

# Test-driven Reinforcement Learning in Continuous Control

Zhao Yu<sup>\*1</sup>, Xiuping Wu<sup>\*2</sup>, Liangjun Ke<sup>1†</sup>

<sup>1</sup> The State Key Laboratory for Manufacturing Systems Engineering, School of Automation Science and Engineering, Xi'an Jiaotong University, No.28 Xianning West Road, Xi'an, Shaanxi 710049, P.R. China

<sup>2</sup>Southampton Business School, University of Southampton, Highfield Campus, SO17 1BJ, United Kingdom  
yuzhaokz@stu.xjtu.edu.cn, xiuping.wu@soton.ac.uk, keljxjtu@xjtu.edu.cn

## Abstract

Reinforcement learning (RL) has been recognized as a powerful tool for robot control tasks. RL typically employs reward functions to define task objectives and guide agent learning. However, since the reward function serves the dual purpose of defining the optimal goal and guiding learning, it is challenging to design the reward function manually, which often results in a suboptimal task representation. To tackle the reward design challenge in RL, inspired by the satisficing theory, we propose a Test-driven Reinforcement Learning (TdRL) framework. In the TdRL framework, multiple test functions are used to represent the task objective rather than a single reward function. Test functions can be categorized as pass-fail tests and indicative tests, each dedicated to defining the optimal objective and guiding the learning process, respectively, thereby making defining tasks easier. Building upon such a task definition, we first prove that if a trajectory return function assigns higher returns to trajectories closer to the optimal trajectory set, maximum entropy policy optimization based on this return function will yield a policy that is closer to the optimal policy set. Then, we introduce a lexicographic heuristic approach to compare the relative distance relationship between trajectories and the optimal trajectory set for learning the trajectory return function. Furthermore, we develop an algorithm implementation of TdRL. Experimental results on the DeepMind Control Suite benchmark demonstrate that TdRL matches or outperforms handcrafted reward methods in policy training, with greater design simplicity and inherent support for multi-objective optimization. We argue that TdRL offers a novel perspective for representing task objectives, which could be helpful in addressing the reward design challenges in RL applications.

**Code** — <https://github.com/KezhiAdore/TdRL>

**Extended version** — <https://arxiv.org/abs/2511.07904>

## 1 Introduction

For an intelligent agent, it is crucial to specify the objective that needs to be achieved (Silver et al. 2021; Silver and Sutton 2025). In classical reinforcement learning (RL),

the objective is typically specified through the reward functions, which steer the agent toward desired behaviors. (Sutton and Barto 2018) Crucially, reward design accounts for both defining optimal behavior and guiding the learning process, making it inherently challenging (Rajagopal 2023; Knox et al. 2023; Booth et al. 2023). Crafting effective reward functions typically demands domain-specific expertise and reward design experience (Knox et al. 2023; Booth et al. 2023). Nevertheless, the handcrafted reward often represents an imperfect proxy for the true optimization objective, as experts typically view reward as a direct evaluation of the relative goodness of each state-action pair instead of evaluating trajectories. (Booth et al. 2023). This mismatch induces fundamental challenges in reinforcement learning, such as reward hacking (Amodei et al. 2016).

To circumvent the challenges associated with manual reward engineering, several approaches have been developed, mainly including Preference-based RL (PbRL, Christiano et al. 2017), Inverse RL (IRL, Ziebart et al. 2008), and reward functions generated by Large Language Models (LLMs, Kwon et al. 2023). PbRL learns reward functions or directly optimizes policies based on human preference feedback, which avoids manual reward design and has demonstrated promising performance in robot control and LLM alignment (Christiano et al. 2017; Ouyang et al. 2022). However, PbRL's human-centric nature introduces several limitations (Casper et al. 2023). First, inherent biases in human preference data necessitate dedicated techniques to address them. Second, the reliance on human judgments may constrain the agent's behavior within human cognitive boundaries. IRL infers reward functions from expert demonstrations, thereby avoiding manual reward designing. However, IRL typically demands extensive expert demonstration data, and the inferred reward functions typically exhibit poor generalization capabilities when evaluated on out-of-distribution data (Arora and Doshi 2021). Recent studies have demonstrated that LLMs can outperform humans in designing reward functions (Kwon et al. 2023; Yu et al. 2023; Du et al. 2023). However, these approaches depend on a human-specified domain knowledge for reward design, and typically require extensive training feedback to polish the reward function (Cao et al. 2025).

Furthermore, real-world applications typically require agents to simultaneously optimize multiple objectives (Vam-

<sup>\*</sup>These authors contributed equally.

<sup>†</sup>Corresponding author

plew et al. 2023). This multi-objective optimization presents significant challenges for reward function design, particularly in balancing different objectives.

When solving real-world multi-objective tasks, humans often do not pursue the optimal solution in a certain metric but rather seek a *satisficing solution* (Simon 1947) across multiple objectives. For example, when driving, people do not blindly minimize time consumption, but rather reach the destination within a certain time while ensuring safety, comfort, and compliance with regulations.

Inspired by this, we propose a test-driven reinforcement learning (TdRL) framework. In the TdRL framework, the agent’s objective is passing given tests instead of maximizing the cumulative return, which implies that TdRL seeks to obtain a satisficing solution across multiple objectives rather than an optimal solution in a single metric. Test functions take trajectories as input and output the test results. According to their functionality, test functions can be categorized into two types: pass-fail tests and indicative tests. Pass-fail tests evaluate whether the policy meets the required criteria, while indicative tests provide informative guiding signals for policy learning. Pass-fail tests and indicative tests correspond to defining optimal behavior and guiding the learning process, respectively.

Specifically, the goal of TdRL is to train a policy such that its interaction trajectories with the environment pass all given pass-fail tests. To solve this problem, we first prove that if a trajectory return function assigns higher returns to trajectories closer to the optimal trajectory set (where the trajectory passes all given pass-fail tests), maximum entropy policy optimization based on this return function will yield a policy that is closer to the optimal policy set (where the trajectories generated by the policy pass all given pass-fail tests). Then, we introduce a lexicographic heuristic approach to compare the relative distance relationship between trajectories and the optimal trajectory set. Additionally, we develop an algorithm implementation of TdRL.

