Automatic Reward Shaping from Multi-Objective Human Heuristics

Yuqing Xie
Tsinghua University

Jiayu Chen
Tsinghua University

Wenhao Tang
Tsinghua University

Ya Zhang
Shanghai Jiao Tong University

Chao Yu
Tsinghua University

Yu Wang
Tsinghua University

Abstract

Designing effective reward functions remains a central challenge in reinforcement learning, especially in multi-objective environments. In this work, we propose **M**ulti-**O**bjective **R**eward **S**haping with **E**xploration, a general framework that automatically combines multiple human-designed heuristic rewards into a unified reward function. MORSE formulates the shaping process as a bi-level optimization problem: the inner loop trains a policy to maximize the current shaped reward, while the outer loop updates the reward function to optimize task performance. To encourage exploration in the reward space and avoid suboptimal local minima, MORSE introduces stochasticity into the shaping process, injecting noise guided by task performance and the prediction error of a fixed, randomly initialized neural network. Experimental results in MuJoCo and Isaac Sim environments show that MORSE effectively balances multiple objectives across various robotic tasks, achieving task performance comparable to those obtained with manually tuned reward functions.

1 Introduction

Reward design remains a fundamental challenge in robot learning, particularly when multiple conflicting objectives must be balanced. A typical task definition might involve the robot traversing a specified distance (e.g., 2 meters) within a given time limit (e.g., 10 seconds) while maintaining the torque below a safety limit (e.g., 45 Nm). While the success criteria for the task are straightforward to define, their sparsity makes them inadequate as rewards for effective policy optimization. [Andrychowicz et al., 2018]

To facilitate policy learning, researchers often design dense heuristic reward functions, such as rewarding higher velocities or penalizing large joint movements to minimize energy cost. However, these heuristics frequently conflict with one another, necessitating labor-intensive manual tuning to combine them into a final reward function. Studies show that over 90% of reinforcement learning (RL) practitioners devote substantial effort to adjusting reward functions, highlighting a critical scalability bottleneck [Booth et al., 2023].

Our work proposes a self-supervised reward shaping framework that automates the reward design process. The approach requires RL practitioners to specify only: (1) a task performance criterion, which scores a trajectory at the end of the episode, and (2) a set of heuristic reward functions, which provide dense feedback for each step. The system then automatically learns an optimal combination of these heuristics through a bi-level optimization process [Zhang et al., 2024b], where the inner loop trains RL policies under the current reward function, and the outer loop adapts the reward function itself to maximize task performance.

2nd Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2025).

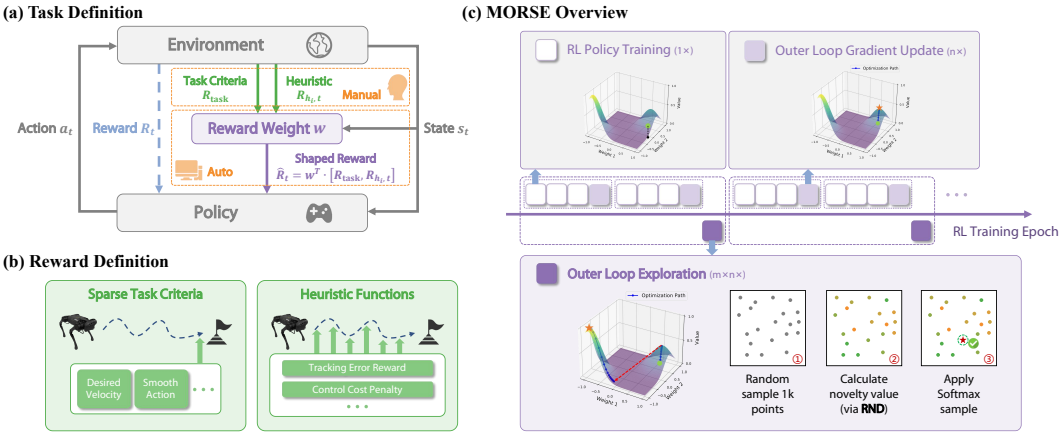


Figure 1: **(a)** Experts provide only task criteria and heuristic functions, instead of a manually tuned reward function. **(b)** Example task criteria and heuristic functions for robotic dog. **(c)** The inner loop trains the RL policy on shaped rewards, while the outer loop, combining gradient updates with exploration, updates reward weights to maximize task criteria.

However, it is not trivial to apply bi-level optimization to shape the rewards for real-world robotic tasks. Consider the robotic dog benchmark in [Margolis and Agrawal, 2023], which defines 15 distinct objectives, constituting an enormous space of valid reward combinations. With such a large reward weight space, the weight–performance landscape becomes highly non-convex with numerous local extrema [Xie et al., 2024]. Under such circumstances, existing bi-level methods [Gupta et al., 2023] fail to converge reliably in complex scenarios, as we will further demonstrate in the experiments.

