

# Supplementary Information: Discovery of the Reward Function for Embodied Reinforcement Learning Agents

Renzhi Lu<sup>1</sup>, Zonghe Shao<sup>1</sup>, Yuemin Ding<sup>2,3\*</sup>, Ruijuan Chen<sup>4</sup>,  
Dongrui Wu<sup>5</sup>, Housheng Su<sup>5</sup>, Tao Yang<sup>6</sup>, Fumin Zhang<sup>7</sup>, Jun Wang<sup>8</sup>,  
Yang Shi<sup>9</sup>, Zhong-Ping Jiang<sup>10</sup>, Han Ding<sup>11</sup>, Hai-Tao Zhang<sup>1\*</sup>

<sup>1</sup>School of Artificial Intelligence and Automation, Engineering Research Center of Autonomous Intelligent Unmanned Systems, Chinese Ministry of Education, State Key Laboratory of Digital Manufacturing Equipment and Technology, Institute of Artificial Intelligence, Huazhong University of Science and Technology, Luoyu Road 1037, Wuhan, 430074, Hubei, China.

<sup>2</sup>Tecnun School of Engineering, University of Navarra, Paseo de Manuel Lardizabal, 13, 20018, San Sebastián, Spain.

<sup>3</sup>Ikerbasque Foundation, Plaza Euskadi 5, 48009, Bilbao, Spain.

<sup>4</sup>Research Center for Applied Mathematics and Interdisciplinary Science, School of Mathematics and Statistics, Wuhan Textile University, No.1 Sunshine Avenue, Wuhan, 430200, Hubei, China.

<sup>5</sup>Key Laboratory of Image Processing and Intelligent Control, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Luoyu Road 1037, Wuhan, 430074, Hubei, China.

<sup>6</sup>State Key Laboratory of Synthetical Automation for Process Industrie, Northeastern University, Wenhua Road 3-11, Shenyang, 110819, Liaoning, China.

<sup>7</sup>Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Sai Kung Peninsula, Hong Kong, China.

<sup>8</sup>Department of Computer Science and Department of Data Science, City University of Hong Kong, Kowloon, Hong Kong, China.

<sup>9</sup>Department of Mechanical Engineering, University of Victoria, 3800 Finnerty Road, Victoria, Canada.

<sup>10</sup>Department of Electrical and Computer Engineering, New York University, 370 Jay Street, Brooklyn, NY 11201, USA.

<sup>11</sup>State Key Laboratory of Digital Manufacturing Equipment and Technology, School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Luoyu Road 1037, Wuhan, 430074, Hubei, China.

\*Corresponding author(s). E-mail(s): [yueminding@tecnun.es](mailto:yueminding@tecnun.es);  
[zht@mail.hust.edu.cn](mailto:zht@mail.hust.edu.cn);

Contributing authors: [rzlu@hust.edu.cn](mailto:rzlu@hust.edu.cn); [zhshao@hust.edu.cn](mailto:zhshao@hust.edu.cn);  
[rjchen@wtu.edu.cn](mailto:rjchen@wtu.edu.cn); [drwu@hust.edu.cn](mailto:drwu@hust.edu.cn); [shs@hust.edu.cn](mailto:shs@hust.edu.cn);  
[yangtao@mail.neu.edu.cn](mailto:yangtao@mail.neu.edu.cn); [eefumin@ust.hk](mailto:eefumin@ust.hk); [jwang.cs@cityu.edu.hk](mailto:jwang.cs@cityu.edu.hk);  
[yshi@uvic.ca](mailto:yshi@uvic.ca); [zjiang@nyu.edu](mailto:zjiang@nyu.edu); [dinghan@hust.edu.cn](mailto:dinghan@hust.edu.cn);

## Supplementary Related Works

### The Optimal Reward Function

Many studies [1–6] have sought to identify the “best” reward function for reinforcement learning (RL), although a formal definition remains elusive. The authors of [7, 8] suggested that an optimal reward function should have a dual structure: a primary component  $R$  that guides the agent’s learning process and a fitness function  $F$  that assesses learning quality. Building on this, works such as [9, 10] proposed that the optimal reward function should promote optimal behavior (fitness) while minimizing the learning time (effort). However, definition of a generalized fitness function is challenging. Some studies have approached the definition of the optimal reward function from specific perspectives [11–13]. For example, [11] defined the optimal reward as that which avoided overfitting of a specific RL algorithm. Moreover, intrinsic motivation [14] has encouraged exploration of whether optimal intrinsic rewards could complement external rewards [15], often in conjunction with reward shaping [16]. However, the focus of this work is the core question of what constitutes an optimal reward function; we do not deal with auxiliary rewards. Recent studies have leveraged supervised learning techniques, framing optimal rewards in terms of expert demonstrations [17, 18] or human preferences [6, 19]. A promising method is separation of the objectives of the reward function designer from those of the RL agent [15, 20]. The optimal reward function is then that which maximizes the external rewards throughout the life cycle of the RL agent [21]. Our perspective aligns with that of [21], which recommended the use of a dedicated objective function for the reward itself; the aim was to maximize a corresponding objective function within the reward space. We suggest that the optimal reward function should minimize the policy regret while keeping the training error low [22], thereby expediting learning.

### Meta Learning in Reinforcement Learning

Meta learning methods have been widely applied to RL tasks [23–28] including hyperparameter optimization [25, 29], model learning [24, 30], and algorithm generalization [28]. In this paper, we focus on a meta learning framework specific to reward function learning [31, 32]. One approach used to reward meta learning involves optimizing the reward functions across diverse task distributions for adaptation to new tasks [33, 34]. A promising alternative is to establish a bilevel optimization framework based on meta learning. Following the theory proposed in [21], researchers in [14] posited that reward learning could be framed as a bilevel optimization problem, where policy optimization in an inner loop shapes the rewards and the reward function parameters are optimized in the outer loop. Similarly, [2] viewed reward learning as a bilevel optimization framework but focused on how the optimized reward function is integrated within the framework. Our approach aligns with this idea. The framework has been expanded in various ways. For example, [35–37] incorporated it into multi-agent RL, allowing each agent to learn a personal reward function. Currently, the key challenges posed by the use of bilevel framework, to optimize reward functions, include effective definition of the upper-level optimization objectives. Usually, the definition is the same as that of the optimal reward function, as explored in [38], which sought to optimize imperfect rewards. Meanwhile, [39] and [5] focused on alignment of reward functions with human preferences. Our approach differs fundamentally from previous methods in terms of how the upper-level optimization objective is defined within the bilevel framework. Prior works often rely on indirect or heuristic formulations, but our method introduces a direct and principled objective. Specifically, we define the optimal reward function by explicitly minimizing policy regret. The policy performance of the agent is the metric used for reward optimization, which effectively resolves the long-standing challenge of specifying the upper-level optimization goal. Thus, our approach establishes a clear and theoretically grounded connection between reward function learning and policy optimization, setting it apart from existing methods.

## Reward Function Design Methods

In the early stages of RL, researchers relied primarily on manual, trial-and-error adjustments to refine the reward functions [40]. However, this approach is time-consuming and often culminates in reward functions that inadvertently guide agents to act contrary to the human intent [22], associated with policy deviations. Therefore, many methods have been developed to optimize reward learning. One widely studied approach is reward shaping [2, 41–43], which involves the creation of auxiliary rewards that supplement the environmental reward function and ensure policy invariance [44]. However, the definition of shaping functions and the attainment of policy invariance have long posed challenges [45]. Another popular approach is inverse RL (IRL), initially proposed by [46], which is aimed at deriving a reward function from expert demonstrations, assuming that such demonstrations represent an optimal policy. Recent advances in deep learning (DL) have extended the performance of IRL [47, 48], suggesting that an adequate number of expert demonstrations can indeed produce an optimal reward function. However, in many cases, the aim of RL is not only to replicate human performance but also to exceed human capabilities, particularly in tasks that are beyond human reach [49, 50]. This limitation constrains the applicability of IRL, although some researchers have sought to surpass expert performance by incorporating additional information and using non-expert demonstrations [51, 52]. Recently, an active reward learning approach has emerged, inspired by the training paradigms of LLMs [53, 54], with its origins in robotics [55]. Active reward learning is aimed at constructing reward functions that align closely with human preferences by modeling those preferences [56–58], even though some researchers have explored whether it is possible to completely bypass reward learning [59–61]. Nonetheless, the capture of sufficiently rich human feedback remains a significant challenge [62]. Other reward learning methods have leveraged characteristics of the agent or environmental [15, 63] to design reward functions, such as encouraging curiosity-driven exploration in maze tasks [16, 64]. All cited methods extensively explored the automatic design or optimization of reward functions. However, such approaches rely heavily on priors. For example, IRL requires high-quality expert demonstrations and ARL requires substantial human preference data. In contrast, our work directly optimizes the reward function using the interactions of the agent during RL, leveraging agent experience (policy meta-gradients). There is essentially no need for prior knowledge or external supervision. By continuously refining the reward function based on the experience accumulated by the agent, our approach renders reward function discovery more autonomous and adaptive. This is particularly helpful when expert demonstrations and/or human feedback are scarce or unavailable.

## Supplementary Theoretical Results

In this section, proofs and analyses of the theorems presented in the main text are provided.

### Proof of Proposition 1

*Proof of Proposition 1.* Before formally stating Proposition 1 in the main text, we again explain the notations for clarity.  $R^*$  is the optimal reward function.  $\pi^*$  represents the optimal policy in the policy space  $\Pi$ , which is often considered fixed.  $\pi_R^*$  indicates that an agent, guided by reward function  $R$ , optimizes its policy  $\pi$  by maximizing the expected return  $G_R^\pi$ , with the ultimate objective of optimizing its policy to  $\pi_R^*$ . Therefore, we have  $\pi_R^* \in \arg \max_{\pi \in \Pi} G_R^\pi$ . We assume  $\pi_{R^*}^* = \pi^*$ .

By Definition 1 in the main text, if the reward function  $R$  is *proximal* to the optimal reward function  $R^*$ , we have:

$$|G_R^\pi - G_{R^*}^\pi| \leq \delta, \forall \pi \in \Pi, \quad (\text{S1})$$

which gives:

$$\begin{aligned} |G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}| &\leq \delta, \\ |G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}| &\leq \delta, \end{aligned} \quad (\text{S2})$$

where  $\delta$  is a small positive constant.

Applying the triangle inequality to the bound  $|G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}|$ , we obtain the following equation:

$$|G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}| \leq |G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}| + |G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}| + |G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}|. \quad (\text{S3})$$

By substituting the bounds obtained in equation (S2), we obtain:

$$\begin{aligned} |G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}| &\leq \delta + |G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}| + \delta, \\ &\leq |G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*}| + 2\delta, \\ &\stackrel{(a)}{\leq} G_{R^*}^{\pi_R^*} - G_{R^*}^{\pi_R^*} + 2\delta, \\ &\stackrel{(b)}{\leq} (G_R^{\pi_R^*} + \delta) - (G_{R^*}^{\pi_R^*} - \delta) + 2\delta, \\ &\leq G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*} + 4\delta, \\ &\stackrel{(c)}{\leq} 4\delta, \end{aligned} \quad (\text{S4})$$

where (a) holds because  $\pi^*$  is the optimal policy under  $R^*$  (i.e.,  $G_{R^*}^{\pi^*} \geq G_{R^*}^{\pi_R^*}, \forall \pi \in \Pi$ ), which implies that:

$$|G_{R^*}^{\pi_R^*} - G_{R^*}^{\pi^*}| = G_{R^*}^{\pi^*} - G_{R^*}^{\pi_R^*} \geq 0, \quad (\text{S5})$$

and (b) holds on expansion of the absolute value inequalities of equation (S2):

$$\begin{aligned} G_R^{\pi_R^*} - G_{R^*}^{\pi_R^*} &\leq \delta, \\ G_{R^*}^{\pi_R^*} - G_R^{\pi_R^*} &\leq \delta, \end{aligned} \quad (\text{S6})$$

and (c) holds because  $\pi_R^*$  is the optimal policy under  $R$ , i.e.,  $G_R^{\pi_R^*} \geq G_R^{\pi}, \forall \pi \in \Pi$ .

As the absolute difference of  $G_R^{\pi_R^*}$  and  $G_{R^*}^{\pi_R^*}$  is bounded by  $4\delta$ , and  $\delta$  is a small positive constant, therefore, by Definition 1 in the main text,  $\pi_R^* \stackrel{R}{\sim} \pi^*$ .

Similar to equation (S4), applying the triangle inequality to the bound  $|G_{R^*}^{\pi_R^*} - G_{R^*}^{\pi^*}|$ , we have:

$$\begin{aligned} |G_{R^*}^{\pi_R^*} - G_{R^*}^{\pi^*}| &\leq |G_{R^*}^{\pi_R^*} - G_R^{\pi_R^*}| + |G_R^{\pi_R^*} - G_{R^*}^{\pi^*}| + |G_R^{\pi^*} - G_{R^*}^{\pi^*}|, \\ &\leq \delta + |G_R^{\pi_R^*} - G_{R^*}^{\pi^*}| + \delta, \\ &\leq |G_R^{\pi_R^*} - G_{R^*}^{\pi^*}| + 2\delta, \\ &\leq G_{R^*}^{\pi^*} - G_R^{\pi_R^*} + 2\delta, \\ &\leq (G_{R^*}^{\pi^*} + \delta) - (G_R^{\pi_R^*} - \delta) + 2\delta, \\ &\leq G_{R^*}^{\pi^*} - G_R^{\pi_R^*} + 4\delta, \\ &\leq 4\delta. \end{aligned} \quad (\text{S7})$$

Therefore, if the reward function  $R$  is *proximal* to the optimal reward function  $R^*$ , we obtain a policy  $\pi_R^*$  that is *proximal* to the optimal policy  $\pi^*$  with  $R$  and  $R^*$ . Thus:

$$R \sim R^* \Rightarrow \pi_R^* \stackrel{R}{\sim} \pi^*, \pi_R^* \stackrel{R^*}{\sim} \pi^*. \quad (\text{S8})$$

□

## Proof of Theorem 2

The meta gradient exhibits significant potential for use during RL. Although we are not the first to explore the application of meta gradients in optimizing reward functions, we present a rigorous theoretical analysis that ensures the applicability of the proposed approach. Furthermore, we clarify that, using the alternating optimization approach proposed in the main text for RL, the policy parameters can be expressed as a meta function of the reward function parameters in equation (13) in the main text. To clarify the subsequent discussion, we here note that the new policy parameter  $\theta'$  is a meta function of the reward function parameter  $\omega$ . This follows directly from the RL process, but there is no direct relationship between the policy parameters  $\theta$  and the reward function parameters  $\omega$ .