The TdRL framework provides several key benefits. First, test functions operate on trajectories instead of state-action pairs, mitigating designer-induced bias (Booth et al. 2023). Second, the task objective is represented by pass-fail tests rather than a scalar reward function, naturally accommodating multi-objective optimization. Finally, test results can evaluate policy performance more accurately than cumulative returns. The main contributions of this paper are summarized as follows:

- To address the reward design challenge, we propose a test-driven reinforcement learning framework, where the task objective is represented by several test functions instead of a single reward function.
- We propose sufficient conditions for trajectory return functions to guarantee policy convergence to the optimal policy set, and present a lexicographic heuristic approach for constructing return functions based on trajectory testing results.
- We develop an algorithm implementation of TdRL and conduct experiments on the DeepMind Control Suite benchmark to show the benefits of TdRL.

## 2 Related Work

**Test-driven reward design** has been adapted to RL to ensure robustness and correctness recently (Jaensch et al. 2022; Fischer et al. 2024). Inspired by the test-driven development principle (Beck 2002), Jaensch et al. (2022) proposed an approach that uses test cases to guide reward design, iteratively refining the policy until all tests are passed. Fischer et al. (2024) extended this idea to inverse reinforcement learning, replacing expert demonstrations with scenario-based testing to learn cost functions via Bayesian inference. Pranger et al. (2024) introduced a state-importance-driven testing method for deep RL, prioritizing high-impact states to efficiently detect policy vulnerabilities. These approaches demonstrate the potential of test-driven frameworks to enhance RL in applications.

**Preference-based reinforcement learning (PbRL)** has emerged to address the challenges of reward design. Instead of relying on handcrafted reward functions, PbRL (Christiano et al. 2017) learns reward models from human preferences over agent behaviors, which in turn guides policy optimization, achieving superior performance compared to handcrafted rewards. PEBBLE method (Lee, Smith, and Abbeel 2021) enhances sample efficiency by integrating off-policy learning and unsupervised pre-training. To further reduce human feedback requirements, a semi-supervised method named SURF (Park et al. 2022) combines data augmentation with PbRL to leverage both labeled and unlabeled experience. More recently, Bukharin et al. (2024) proposed a hierarchical reward modeling framework called HERON that structures preference learning based on the importance of feedback signals, improving performance in sparse-reward scenarios.

Test-driven methods have been applied in RL recently, but they often require *manual* design of approaches to process test results. We introduce the first theoretical framework for test-driven reinforcement learning that is learning-based. Algorithmically, we adapt the PbRL paradigm of learning the return function from trajectory comparison.

## 3 Preliminaries

A Markov Decision Process (MDP) could be described by a tuple  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$  where  $\mathcal{S}$  is the state space,  $\mathcal{A}$  is the action space,  $\mathcal{P}$  is the transition probability function,  $r \in \mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the reward function, and  $\gamma$  is the discount factor. The goal of reinforcement learning (Sutton and Barto 2018) is to find a policy  $\pi \in \Pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$  that maximizes the expected cumulative reward:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} r(s_t, a_t) \right]. \quad (1)$$

A trajectory  $\tau \in \mathcal{T}$  represents a sequence of state-action pairs generated through policy-environment interactions, beginning at some initial state. The trajectory return function  $R$  evaluates a trajectory by summing rewards across all constituent state-action pairs:  $R(\tau) = \sum_{(s,a) \in \tau} r(s, a)$ . Given these definitions, the reinforcement learning optimization objective is formally expressed as:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\tau \sim \pi} [R(\tau)]. \quad (2)$$

### 3.1 Maximum Entropy Reinforcement Learning

Maximum Entropy Reinforcement Learning (MERL, Haarnoja et al. 2018) extends the standard reinforcement learning framework by incorporating an entropy maximization objective. This approach simultaneously optimizes for both cumulative reward maximization and policy entropy, thereby improving exploration capacity and algorithmic robustness. The objective of MERL could be written as:

$$\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[ \sum_{t=0}^T r(s_t, a_t) + \alpha H(\pi(\cdot | s_t)) \right], \quad (3)$$

where  $H(\pi(\cdot | s_t))$  denotes the policy’s entropy at state  $s_t$ , and  $\alpha$  controls the trade-off between the reward and entropy terms. When policy  $\pi_1$  is optimized via maximum entropy reinforcement learning with respect to the trajectory return function  $R(\tau) = \sum_{(s,a) \in \tau} r(s,a)$ , the resulting policy  $\pi_2$  follows:

$$\pi_2(\tau) = \frac{1}{Z} \pi_1(\tau) \exp\left(\frac{1}{\alpha} R(\tau)\right), \quad (4)$$

where  $Z$  is the partition function,  $\pi(\tau)$  represents the probability of generating trajectory  $\tau$  using policy  $\pi$ . The proof of Equation (4) is in Appendix E.

### 3.2 Preference-based Reinforcement Learning

Preference-based reinforcement learning (Christiano et al. 2017) is a class of reinforcement learning algorithms that learns a policy by comparing the rewards of different trajectories. A preference relation  $\succ$  is defined on  $\mathcal{T}$  as follows:

$$\tau_1 \succ \tau_2 \Leftrightarrow R(\tau_1) \geq R(\tau_2). \quad (5)$$

Following the Bradley-Terry model (Bradley and Terry 1952), we could use a reward function estimate  $\hat{r}$  as the preference predictor, as  $\hat{r}$  could be viewed as a latent factor to explain the preference. We assume the preference probability of a trajectory depends exponentially on its return:

$$\hat{P}[\tau_1 \succ \tau_2] = \frac{\exp(\hat{R}(\tau_1))}{\exp(\hat{R}(\tau_1)) + \exp(\hat{R}(\tau_2))}, \quad (6)$$

For a given dataset  $\mathcal{D}$  of triples  $(\tau_1, \tau_2, \mu)$ , where  $\tau_1$  and  $\tau_2$  are different trajectories and  $\mu$  is a distribution over  $\{1, 2\}$  indicating which trajectory is preferred, we could learn the reward function by minimize the following loss function :

$$\begin{aligned} \mathcal{L}(\hat{r}) = & - \sum_{(\tau_1, \tau_2, \mu) \in \mathcal{D}} \mu(1) \log \hat{P}[\tau_1 \succ \tau_2] \\ & + \mu(2) \log(1 - \hat{P}[\tau_1 \succ \tau_2]). \end{aligned} \quad (7)$$