To overcome these challenges, we introduce **M**ulti-**O**bjective **R**eward **S**haping with **E**xploration (MORSE). MORSE augments the outer loop of bi-level optimization with controlled stochastic exploration, thereby facilitating training when the reward space is large. Concretely, if the policy plateaus without reaching the target objective, the algorithm initiates outer-loop exploration: it samples a new reward weight and resumes gradient-based bi-level updates from this point. Exploration relies on novelty scores computed via Random Network Distillation (RND), where the prediction error against a fixed, randomly initialized network serves as the novelty measure. New starting points are then selected through softmax sampling over these scores.

The experiments are organized as follows. We begin with a motivating example in a custom multi-objective CartPole environment and show that unguided exploration helps conventional bi-level approaches to escape from local optima. We then evaluate MORSE in 8 challenging robotics domains, covering a 9-objective quadruped locomotion task and a 5-objective robotic manipulation task. Across all environments, MORSE consistently outperforms vanilla bi-level optimization and achieves performance comparable to that obtained with human-engineered reward functions.

2 Related Work

2.1 Multi-Objective Reinforcement Learning

To incorporate the trade-offs between multiple objectives, different multi-objective reinforcement learning (MORL) methods have been proposed. These methods can be roughly divided into two categories, that is, single-policy and multi-policy [Hayes et al., 2022, Felten et al., 2023].

Single-policy methods yield a policy capable of adapting to any preference utility at test time. This can be done either during training, where the policy maximizes expected scalarized returns (ESR) for potential utilities, [Roijers et al., 2018, Siddique et al., 2020] or during execution, where the policy selects actions based on test-time utility[Van Moffaert et al., 2013a,b, Issabekov and Vamplew, 2012, Reymond et al., 2023].

Multi-policy methods, on the other hand, generate a collection of policies that form a Pareto Front or Coverage Set, capturing inter-objective relationships. They can iteratively sample utility functions and derive corresponding policies to construct an exhaustive coverage set [Parisi et al., 2014, Mossalam et al., 2016, Parisi et al., 2017, Natarajan and Tadepalli, 2005], or generate multiple policies concurrently and employ specific criteria such as hypervolume value or Monte Carlo Tree Search to select actions[Castelletti et al., 2012, Ruiz-Montiel et al., 2017, Van Moffaert and Nowé, 2014, Yang et al., 2019, Lu et al., 2022, Reymond et al., 2022].

In general, MORL focuses on recovering the Pareto front that captures the trade-offs between multiple objectives, but it does not learn how to choose or optimize the reward weights. It assumes that the appropriate weights are given at test time. Our method aims to solve the orthogonal problem. Our goal is to automatically adjust the reward weights of auxiliary rewards to better guide the RL policy toward higher success rates. Therefore, our problem setup fundamentally differs from MORL.

2.2 Reward Shaping in Reinforcement Learning

Designing effective reward functions poses fundamental challenges in RL: sparse rewards yield inefficient learning, while poorly shaped rewards may induce suboptimal behaviors. To address these challenges, existing reward shaping methods can be broadly categorized into four main approaches.

Heuristic-based intrinsic reward methods encourage exploration through visitation frequency counts [Tang et al., 2017, Lobel et al., 2023, Machado et al., 2020, Fu et al., 2017] or curiosity-driven uncertainty bonuses [Pathak et al., 2017, 2019, Zhang et al., 2020, Burda et al., 2018], but require careful manual tuning of intrinsic and extrinsic reward balances. More automated approaches include differentiable intrinsic reward learning (LIRPG Zheng et al. [2018]), trajectory preference-based reward inference (SORS Memarian et al. [2021]), and composite reward functions combining task and exploration objectives (EXPLORS Devidze et al. [2022]). With the rapid development of large language models (LLMs), several recent works have explored using LLMs to assist in constructing reward functions[Ma et al., 2023, Zhang et al., 2024a]. These approaches typically require the LLM to define both the heuristic functions and the reward weights, and to iteratively refine them over multiple optimization rounds, which leads to substantial computational overhead.

Most relevant to our work are bi-level optimization methods [Hu et al., 2020, Gupta et al., 2023] that formulate reward shaping as $\max_\phi \mathcal{R}_{task}(\pi_\theta^*)$ s.t. $\pi_\theta^* = \arg\max_\pi \mathcal{R}_\phi(s, a)$, where ϕ parameterizes the shaped reward. These method assume (i) smooth reward-performance landscapes and (ii) sufficient gradient signals from heuristic rewards. However, such assumptions are often violated in robotics tasks, causing outer-loop optimization to stagnate in local optima. Our method addresses these limitations by adding stochasticity that considers task performance and novelty measurement in reward weight space.

3 Preliminary

3.1 Task Formulation

We formulate each task as a Markov decision process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \hat{R}, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the state transition function, $\hat{R}$: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the task-specific reward function, γ is the discount factor. The goal is to find a policy π_θ that maximizes the discounted cumulative task reward, which is defined as $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\Sigma_{t=0}^{T} \gamma^t \hat{R}(s_t, a_t)]$, where $\tau = (s_0, a_0, \ldots, s_T)$ denotes a trajectory sampled under π_θ, and T is the episode horizon.