*Proof of Theorem 2.* Given the current reward function  $R_\omega$  and the agent's policy  $\pi_\theta$ , the agent generates trajectories within the world model  $\mathcal{M}$  and optimizes its policy from  $\theta$  to  $\theta'$  on the basis of equation (10) in the main text during the lower-level update step. We now reformulate the equation (10) in the main text as follows:

$$\theta' = \theta + \alpha \cdot \mathbb{E}_{\mu_{s,a}} [Q_{R_\omega}^{\pi_\theta}(s, a) \cdot \frac{\partial \log \pi_\theta(s, a)}{\partial \theta}]. \quad (\text{S9})$$

Our key insight is that the policy parameters  $\theta$  can be viewed as a meta function of the reward function parameter  $\omega$  because agents interact with the environment and optimize the policy under the guidance of that reward function. Therefore, mathematically, the gradient of the upper-level objective with respect to the reward parameters is

$$\frac{d}{d\omega} \mathcal{J}^{upper}(\theta', \omega) = \frac{\partial}{\partial \omega} \mathcal{J}^{upper}(\theta', \omega) + \frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega) \cdot \frac{d\theta'}{d\omega}. \quad (\text{S10})$$

Here, both equation (13) in the main text and equation (S10) indicate that the policy parameters  $\theta$  should be modeled as a meta function of the reward parameters  $\omega$ . However, we now develop a more precise formulation to facilitate the subsequent proof of equation (S10).

The first term  $\frac{\partial}{\partial \omega} \mathcal{J}^{upper}(\theta', \omega)$  can be formulated as:

$$\frac{\partial}{\partial \omega} \mathcal{J}^{upper}(\theta', \omega) = \mathbb{E}_{\mu'_{s,a}} \left[ Q_{\bar{R}}^{\pi_{\theta'}}(s, a) \cdot \frac{\partial \log \pi_{\theta'}(s, a)}{\partial \omega} \right], \quad (\text{S11})$$

where  $\mu'_{s,a}$  represents the state-action distribution under the policy  $\pi_{\theta'}$ , i.e.,  $s \sim \rho^{\pi_{\theta'}}, a \sim \pi_{\theta'}(\cdot | s)$ .

For  $\frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega)$ , based on the definition of  $G_R^\pi$ , proceeding in manner similar to the lower-level optimization of Lemma 1 in the main text shows that the optimization may be defined as:

$$\frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega) = \mathbb{E}_{\mu'_{s,a}} \left[ Q_{\bar{R}}^{\pi_{\theta'}}(s, a) \cdot \frac{\partial \log \pi_{\theta'}(s, a)}{\partial \theta'} \right]. \quad (\text{S12})$$

$\frac{d\theta'}{d\omega}$  can be expressed as follows:

$$\begin{aligned}\frac{d\theta'}{d\omega} &= \frac{\partial\theta'}{\partial\theta} \cdot \frac{d\theta}{d\omega} + \frac{\partial\theta'}{\partial\omega} \\ &= \left( I + \alpha \cdot \mathbb{E}_{\mu_{s,a}} \left[ Q_{R_\omega}^{\pi_\theta}(s, a) \cdot \frac{\partial^2 \log \pi_\theta(s, a)}{\partial\theta^2} \right] \right) \cdot \frac{d\theta}{d\omega} \\ &\quad + \alpha \cdot \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial\omega} \cdot \frac{\partial \log \pi_\theta(s, a)}{\partial\theta} \right] \\ &\quad + \alpha \cdot \mathbb{E}_{\mu_{s,a}} \left[ Q_{R_\omega}^{\pi_\theta}(s, a) \cdot \frac{\partial^2 \log \pi_\theta(s, a)}{\partial\theta \partial\omega} \right],\end{aligned}\tag{S13}$$

where  $\mu_{s,a}$  represents the state-action distribution under the policy  $\pi_\theta$ .

If we restrict the analysis to back-propagation through a single step, the first additive term in equation (S13) is zero. We have:

$$\frac{d\theta'}{d\omega} = \frac{d\theta}{d\omega} + \alpha \cdot \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial\omega} \cdot \frac{\partial \log \pi_\theta(s, a)}{\partial\theta} + Q_{R_\omega}^{\pi_\theta}(s, a) \cdot \frac{\partial^2 \log \pi_\theta(s, a)}{\partial\theta \partial\omega} \right].\tag{S14}$$

**Supplementary Assumption 1.** Assume that the policy is a meta function of the reward, i.e.,  $\theta(\omega)$ . Then, changes in the reward affect the policy parameters by influencing policy updates indirectly rather than directly impacting the policy parameters.

Under Supplementary Assumption 1, we approximate equation (S11) and the second additional term in equation (S14) as 0.

$$\begin{aligned}\mathbb{E}_{\mu'_{s,a}} \left[ Q_R^{\pi_{\theta'}}(s, a) \cdot \frac{\partial \log \pi_\theta(s, a)}{\partial\omega} \right] &\approx 0, \\ \mathbb{E}_{\mu_{s,a}} \left[ Q_{R_\omega}^{\pi_\theta}(s, a) \cdot \frac{\partial^2 \log \pi_\theta(s, a)}{\partial\theta \partial\omega} \right] &\approx 0.\end{aligned}\tag{S15}$$

On the basis of equation (S15), the meta gradient of equation (S14) can be expressed as:

$$\frac{d\theta'}{d\omega} = \alpha \cdot \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial\omega} \cdot \frac{\partial \log \pi_\theta(s, a)}{\partial\theta} \right].\tag{S16}$$

We observe that, under these two assumptions, our results align with the findings of [15, 16]. Therefore, the gradient of the upper-level objective with respect to the reward parameters of equation (S10) can be formulated as:

$$\begin{aligned}&\frac{d}{d\omega} \mathcal{J}^{upper}(\theta', \omega) \\ &= \frac{\partial}{\partial\omega} \mathcal{J}^{upper}(\theta', \omega) + \frac{\partial}{\partial\theta'} \mathcal{J}^{upper}(\theta', \omega) \cdot \frac{d\theta'}{d\omega} \\ &= \alpha \cdot \mathbb{E}_{\mu'_{s,a}} \left[ Q_R^{\pi_{\theta'}}(s, a) \cdot \frac{\partial \log \pi_\theta(s, a)}{\partial\theta'} \right] \cdot \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial\omega} \cdot \frac{\partial \log \pi_\theta(s, a)}{\partial\theta} \right].\end{aligned}\tag{S17}$$

By abbreviating the full partial differentials, this simplifies to:

$$[\nabla_\omega \mathcal{J}^{upper}]_\omega = \mathbb{E}_{\mu'_{s,a}} \left[ Q_R^{\pi_{\theta'}} \cdot [\nabla_{\theta'} \log \pi_\theta]_{\theta'} \right] \cdot \mathbb{E}_{\mu_{s,a}} \left[ [\nabla_\omega Q_{R_\omega}^{\pi_\theta}]_\omega \cdot [\nabla_\theta \log \pi_\theta]_\theta \right],\tag{S18}$$

where we omit the factor  $\alpha$ .

□

### Proof of Theorem 3

*Proof of Theorem 3.* On the basis of Theorem 2, the gradient of the upper-level objective with respect to the reward parameters is:

$$\frac{d}{d\omega} \mathcal{J}^{upper}(\theta', \omega) \approx \frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega) \cdot \frac{d\theta'}{d\omega}. \quad (\text{S19})$$

In equation (S19), the two terms are as follows:

$$\begin{aligned} \frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega) &= \mathbb{E}_{\mu'_{s,a}} \left[ Q_R^{\pi_{\theta'}}(s, a) \cdot \frac{\partial \log \pi_{\theta}(s, a)}{\partial \theta'} \right], \\ \frac{d\theta'}{d\omega} &\approx \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial Q_{R,\omega}^{\pi_{\theta}}(s, a)}{\partial \omega} \cdot \frac{\partial \log \pi_{\theta}(s, a)}{\partial \theta} \right]. \end{aligned} \quad (\text{S20})$$

To illustrate the value-based approximation of the meta gradient term more clearly, we first introduce the assumption that the action space  $\mathcal{A}$  is a finite discrete set, referred to as Supplementary Assumption 2.

**Supplementary Assumption 2.** *Assume that the action space is finite and discrete.*

Supplementary Assumption 2 allows us to approximate the policy parameters  $\theta$  on the basis of Supplementary Lemma 1.

**Supplementary Lemma 1** ([65]). *When the action space  $\mathcal{A}$  is finite and discrete, the policy can be parameterized as a normalization of the action preferences. This can be mathematically formalized as:*

$$\pi_{\theta}(s, a) := \frac{\exp \theta^{\top} \mathbf{x}_{s,a}}{\sum_{a' \in \mathcal{A}} \exp \theta^{\top} \mathbf{x}_{s,a'}}, \forall s \in \mathcal{S}, a \in \mathcal{A}, \quad (\text{S21})$$

where  $\mathbf{x}_{s,a} \in \mathbb{R}^d$  is the soft feature vector of the state-action pair and  $d$  is the dimension of the feature vector.

On the basis of Supplementary Lemma 1, the policy gradient can be formulated as

$$\frac{\partial \log \pi_{\theta}(s, a)}{\partial \theta} = \mathbf{x}_{s,a} - \sum_{a' \in \mathcal{A}} \mathbf{x}_{s,a'} \cdot \pi_{\theta}(s, a'). \quad (\text{S22})$$

Therefore,  $\frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega)$  can be simplified as follows:

$$\begin{aligned} &\frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega) \\ &= \mathbb{E}_{\mu'_{s,a}} \left[ Q_R^{\pi_{\theta'}}(s, a) \cdot \frac{\partial \log \pi_{\theta}(s, a)}{\partial \theta'} \right] \\ &= \mathbb{E}_{\mu'_{s,a}} \left[ Q_R^{\pi_{\theta'}}(s, a) \cdot \left( \mathbf{x}_{s,a} - \sum_{a' \in \mathcal{A}} \pi_{\theta'}(s, a') \cdot \mathbf{x}_{s,a'} \right) \right] \\ &= \mathbb{E}_{s \sim \rho^{\pi_{\theta'}}} \left[ \sum_{a \in \mathcal{A}} \pi_{\theta'}(s, a) \cdot \left( \mathbf{x}_{s,a} - \sum_{a' \in \mathcal{A}} \pi_{\theta'}(s, a') \cdot \mathbf{x}_{s,a'} \right) \cdot Q_R^{\pi_{\theta'}}(s, a) \right] \\ &= \mathbb{E}_{s \sim \rho^{\pi_{\theta'}}} \left[ \sum_{a \in \mathcal{A}} \pi_{\theta'}(s, a) \cdot \mathbf{x}_{s,a} \cdot Q_R^{\pi_{\theta'}}(s, a) - V_R^{\pi_{\theta'}}(s) \cdot \sum_{a' \in \mathcal{A}} \pi_{\theta'}(s, a') \cdot \mathbf{x}_{s,a'} \right]. \end{aligned} \quad (\text{S23})$$

Using the relationship between the value function  $V_R^\pi(s)$  and the action-value function  $Q_R^\pi(s, a)$ , we rewrite equation (S23) as follows:

$$\frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega) = \mathbb{E}_{s \sim \rho^{\pi_{\theta'}}} \left[ \sum_{a \in \mathcal{A}} \pi_{\theta'}(s, a) \cdot \left( Q_R^{\pi_{\theta'}}(s, a) - V_R^{\pi_{\theta'}}(s) \right) \cdot \mathbf{x}_{s,a} \right]. \quad (\text{S24})$$

By recognizing that the expectation operator is linear, we can combine the summation and expectation into a single expectation over the state-action pairs, leading to the simplified form as follows:

$$\frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega) = \mathbb{E}_{\mu'_{s,a}} \left[ \left( Q_R^{\pi_{\theta'}}(s, a) - V_R^{\pi_{\theta'}}(s) \right) \cdot \mathbf{x}_{s,a} \right]. \quad (\text{S25})$$