## 4 Method

In this section, we introduce Test-driven reinforcement learning. We first illustrate the notation and goal in TdRL, then we establish sufficient conditions for trajectory return functions to guarantee policy convergence to the optimal policy set. Additionally, we introduce a lexicographic heuristic approach to compare trajectories for learning trajectory return function. Finally, we detail the implementation of the TdRL algorithm. Figure 1 demonstrates the main procedure of the TdRL algorithm. It follows an iterative procedure mainly consisting of four stages:

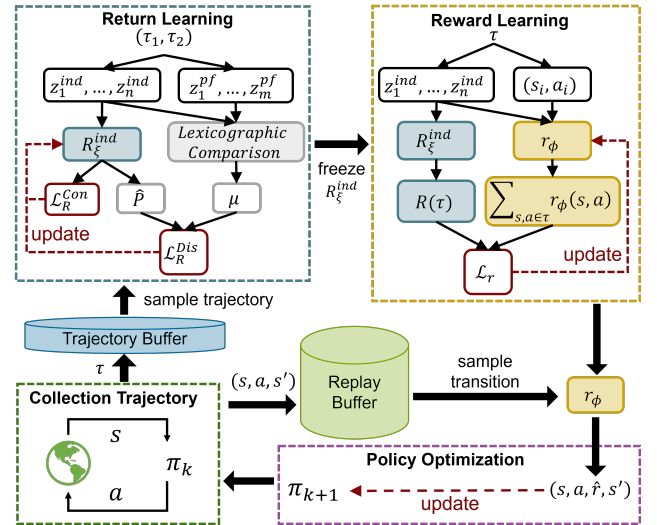


Figure 1: The main procedure of the TdRL algorithm.

- *Collection Trajectory*: The agent interacts with the environment to collect trajectories.
- *Return Learning*: The return function is updated by the comparison results of trajectories.
- *Reward Learning*: The reward function is updated and rewards are recalculated for transitions in the replay buffer.
- *Policy Optimization*: The policy network is optimized using transitions in the replay buffer.

### 4.1 Notation and Goal

TdRL aims to find a policy whose interaction trajectories with the environment pass all given pass-fail tests. Test functions map a trajectory to a test result. Specifically, test functions can be categorized into two types based on their functionality: pass-fail tests and indicative tests. A pass-fail test outputs a binary value judging whether a trajectory meets the required criteria. In contrast, an indicative test outputs a real number, quantifying the performance of a trajectory in a specific metric. Formally, we use  $z^{pf} \in \mathcal{Z}^{pf} : \mathcal{T} \rightarrow \{0, 1\}$  to denote pass-fail test function, and  $z^{ind} \in \mathcal{Z}^{ind} : \mathcal{T} \rightarrow \mathbb{R}$  to denote indicative test function.

The task objective in TdRL can be formulated as  $\langle S, \mathcal{A}, \mathcal{P}, \mathcal{Z}^{pf}, \mathcal{Z}^{ind}, \gamma \rangle$ , where  $\mathcal{Z}^{pf} = \{z_1^{pf}, \dots, z_m^{pf}\}$  represents a set of  $m$  pass-fail test functions and  $\mathcal{Z}^{ind} = \{z_1^{ind}, \dots, z_n^{ind}\}$  represents a set of  $n$  indicative test functions. The goal of TdRL is to find a policy  $\pi^*$  such that the resulting trajectory  $\tau$  from the interaction between  $\pi^*$  and the environment satisfies:

$$\mathbb{E}_{\tau \sim \pi^*} \left[ \sum_{i=1}^m z_i^{pf}(\tau) \right] = m. \quad (8)$$

Following the above definition, we define the optimal trajectory set as the collection of all trajectories that pass all pass-fail tests, denoted as:

$$\tilde{\mathcal{T}} = \{\tau | \tau \in \mathcal{T}, \sum_{i=1}^m z_i^{pf}(\tau) = m\}. \quad (9)$$

Let  $\tilde{\mathcal{T}}_i = \{\tau | \tau \in \mathcal{T}, z_i^{pf}(\tau) = 1\}$  denote the set of trajectories that pass the pass-fail test  $z_i^{pf}$ . Then, we have  $\tilde{\mathcal{T}} = \cap_{i=1}^m \tilde{\mathcal{T}}_i$ .

Similarly, we define the optimal policy set as the collection of all policies whose interaction with the environment generates trajectories that pass all pass-fail tests, denoted as:

$$\tilde{\Pi} = \{\pi | \pi \in \Pi, \mathbb{E}_{\tau \sim \pi} [\sum_{i=1}^m z_i^{pf}(\tau)] = m\}. \quad (10)$$

Let  $\tilde{\Pi}_i = \{\pi | \pi \in \Pi, \mathbb{E}_{\tau \sim \pi} [z_i^{pf}(\tau)] = 1\}$  denote the set of policies whose interaction with the environment generates trajectories that pass the pass-fail test  $z_i^{pf}$ . Then, we have  $\tilde{\Pi} = \cap_{i=1}^m \tilde{\Pi}_i$ .

## 4.2 Test Functions to Return Function

The goal of TdRL is to find a policy  $\pi^*$  that generates trajectories passing all pass-fail tests. In other words, we need to find a policy in  $\tilde{\Pi}$ . To achieve this, we could construct a trajectory return function  $R$  such that policy optimization based on this function could converge to a policy belonging to  $\tilde{\Pi}$ . Let  $d(\tau_1, \tau_2)$  denote the distance between trajectories  $\tau_1$  and  $\tau_2$  in the trajectory space, and the distance between a trajectory  $\tau$  and a trajectory set  $\tilde{\mathcal{T}}$  is defined as:

$$d(\tau, \tilde{\mathcal{T}}) = \min_{\tau' \in \tilde{\mathcal{T}}} d(\tau, \tau'). \quad (11)$$