Designing an effective reward function $\hat{R}$ is challenging, as it often requires balancing multiple conflicting objectives. In the proposed framework, we gradually learn $\hat{R}$ from (1) a sparse task-completion metric R_{task} provided by a human expert (e.g., a binary indicator of success at episode termination) and (2) a set of n auxiliary heuristic reward functions $\{R_{h_i}\}_{i=1}^n$, each capturing a separate objective of the task. Here, the human expert only needs to determine the terminal success criterion and provide heuristic guidance; the algorithm automatically finishes reward shaping.

Formally, the algorithm balances all reward components $\mathbf{R} = (R_{\text{task}}, R_{h_1}, \ldots, R_{h_n})$ to maximize task reward R_{task}. We learn a state-dependent weight function $w_\phi : \mathcal{S} \to \mathbb{R}^{n+1}$ to produce the

composite reward:

$$R_\phi(s,a) = w_\phi(s)^\top \mathbf{R}(s,a),$$

where $\mathbf{R}(s,a) \in \mathbb{R}^{n+1}$ is the concatenated vector of all reward components.

3.2 Bi-level Optimization for Reinforcement Learning

The above reward shaping process can be solved via bi-level optimization. The outer loop optimizes the reward weight $w_\phi(s)$ to maximize the accumulated task reward $J(\theta,\phi)$:

$$J(\theta,\phi) = E_{\pi_\theta, w_\phi}[\Sigma_{t=0}^{T} \gamma^t R_{\text{task}}(s_t,a_t)]$$

where as the inner loop optimizes the policy π_θ to maximize the weighted return $G(\theta)$:

$$G_\phi(\theta) = E_{\pi_\theta}[\Sigma_{t=0}^{T} \gamma^t R_\phi(s,a)] = E_{\pi_\theta}[\Sigma_{t=0}^{T} \gamma^t w_\phi(s_t) \cdot \mathbf{R}(s_t,a_t)]$$

The inner loop can be optimized using any reinforcement learning algorithm, while the outer loop can be solved using various bi-level optimization techniques. In this work, we choose the Implicit Function (IF)-based approach [Zhang et al., 2024b].

According to the chain rule, the gradient computation for the outer loop can be decomposed as:

$$\frac{\mathrm{d}J(\theta,\phi)}{\mathrm{d}\phi} = \frac{\partial J(\theta,\phi)}{\partial \phi} + \frac{\partial J(\theta,\phi)}{\partial \theta} \cdot \frac{\mathrm{d}\theta}{\mathrm{d}\phi}$$

Since w_ϕ does not explicitly influences $J(\theta,\phi)$, the term $\frac{\partial J(\theta,\phi)}{\partial \phi}$ equals zero. The next term $\frac{\partial J(\theta,\phi)}{\partial \theta}$ is the policy gradient calculated using the task reward, R_{task}. The third term $\frac{\mathrm{d}\theta}{\mathrm{d}\phi}$ captures how the optimal inner loop policy responds to changes in the reward weight w_ϕ. It is determined by $\frac{\mathrm{d}\pi^*_{\theta;\phi}}{\mathrm{d}\phi}$, where $\pi^*_{\theta;\phi}$ is the optimal policy obtained in the inner loop with w_ϕ.

We briefly discuss how to calculate the second term. During convergence, we have $\nabla_\theta G_\phi(\theta) = 0$, leading to the following expression for $\frac{\mathrm{d}\pi^*_{\theta;\phi}}{\mathrm{d}\phi}$:

$$\frac{\mathrm{d}\pi^*_{\theta;\phi}}{\mathrm{d}\phi} = -\left(\nabla^2_\phi \mathbb{E}_\pi\left[\sum_{t=0}^{T} \gamma^t R_\theta\right]\right)^{-1} \nabla_\phi \mathbb{E}_\pi\left[\sum_{t=0}^{T} \gamma^t R_\theta\right]$$

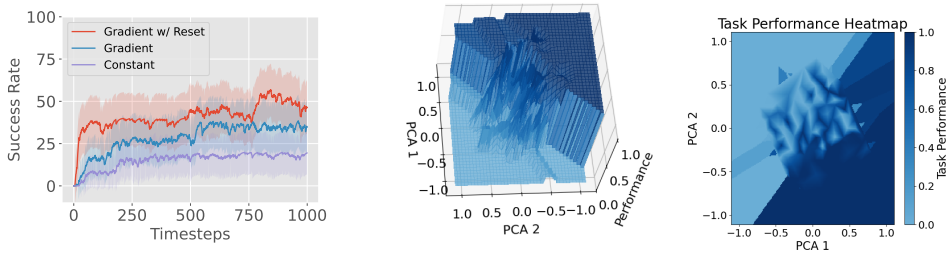
The two terms above can be computed using the second-order and first-order derivatives of the inner loop objective function. However, computing the inverse Hessian matrix $H = \nabla^2_\phi \mathbb{E}_\pi\left[\sum_{t=0}^{T} \gamma^t R_\theta\right]$ is computationally expensive. Therefore, we use the Neumann Series method to approximate the inverse of the Hessian matrix :

$$H^{-1} \approx \sum_{i=0}^{K-1} (I-H)^i$$

4 A Motivation Example: Multi-Objective CartPole with Extremely Sparse Reward

In this section, we extend the classic CartPole to a multi-objective setting where the agent must balance the pole while moving the cart as far right as possible. This demonstrates how vanilla bi-level optimization struggles with conflicting objectives and highly sparse task rewards. The environment retains the original observation and action spaces but uses a composite reward with four components: (1) a +100 task reward for surviving all 100 steps; (2) a +1 survival heuristic per step; (3) a +1 position heuristic when the cart's position exceeds 0.5; and (4) a -1 interference penalty when the agent's action matches a PD controller's stabilization attempt. These modifications make the multi-objective Cartpole significantly harder. We perform a grid search for potential reward weights, and the best performing configuration achieved approximately a 50% success rate.