Similar to equation (S23), given the assumption in equation (S21), we can simplify  $\frac{d\theta'}{d\omega}$  to:

$$\begin{aligned} & \frac{d\theta'}{d\omega} \\ &= \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial \omega} \cdot \frac{\partial \log \pi_\theta(s, a)}{\partial \theta} \right] \\ &= \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial \omega} \cdot \left( \mathbf{x}_{s,a} - \sum_{a' \in \mathcal{A}} \pi_\theta(s, a') \cdot \mathbf{x}_{s,a'} \right) \right] \\ &= \mathbb{E}_{s \sim \rho^{\pi_\theta}} \left[ \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \cdot \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial \omega} \cdot \left( \mathbf{x}_{s,a} - \sum_{a' \in \mathcal{A}} \pi_\theta(s, a') \cdot \mathbf{x}_{s,a'} \right) \right] \\ &= \mathbb{E}_{s \sim \rho^{\pi_\theta}} \left[ \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \cdot \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial \omega} \cdot \mathbf{x}_{s,a} - \frac{\partial}{\partial \omega} \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \cdot Q_{R_\omega}^{\pi_\theta}(s, a) \cdot \sum_{a' \in \mathcal{A}} \pi_\theta(s, a') \cdot \mathbf{x}_{s,a'} \right] \\ &= \mathbb{E}_{s \sim \rho^{\pi_\theta}} \left[ \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \cdot \frac{\partial Q_{R_\omega}^{\pi_\theta}(s, a)}{\partial \omega} \cdot \mathbf{x}_{s,a} - \sum_{a' \in \mathcal{A}} \pi_\theta(s, a') \cdot \mathbf{x}_{s,a'} \cdot \frac{\partial V_{R_\omega}^{\pi_\theta}(s)}{\partial \omega} \right]. \end{aligned} \quad (\text{S26})$$

Similar to the equations (S23–S25), we next simplify equation (S26) to:

$$\frac{d\theta'}{d\omega} = \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial}{\partial \omega} \left( Q_{R_\omega}^{\pi_\theta}(s, a) - V_{R_\omega}^{\pi_\theta}(s) \right) \cdot \mathbf{x}_{s,a} \right]. \quad (\text{S27})$$

Finally, we consider the following:

$$\begin{aligned} & \frac{\partial}{\partial \theta'} \mathcal{J}^{upper}(\theta', \omega) \cdot \frac{d\theta'}{d\omega} \\ &= \mathbb{E}_{\mu'_{s,a}} \left[ \left( Q_R^{\pi_{\theta'}}(s, a) - V_R^{\pi_{\theta'}}(s) \right) \cdot \mathbf{x}_{s,a} \right] \cdot \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial}{\partial \omega} \left( Q_{R_\omega}^{\pi_\theta}(s, a) - V_{R_\omega}^{\pi_\theta}(s) \right) \cdot \mathbf{x}_{s,a} \right]. \end{aligned} \quad (\text{S28})$$

Now, we analyze the scenario that is not covered under Assumption 2, (i.e., that the action space is continuous and infinite). This is the case for most RL problems. During deep reinforcement learning (DRL), the policy of the embodied RL agent is modeled by deep neural networks (DNNs) wherein the feature vectors  $\mathbf{x}_{s,a}$  and parameters  $\theta$  are unified. Therefore, equation (S28) can be directly expressed as:

$$\frac{d}{d\omega} \mathcal{J}^{upper}(\theta', \omega) = \mathbb{E}_{\mu'_{s,a}} \left[ Q_R^{\pi_{\theta'}}(s, a) - V_R^{\pi_{\theta'}}(s) \right] \cdot \mathbb{E}_{\mu_{s,a}} \left[ \frac{\partial}{\partial \omega} \left( Q_{R_\omega}^{\pi_\theta}(s, a) - V_{R_\omega}^{\pi_\theta}(s) \right) \right]. \quad (\text{S29})$$

On abbreviating the full and partial differentials, equation (S29) simplifies to:

$$[\nabla_{\omega} \mathcal{J}^{upper}]_{\omega} = \mathbb{E}_{\mu'_{s,a}} \left[ A_R^{\pi_{\theta'}}(s, a) \right] \cdot \mathbb{E}_{\mu_{s,a}} \left[ \nabla_{\omega} A_{R_{\omega}}^{\pi_{\theta}}(s, a) \right], \quad (\text{S30})$$

where  $A_R^{\pi}(s, a)$  is the advantage function of MDP  $\mathcal{M}[R]$ . Typically,  $A_R^{\pi}(s, a)$  denotes the value of the advantage of choosing action  $a$  in state  $s$  under policy  $\pi$  determined by the reward function  $R$ , which is defined as follows:

$$A_R^{\pi}(s, a) = Q_R^{\pi}(s, a) - V_R^{\pi}(s), \quad (\text{S31})$$

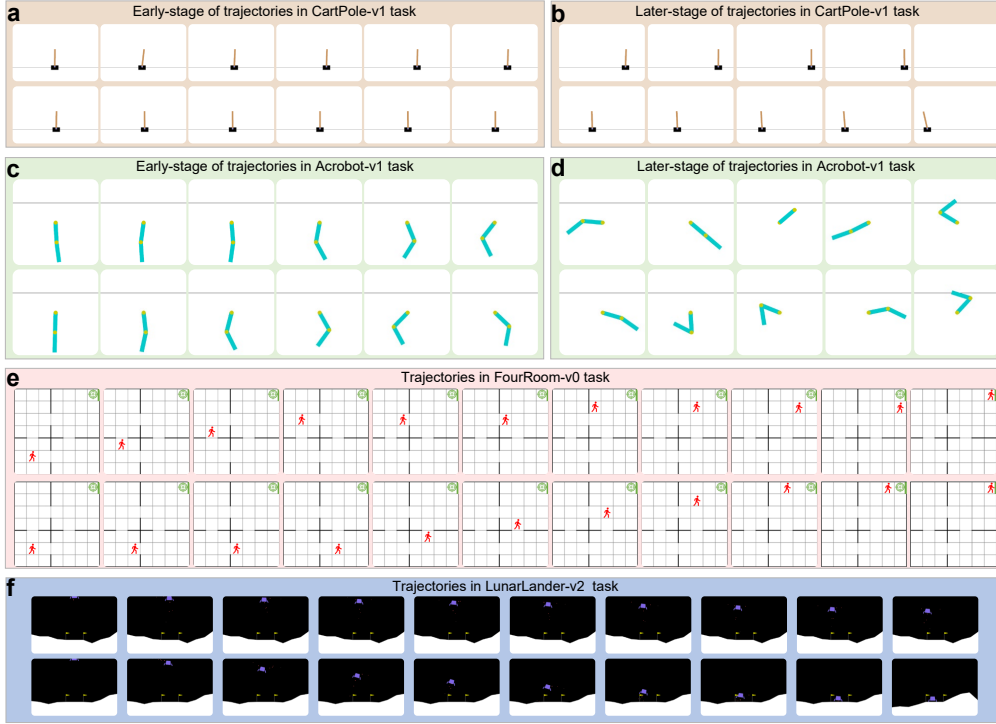
where  $Q_R^{\pi_{\theta}}(s, a)$  represents the expected return of taking action  $a$  in state  $s$  under reward function  $R$  and  $V_R^{\pi_{\theta}}(s)$  denotes the expected return when following policy  $\pi_{\theta}$  in state  $s$  under reward function  $R$ .

□

# Supplementary Experimental Results

## Supplementary Experimental Results of Sparse-Reward Task

The behaviors of agents receiving different rewards are revealed by their interaction trajectories in Supplementary Fig. 1. Each subplot features two rows of trajectories, where the first row is that for the agent receiving sparse reward signals and the second row corresponds to the agent learned with the discovered reward function. The results in Supplementary Fig. 1 suggest that agents given discovered reward function achieve their objectives more efficiently and effectively (e.g., in the *LunarLander-v2* and *CartPole-v1* tasks). These results are consistent with the experimental conclusions presented in Fig. 3(c-d) in the main text.

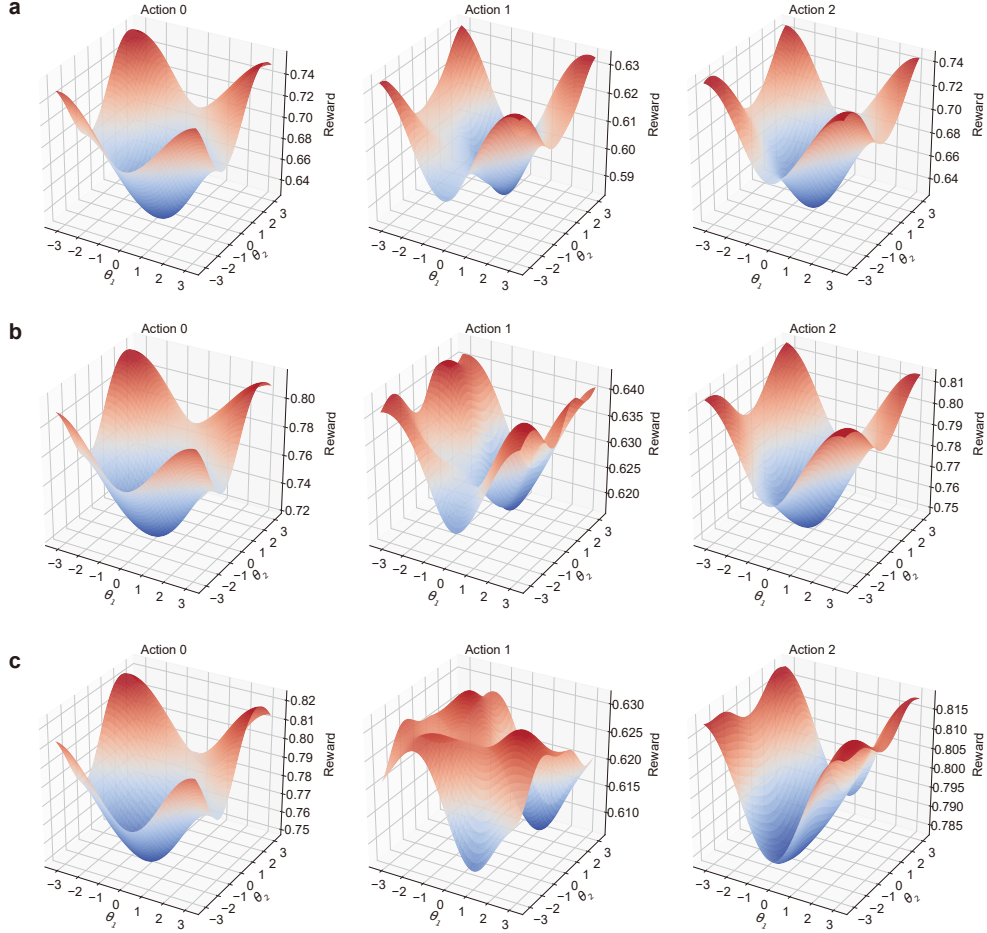


**Supplementary Fig. 1 The trajectories of embodied RL agents exploring tasks when receiving different rewards.** For each subplot, the first row represents the agent that learns with sparse reward signals, and the second row represents the agent that learns with the discovered reward function. **a** Interaction trajectory segments of the *CartPole-v1* task, sampled every 20 frames from the initial agent state. **b** The interaction trajectory segments before termination of the *CartPole-v1* task, sampled every 20 frames. **c** The interaction trajectory segments of the *Acrobot-v1* task, sampled every five frames from the initial agent state. **d** The interaction trajectory segments before termination of the *Acrobot-v1* task, sampled every five frames. **e** The complete interaction trajectories of the *FourRoom-v0* task, sampled at every frame. **f** The interaction trajectories of the *LunarLander-v2* task, sampled every eight frames. During these segments, the agent that receive reward signals from the discovered reward function successfully attains the goal, whereas the agent receiving sparse reward signals does not.

Fig. 4(e-g) in the main text illustrates the reward surfaces of the discovered reward function about the two joint angles  $\theta_1$  and  $\theta_2$  when the angular velocities  $\dot{\theta}_1$  and  $\dot{\theta}_2$  are 0. Supplementary Figs. 2-4 present the reward function as the joint angular velocities  $\dot{\theta}_1$  and  $\dot{\theta}_2$  vary from -2 to 2.

Supplementary Fig. 2 shows the reward distributions when  $\dot{\theta}_1 = -2$  and  $\dot{\theta}_2$  varies from -2 (Supplementary Fig. 2(a)) to 0 (Supplementary Fig. 2(b)) and 2 (Supplementary Fig. 2(c)). Our reward function guides nuanced, state-dependent action selection in the *Acrobot-v1* environment. At each fixed joint velocity, the surfaces show that the rewards when applying -1 or 1 torque (action 0 and 2) are highest when the joint angles deviate significantly from the

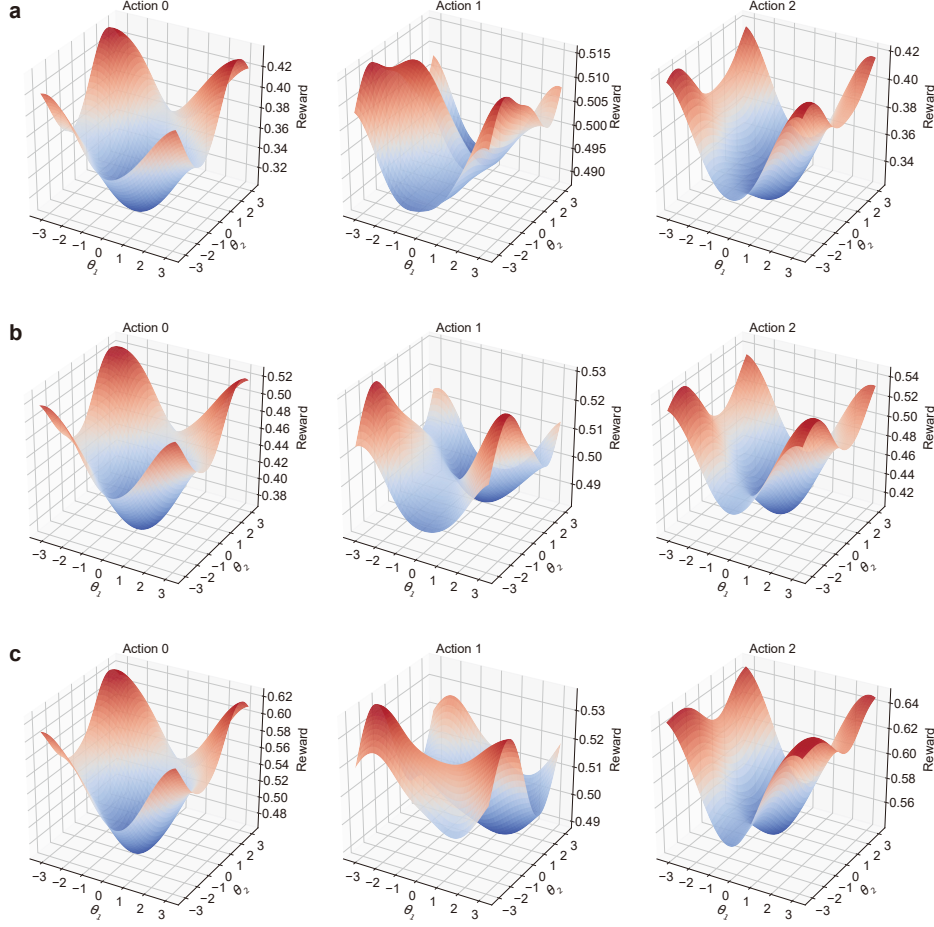
downward equilibrium position. It indicates that the agent is encouraged to generate momentum and initiate the swing-up motion when the system lies far from the goal. The smoothness of, and clear separation between, the action-reward regions in the state space show that the learned reward function not only captures the nonlinear dynamics but also creates meaningful gradients for all relevant state-action pairs. This allows the agent to switch adaptively between attainment of the goal and stabilization, leading to more effective and robust policy learning compared to sparse or naive rewards.