For a policy  $\pi$ , let  $P_\pi : \mathcal{T} \rightarrow [0, 1]$  denote the distribution of trajectories generated through interaction with the environment, satisfying  $\int_{\mathcal{T}} P_\pi(\tau) d\tau = 1$ . The distance between policies  $\pi_1$  and  $\pi_2$  is defined by the Wasserstein- $p$  distance (Villani et al. 2008) between their corresponding trajectory distributions  $P_{\pi_1}$  and  $P_{\pi_2}$ :

$$d(\pi_1, \pi_2) = W_p(P_{\pi_1}, P_{\pi_2}) = \left( \inf_{\gamma \in \Gamma(P_{\pi_1}, P_{\pi_2})} \int_{\mathcal{T} \times \mathcal{T}} d(\tau_1, \tau_2)^p d\gamma(\tau_1, \tau_2) \right)^{\frac{1}{p}}, \quad (12)$$

where  $\Gamma(P_{\pi_1}, P_{\pi_2})$  denotes the set of all joint probability distributions whose marginal distributions are  $P_{\pi_1}$  and  $P_{\pi_2}$ , respectively. The distance between policy  $\pi$  and policy set  $\tilde{\Pi}$  is defined as:

$$d(\pi, \tilde{\Pi}) = \min_{\pi' \in \tilde{\Pi}} d(\pi, \pi'). \quad (13)$$

**Theorem 1.** *If there exists a trajectory return function  $R(\tau)$  that is monotonically non-increasing with respect to the distance between a trajectory  $\tau$  and the optimal trajectory set  $\tilde{\mathcal{T}}$ , such that:*

$$d(\tau_1, \tilde{\mathcal{T}}) \leq d(\tau_2, \tilde{\mathcal{T}}) \implies R(\tau_1) \geq R(\tau_2).$$

*Suppose policy  $\pi_2$  is obtained by optimizing policy  $\pi_1$  using a maximum entropy algorithm with respect to  $R$ . Then, policy  $\pi_2$  is closer to the optimal policy set  $\tilde{\Pi}$  than  $\pi_1$ :*

$$d(\pi_1, \tilde{\Pi}) \geq d(\pi_2, \tilde{\Pi}).$$

The proof of Theorem 1 is shown in Appendix F. Theorem 1 demonstrates that if there exists a trajectory return function  $R$  that assigns higher values to trajectories closer to the optimal trajectory set  $\tilde{\mathcal{T}}$ , then performing maximum entropy policy optimization on this return function will yield a policy that is closer to the policy set  $\tilde{\Pi}$  than the original policy. To obtain a trajectory reward function  $R$  that satisfies the above conditions, we consider two key issues: how to construct the trajectory return function and how to design a loss function for learning such a return function.

There typically exist numerous natural indicative signals in environments. The key challenge lies in effectively combining these indicative signals into a reward signal that can properly guide the learning process (Juechems and Summerfield 2019; Silver and Sutton 2025). Motivated by this insight, our approach does not directly construct a function mapping trajectories to returns. Instead, we regard indicative test results as indicative signals, and we utilize indicative test functions to construct a trajectory return function. Appendix G provides a detailed interruption about the indicative test function combination.

Specifically, we employ a fully connected network that takes an input vector of dimension  $n$  and produces a scalar output parameterized by  $\xi$  to construct a return mapping function  $R_\xi^{ind}$ . Then, the trajectory return function  $R$  could be constructed as a composite function of  $R_\xi^{ind}$  and the  $n$  indicative test functions:

$$R(\tau) = R_\xi^{ind}(z_1^{ind}(\tau), z_2^{ind}(\tau), \dots, z_n^{ind}(\tau)). \quad (14)$$

Following the Bradley-Terry model (Bradley and Terry 1952), if we consider the distance to the optimal trajectory set as a measure of trajectory quality, we can estimate the probability of distance relationships between trajectories by the return function. For convenience, we define  $\tilde{d}(\tau)$  as the distance between trajectory  $\tau$  and the optimal trajectory set  $\tilde{\mathcal{T}}$ . Then, the probability of  $\tau_1$  being closer to  $\tilde{\mathcal{T}}$  than  $\tau_2$  can be estimated as:

$$\hat{P}[\tilde{d}(\tau_1) < \tilde{d}(\tau_2)] = \frac{\exp(R(\tau_1))}{\exp(R(\tau_1)) + \exp(R(\tau_2))}. \quad (15)$$

Suppose  $\mu \in \{0, 0.5, 1\}$  represents the true probability that  $\tau_1$  is closer to  $\tilde{\mathcal{T}}$  than  $\tau_2$ , where  $\mu = 0$  means  $\tilde{d}(\tau_1) > \tilde{d}(\tau_2)$ ,  $\mu = 1$  means  $\tilde{d}(\tau_1) < \tilde{d}(\tau_2)$ , and  $\mu = 0.5$  means  $\tilde{d}(\tau_1) = \tilde{d}(\tau_2)$ . Then, we could construct the following distance-based loss function:

$$\begin{aligned} \mathcal{L}_R^{Dis} = & - \sum_{(\tau_1, \tau_2, \mu) \in \mathcal{D}} \left[ \mu \cdot \log \hat{P}[\tilde{d}(\tau_1) < \tilde{d}(\tau_2)] \right. \\ & \left. + (1 - \mu) \cdot \log \left( 1 - \hat{P}[\tilde{d}(\tau_1) < \tilde{d}(\tau_2)] \right) \right], \quad (16) \end{aligned}$$

where  $\mathcal{D}$  is a dataset of trajectory pairs  $(\tau_1, \tau_2)$  and their corresponding probabilities  $\mu$ . Furthermore, to ensure numerical stability during return function learning, we introduce a penalty term:

$$\mathcal{L}_R^{Penalty} = \sum_{(\tau_i, \tau_j, \mu) \in \mathcal{D}} \sum_{i \in \{1, 2\}} \left( R(\tau_i) - \tilde{R}(\tau_i) \right)^2, \quad (17)$$

where  $\tilde{R}$  denotes the trajectory return values computed before network updating.

Then, we decompose the learned trajectory return function into a state-action reward function for policy learning. We parameterize the reward function as  $r_\phi(s, a)$  and construct the reward learning loss:

$$\mathcal{L}_r = \sum_{\tau \in \mathcal{D}} \left[ R(\tau) - \sum_{(s,a) \in \tau} r_\phi(s, a) \right]^2. \quad (18)$$

By minimizing the loss  $\mathcal{L}_R^{Dis}$  and  $\mathcal{L}_R^{Penalty}$ , we can learn the return mapping function  $R_\xi^{ind}$ . Then we could decompose the trajectory return to state-action reward by learning a reward function through minimizing the loss  $\mathcal{L}_r$ .

Computing  $\mathcal{L}_R^{Dis}$  requires the relative distance relationships between trajectories. However, during the learning process, the trajectories contained in the optimal trajectory set  $\tilde{\mathcal{T}}$  remain unknown, making it intractable to directly compute the distances  $\tilde{d}(\tau)$ . Notably, we do not require exact values of  $\tilde{d}(\tau)$  when computing the loss  $\mathcal{L}_R^{Dis}$ , but only need to determine the relative distance relationships between trajectories, specifically, whether  $\tilde{d}(\tau_1) > \tilde{d}(\tau_2)$ ,  $\tilde{d}(\tau_1) < \tilde{d}(\tau_2)$  or  $\tilde{d}(\tau_1) = \tilde{d}(\tau_2)$ . In the next section, we will introduce a lexicographic heuristic approach to construct the relative distance relationships between trajectories.