We use REINFORCE in the inner loop for policy training, with the outer loop modeling rewards as $w(s) \cdot \mathbf{R}(s,a)$ through a weight network $w(s)$ that maps states to reward weights. We compare three update strategies for $w(s)$: **Constant** (randomly initialized weights with no update), **Gradient**



(a) **Gradient** failure case.

(b) Weight–performance relation.

Figure 2: Lesson from the motivation example: updating reward weights with outer-loop gradient is not sufficient; periodic resetting reward weights helps to escape from poor local optima.

(updated with standard bi-level optimization), and **Gradient w/ Reset** (resetting the weight network parameters every five gradient updates).

We evaluate each method over 10 trials with different random seeds, and report the results in Fig. 2a. Compared with **Constant**, **Gradient** improves the task success rate to 30%, while **Gradient w/ Reset**, which periodically resets the reward weight, further increases the success rate to 50%.

The limitation of **Gradient** arises from two inherent properties of multi-objective problems: (1) adding competing objectives increases task complexity and expands the search space for reward weights; and (2) task criteria provide extremely sparse feedback, yielding weak gradient signals for backpropagation. Starting with a poor initial weight often leads the policy to converge to a sub-optimal solution, which in turn prevents the outer loop from making meaningful gradient updates. Consequently, gradient-based optimization is only effective when the number of objectives is small, the reward space is limited, the solution space is broad, and the mapping from weights to performance is smooth and continuous, avoiding rugged, non-convex landscapes.

However, real-world tasks rarely satisfy these conditions. To illustrate this, we visualize the Walker2d task in the MuJoCo domain in Fig. 2b. Since Walker2d has three objectives, we project the 3D reward weight space into 2D via Principal Component Analysis (PCA), and use the z-axis to represent the optimal task performance achieved under each weight. By sampling around 100 weights and interpolating their outcomes, we construct the performance surface. The resulting landscape exhibits numerous local optima, violating the assumptions under which **Gradient** succeeds.

Interestingly, our experiments show that randomly resetting reward weights helps escape poor local optima, leading to substantially better performance. We will provide more rigorous empirical validation of these findings in Sec. 6.

5 MORSE: Multi-Objective Reward Shaping with Exploration

The motivation example suggests that introducing noise into the bi-level optimization can potentially discover superior reward function and improve convergence performance. Inspired by this insight, we present a general framework to enhance bi-level optimization and propose a pratical technique, MORSE. In this section, we present MORSE, a stochastic optimization method that enhances exploration in the outer loop through selective noise injection based on two metrics, reward space novelty and task performance.

5.1 Exploration Guided by Task Performance and Reward Space Novelty

Each episode receives a task reward of 1 if it satisfies task success criteria, otherwise 0. We use P to denote task performance, which is the fraction of successful trajectories over all rollouts obtained using the current policy. Exploration is triggered with probability $1 - P$.

Reward-space novelty is measured using a curiosity-driven metric based on Random Network Distillation. Specifically, we introduce two networks: a fixed, randomly initialized target network f_{target} and a trainable predictor f_{pred}, both mapping reward weights to scalar values. Novelty is quanti-

fied as the prediction error between them, with larger discrepancies indicating more novel reward weights. The predictor is trained with the reward weights previously used for RL training, minimizing the MSE loss $= \|f_{\text{pred}}(w(s)) - f_{\text{target}}(w(s))\|_2$. We define the novelty metric for a weight candidate w as $V_{\text{novelty}}(w) = \|f_{\text{pred}}(w(s)) - f_{\text{target}}(w(s))\|_2$. Larger score indicates that w lies in a less explored region.

Outer-loop exploration proceeds by randomly sampling N candidate reward weights $W = \{w_i\}_{i=1}^N$ from the weight space. Each candidate is evaluated by the novelty metric $V_{\text{novelty}}(w_i)$, and a new weight W' is selected via softmax sampling over these scores.

Algorithm 1 Outer Loop Exploration

Require: History reward weights W_{past}, sample size N, the current task performance R_{curr}, the variation in task performance between two training epochs ΔR, improvement threshold α

if $\Delta R < \alpha$ **then**
  Train the predictor network f_{pred} with $\|f_{\text{pred}}(w) - f_{\text{target}}(w)\|_2, w \in W_{past}$
  Let $P_{curr} = \text{normalize}(R_{curr})$. With probablity $1 - P_{curr}$, perform the following steps:
  Randomly sample N points $W = \{w_i\}_{i=1}^N$ from the reward weight space
  Compute novelty metric: $V_{\text{novelty}}(w_i), \quad \forall w_i \in W$
  Sample from W based on softmax probabilities of V_{novelty}: $w \sim \textbf{Softmax}(V_{novelty})$
end if

5.2 Overall Algorithm

Combining the bi-level optimization and outer loop exploration, we present a realization of Multi-Objective Reward Shaping with Exploration, which is summarized in Alg. 2.