**Supplementary Fig. 2** Reward surfaces of the discovered reward function for different values of  $\dot{\theta}_2$  at fixed  $\dot{\theta}_1 = -2$  in the *Acrobot-v1* task. **a** Reward surfaces for all actions when  $\dot{\theta}_1 = -2$  and  $\dot{\theta}_2 = -2$ . **b** Reward surfaces for all actions when  $\dot{\theta}_1 = -2$  and  $\dot{\theta}_2 = 0$ . **c** Reward surfaces for all actions when  $\dot{\theta}_1 = -2$  and  $\dot{\theta}_2 = 2$ .

Supplementary Fig. 3 illustrates the learned reward surfaces for a fixed  $\dot{\theta}_1 = 0$  but a varying  $\dot{\theta}_2$ . This shows how our reward function adapts to the agent joint angle configurations and the velocity of the second joint. Compared to the previous setting with  $\dot{\theta}_1 = -2$ , the overall reward values are higher across all actions, indicating a shift in preference toward more conservative control when the first joint is stationary. In all subplots, the reward for action 0 (-1 torque) is elevated when the arms are more extended (i.e., when  $\theta_1, \theta_2$  are far from 0), encouraging the agent to initiate momentum when in spread-out configurations. Action 2 (1 torque) also shows high rewards in symmetric but mirrored regions, enabling directional control depending on the configuration. In contrast, action 1 (0 torque) receives the highest reward near neutral angular positions, especially when  $\dot{\theta}_2$  is small or positive (e.g., Supplementary Fig. 3(b) and

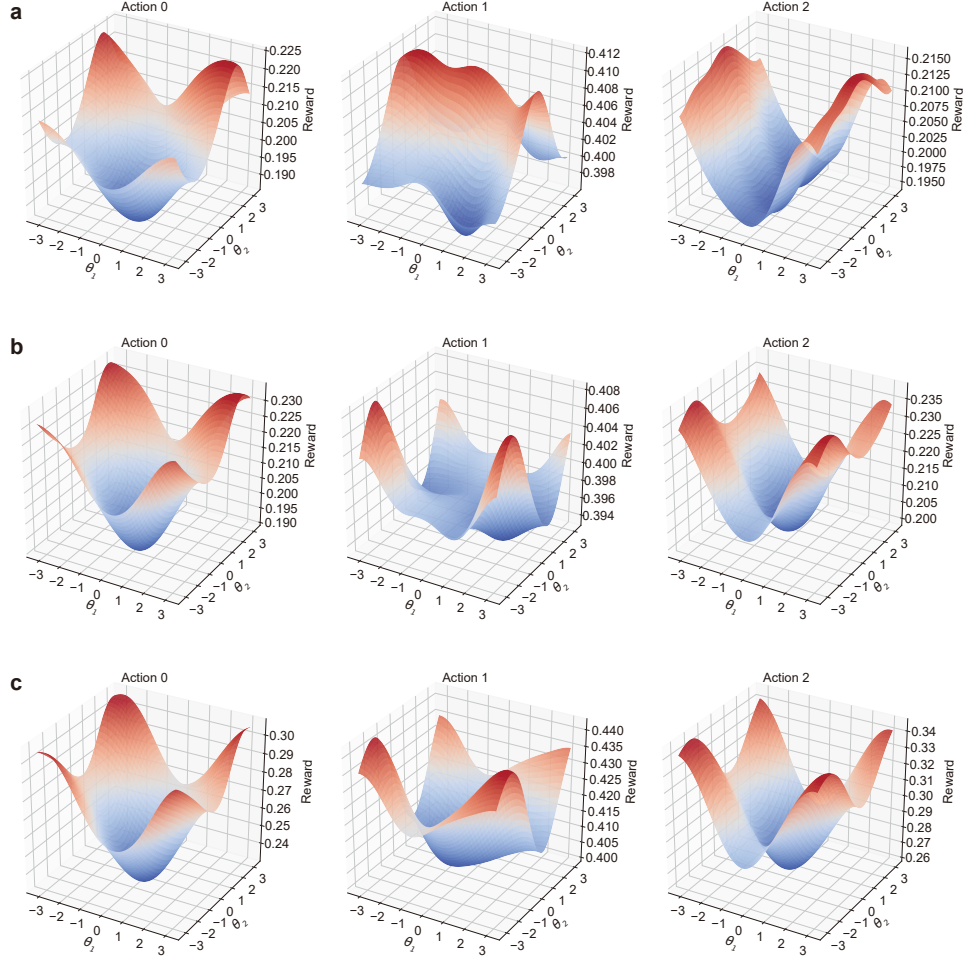
Supplementary Fig. 3(c)). This implies that the agent is encouraged to apply 0 torque when the system nears the goal region or already moves in the desired direction.



**Supplementary Fig. 3** Reward surfaces of the discovered reward function for different values of  $\dot{\theta}_2$  at fixed  $\dot{\theta}_1 = 0$  in the *Acrobot-v1* task. **a** Reward surfaces for all actions when  $\dot{\theta}_1 = 0$  and  $\dot{\theta}_2 = -2$ . **b** Reward surfaces for all actions when  $\dot{\theta}_1 = 0$  and  $\dot{\theta}_2 = 0$ . **c** Reward surfaces for all actions when  $\dot{\theta}_1 = 0$  and  $\dot{\theta}_2 = 2$ .

Supplementary Fig. 4 shows the reward surfaces learned under a fixed angular velocity  $\dot{\theta}_1 = 2$ , where the agent already possesses strong positive momentum. In this setting, action 0 and 2 exhibit higher and more structured rewards in regions with extreme joint angles, promoting active exploitation of momentum to reach the goal. Action 1 (0 torque) consistently receives lower rewards, reflecting its limited utility in high-energy states.

Compared to Supplementary Figs. 2-4, where  $\dot{\theta}_1$  varies from  $-2$  to  $2$ , we observe that the reward function engages in dynamic reshaping based on the angular velocity. When  $\dot{\theta}_1 = -2$ , the rewards function encourages aggressive torque to initiate swing-up; at  $\dot{\theta}_1 = 0$ , the reward tends to be close to the equilibrium action 1, indicating that the existing behavior is expected to be maintained when the agent reaches a limit or an equilibrium state; and at  $\dot{\theta}_1 = 2$ , the reward shifts back to torque-driven actions to sustain motion. This adaptive reward shaping enables the agent to select context-appropriate actions across varying dynamic regimes, offering more informative guidance than sparse or hand-tuned alternatives.

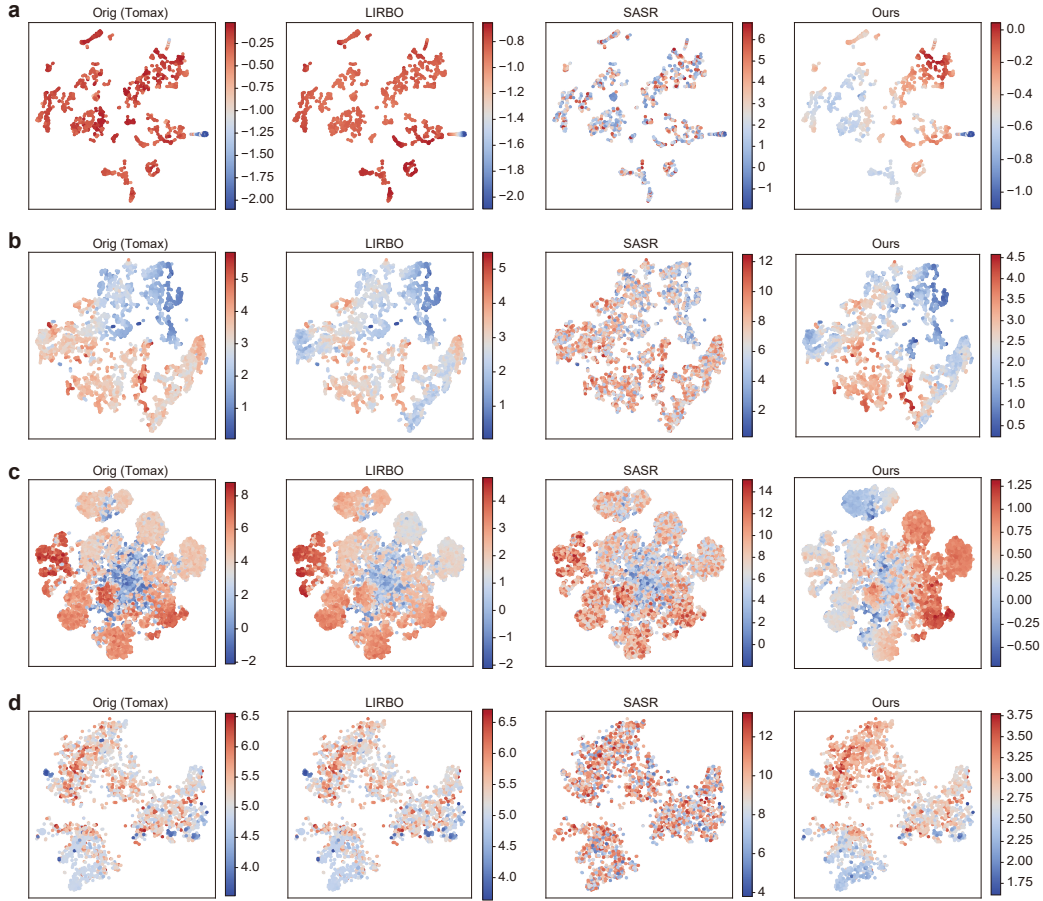


**Supplementary Fig. 4** Reward surfaces of the discovered reward function for different values of  $\dot{\theta}_2$  at fixed  $\dot{\theta}_1 = 2$  in the *Acrobot-v1* task. **a** Reward surfaces for all actions when  $\dot{\theta}_1 = 2$  and  $\dot{\theta}_2 = -2$ . **b** Reward surfaces for all actions when  $\dot{\theta}_1 = 2$  and  $\dot{\theta}_2 = 0$ . **c** Reward surfaces for all actions when  $\dot{\theta}_1 = 2$  and  $\dot{\theta}_2 = 2$ .

## Supplementary Experimental Results of High-Dimensional Control Task

In this section, we describe the reward function obtained by the proposed approach and the comparative methods in the main text. It should be pointed out that Tomax [66] did not feature a reward function because no such function was designed during RL training. Tomax changed the way the reward function was used from maximizing the cumulative return to maximizing the maximum historical reward signal. Therefore, the reward distribution of Tomax is the same as the Orig (a hand-designed reward function).

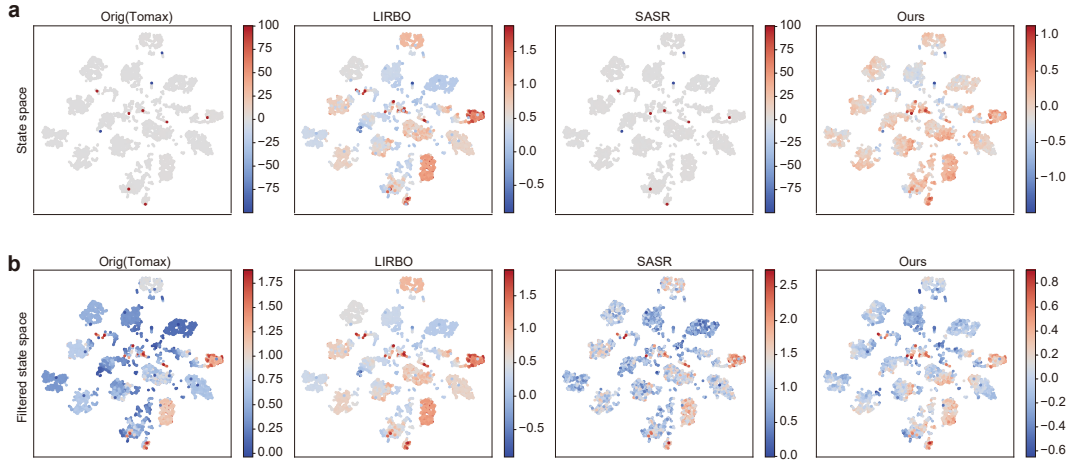
As shown in Supplementary Fig. 5, since LIRBO learns an intrinsic reward function that maximizes the agent performance under the external reward function, the learned reward distribution is very similar to that of the hand-designed external reward function. Moreover, as LIBRO is constrained by the external reward, it is difficult to address deficiencies that are not considered during manual reward design. SASR, in contrast, adds a shaping function to the hand-designed reward signal. Although this would appear to be effective in principle, the shaped reward signal is not smooth, which may introduce extra noise during learning. The reward distribution of Tomax is identical to that of the hand-designed reward function.