### 4.3 Lexicographic Trajectory Comparison

For trajectories  $\tau_1$  and  $\tau_2$ , we need to compare their distances to the optimal trajectory set  $\tilde{\mathcal{T}}$ , i.e.,  $\tilde{d}(\tau_1)$  and  $\tilde{d}(\tau_2)$ . However, directly computing  $\tilde{d}(\tau)$  is infeasible as  $\tilde{\mathcal{T}}$  is unknown. Under such limited information conditions, we employ a lexicographic heuristic (Özgür Şimşek 2020) approach to compare the distances. The lexicographic heuristic is efficient, practical, and adaptable, which is a fast and frugal strategy originally from human decision-making (Gigerenzer and Goldstein 1996; Katsikopoulos 2013; Katsikopoulos et al. 2021). Specifically, the lexicographic method sequentially evaluates information in priority order and makes decisions based on the first criterion that meets predefined thresholds (e.g., exceeding certain values).

Building upon the definitions of pass-fail tests and indicative tests, along with the relationship between  $\tilde{\mathcal{T}}_i$  and  $\tilde{\mathcal{T}}$ , we could derive the following priors (The interpretation of the comparison priors is detailed in Appendix H.):

- Pass-fail tests take precedence over indicative tests
- A trajectory satisfying more pass-fail tests is closer to  $\tilde{\mathcal{T}}$
- A trajectory passing more challenging tests (corresponding to smaller  $\tilde{\mathcal{T}}_i$ ) is closer to  $\tilde{\mathcal{T}}$
- All trajectories within  $\tilde{\mathcal{T}}$  have zero distance to  $\tilde{\mathcal{T}}$
- Under-optimized indicators should be prioritized

Building upon the established priors, we could compute the probability  $\mu$  that  $\tau_1$  is closer to  $\tilde{\mathcal{T}}$  than  $\tau_2$  through the following lexicographical procedure (more details are shown in Appendix H):

---

### Algorithm 1: TdRL

---

**Require:** frequency of return network update  $K$

**Require:** pass-fail tests  $\{z_1^{pf}, z_2^{pf}, \dots, z_m^{pf}\}$ , indicative tests  $\{z_1^{ind}, z_2^{ind}, \dots, z_n^{ind}\}$

- 1: Initial  $\pi_\theta, R_\xi^{ind}, r_\phi$ , and  $\tau = \emptyset$
  - 2: Initial trajectory buffer  $\mathcal{D} = \emptyset$ , replay buffer  $\mathcal{B} = \emptyset$ .
  - 3: **for** each iteration **do**
  - 4:   Sample action  $a$  from  $\pi_\theta(\cdot|s)$
  - 5:   Exec action  $a$  in environment and get  $(s', done)$
  - 6:   Update Trajectory  $\tau \leftarrow \tau \cup \{(s, a, s')\}$
  - 7:   Store transition  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s, a, s', r_\phi(s, a), done)\}$
  - 8:   **if**  $done$  **then**
  - 9:     Store trajectory  $\mathcal{D} \leftarrow \mathcal{D} \cup \tau$ , reset  $\tau = \emptyset$
  - 10:   **end if**
  - 11:   **if** iteration %  $K == 0$  **then**
  - 12:     **for** each gradient step **do**
  - 13:       Sample minibatch  $\{(\tau_1, \tau_2)_j\}_{j=1}^D \sim \mathcal{D}$
  - 14:       Compute  $\mu$  for each pair  $(\tau_1, \tau_2)$  by the lexicographic trajectory comparison approach
  - 15:       Optimize  $\mathcal{L}_R^{Dis}$  in (16) and  $\mathcal{L}_R^{Penalty}$  in (17) with respect to  $\xi$
  - 16:       Optimize  $\mathcal{L}_r$  in (18) with respect to  $\phi$
  - 17:     **end for**
  - 18:     Relabel entire replay buffer  $\mathcal{B}$  using  $r_\phi$
  - 19:   **end if**
  - 20:   **for** each gradient step **do**
  - 21:     Sample random minibatch from  $\mathcal{B}$
  - 22:     Update policy network  $\theta$
  - 23:   **end for**
  - 24: **end for**
- 

1. If  $\sum_{i=1}^m z_i^{pf}(\tau_j) = m, \forall \tau \in \{\tau_1, \tau_2\}$ , return  $\mu = 0.5$ .
2. Compare pass-fail test passing count:
  - If  $\sum_{i=1}^m z_i^{pf}(\tau_1) > \sum_{i=1}^m z_i^{pf}(\tau_2)$ , return  $\mu = 1$
  - If  $\sum_{i=1}^m z_i^{pf}(\tau_1) < \sum_{i=1}^m z_i^{pf}(\tau_2)$ , return  $\mu = 0$
3. Sort pass-fail tests in ascending order of historical pass rates (i.e., descending difficulty) as  $\{z_{k_1}^{pf}, \dots, z_{k_m}^{pf}\}$ . Sequentially compare  $z_{k_i}^{pf}(\tau_1)$  and  $z_{k_i}^{pf}(\tau_2)$ .
  - If  $z_{k_i}^{pf}(\tau_1) > z_{k_i}^{pf}(\tau_2)$ , return  $\mu = 1$
  - If  $z_{k_i}^{pf}(\tau_1) < z_{k_i}^{pf}(\tau_2)$ , return  $\mu = 0$
4. Sort indicative tests in descending order of the skewness of historical test results (least-optimized first) as  $\{z_{l_1}^{ind}, \dots, z_{l_n}^{ind}\}$ . Sequentially compare  $z_{l_i}^{ind}(\tau_1)$  and  $z_{l_i}^{ind}(\tau_2)$ .
  - If  $z_{l_i}^{ind}(\tau_1) > z_{l_i}^{ind}(\tau_2)$ , return  $\mu = 1$
  - If  $z_{l_i}^{ind}(\tau_1) < z_{l_i}^{ind}(\tau_2)$ , return  $\mu = 0$
5. Return  $\mu = 0.5$ , i.e., trajectories are indistinguishable

