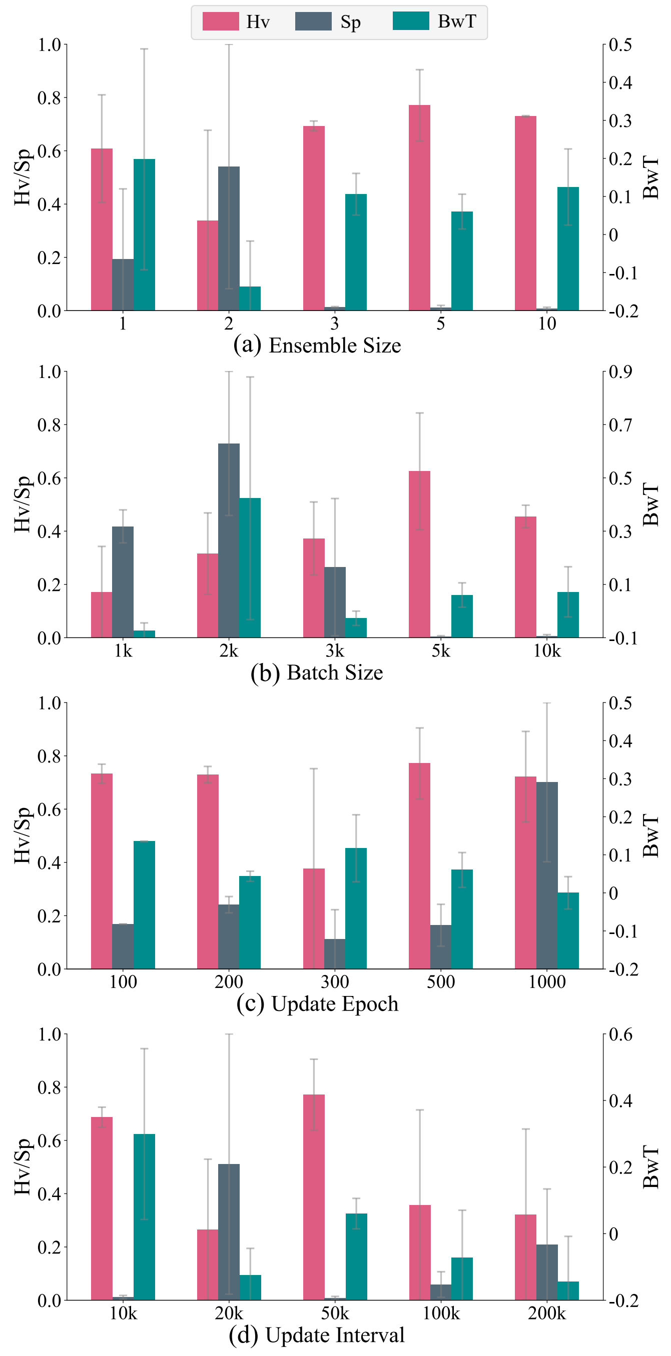


Figure 4: Sensitivity studies in Hopper.

D.2 Sensitivity Studies

The complete results of sensitivity studies on multiple hyper-parameters, including the (1) ensemble size, (2) batch size, (3) number of update epochs, and (4) update interval of training the multi-objective reward model, is shown in Figure 4.

In the main body of the paper, we have discussed the sensitivity of the ensemble size. For the batch size of training the reward model, on one hand, the training data provided by a small minibatch may not be sufficient to obtain a stable and accurate reward model. On the other hand, a large batch size will slow down the training process. As shown in Figure 4(b), we find that when the batch size is less than 5000, the performance of CORE3 improves with an increase in batch size. Nevertheless, the HV and SP doesn't get better when we increase batch size from 5000 to 10000. Furthermore, another adjustable hyperparameter, the number of update epochs, also has an impact on the amount of data for model training. Figure 4(c) shows that a small number of epochs can lead to underfitting, while a large number results in overfitting of the reward model. We set it to an appropriate number of 500. The last analyzed hyperparameter is the update interval. As shown in Figure 4(d), a large update interval leads to insufficient model training, causing catastrophic forgetting of the agent, while an update interval that is too small significantly reduces training efficiency without yielding improvements. We find that an update interval of 5k steps performs the best.

References

[Agarap, 2018] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[Basaklar *et al.*, 2023] Toygun Basaklar, Suat Gumussoy, and Ümit Y. Ogras. PD-MORL: preference-driven multi-objective reinforcement learning algorithm. In *ICLR*, 2023.

[Chung *et al.*, 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[Fujimoto *et al.*, 2018] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, pages 1587–1596, 2018.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kirkpatrick *et al.*, 2017] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[Rolnick *et al.*, 2019] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *NeurIPS*, pages 350–360, 2019.

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033, 2012.

[Xu *et al.*, 2020] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *ICML*, pages 10607–10616, 2020.

[Yang *et al.*, 2019] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *NeurIPS*, page 14636–14647, 2019.