**Supplementary Fig. 5** Reward distributions in the state space, each point represents a state projected onto a 2D space via the t-SNE method. The color of each point indicates the reward, with red denoting higher rewards and blue representing lower rewards. **a** The reward distribution of the different reward function in *Reacher-v2*. **b** The reward distribution of the different reward function in *Hopper-v2*. **c** The reward distribution of the different reward function in *HalfCheetah-v2*. **d** The reward distribution of the different reward function in *Humanoid-v2*.

## Supplementary Experimental Results of Real-world tasks

Supplementary Fig. 6 shows the distributions of reward functions in the original and filtered state space derived using different methods. Supplementary Fig. 6(a) shows that the original reward function assigns reward signals that are either too high or too low to the critical states. Even in the filtered state space of Supplementary Fig. 6(b), the original reward remains uniform and non-informative. The reward does not distinguish useful states. LIRBO replaces the original reward function by learning an intrinsic reward function, and it exhibits a distribution similar to the external reward signal. The SASR auxiliary reward function is directly added to the external reward, preserving recognition of critical states. The reward function discovered in this work possesses all desired qualities. It recognizes the critical states and the distribution is similar to that of the hand-designed reward function, demonstrating that the reward function learns the dynamics behind the task.



**Supplementary Fig. 6** Reward distributions in the state space, each point represents a state projected onto a 2D space via the t-SNE method. The color of each point indicates the reward, with red denoting higher rewards and blue representing lower rewards. **a** The reward distribution of the different reward function in original state space. **b** The reward distribution of the different reward function in filtered state space.

## Supplementary Methods

In this section, the algorithm underlying the proposed approach is discussed. The pseudocode for the discovery of the optimal RL reward function is that of Algorithm 3. We begin by introducing our data-driven model optimization methods, which are summarized in Algorithm 1. These methods leverage empirical data to improve model performance and facilitate better decision-making. Typically, data-driven optimization commences with collection of relevant datasets (generated via RL in this work) that may include historical performance metrics, user interactions, or simulated outcomes.

---

### Algorithm 1 Data-driven method for model optimization

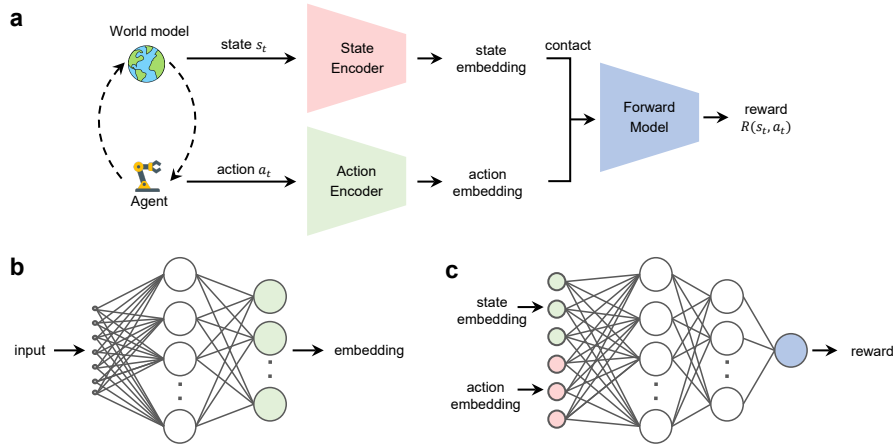
---

**Input:** Dataset  $(X, Y)$ , batch size  $B$ , learning rate  $\alpha$ , epochs  $K$ , model  $f(\cdot; \phi)$

**Output:** Optimized model parameters  $\phi$

- 1: Initialize model parameters  $\phi$  randomly
  - 2: **for** epoch  $k = 1$  to  $K$  **do**
  - 3:   Shuffle the training dataset  $(X, Y)$
  - 4:   **for**  $t = 0$  to  $N/B$  **do**
  - 5:     Select a mini-batch  $(X_{batch}, Y_{batch})$  of size  $B$
  - 6:     Calculate the predictions  $\hat{Y}_{batch} = f(X_{batch}; \phi)$
  - 7:     Calculate the loss  $L = \frac{1}{B} \sum_{i=1}^B \ell(Y_{batch}^i, \hat{Y}_{batch}^i)$
  - 8:     Calculate the gradients  $\nabla_{\phi} L$
  - 9:     Update the parameters:  $\phi \leftarrow \phi - \alpha \cdot \nabla_{\phi} L$
  - 10:   **end for**
  - 11: **end for**
- 

Next, implementation of the reward function optimization step of the proposed approach is outlined. This corresponds to the upper-level of the framework depicted in Fig. 2(d) in the main text. In our work, the reward function is constructed by DNNs, as shown in Supplementary Fig. 7. The reward function separately encodes the observational state of the world model and the action of the embodied RL agent and then concatenates them. The concatenated embeddings are subsequently processed through a forward model to generate the reward signal.



**Supplementary Fig. 7 Architecture of the reward function utilized in the main text.** **a** Overview of the architecture of the reward function, including the state encoder, the action encoder, and the forward model that generate the reward signal. **b** The state encoder and action encoder are fully connected neural networks that receive states or actions as inputs and then produce their respective embeddings. **c** The forward model concatenates the outputs of the state encoder and action encoder and processes them using a deep neural network to generate the reward signal. Icons from Icons8.com.

---

**Algorithm 2** The reward function optimization step in upper-level optimization

---

**Input:** Trajectory buffer  $\mathcal{D}$ , reward function  $R_\omega$ , set objective reward  $\bar{R}$

**Output:** Updated reward function  $R_\omega$

- 1: Decompose the trajectories in  $\mathcal{D}$  into steps
  - 2: Randomly sample  $N$  steps from  $\mathcal{D}$  to create the subset  $\mathcal{D}_\tau = \{\tau_1, \tau_2, \dots, \tau_N\}$
  - 3: Calculate the standard return  $\bar{U}$  under setting  $\bar{R}$
  - 4: Construct a model  $V$  that approximates the standard state values
  - 5: **for** step  $\tau^{(i)} = (s^{(i)}, a^{(i)}, r_\omega^{(i)}, \bar{U}^{(i)}, \mu^{(i)})$  in  $\mathcal{D}_\tau$  **do**
  - 6:     Derive the action probability  $\pi(s^{(i)}, a^{(i)})$  from the policy distribution  $\mu^{(i)}$
  - 7:     Estimate the state value  $\bar{V}(s^{(i)})$  employing the value model  $V$
  - 8:     Sample  $K$  actions from the action space to form the set  $\mathcal{A}$
  - 9:     **for** action  $a'_k$  in  $\mathcal{A}$  **do**
  - 10:         Calculate the reward  $r_k(s^{(i)}, a'_k)$  using the reward function  $R_\omega$
  - 11:         Derive the action probability  $\pi(s^{(i)}, a'_k)$  from the distribution  $\mu^{(i)}$
  - 12:     **end for**
  - 13:     Estimate the reward center  $r_c^{(i)}$  over the sampled actions:  $r_c^{(i)} = \frac{1}{K} \sum_{k=1}^K \pi(s^{(i)}, a'_k) \cdot r_k(s^{(i)}, a'_k)$
  - 14:     Calculate the gradient term of  $\tau$ :  $g_{\tau_i} = (\bar{V}(s^{(i)}) - \bar{U}^{(i)}) \cdot (r_\omega^{(i)} - r_c^{(i)})$
  - 15:     **end for**
  - 16: Estimate the expected gradient:  $\nabla_\omega[\mathcal{J}^{upper}]_\omega = \frac{1}{N} \sum_{i=1}^N \pi(s^{(i)}, a^{(i)}) \cdot g_{\tau_i}$
  - 17: Update the reward function:  $\omega' \leftarrow \omega + \beta \cdot \nabla_\omega[\mathcal{J}^{upper}]_\omega$
- 

In Algorithm 2, it is assumed that all interactions between the embodied RL agent and the world model are complete, and that the corresponding data are stored in the buffer. Reward function optimization involves several critical steps, including action sampling (line 9), gradient accumulation (line 16), and model updating (line 17). Initially, trajectories from buffer  $\mathcal{D}_\xi$  are decomposed into their individual steps  $\tau$ , constructed as tuples  $(s, a, r_\omega, \bar{U}, \mu)$ . A subset of  $N$  steps is then randomly sampled for model optimization. For each step  $\tau$ , the action probability is derived from the current policy, and the standard state value  $\bar{V}(s)$  is estimated via the maintained value model. A set of actions is sampled from the policy distribution, and for each sampled action  $a' \in \mathcal{A}$ , the reward  $r_\omega(s, a')$  is obtained from the reward function  $R_\omega$ , along with the corresponding action probability  $\pi(s, a')$ . The reward center  $r_c$  and gradient terms  $g_\tau$  are then computed, as shown in Algorithm 2. Then, we accumulate the gradient over the sampled steps (line 14 in Algorithm 2). We rewrite the term of equation (S30) as follows:

$$\begin{aligned} \nabla_\omega[\mathcal{J}^{upper}]_\omega &= \mathbb{E}_{\mu_{s,a}} \left[ \nabla_\omega \left[ A_R^{\pi_\theta}(s, a) \cdot A_{R_\omega}^{\pi_\theta}(s, a) \right]_\omega \right] \\ &\approx \frac{1}{N} \sum_{\tau \in \mathcal{D}_\tau} [\pi(s, a) \cdot (\bar{V}(s) - \bar{U}) \cdot (r_\omega - r_c)]. \end{aligned} \quad (\text{S32})$$

Ultimately, the reward function  $R_\omega$  is updated on the basis of Theorem 3, ensuring joint optimization of the reward and value models. By iterating through these steps, the reward function is effectively refined.

$$\omega' \leftarrow \omega + \beta \cdot \nabla_\omega[\mathcal{J}^{upper}]_\omega. \quad (\text{S33})$$

Application of the optimal reward function discovery approach within RL is detailed in Algorithm 3. This approach is formulated within a general RL framework, thus independent of specific RL algorithms. The method commences with initialization of the policy model  $\pi$  and the reward function  $R$  using randomly assigned weights. During each episode, the state is initialized on the basis of the world model  $\mathcal{M}$ , and a loop is executed until a terminal state is attained. Within this loop, the RL agent derives the action distribution  $\mu$  from the policy model, samples an action, executes it, observes the subsequent state from the viewpoint of the world model  $\mathcal{M}$ , and receives reward  $r$  from the reward function  $R_\omega$ . The policy model is

then updated via various RL methods (line 12 in Algorithm 3). Although the specifics of the different RL methods vary in terms of policy optimization, the proposed approach is unaffected by such variations. On completion of an episode, the trajectory is stored in the reward buffer  $\mathcal{D}_\xi$ . The reward function is subsequently refined using the method outlined in Algorithm 2, which ensures continuous refinement. By integrating these components, the proposed approach efficiently discovers the reward function  $R^*$ , thereby improving agent performance in complex environments. Notably, compared to standard RL methods, the proposed approach requires only a single additional line of code (line 15 in Algorithm 3) to enable co-optimization of the reward function and policy model, disregarding other implementation details.

---

**Algorithm 3** Approach for discovering the optimal reward function for embodied RL agents

---

**Input:** World model  $\mathcal{M}$ , agent policy  $\pi$ , reward function  $R$ , trajectory buffer  $\mathcal{D}_\xi$

**Output:** The optimal policy  $\pi^*$ , the optimal reward  $R^*$

- 1: Initialize the policy  $\pi$  with random weights  $\theta_0$
  - 2: Initialize the reward function  $R$  with random weights  $\omega_0$
  - 3: Initialize the trajectory buffer  $\mathcal{D}_\xi$  as empty
  - 4: **while** policy and reward are not optimal **do**
  - 5:     Initialize state  $s$  from world model  $\mathcal{M}$
  - 6:     **while** not terminal state **do**
  - 7:         Obtain the action distribution  $\mu$  from policy model  $\pi_\theta$
  - 8:         Make a decision and perform an action  $a$
  - 9:         Observe the next state  $s'$  in world model  $\mathcal{M}$
  - 10:        Calculate the reward  $r_\omega$  using the reward function  $R_\omega$  on the basis of state  $s$  and action  $a$
  - 11:        Transfer the state to  $s'$  from  $s$
  - 12:        Update the policy model  $\pi_{\theta'} \leftarrow \pi_\theta$     $\triangleright$  Lower-level: Update the policy based on the RL method
  - 13:     **end while**
  - 14:     Store the trajectory  $\xi$  of this episode in buffer  $\mathcal{D}_\xi$
  - 15:     Update the reward function  $R_{\omega'} \leftarrow R_\omega$     $\triangleright$  Upper-level: Update the reward based on Algorithm 2
  - 16: **end while**
-

## Supplementary Experiment Details

### Detailed RL Algorithms

Algorithm 3 presents the approach used to integrate the discovery of the optimal reward function with the embodied RL. In this section, the learning process of embodied RL agents described in the main text is further discussed employing Algorithms 4–7, including the DQN (Algorithm 4), PPO (Algorithm 5), SAC (Algorithm 6), and TD3 (Algorithm 7). The RL agents used in the main text are sourced from CleanRL [67]. Rather than elaborating on the technical details of the different types of agents, we demonstrate that the proposed optimal reward function discovery approach can be rapidly applied to most embodied RL agents.