### 4.4 TdRL Algorithm

Algorithm 1 presents the detailed procedure of the TdRL algorithm. TdRL employs  $\mathcal{L}_R^{Dis}$  to learn the return function

and constrains the update step via  $\mathcal{L}_R^{Penalty}$ . However, since the gradient magnitudes of the two losses can not be directly comparable, selecting appropriate weights for balancing them is challenging. Specifically,  $\mathcal{L}_R^{Dis}$  is a cross-entropy loss, whose gradient scales with the probability difference, whereas  $\mathcal{L}_R^{Penalty}$  is an MSE loss, whose gradient depends on the variation in the return output. To address this issue, we propose two methods: *gradient norm* (GN) and *early stop* (ES). Both methods compute the gradients of  $\mathcal{L}_R^{Dis}$  ( $\nabla_{\xi} \mathcal{L}_R^{Dis}$ ) and  $\mathcal{L}_R^{Penalty}$  ( $\nabla_{\xi} \mathcal{L}_R^{Penalty}$ ) before each network update and use their sum to update the network parameters. The key distinction lies in the fact that GN rescales the MSE gradient to match the cross-entropy gradient L2-norm when the former exceeds the latter, and ES stops training if the MSE gradient L2-norm surpasses a predefined multiple ( $K^{ES}$ ) of the cross-entropy gradient L2-norm.

## 5 Experiments

We design our experiments to investigate the benefits of using test functions for representing task objectives, as well as the performance of the TdRL algorithm.

### 5.1 Setups

We evaluate TdRL on several continuous robot control tasks from DeepMind Control Suite (DM-Control, Tunyasuvunakool et al. 2020). We employ the Soft Actor-Critic (SAC, Haarnoja et al. 2018) algorithm as the backbone for the implementation of TdRL. To enhance experience diversity in early training, we use unsupervised RL for warm-up. Hyperparameters for the algorithm implementation are detailed in Appendix C. In Section 4.4, we introduce two approaches—*gradient norm* and *early stop*—to balance the two loss terms  $\mathcal{L}_R^{Dis}$  and  $\mathcal{L}_R^{Penalty}$  in return function learning. For notational clarity, we refer to the TdRL variants employing these methods as TdRL-GN and TdRL-ES.

### 5.2 Main Results

We treat the environment rewards in DM-Control as the oracle rewards. To focus on the efficiency of TdRL, we only use the components of the oracle rewards in environments to construct the test functions. For instance, in the *Walker-Walk* task, the oracle reward is derived from three components: torso height, torso upright, and move speed. Our test functions for this task also only use these components. Detailed test functions for each task are provided in Appendix I. More experiment results are shown in D.

**TdRL eliminates manual objective weighting while achieving performance comparable to or better than carefully handcrafted reward functions.** Figure 3 demonstrates that TdRL achieves comparable or superior performance to SAC with oracle rewards in continuous robot control tasks. Since TdRL requires learning the reward function during policy optimization, its performance improvement during early training stages is slower compared to using oracle rewards. Furthermore, both TdRL-GN and TdRL-ES exhibit strong performance, indicating the effectiveness of combining cross-entropy and MSE loss in both approaches.

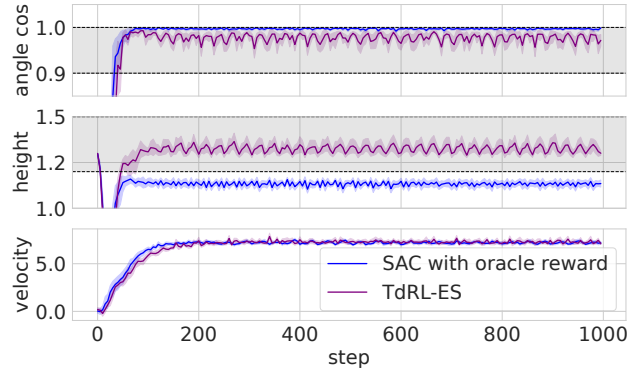


Figure 2: Performance comparison in multi-objectives between SAC with oracle reward and TdRL in the *Walker-Run* task. The gray shaded area represents the predefined performance threshold for each metric.

#### TdRL can also be applied to on-policy RL algorithms.

Although TdRL is theoretically grounded in maximum entropy reinforcement learning, the TdRL algorithm can also be applied to on-policy RL methods. We maintain the reward learning component of TdRL while replacing its policy update algorithm with Proximal Policy Optimization (PPO, Schulman et al. 2017). Figure 3 demonstrates that TdRL with PPO achieves comparable performance to PPO with oracle rewards on several tasks, while exhibiting significant performance gaps on others. These results suggest TdRL’s potential for integration with on-policy reinforcement learning algorithms.

**TdRL achieves a satisficing solution across multiple objectives rather than an optimal solution in a single metric.** Figure 2 shows that the performance comparison in multi-objectives between SAC with oracle reward and TdRL in the *Walker-Run* task. In the *Walker-Run* task, there are three primary objectives. The cosine of the torso angle should be within [0.9, 1], ensuring the robot’s upper body remains upright. The torso height should exceed 1.2, ensuring the robot maintains a standing posture. The velocity in the x-axis should reach 8. The reward function for this task in DM-Control is defined as  $((3 * stand + upright)/4) * (5 * move + 1)/6$ , where *upright*, *stand*, and *move* respectively represent the rewards of uprightiness, standing height, and move speed of the robot. Figure 2 shows that despite *stand* having a higher weight than *upright* in the reward function, the trained policy achieves the desired uprightiness but fails to maintain sufficient standing height.