In each training epoch, we rollout data under the current policy π_θ and update θ via any on-policy or off-policy RL optimizer. Every T_{grad} epochs, the multi-objective weight vector ϕ is updated by ascending the gradient of $J(\phi)$, and, at a lower explore frequency $T_{explore}$, we reset ϕ through outer loop exploration and reinitialize the policy's parameters to encourage the policy to retain high entropy. By jointly optimizing θ and ϕ while periodically reinitializing the policy, MORSE balances efficient exploitation of learned reward compositions with continual exploration in the multi-objective space, yielding robust policies that negotiate conflicting objectives without manual reward engineering.

Algorithm 2 Multi-Objective Reward Shaping with Exploration (MORSE)

Require: Total training epoch N_{epoch}, RL policy π_θ, outer loop update interval T_{grad}, exploration interval $T_{explore}$

for $e = 0$ to N_{epoch} **do**
  Rollout: Execute policy π_θ to collect experience
  Update policy: Perform RL updates to optimize π_θ
  if $e \bmod T_{grad} = 0$ **then**
    Update reward weight: $\phi_{e+1} \leftarrow \phi_e + \nabla_\phi J(\phi_e)$
    if $e \bmod T_{explore} = 0$ **then**
      $w_\phi \leftarrow$ **MORSE_exploration**(ϕ_e)
      Reset policy: Reset the policy network parameters
    end if
  end if
end for

6 Experiments

We evaluate MORSE in 8 robotic tasks: 3 MuJoCo locomotion tasks, *Halfcheetah*, *Hopper*, and *Walker2d*, and 5 Isaac Sim tasks, including 3 locomotion tasks, *Quadcopter* (Isaac-Quadcopter-Direct-v0), *Unitree-A1-2obj* (modified Isaac-Velocity-Flat-Unitree-A1-v0), *Unitree-A1-9obj* (original Isaac-Velocity-Flat-Unitree-A1-v0), and 2 manipulation tasks, *Reach* (Isaac-Reach-Franka-v0), *Lift* (Isaac-Lift-Cube-Franka-IK-Rel-v0). These tasks involve 2, 3, 3, 3, 2, 9, 4, and 5 reward objectives, respectively.