---

**Algorithm 4** Approach for discovering the optimal reward function for DQN [68] agent

---

**Input:** World model  $\mathcal{M}$ , Q-network  $Q_\theta$ , target network  $Q_{\theta'}$ , reward function  $R_\omega$ , replay buffer  $\mathcal{D}$ , trajectory buffer  $\mathcal{D}_\xi$ , batch size  $B$ , discount factor  $\gamma$ , max steps  $T$ , policy optimization steps  $t_p$ , reward optimization steps  $t_r$ , target optimization steps  $t_c$ , learning rate  $\alpha$

**Output:** Trained DQN agent

```
1: Initialize the Q-network  $Q_\theta$  with random weights
2: Initialize the target network  $Q_{\theta'} \leftarrow Q_\theta$ 
3: Initialize the reward function  $R_\omega$  with random weights
4: Initialize the replay buffer  $\mathcal{D}$  and trajectory buffer  $\mathcal{D}_\xi$  as empty
5: Initialize the exploration rate  $\epsilon$ 
6: for  $t = 1$  to  $T$  do
7:   Observe the current state  $s$  from world model  $\mathcal{M}$ 
8:   Select action  $a$  with probability  $\epsilon$  or  $\arg \max_a Q_\theta(s, a)$ 
9:   Execute action  $a$ , and observe the next state  $s'$  of world model  $\mathcal{M}$ 
10:  Obtain reward  $r$  from reward function  $R_\omega$ 
11:  Store the transition  $(s, a, r, s')$  in  $\mathcal{D}$ 
12:  if terminal state then
13:    Calculate the cumulative return  $\bar{G}$  based on the set reward  $\bar{R}$  for each step of the
    trajectory
14:    Store the trajectory in  $\mathcal{D}_\xi$ 
15:  end if
16:  if  $t \bmod t_p = 0$  then
17:    Sample a mini-batch of transitions from  $\mathcal{D}$ 
18:    Compute the target  $y = r + \gamma \cdot \max_{a'} Q_{\theta'}(s', a')$ 
19:    Update  $\theta$  by minimizing the loss:  $\mathcal{L}(\theta) = \frac{1}{B} \sum (y - Q_\theta(s, a))^2$ 
20:  end if
21:  if  $t \bmod t_r = 0$  then
22:    Sample a mini-batch of trajectories from  $\mathcal{D}_\xi$ 
23:    Compute the gradient and update  $\omega$  via Algorithm 2:  $R_{\omega'} \leftarrow R_\omega$ 
24:  end if
25:  if  $t \bmod t_c = 0$  then
26:    Update the target network:  $Q_{\theta'} \leftarrow Q_\theta$ 
27:  end if
28:  Reduce the exploration rate  $\epsilon$ 
29: end for
```

---

---

**Algorithm 5** Approach for discovering the optimal reward function for PPO [69] agent

---

**Input:** World model  $\mathcal{M}$ , policy network  $\pi_\theta$ , value function  $V_\phi$ , reward function  $R_\omega$ , clip parameter  $\epsilon$ , learning rates  $\alpha_\pi$ ,  $\alpha_V$ , and  $\alpha_r$ , policy optimization steps  $T$ , discount factor  $\gamma$ , trajectory buffer  $\mathcal{D}_\xi$ , number of epochs  $K$ , mini-batch size  $M$

**Output:** Trained PPO agent

- 1: Initialize the policy parameters  $\theta$  and  $\phi$
- 2: Initialize the reward parameters  $\omega$
- 3: Initialize the world model and obtain the initial state  $s_0$
- 4: **while** not converged **do**
- 5:   Collect a trajectory  $\{(s_t, a_t, r_t, s_{t+1})\}$  of length  $T$  by acting in the world model via  $\pi_\theta$
- 6:   Calculate the cumulative return  $\bar{G}$  based on the set reward  $\bar{R}$  for each step of the trajectory
- 7:   Store the trajectory in buffer  $\mathcal{D}_\xi$
- 8:   Compute the advantages  $\hat{A}_t$  via generalized advantage estimation (GAE):

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}$$

- 9:   Compute the target values for the value function:  $\hat{R}_t = r_t + \gamma \hat{R}_{t+1}$
- 10:   Compute the old action probabilities as  $\pi_\theta: \pi_\theta(a_t|s_t)$
- 11:   **for** epoch  $k = 1$  to  $K$  **do**
- 12:     Shuffle the collected trajectories into mini-batches of size  $M$
- 13:     **for** each mini-batch **do**
- 14:       Compute the policy ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

- 15:       Compute the clipped surrogate objective:

$$L^{\text{clip}}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]$$

- 16:       Compute the value loss:  $L^{\text{value}}(\phi) = \mathbb{E}_t \left[ \left( V_\phi(s_t) - \hat{R}_t \right)^2 \right]$
- 17:       Compute the total loss with the entropy regularization term  $H$ :

$$L^{\text{total}} = L^{\text{clip}}(\theta) - c_1 L^{\text{value}}(\phi) + c_2 H(\pi_\theta)$$

- 18:       Update the policy parameters:  $\theta \leftarrow \theta + \alpha_\pi \nabla_\theta L^{\text{total}}$
  - 19:       Update the value function parameters:  $\phi \leftarrow \phi - \alpha_V \nabla_\phi L^{\text{value}}$
  - 20:     **end for**
  - 21:   **end for**
  - 22:   Sample a mini-batch of trajectories from  $\mathcal{D}_\xi$
  - 23:   Compute the gradient and update  $\omega$  on the basis of Algorithm 2:  $\omega \leftarrow \omega + \alpha_r \cdot \nabla_\omega \mathcal{J}^{\text{upper}}$
  - 24: **end while**
-

---

**Algorithm 6** Approach for discovering the optimal reward function for SAC [70] agent

---

**Input:** World model  $\mathcal{M}$ , policy network  $\pi_\phi$ , two Q-value networks  $Q_{\theta_1}$  and  $Q_{\theta_2}$ , two target networks  $\bar{Q}_{\theta_1}$  and  $\bar{Q}_{\theta_2}$ , reward function  $R_\omega$ , temperature parameter  $\alpha$ , learning rates  $\alpha_\pi, \alpha_Q, \alpha_\alpha$ , and  $\alpha_r$ , replay buffer  $\mathcal{D}$ , trajectory buffer  $\mathcal{D}_\xi$ , policy optimization steps  $t_p$ , reward optimization steps  $t_r$ , discount factor  $\gamma$ , target smoothing coefficient  $\tau$

**Output:** Trained SAC agent

- 1: Initialize parameters  $\phi$ ,  $\theta_1$ , and  $\theta_2$  and target networks  $\bar{\theta}_1 \leftarrow \theta_1$ ,  $\bar{\theta}_2 \leftarrow \theta_2$
- 2: Initialize reward parameters  $\omega$
- 3: Initialize replay buffer  $\mathcal{D}$  and trajectory buffer  $\mathcal{D}_\xi$
- 4: Initialize the world model and determine the initial state  $s_0$
- 5: **for**  $t = 1$  to  $T$  **do**
- 6:     Collect action  $a_t \sim \pi_\phi(a_t|s_t)$  and observe the next state  $s_{t+1}$  from the world model  $\mathcal{M}$
- 7:     Obtain the reward  $r_t$  from the reward function  $R_\omega$
- 8:     Store the transition  $(s_t, a_t, r_t, s_{t+1})$  in replay buffer  $\mathcal{D}$
- 9:     **if** terminal state **then**
- 10:         Calculate the cumulative return  $\bar{G}$  based on the set reward  $\bar{R}$  for each step of the trajectory
- 11:         Store the trajectory in  $\mathcal{D}_\xi$
- 12:     **end if**
- 13:     **if**  $t \bmod t_p = 0$  **then**
- 14:         Sample a mini-batch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $\mathcal{D}$
- 15:         Compute the target for the soft Q value:

$$y_i = r_i + \gamma \mathbb{E}_{a_{i+1} \sim \pi_\phi} [\min(\bar{Q}_{\theta_1}(s_{i+1}, a_{i+1}), \bar{Q}_{\theta_2}(s_{i+1}, a_{i+1})) - \alpha \log \pi_\phi(a_{i+1}|s_{i+1})]$$

- 16:         Update the Q-function parameters by minimizing the loss:

$$L_Q(\theta) = \frac{1}{N} \sum_i (Q_\theta(s_i, a_i) - y_i)^2$$

$$\theta_1 \leftarrow \theta_1 - \alpha_Q \nabla_{\theta_1} L_Q, \quad \theta_2 \leftarrow \theta_2 - \alpha_Q \nabla_{\theta_2} L_Q$$

- 17:         Update the policy network by minimizing the objective:

$$L_\pi(\phi) = \frac{1}{N} \sum_i \mathbb{E}_{a_i \sim \pi_\phi} [\alpha \log \pi_\phi(a_i|s_i) - Q_{\theta_1}(s_i, a_i)]$$

$$\phi \leftarrow \phi - \alpha_\pi \nabla_\phi L_\pi$$

- 18:         Adjust the temperature parameter  $\alpha$ :

$$L_\alpha(\alpha) = -\frac{1}{N} \sum_i \mathbb{E}_{a_i \sim \pi_\phi} [\alpha (\log \pi_\phi(a_i|s_i) + \mathcal{H}_{\text{target}})]$$

$$\alpha \leftarrow \alpha - \alpha_\alpha \nabla_\alpha L_\alpha$$

- 19:         Update the target networks:

$$\bar{\theta}_1 \leftarrow \tau \theta_1 + (1 - \tau) \bar{\theta}_1, \quad \bar{\theta}_2 \leftarrow \tau \theta_2 + (1 - \tau) \bar{\theta}_2$$

- 20:     **end if**
  - 21:     **if**  $t \bmod t_r = 0$  **then**
  - 22:         Sample a mini-batch of trajectories from  $\mathcal{D}_\xi$
  - 23:         Compute the gradient and update  $\omega$  via Algorithm 2:  $\omega \leftarrow \omega + \alpha_r \cdot \nabla_\omega \mathcal{J}^{\text{upper}}$
  - 24:     **end if**
  - 25: **end for**
-

---

**Algorithm 7** Approach for discovering the optimal reward function for TD3 [71] agent

---

**Input:** World model  $\mathcal{M}$ , actor network  $\pi_\phi$ , two critic networks  $Q_{\theta_1}$  and  $Q_{\theta_2}$ , target networks  $\pi_{\bar{\phi}}$ ,  $\bar{Q}_{\theta_1}$ , and  $\bar{Q}_{\theta_2}$ , reward function  $R_\omega$ , replay buffer  $\mathcal{D}$ , trajectory buffer  $\mathcal{D}_\xi$ , exploration noise  $\mathcal{N}$ , target policy smoothing noise  $\mathcal{N}_{\text{target}}$ , discount factor  $\gamma$ , soft update rate  $\tau$ , policy optimization steps  $t_p$ , reward optimization steps  $t_r$ , delay parameter  $d$ , mini-batch size  $M$ , learning rates  $\alpha_\pi$ ,  $\alpha_Q$ , and  $\alpha_r$

**Output:** Trained TD3 agent

- 1: Initialize parameters  $\phi$ ,  $\theta_1$ ,  $\theta_2$  and target networks  $\bar{\phi} \leftarrow \phi$ ,  $\bar{\theta}_1 \leftarrow \theta_1$ ,  $\bar{\theta}_2 \leftarrow \theta_2$
- 2: Initialize replay buffer  $\mathcal{D}$  and trajectory buffer  $\mathcal{D}_\xi$
- 3: Initialize the environment and determine the initial state  $s_0$
- 4: **for**  $t = 1$  to  $T$  **do**
- 5:   Select action  $a_t = \pi_\phi(s_t) + \mathcal{N}$ , clip if necessary
- 6:   Execute  $a_t$ , observe the next state  $s_{t+1}$
- 7:   Obtain reward  $r_t$  from reward function  $R_\omega$
- 8:   Store the transition  $(s_t, a_t, r_t, s_{t+1})$  in  $\mathcal{D}$
- 9:   **if** terminal state **then**
- 10:     Calculate the cumulative return  $\bar{G}$  based on the set reward  $\bar{R}$  for each step of the trajectory
- 11:     Store the trajectory in  $\mathcal{D}_\xi$
- 12:   **end if**
- 13:   **if**  $t \bmod t_p = 0$  **then**
- 14:     Sample a mini-batch of  $M$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $\mathcal{D}$
- 15:     Add smoothing noise to the target policy:

$$\tilde{a}_{i+1} = \pi_{\bar{\phi}}(s_{i+1}) + \text{clip}(\mathcal{N}_{\text{target}}, -c, c)$$

- 16:     Compute the target for the Q value:

$$y_i = r_i + \gamma \min(\bar{Q}_{\theta_1}(s_{i+1}, \tilde{a}_{i+1}), \bar{Q}_{\theta_2}(s_{i+1}, \tilde{a}_{i+1}))$$

- 17:     Update the critic networks by minimizing the loss:

$$L_Q(\theta) = \frac{1}{M} \sum_i (Q_\theta(s_i, a_i) - y_i)^2$$

$$\theta_1 \leftarrow \theta_1 - \alpha_Q \nabla_{\theta_1} L_Q, \quad \theta_2 \leftarrow \theta_2 - \alpha_Q \nabla_{\theta_2} L_Q$$