In TdRL, however, for each of the three metrics - uprightiness, stand height, and move speed - we constructed both pass-fail tests and indicative tests (the test functions are detailed in the Appendix I). Notably, we do not need to preset the weights to combine them; instead, the pass-fail tests define the passing conditions for each metric. Figure 2 demonstrates that the TdRL policy fulfills all task metrics more effectively. While TdRL does not always achieve the optimum in individual metrics (e.g., upper-body uprightiness), it consistently attains a satisficing solution across multiple ob-

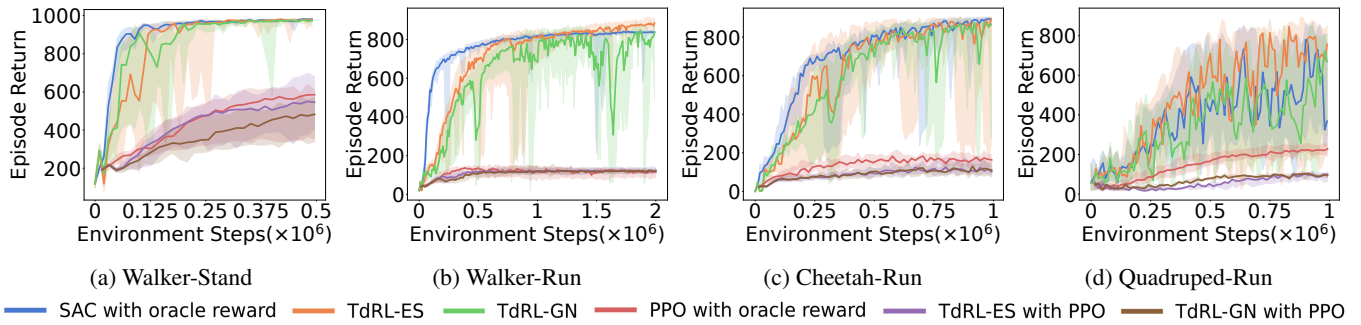


Figure 3: Performance comparison of algorithms on DM-Control tasks. Each algorithm runs with 10 different random seeds. Following Agarwal et al. (2021), the solid lines represent the interquartile mean (IQM) of episode returns while the shaded areas indicate 95% confidence intervals.

jectives. In the *Walker-Run* task, TdRL matches the speed of the SAC using oracle reward while satisfying both upright-ness and stand height requirements.

**TdRL focuses on the performance of trajectories rather than the quality of state-action pairs.** Figure 2 shows that the TdRL-trained policy underperforms the policy trained by SAC with oracle rewards across all metrics in stability. This discrepancy likely arises because TdRL operates at the trajectory level rather than evaluating state-action pairs. Such trajectory-level evaluation is more intuitive for designers, avoiding task objectives that overly emphasize state quality (Booth et al. 2023), which may lead to reward hacking (Amodei et al. 2016). Furthermore, TdRL can also enhance stability for specific metrics through introducing additional test functions (e.g., the variance of the robot’s standing height).

### 5.3 Ablation Study

**Impact of reward learning approaches** Figure 4(a) illustrates the impact of different reward learning approaches in the TdRL algorithm. TdRL with no penalty denotes the variant without applying  $\mathcal{L}_R^{Penalty}$  to constrain the trajectory return function learning, while TdRL with direct reward learning refers to directly learning the reward function from trajectory comparisons rather than first learning the return function followed by decomposition. Experimental results demonstrate that both omitting the penalty term and directly learning the reward function lead to training instability and degraded policy performance. Specifically, the absence of a penalty causes uncontrolled growth in return values during learning, potentially inducing numerical instability. When directly learning the reward function,  $\tanh$  activation is typically applied in the reward network’s output layer to bound its outputs, which requires continuous rescaling during later training stages to accommodate reward learning, ultimately resulting in training instability and performance deterioration.

**Hyperparameters choice** TdRL-ES stops training when the L2-norm of  $\mathcal{L}_R^{Penalty}$ ’s gradient exceeds a predefined multiple ( $K^{ES}$ ) of  $\mathcal{L}_R^{Dis}$ ’s gradient L2-norm. The scaling factor  $K^{ES}$  is a tunable parameter requiring configuration. We investigated how different multiple  $K^{ES}$  values affect

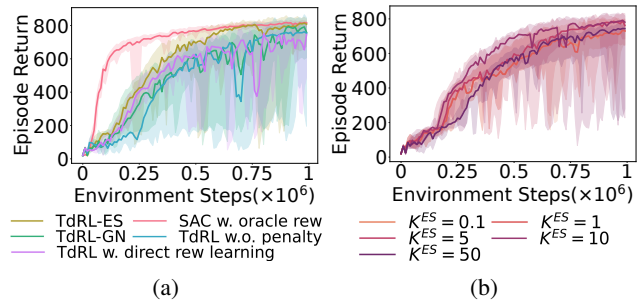


Figure 4: **Left:** the performance of TdRL with different reward learning methods. **Right:** the performance of TdRL-ES with varying values of multiple  $K^{ES}$ .

algorithm performance. Figure 4(b) indicates that both excessively large and small multiple  $K^{ES}$  values degrade performance. Based on experimental results, we recommend setting multiple  $K^{ES}$  to 10 as a general guideline.

## 6 Conclusion

To tackle the challenges in reward design for reinforcement learning, this paper introduces a test-driven reinforcement learning (TdRL) framework. Instead of relying on a scalar reward function, TdRL represents task objectives using multiple test functions. The test functions are defined on trajectories rather than state-action pairs, which is more intuitive for designers. Besides, each test function only needs to represent a single objective, and designers do not need to consider their weights, greatly reducing the complexity of the reinforcement learning task design.

We propose sufficient conditions for trajectory return functions to guarantee policy convergence to the optimal policy set. Then, we introduce a lexicographic trajectory comparison approach for return learning. Furthermore, we present an implementation of TdRL, and the experimental results show that TdRL matches or outperforms reward-based methods in task training, with greater design simplicity and inherent support for multi-objective optimization. TdRL offers a viable approach to tackling reward design challenges in reinforcement learning.

## Acknowledgements

We thank all anonymous reviewers for their constructive feedback, which helped improve our paper. We also thank Mr. Yuxin Cheng from the University of Hong Kong for his helpful discussions.