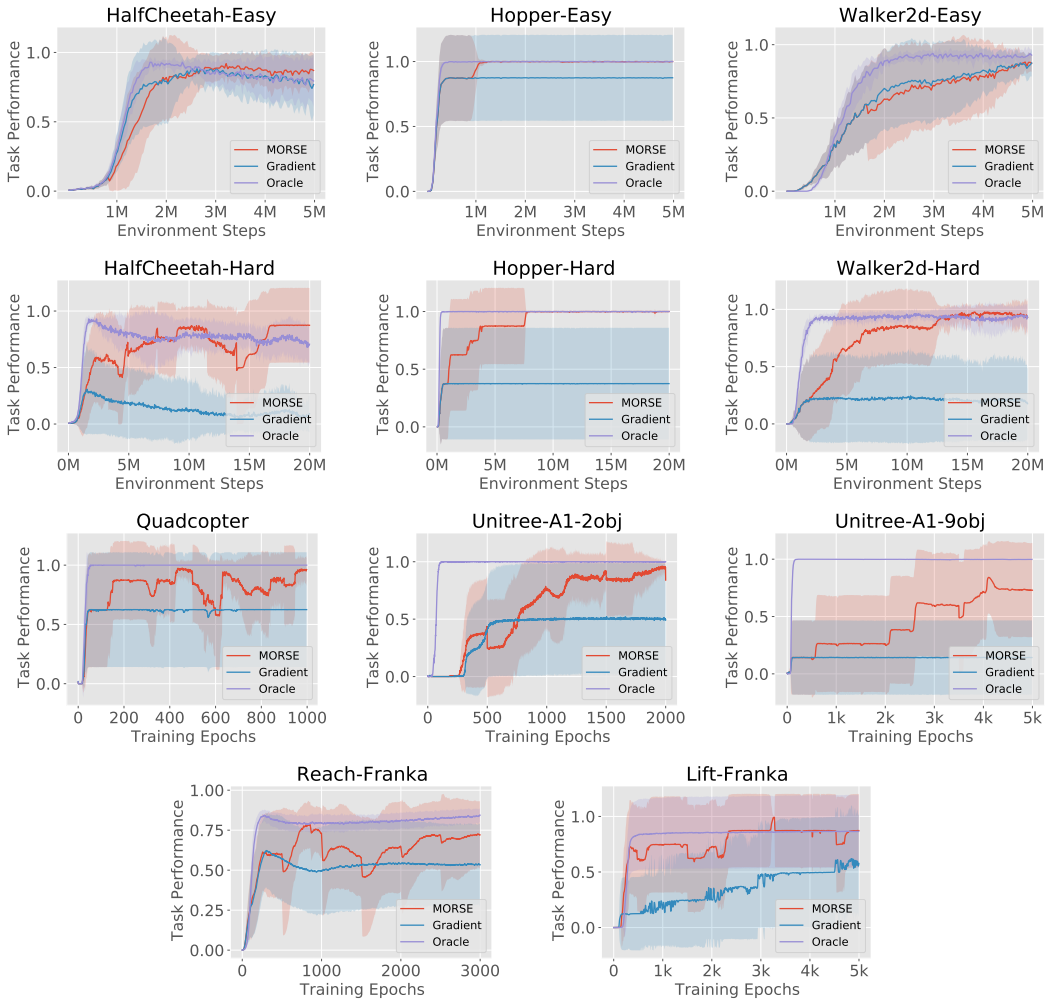


Figure 3: Main results in the MuJoCo and Isaac Sim environments. The results show that **MORSE** outperforms the vanilla bi-level optimization method and is comparable with **Oracle**, whose reward weights are tuned by human expert.

6.1 Experiment Setup

Task Criteria and Heuristic Functions. An episode is considered successful only if the agent survives to the end, reaches the target goal (e.g., specific velocity or position), and its entire trajectory remains within a predefined action threshold. In our experiments, the action threshold is determined by training a policy with oracle weights defined in the codebase and measuring the maximum action features (e.g., torque or angular velocity) of the policy rollouts. For practical applications, these action thresholds are often set empirically, as in [Chen et al., 2025]. Heuristic rewards correspond to the single-objective reward components defined in the original codebase. Detailed definitions of task criteria and heuristic functions for each environment are provided in the Appendix.

Implementation Details. Given the distinct API and rollout logic between MuJoCo and Isaac Sim, we implement MORSE using two different frameworks, Stable-Baselines3[Raffin et al., 2021] for MuJoCo and rsl-rl[Rudin et al., 2022] for Isaac Sim. In Stable-Baselines3, PPO's critic is extended to a multi-head critic that predicts returns for each heuristic function. In rsl-rl, we keep a single-head critic but store all reward components in the buffer.

Reward Weight Constraints. For MuJoCo tasks, we design two difficulty levels. **Easy** setting intialize reward weights randomly within $[0, 1]$ and maintains this range for weight update. In **Hard**, the range of reward weight is $[-1, 1]$. The larger range makes it significantly more challenging to

identify the optimal reward function. For Isaac Sim tasks, weights are initialized within $[0, 1]$ and clipped accordingly.

All experiments are repeated 8 times with different random seeds. In the plots, solid lines in the plots denote the means, and shaded regions denote the standard deviations. For more details on hyperparameter settings, please refer to the Appendix.

6.2 Main Results

We compare the following methods. **Oracle**: The policy is trained with the constant reward weight tuned by human expert. **Gradient**: The reward weight is updated using only the bi-level optimization gradient, equivalent to Barfi[Gupta et al., 2023]. **MORSE**: The reward weights are reset with MORSE.

Fig. 3 reports the main experiment results. Across all environments, **MORSE** consistently outperforms **Gradient** and achieves performance comparable to **Oracle**, successfully identifying reward weights that enable high policy success rates. Furthermore, the results on MuJoCo-easy and MuJoCo-hard tasks show that, as the reward space expands, the performance of **Gradient** degrades sharply, whereas **MORSE** maintains stable performance. This again validates the insights in Sec. 4: gradient-based optimization method is strongly sensitive to the initial weight selection and the size of exploration space, and is also prone to suboptimal convergence, whereas exploration helps it escape local optima. These findings highlight the critical role of outer-loop exploration in complex robotic tasks and demonstrate the potential of **MORSE** as a general framework for reward shaping.

6.3 Ablation Studies

MORSE incorporates three key design choices: (1) during gradient-based bi-level optimization, if the policy converges to unsatisfactory performance, the algorithm jumps to a new reward weight; (2) during exploration, the new starting weight is sampled based on the RND novelty metric; (3) when resetting the reward weight, the actor of the RL policy is also reset to encourage sufficient policy exploration.

We conduct an ablation study in the MuJoCo-Hard domain, comparing **MORSE** with three variants: **wo/ Performance Metric**, which resets weights at fixed intervals, regardless of policy performance; **wo/ Novelty Metric**, which replaces RND with random sampling during exploration; and **wo/ Reset Policy**, which keeps the policy unchanged when resetting reward weights. Results are shown in Fig. 4.

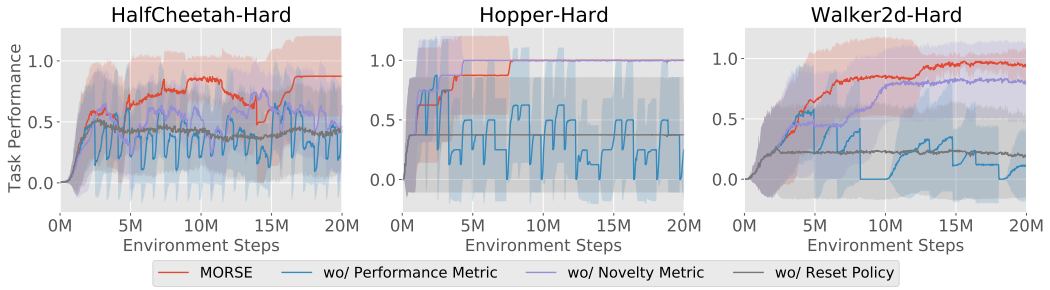


Figure 4: Ablate key components of MORSE in MuJoCo hard domain.