- 18:   **end if**
- 19:   **if**  $t \bmod d = 0$  **then**
- 20:     Update the actor network by maximizing the Q value:

$$L_\pi(\phi) = -\frac{1}{M} \sum_i Q_{\theta_1}(s_i, \pi_\phi(s_i))$$

$$\phi \leftarrow \phi - \alpha_\pi \nabla_\phi L_\pi$$

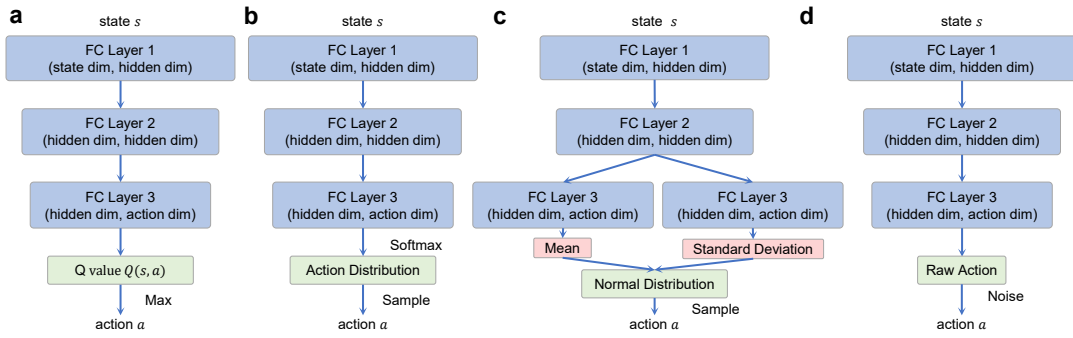
- 21:     Update the target networks:

$$\bar{\phi} \leftarrow \tau \phi + (1 - \tau) \bar{\phi}, \quad \bar{\theta}_1 \leftarrow \tau \theta_1 + (1 - \tau) \bar{\theta}_1, \quad \bar{\theta}_2 \leftarrow \tau \theta_2 + (1 - \tau) \bar{\theta}_2$$

- 22:   **end if**
  - 23:   **if**  $t \bmod t_r = 0$  **then**
  - 24:     Sample a mini-batch of trajectories from  $\mathcal{D}_\xi$
  - 25:     Compute the gradient and update  $\omega$  via Algorithm 2:  $\omega \leftarrow \omega + \alpha_r \cdot \nabla_\omega \mathcal{J}^{\text{upper}}$
  - 26:   **end if**
  - 27: **end for**
-

## Agent Structure and Hyperparameters setting

The agent architectures used in the main text are shown in Supplementary Fig. 8, including deep Q-network (DQN) [68], proximal policy optimization (PPO) [69], soft actor-critic (SAC) [70], and twin delayed deep deterministic policy gradient (TD3) [71]. In Supplementary Fig. 8(a), the DQN agent computes the Q-values  $Q(s, a)$  for all actions and selects the action with the highest value. In Supplementary Fig. 8(b), the discrete PPO agent outputs a probability distribution over the actions, from which an action is sampled. In Supplementary Fig. 8(c), the continuous PPO and SAC agents generate an action distribution by computing means and standard deviations, and then sampling actions accordingly. In Supplementary Fig. 8(d), the TD3 agent directly outputs an action with added noise to encourage exploration. Although the tasks differ, the agent architectures are similar. Depending on the complexity of the task, the architectural parameters can be scaled up to enhance the learning capacities of the embodied RL agents.



**Supplementary Fig. 8 Architecture of the agent utilized in the main text.** **a** The DQN agent, with the state serving as the input, obtains the value of each action and selects the action with the highest value. **b** The discrete PPO agent, again with the state serving as the input, obtains the probability distribution of each action and samples an action. **c** The continuous PPO and SAC agents, using the state as the input, construct an action distribution and sample an action from that distribution. **d** The TD3 agent, with the state serving as the input, directly obtains an action, as well as add a certain amount of noise.

The hyperparameters of the various agents and task parameter settings of the main text are listed in Supplementary Table 1. As stated in the main text, the experimental hyperparameters were not specifically fine-tuned for individual tasks. Multiple experiments were conducted to ensure comprehensive evaluation of the algorithms. Only the random seed changed across the experiments.

**Supplementary Table 1** The experimental parameter setting for all tasks.

| Parameter                   | Sparse-reward task           |                      | High-dimensional task |                    | Energy management task |                    | Unmanned systems task |                    |                    |                    |                    |
|-----------------------------|------------------------------|----------------------|-----------------------|--------------------|------------------------|--------------------|-----------------------|--------------------|--------------------|--------------------|--------------------|
|                             | DQN                          | PPO                  | PPO                   | SAC                | DQN                    | PPO                | SAC                   | PPO                | SAC                | TD3                |                    |
| Agent                       | hidden dim                   | [120, 84]            | [64, 64]              | [64, 64]           | [256, 256]             | [64, 32]           | [64, 64]              | [64, 64]           | [64, 64]           | [256, 256]         | [256, 256]         |
|                             | max time steps               | $5 \times 10^5$      | $5 \times 10^5$       | $1 \times 10^6$    | $1 \times 10^6$        | -                  | -                     | -                  | $1 \times 10^6$    | $1 \times 10^6$    | $1 \times 10^6$    |
|                             | buffer size                  | $1 \times 10^4$      | -                     | -                  | $1 \times 10^6$        | $1 \times 10^4$    | $1 \times 10^4$       | $1 \times 10^4$    | -                  | $1 \times 10^6$    | $1 \times 10^6$    |
|                             | batch size                   | 128                  | 64                    | 64                 | 256                    | 64                 | 64                    | 64                 | 2,048              | 256                | 256                |
|                             | learning rate                | $2.5 \times 10^{-4}$ | $1 \times 10^{-4}$    | $3 \times 10^{-4}$ | $3 \times 10^{-4}$     | $3 \times 10^{-4}$ | $3 \times 10^{-4}$    | $3 \times 10^{-4}$ | $3 \times 10^{-4}$ | $3 \times 10^{-4}$ | $3 \times 10^{-4}$ |
|                             | policy optimization steps    | 10                   | 1024                  | 2,048              | 2                      | -                  | -                     | -                  | 5,000              | 2                  | 2                  |
|                             | offline optimization rounds  | -                    | -                     | -                  | -                      | 200                | 200                   | 200                | -                  | -                  | -                  |
|                             | running time                 | -                    | -                     | -                  | -                      | 1 month            | 1 month               | 1 month            | -                  | -                  | -                  |
|                             | learning start steps         | 5,000                | -                     | -                  | 5,000                  | -                  | -                     | 5,000              | -                  | 5,000              | 25,000             |
|                             | target optimization steps    | 500                  | -                     | -                  | -                      | -                  | -                     | -                  | -                  | 1                  | -                  |
|                             | target smoothing coefficient | -                    | -                     | -                  | 0.05                   | -                  | -                     | -                  | -                  | 0.05               | 0.05               |
|                             | target network update rate   | 1                    | -                     | -                  | 1                      | -                  | -                     | -                  | -                  | 0.05               | 0.05               |
|                             | exploration noise rate       | 0.5                  | -                     | -                  | -                      | -                  | -                     | -                  | -                  | 0.2                | 0.1                |
|                             | policy training epochs       | -                    | 10                    | 10                 | -                      | -                  | 10                    | -                  | 10                 | -                  | -                  |
|                             | clip rate                    | -                    | 0.2                   | 0.2                | -                      | -                  | 0.2                   | -                  | 0.2                | -                  | -                  |
|                             | gae lambda                   | -                    | 0.95                  | 0.95               | -                      | -                  | -                     | -                  | 0.95               | -                  | -                  |
| entropy regularization rate | -                            | -                    | -                     | 0.2                | -                      | -                  | 0.2                   | -                  | 0.2                | -                  |                    |
| policy noise rate           | -                            | -                    | -                     | -                  | -                      | -                  | -                     | -                  | -                  | 0.2                |                    |
| noise clip rate             | -                            | -                    | -                     | -                  | -                      | -                  | -                     | -                  | -                  | 0.5                |                    |
| Reward                      | trajectory buffer size       | 100                  | 100                   | 100                | 100                    | 100                | 100                   | 100                | 100                | 100                | 100                |
|                             | learning rate                | $1 \times 10^{-4}$   | $3 \times 10^{-4}$    | $1 \times 10^{-4}$ | $1 \times 10^{-4}$     | $1 \times 10^{-4}$ | $1 \times 10^{-4}$    | $1 \times 10^{-4}$ | $3 \times 10^{-4}$ | $3 \times 10^{-4}$ | $3 \times 10^{-4}$ |
|                             | reward optimization steps    | 1,000                | 1,024                 | 2,048              | 2,048                  | -                  | -                     | -                  | 5,000              | 5,000              | 5,000              |
|                             | sample action number         | -                    | -                     | 1,000              | 1,000                  | -                  | -                     | -                  | 1,000              | 1,000              | 1,000              |

## Implementation Details

### Reward Function Discovery approach

All experiments employed a computing cluster with a NVIDIA GeForce RTX 4090. All experiments were run at least 5 times in parallel. All hyperparameters are listed in Supplementary Table 1. Specifically, we integrated reward function discovery into the standard RL procedure, using the “reward optimization steps” parameter to control how often the reward function was updated. Although the details varied slightly across the different RL agents (Algorithms 4–7), we made effort to ensure that our proposed reward function discovery method was broadly applicable and generalizable to a wide range of RL agents.

### Comparison Methods

In the experimental setup, several state-of-the-art reward function design methods were used for comparison. Specifically:

**SASR** [43] introduced a self-adaptive success rate-based reward shaping method to address the sparse-reward problem in RL, wherein the shaped reward functions were constructed by leveraging the success rates obtained from past interactions. These success rates were sampled from a dynamically updated beta distribution, which was efficiently estimated using kernel density estimation with random Fourier features. This approach enhanced the efficiency and adaptability of reward shaping in environments with sparse feedback.

**Tomax** [66] recognized the importance of reward maximization in RL, which focused on utilizing reward functions more efficiently. Specifically, a max-reward RL method was used to maximize the maximum reward encountered within an episode, rather than the cumulative return. This reformulation simplified the design and optimization of reward functions.

**LIRBO** [14] formulated the intrinsic RL reward learning problem as a bilevel optimization problem and employed the self-tuning networks approach for hyperparameter optimization. This method aimed to train the intrinsic reward function more efficiently. A best-response function was learned, and a neural network module was introduced. This engaged in conditional layer normalization that embed the parameters of the reward function into the policy network. This integration ensured a stable and efficient training process. By avoiding the high computational cost of backpropagation throughout the entire training process, LIRBO improved both the scalability and practicality of intrinsic reward learning.

### Reward Function Visualization Methods

In our approach, the reward function is parameterized (as shown in Supplementary Fig. 7). The weights are continuously updated via a meta-gradient based optimization. Therefore, it is challenging to write a closed-form analytical expression for the function. In sparse-reward tasks, the agent observations are simple and the action space is discrete, facilitating visualization of reward function surfaces in the state-action space. In high-dimensional control tasks and real-world tasks, both the observation space and the action space become significantly more complex and are typically continuous. To address this, we employed the t-SNE method [72] to project the high-dimensional state space onto two dimensions, enabling us to visualize the distribution of reward signals within the state space.

**Reward Curves.** Fig. 4(a-d) in the main text present the reward signal curves observed during a single interaction between the agent and the task. Specifically, we used a well-trained agent to interact with the task for one complete episode, and recorded the reward signals generated by different reward functions. This allowed us to compare the reward signals given to the agent at the same states when different reward function designs were used.

**Reward Surfaces.** We considered the *Acrobot-v1* task, a classic benchmark in RL that involves underactuated control. The system consists of a two-link pendulum with one end fixed and a single actuated joint between the two links. The goal is to apply one of three discrete  $-1, 0$ , or  $+1 \text{ N} \cdot \text{m}$  torques to the actuated joint to swing the free end of the outer link above a target height, starting from the initial state where both links hang downwards. The observation space includes the sines and cosines of both joint angles  $\theta_1, \theta_2$ , and their angular velocities  $\dot{\theta}_1, \dot{\theta}_2$ . This yields a 6-dimensional continuous state representation. The agent must

learn to coordinate oscillatory motions when building sufficient momentum to attain the goal. We systematically varied the two joint angles from  $-\pi$  to  $\pi$  while keeping the joint angular velocities fixed, thereby generating 6-dimensional states. In total, 100 states were generated. For each state, the reward function delivers reward signals associated with different actions, and the rewards are then used to construct the reward surfaces as functions of the two joint angles.

**t-SNE.** To visualize the distribution of reward signals in high-dimensional state spaces, we associate each high-dimensional state with the corresponding reward value and project these state representations into a 2-dimensional space using the t-distributed Stochastic Neighbor Embedding (t-SNE) method [72]. t-SNE is a nonlinear dimensionality reduction technique that preserves local data structure by mapping similar states close together in the 2D space. In our visualization, each point represents a state, and the color thereof the associated reward signal (red: higher rewards; blue: lower rewards). This enables an intuitive interpretation of how reward signals are distributed across complex, high-dimensional state spaces.