## References

- Agarwal, R.; Schwarzer, M.; Castro, P. S.; Courville, A.; and Bellemare, M. G. 2021. Deep reinforcement learning at the edge of the statistical precipice. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713845393.
- Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete Problems in AI Safety. arXiv:1606.06565.
- Arora, S.; and Doshi, P. 2021. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297: 103500.
- Beck. 2002. *Test Driven Development: By Example*. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN 0321146530.
- Booth, S.; Knox, W. B.; Shah, J.; Niekum, S.; Stone, P.; and Allievi, A. 2023. The Perils of Trial-and-Error Reward Design: Misdesign through Overfitting and Invalid Task Specifications. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(5): 5920–5929.
- Bradley, R. A.; and Terry, M. E. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4): 324–345.
- Bukharin, A.; Li, Y.; He, P.; and Zhao, T. 2024. Deep Reinforcement Learning from Hierarchical Preference Design. arXiv:2309.02632.
- Cao, Y.; Zhao, H.; Cheng, Y.; Shu, T.; Chen, Y.; Liu, G.; Liang, G.; Zhao, J.; Yan, J.; and Li, Y. 2025. Survey on Large Language Model-Enhanced Reinforcement Learning: Concept, Taxonomy, and Methods. *IEEE Transactions on Neural Networks and Learning Systems*, 36(6): 9737–9757.
- Casper, S.; Davies, X.; Shi, C.; Gilbert, T. K.; Scheurer, J.; Rando, J.; Freedman, R.; Korbak, T.; Lindner, D.; Freire, P.; Wang, T. T.; Marks, S.; Ségerie, C.-R.; Carroll, M.; Peng, A.; Christoffersen, P. J. K.; Damani, M.; Slocum, S.; Anwar, U.; Siththaranjan, A.; Nadeau, M.; Michaud, E. J.; Pfau, J.; Krashenninnikov, D.; Chen, X.; Langosco, L.; Hase, P.; Biyik, E.; Dragan, A. D.; Krueger, D.; Sadigh, D.; and Hadfield-Menell, D. 2023. Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback. *Trans. Mach. Learn. Res.*, 2023.
- Christiano, P.; Leike, J.; Brown, T. B.; Martic, M.; Legg, S.; and Amodei, D. 2017. Deep reinforcement learning from human preferences. arXiv:1706.03741.
- Du, Y.; Watkins, O.; Wang, Z.; Colas, C.; Darrell, T.; Abbeel, P.; Gupta, A.; and Andreas, J. 2023. Guiding Pre-training in Reinforcement Learning with Large Language Models. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, 8657–8677. PMLR.
- Fischer, J.; Werling, M.; Lauer, M.; and Stiller, C. 2024. Test-Driven Inverse Reinforcement Learning Using Scenario-Based Testing. In *2024 IEEE Intelligent Vehicles Symposium (IV)*, 827–834.
- Gigerenzer, G.; and Goldstein, D. G. 1996. Reasoning the fast and frugal way: models of bounded rationality. *Psychological review*, 103(4): 650.
- Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic Algorithms and Applications. *CoRR*, abs/1812.05905.
- Jaensch, F.; Kübler, K.; Schwarz, E.; and Verl, A. 2022. Test-Driven Reward Function for Reinforcement Learning: A Contribution towards Applicable Machine Learning Algorithms for Production Systems. *Procedia CIRP*, 112: 103–108. 15th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 14–16 July 2021.
- Juechems, K.; and Summerfield, C. 2019. Where Does Value Come From? *Trends in Cognitive Sciences*, 23(10): 836–850.
- Katsikopoulos, K. V. 2013. Why Do Simple Heuristics Perform Well in Choices with Binary Attributes? *Decision Analysis*, 10(4): 327–340.
- Katsikopoulos, K. V.; Şimşek, Ö.; Buckmann, M.; and Gigerenzer, G. 2021. *Classification in the Wild: The Science and Art of Transparent Decision Making*. The MIT Press. ISBN 9780262363228.
- Knox, W. B.; Allievi, A.; Banzhaf, H.; Schmitt, F.; and Stone, P. 2023. Reward (Mis)design for autonomous driving. *Artificial Intelligence*, 316: 103829.
- Kwon, M.; Xie, S. M.; Bullard, K.; and Sadigh, D. 2023. Reward Design with Language Models. In *The Eleventh International Conference on Learning Representations*.
- Lee, K.; Smith, L. M.; and Abbeel, P. 2021. PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 6152–6163. PMLR.
- Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C. L.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; Schulman, J.; Hilton, J.; Kelton, F.; Miller, L.; Simens, M.; Askell, A.; Welinder, P.; Christiano, P.; Leike, J.; and Lowe, R. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713871088.
- Park, J.; Seo, Y.; Shin, J.; Lee, H.; Abbeel, P.; and Lee, K. 2022. SURF: Semi-supervised Reward Learning with Data Augmentation for Feedback-efficient Preference-based Reinforcement Learning. In *International Conference on Learning Representations*.

- Pranger, S.; Chockler, H.; Tappler, M.; and Könighofer, B. 2024. Test Where Decisions Matter: Importance-driven Testing for Deep Reinforcement Learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Rajagopal, S. 2023. What's Next if Reward is Enough? Insights for AGI from Animal Reinforcement Learning. *Journal of Artificial General Intelligence*, 14(1): 15–40.
- Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms.
- Silver, D.; Singh, S.; Precup, D.; and Sutton, R. S. 2021. Reward is enough. *Artificial Intelligence*, 299: 103535.
- Silver, D.; and Sutton, R. S. 2025. Welcome to the era of experience. *Google AI*, 1.
- Simon, H. A. 1947. Administrative behavior; a study of decision-making processes in administrative organization. *Administrative behavior; a study of decision-making processes in administrative organization.*, xvi, 259–xvi, 259. Place: Oxford, England Publisher: Macmillan.
- Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book. ISBN 978-0-262-03924-6.
- Tunyasuvunakool, S.; Muldal, A.; Doron, Y.; Liu, S.; Bohez, S.; Merel, J.; Erez, T.; Lillicrap, T.; Heess, N.; and Tassa, Y. 2020. dm\_control: Software and tasks for continuous control. *Software Impacts*, 6: 100022.
- Vamplew, P.; Smith, B. J.; Källström, J.; Ramos, G.; Rădulescu, R.; Roijers, D. M.; Hayes, C. F.; Hentz, F.; Mannion, P.; Libin, P. J.; Dazeley, R.; and Foale, C. 2023. Scalar Reward is Not Enough. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, 839–841. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450394321.
- Villani, C.; et al. 2008. *Optimal transport: old and new*, volume 338. Springer.
- Yu, W.; Gileadi, N.; Fu, C.; Kirmani, S.; Lee, K.-H.; Arenas, M. G.; Chiang, H.-T. L.; Erez, T.; Hasenclever, L.; Humpalik, J.; brian ichter; Xiao, T.; Xu, P.; Zeng, A.; Zhang, T.; Heess, N.; Sadigh, D.; Tan, J.; Tassa, Y.; and Xia, F. 2023. Language to Rewards for Robotic Skill Synthesis. In *7th Annual Conference on Robot Learning*.
- Ziebart, B. D.; Maas, A.; Bagnell, J. A.; and Dey, A. K. 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, 1433–1438. AAAI Press. ISBN 9781577353683.
- Özgür Şimşek. 2020. Lexicographic Decision Rule. In *Oxford Research Encyclopedia of Politics*. Oxford University Press.