The discrepency between **MORSE** and **wo/ Performance Metric** highlights the importance of reset conditioning based on task performance. In **wo/ Performance Metric**, the reward weight switches periodically even after it reaches the global optimum, which degrades final performance.

wo/ Novelty Metric randomly samples a new reward weight during exploration. For simpler tasks (Hopper) where many reward weights yield high performance, the novelty metric has less impact. However, for harder tasks like Walker2D, selecting the most novel reward weight speeds up the coverage of the whole reward space.

Comparing **wo/ Reset Policy** with **MORSE**, we find that resetting the policy during outer loop exploration is essential for the policy to maintain high entropy and therefore adapt to reward changes.

7 Conclusion and Future Work

Our goal is to reduce the burden of manual reward shaping in RL, simplifying both task design and training. In MORSE, human experts only need to specify sparse task criterion and dense heuristic functions without defining their interactions, while the policy automatically learns to solve complex robotic tasks. We hope this framework can make RL training more scalable and broadly applicable.

There remain several directions for improvement. In outer-loop exploration, novelty and performance metrics could be combined more effectively, for example by considering the historical performance of reward weights and adopting Upper Confidence Bound[Auer et al., 2002] to better balance exploration and exploitation. Moreover, the current design requires resetting the whole policy when switching reward weights. Incorporating successor features[Barreto et al., 2017] into the policy structure may enable more efficient reuse of past knowledge and faster adaptation to new reward weights.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2018. URL `https://arxiv.org/abs/1707.01495`.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Serena Booth, W Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, Feb 2023.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Andrea Castelletti, Francesca Pianosi, and Marcello Restelli. Tree-based fitted q-iteration for multi-objective markov decision problems. In *The 2012 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2012.

Jiayu Chen, Chao Yu, Yuqing Xie, Feng Gao, Yinuo Chen, Shu'ang Yu, Wenhao Tang, Shilong Ji, Mo Mu, Yi Wu, et al. What matters in learning a zero-shot sim-to-real rl policy for quadrotor control? a comprehensive study. *IEEE Robotics and Automation Letters*, 2025.

Rati Devidze, Parameswaran Kamalaruban, and Adish Singla. Exploration-guided reward shaping for reinforcement learning under sparse rewards. *Advances in Neural Information Processing Systems*, 35:5829–5842, 2022.

Florian Felten, Lucas N. Alegre, Ann Nowé, Ana L. C. Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno Castro da Silva. A toolkit for reliable benchmarking and research in multi-objective reinforcement learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023)*, 2023.

Justin Fu, John Co-Reyes, and Sergey Levine. Ex2: Exploration with exemplar models for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Dhawal Gupta, Yash Chandak, Scott Jordan, Philip S Thomas, and Bruno C da Silva. Behavior alignment via reward function optimization. *Advances in Neural Information Processing Systems*, 36:52759–52791, 2023.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1):1–18, 2003.

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

Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33:15931–15941, 2020.

Rustam Issabekov and Peter Vamplew. An empirical comparison of two common multiobjective reinforcement learning algorithms. In *AI 2012: Advances in Artificial Intelligence: 25th Australasian Joint Conference, Sydney, Australia, December 4-7, 2012. Proceedings 25*, pages 626–636. Springer, 2012.

Sam Lobel, Akhil Bagaria, and George Konidaris. Flipping coins to estimate pseudocounts for exploration in reinforcement learning. In *International Conference on Machine Learning*, pages 22594–22613. PMLR, 2023.

Haoye Lu, Daniel Herman, and Yaoliang Yu. Multi-objective reinforcement learning: Convexity, stationarity and pareto optimality. In *The Eleventh International Conference on Learning Representations*, 2022.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5125–5133, 2020.

Gabriel B Margolis and Pulkit Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. In *Conference on Robot Learning*, pages 22–31. PMLR, 2023.

Farzan Memarian, Wonjoon Goo, Rudolf Lioutikov, Scott Niekum, and Ufuk Topcu. Self-supervised online reward shaping in sparse-reward environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2369–2375. IEEE, 2021.

Hossam Mossalam, Yannis M Assael, Diederik M Roijers, and Shimon Whiteson. Multi-objective deep reinforcement learning. *arXiv preprint arXiv:1610.02707*, 2016.

Sriraam Natarajan and Prasad Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 601–608, 2005.

Simone Parisi, Matteo Pirotta, Nicola Smacchia, Luca Bascetta, and Marcello Restelli. Policy gradient approaches for multi-objective sequential decision making. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 2323–2330. IEEE, 2014.

Simone Parisi, Matteo Pirotta, and Jan Peters. Manifold-based multi-objective policy search with sample reuse. *Neurocomputing*, 263:3–14, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

Mathieu Reymond, Eugenio Bargiacchi, and Ann Nowé. Pareto conditioned networks. *arXiv preprint arXiv:2204.05036*, 2022.