## Supplementary References

- [1] Zhang, H., Parkes, D.C., Chen, Y.: Policy teaching through reward function learning. In: Proceedings of the 10th ACM Conference on Electronic Commerce, pp. 295–304 (2009)
- [2] Hu, Y., Wang, W., Jia, H., Wang, Y., Chen, Y., Hao, J., Wu, F., Fan, C.: Learning to utilize shaping rewards: a new approach of reward shaping. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, pp. 15931–15941 (2020)
- [3] Devidze, R., Radanovic, G., Kamalaruban, P., Singla, A.: Explicable reward design for reinforcement learning agents. In: Proceedings of the 35th International Conference on Neural Information Processing Systems, pp. 20118–20131 (2021)
- [4] Icarte, R.T., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research* **73**, 173–208 (2022)
- [5] Gupta, D., Chandak, Y., Jordan, S.M., Thomas, P.S., Silva, B.C.: Behavior alignment via reward function optimization. In: Proceedings of the 37th International Conference on Neural Information Processing Systems, pp. 52759–52791 (2023)
- [6] Knox, W.B., Hatgis-Kessell, S., Adalgeirsson, S.O., Booth, S., Dragan, A., Stone, P., Niekum, S.: Learning optimal advantage from preferences and mistaking it for reward. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, pp. 10066–10073 (2024)
- [7] Singh, S., Lewis, R.L., Barto, A.G.: Where do rewards come from. In: Proceedings of the Annual Conference of the Cognitive Science Society, pp. 2601–2606 (2009)
- [8] Grunitzki, R., Silva, B.C., Bazzan, A.L.: A flexible approach for designing optimal reward functions. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, pp. 1559–1561 (2017)
- [9] Sorg, J., Singh, S., Lewis, R.L.: Reward design via online gradient ascent. In: Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 2, pp. 2190–2198 (2010)
- [10] Grunitzki, R., Silva, B.C., Bazzan, L.A.: Towards designing optimal reward functions in multi-agent reinforcement learning problems. In: International Joint Conference on Neural Networks, pp. 1–8 (2018)
- [11] Booth, S., Knox, W.B., Shah, J., Niekum, S., Stone, P., Allievi, A.: The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, pp. 5920–5929 (2023)
- [12] Banihashem, K., Singla, A., Gan, J., Radanovic, G.: Admissible policy teaching through reward design. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, pp. 6037–6045 (2022)
- [13] Devidze, R., Kamalaruban, P., Singla, A.: Informativeness of reward functions in reinforcement learning. In: Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, pp. 444–452 (2024)
- [14] Stadie, B., Zhang, L., Ba, J.: Learning intrinsic rewards as a bi-level optimization problem. In: Conference on Uncertainty in Artificial Intelligence, pp. 111–120 (2020)

- [15] Zheng, Z., Oh, J., Singh, S.: On learning intrinsic rewards for policy gradient methods. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 4649–4659 (2018)
- [16] Devidze, R., Kamalaruban, P., Singla, A.: Exploration-guided reward shaping for reinforcement learning under sparse rewards. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, pp. 5829–5842 (2022)
- [17] Palan, M., Shevchuk, G., Charles Landolfi, N., Sadigh, D.: Learning reward functions by integrating human demonstrations and preferences. In: Robotics: Science and Systems (2019)
- [18] Balakrishnan, S., Nguyen, Q.P., Low, B.K.H., Soh, H.: Efficient exploration of reward functions in inverse reinforcement learning via bayesian optimization. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, pp. 4187–4198 (2020)
- [19] Knox, W.B., Hatgis-Kessell, S., Booth, S., Niekum, S., Stone, P., Allievi, A.: Models of human preference for learning reward functions. preprint arXiv:2206.02231 (2022)
- [20] Sorg, J., Singh, S., Lewis, R.: Internal rewards mitigate agent boundedness. In: Proceedings of the 27th International Conference on Machine Learning, pp. 1007–1014 (2010)
- [21] Zheng, Z., Oh, J., Hessel, M., Xu, Z., Kroiss, M., Hasselt, H., Silver, D., Singh, S.: What can learned intrinsic rewards capture? In: Proceedings of the 37th International Conference on Machine Learning, pp. 11436–11446 (2020)
- [22] Fluri, L., Lang, L., Abate, A., Forré, P., Krueger, D., Skalse, J.: The perils of optimizing learned reward functions: Low training error does not guarantee low regret. preprint arXiv:2406.15753 (2024)
- [23] Xu, Z., Hasselt, H., Silver, D.: Meta-gradient reinforcement learning. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 2402–2413 (2018)
- [24] Oh, J., Hessel, M., Czarnecki, W.M., Xu, Z., Hasselt, H., Singh, S., Silver, D.: Discovering reinforcement learning algorithms. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, pp. 1060–1070 (2020)
- [25] Liu, X., Wu, J., Chen, S.: A context-based meta-reinforcement learning approach to efficient hyperparameter optimization. *Neurocomputing* **478**, 89–103 (2022)
- [26] Pong, V.H., Nair, A.V., Smith, L.M., Huang, C., Levine, S.: Offline meta-reinforcement learning with online self-supervision. In: Proceedings of the 39th International Conference on Machine Learning, pp. 17811–17829 (2022)
- [27] Ren, Z., Liu, A., Liang, Y., Peng, J., Ma, J.: Efficient meta reinforcement learning for preference-based fast adaptation. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, pp. 15502–15515 (2022)
- [28] Hattori, R., Hedrick, N.G., Jain, A., Chen, S., You, H., Hattori, M., Choi, J.-H., Lim, B.K., Yasuda, R., Komiyama, T.: Meta-reinforcement learning via orbitofrontal cortex. *Nature Neuroscience* **26**(12), 2182–2191 (2023)
- [29] Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel programming for hyperparameter optimization and meta-learning. In: Proceedings of the 35th International Conference on Machine Learning, pp. 1568–1577 (2018)

- [30] Clavera, I., Rothfuss, J., Schulman, J., Fujita, Y., Asfour, T., Abbeel, P.: Model-based reinforcement learning via meta-policy optimization. In: Conference on Robot Learning, pp. 617–629 (2018)
- [31] Li, K., Gupta, A., Reddy, A., Pong, V.H., Zhou, A., Yu, J., Levine, S.: Mural: Meta-learning uncertainty-aware rewards for outcome-driven reinforcement learning. In: Proceedings of the 38th International Conference on Machine Learning, pp. 6346–6356 (2021)
- [32] Nussenbaum, K., Hartley, C.A.: Understanding the development of reward learning through the lens of meta-learning. *Nature Reviews Psychology*, 1–15 (2024)
- [33] Zou, H., Ren, T., Yan, D., Su, H., Zhu, J.: Reward shaping via meta-learning. preprint arXiv:1901.09330 (2019)
- [34] Zou, H., Ren, T., Yan, D., Su, H., Zhu, J.: Learning task-distribution reward shaping with meta-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 11210–11218 (2021)
- [35] Du, Y., Han, L., Fang, M., Dai, T., Liu, J., Tao, D.: Liir: learning individual intrinsic reward in multi-agent reinforcement learning. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp. 4403–4414 (2019)
- [36] Chu, Z., Wang, N., Wang, H.: Multi-objective intrinsic reward learning for conversational recommender systems. In: Proceedings of the 37th International Conference on Neural Information Processing Systems, pp. 17702–17712 (2023)
- [37] Yang, C., Xu, P., Zhang, J.: Learning individual potential-based rewards in multi-agent reinforcement learning. *IEEE Transactions on Games*, 1–12 (2024)
- [38] Li, J., Hu, X., Xu, H., Liu, J., Zhan, X., Jia, Q.-S., Zhang, Y.-Q.: Mind the gap: Offline policy optimization for imperfect rewards. In: Proceedings of the 11th International Conference on Learning Representations (2023)
- [39] Liu, R., Bai, F., Du, Y., Yang, Y.: Meta-reward-net: implicitly differentiable reward learning for preference-based reinforcement learning. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, pp. 22270–22284 (2022)
- [40] Silver, D., Singh, S., Precup, D., Sutton, R.S.: Reward is enough. *Artificial Intelligence* **299**, 103535 (2021)
- [41] Lu, R., Hong, S.H., Yu, M.: Demand response for home energy management using reinforcement learning and artificial neural network. *IEEE Transactions on Smart Grid* **10**(6), 6629–6639 (2019)
- [42] Lu, R., Jiang, Z., Wu, H., Ding, Y., Wang, D., Zhang, H.-T.: Reward shaping-based actor-critic deep reinforcement learning for residential energy management. *IEEE Transactions on Industrial Informatics* **19**(3), 2662–2673 (2023)
- [43] Ma, H., Luo, Z., Vo, T.V., Sima, K., Leong, T.-Y.: Highly efficient self-adaptive reward shaping for reinforcement learning. In: Proceedings of the 13th International Conference on Learning Representations (2025)
- [44] Ng, A.Y., Harada, D., Russell, S.J.: Policy invariance under reward transformations: Theory and application to reward shaping. In: Proceedings of the 16th International Conference on Machine Learning, pp. 278–287 (1999)
- [45] Gupta, A., Pacchiano, A., Zhai, Y., Kakade, S.M., Levine, S.: Unpacking reward shaping:

- understanding the benefits of reward engineering on sample complexity. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, pp. 15281–15295 (2022)
- [46] Ng, A.Y., Russell, S.J.: Algorithms for inverse reinforcement learning. In: Proceedings of the 17th International Conference on Machine Learning, pp. 663–670 (2000)
  - [47] Wulfmeier, M., Rao, D., Wang, D.Z., Ondruska, P., Posner, I.: Large-scale cost function learning for path planning using deep inverse reinforcement learning. *The International Journal of Robotics Research* **36**(10), 1073–1087 (2017)
  - [48] Wu, F., Gu, Z., Wu, H., Wu, A., Zhao, Y.: Infer and adapt: Bipedal locomotion reward learning from demonstrations via inverse reinforcement learning. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 16243–16250 (2024)
  - [49] Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., Scaramuzza, D.: Champion-level drone racing using deep reinforcement learning. *Nature* **620**(7976), 982–987 (2023)
  - [50] Mankowitz, D.J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J.-B., Ahern, A., *et al.*: Faster sorting algorithms discovered using deep reinforcement learning. *Nature* **618**(7964), 257–263 (2023)
  - [51] Chen, L., Paleja, R., Gombolay, M.: Learning from suboptimal demonstration via self-supervised reward regression. In: Conference on Robot Learning, pp. 1262–1277 (2021)
  - [52] Xu, H., Zhan, X., Yin, H., Qin, H.: Discriminator-weighted offline imitation learning from suboptimal demonstrations. In: Proceedings of the 39th International Conference on Machine Learning, pp. 24725–24742 (2022)
  - [53] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., *et al.*: Training language models to follow instructions with human feedback. In: Proceedings of the 36th International Conference on Neural Information Processing Systems, pp. 27730–27744 (2022)
  - [54] Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A.M., Hauth, A., *et al.*: Gemini: a family of highly capable multimodal models. preprint arXiv:2312.11805 (2023)
  - [55] Daniel, C., Viering, M., Metz, J., Kroemer, O., Peters, J.: Active reward learning. In: *Robotics: Science and Systems*, vol. 98 (2014)
  - [56] Christiano, P.F., Leike, J., Brown, T.B., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 4302–4310 (2017)
  - [57] Ibarz, B., Leike, J., Pohlen, T., Irving, G., Legg, S., Amodei, D.: Reward learning from human preferences and demonstrations in atari. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 8022–8034 (2018)
  - [58] Büyük, E., Huynh, N., Kochenderfer, M.J., Sadigh, D.: Active preference-based gaussian process regression for reward learning and optimization. *The International Journal of Robotics Research* **43**(5), 665–684 (2024)
  - [59] An, G., Lee, J., Zuo, X., Kosaka, N., Kim, K.-M., Song, H.O.: Direct preference-based policy optimization without reward modeling. In: Proceedings of the 37th International Conference on Neural Information Processing Systems, pp. 70247–70266 (2023)

- [60] Hejna, J., Sadigh, D.: Inverse preference learning: preference-based rl without a reward function. In: Proceedings of the 37th International Conference on Neural Information Processing Systems, pp. 18806–18827 (2023)
- [61] Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C.D., Finn, C.: Direct preference optimization: your language model is secretly a reward model. In: Proceedings of the 37th International Conference on Neural Information Processing Systems, pp. 53728–53741 (2023)
- [62] Wallace, B., Dang, M., Rafailov, R., Zhou, L., Lou, A., Purushwalkam, S., Ermon, S., Xiong, C., Joty, S., Naik, N.: Diffusion model alignment using direct preference optimization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8228–8238 (2024)
- [63] Knox, W.B., Allievi, A., Banzhaf, H., Schmitt, F., Stone, P.: Reward (mis) design for autonomous driving. *Artificial Intelligence* **316**, 103829 (2023)
- [64] Yuan, M., Li, B., Jin, X., Zeng, W.: Automatic intrinsic reward shaping for exploration in deep reinforcement learning. In: Proceedings of the 40th International Conference on Machine Learning, pp. 40531–40554 (2023)
- [65] Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* **8**, 229–256 (1992)
- [66] Veviurko, G., Böhmer, W., De Weerd, M.: To the max: reinventing reward in reinforcement learning. In: Proceedings of the 41st International Conference on Machine Learning, pp. 49455–49470 (2024)
- [67] Huang, S., Dossa, R.F.J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., Araújo, J.G.M.: Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research* **23**(274), 1–18 (2022)
- [68] Mnih, V.: Playing atari with deep reinforcement learning. preprint arXiv:1312.5602 (2013)
- [69] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. preprint arXiv:1707.06347 (2017)
- [70] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th International Conference on Machine Learning, pp. 1861–1870 (2018)
- [71] Fujimoto, S., Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Proceedings of the 35th International Conference on Machine Learning, pp. 1587–1596 (2018)
- [72] Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. *Journal of Machine Learning Research* **9**, 2579–2605 (2008)