Mathieu Reymond, Conor F Hayes, Denis Steckelmacher, Diederik M Roijers, and Ann Nowé. Actor-critic multi-objective reinforcement learning for non-linear utility functions. *Autonomous Agents and Multi-Agent Systems*, 37(2):23, 2023.

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

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 91–100. PMLR, 2022. URL `https://proceedings.mlr.press/v164/rudin22a.html`.

Manuela Ruiz-Montiel, Lawrence Mandow, and José-Luis Pérez-de-la Cruz. A temporal difference method for multi-objective reinforcement learning. *Neurocomputing*, 263:15–25, 2017.

Umer Siddique, Paul Weng, and Matthieu Zimmer. Learning fair policies in multi-objective (deep) reinforcement learning with average and discounted rewards. In *International Conference on Machine Learning*, pages 8905–8915. PMLR, 2020.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512, 2014.

Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pages 191–199. IEEE, 2013a.

Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé. Hypervolume-based multi-objective reinforcement learning. In *Evolutionary Multi-Criterion Optimization: 7th International Conference, EMO 2013, Sheffield, UK, March 19-22, 2013. Proceedings 7*, pages 352–366. Springer, 2013b.

Yuqing Xie, Chao Yu, Hongzhi Zang, Feng Gao, Wenhao Tang, Jingyi Huang, Jiayu Chen, Botian Xu, Yi Wu, and Yu Wang. Multi-uav behavior-based formation with static and dynamic obstacles avoidance via reinforcement learning. *arXiv preprint arXiv:2410.18495*, 2024.

Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Advances in neural information processing systems*, 32, 2019.

Chen Bo Calvin Zhang, Zhang-Wei Hong, Aldo Pacchiano, and Pulkit Agrawal. Orso: Accelerating reward design via online reward selection and policy optimization. *arXiv preprint arXiv:2410.13837*, 2024a.

Yihua Zhang, Prashant Khanduri, Ioannis Tsaknakis, Yuguang Yao, Mingyi Hong, and Sijia Liu. An introduction to bilevel optimization: Foundations and applications in signal processing and machine learning. *IEEE Signal Processing Magazine*, 41(1):38–59, 2024b.

Yunzhi Zhang, Pieter Abbeel, and Lerrel Pinto. Automatic curriculum learning through value disagreement. *Advances in Neural Information Processing Systems*, 33:7648–7659, 2020.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *Advances in neural information processing systems*, 31, 2018.

A Practical Guidelines for Deploying MORSE

When deploying MORSE, we identify several important conditions which we recommend practitioners to follow.

- Make sure the inner-loop policy converges before performing outer-loop updates (including gradient updates and exploration). For bi-level gradient to hold, the policy must reach its local optimum, otherwise the gradient would be distorted. As for exploration, it relies on the policy's current performance to decide whether to reset weights.
- Since we approximate the Hessian numerically, monitoring its stability is crucial. If the series diverges, the update should be skipped.
- Use slower outer-loop updates. Slowing down outer-loop updates, especially in the early stage of training, improves stability.
- Although MORSE enhances exploration of the reward weight space, extremely high-dimensional settings remain challenging. Stronger priors that narrow the weight range can significantly improve performance. If the search space is large with only a small valid region, more training iterations are required for sufficient exploration.
- Resetting the policy ensures high action entropy under new reward weights, enabling effective exploration.

We also note that, MORSE is a modular framework, and its components can be flexibly adapted.

- For outer-loop gradient calculation, while we adopt implicit differentiation, other meta-learning approaches could be used, such as [Hu et al., 2020]. Our choice balances computational complexity against gradient approximation accuracy.
- Users can employ other outer-loop exploration methods with different novelty metrics. For example, for simpler MuJoCo tasks, random sampling is often sufficient.
- For inner-loop policy updates, users can choose any RL algorithms they like, such as PPO or SAC.

B Experimental Settings

B.1 Task Definition

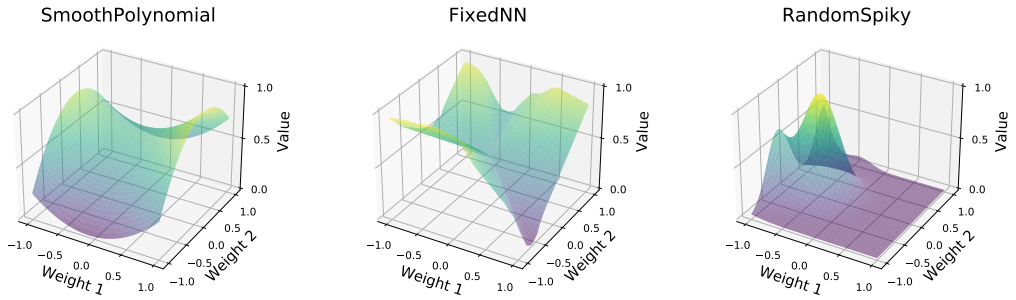


Figure 5: Example optimizaiton landscapes.

In Sec. C.1, we design three classes of synthetic functions to simulate optimization landscapes of varying difficulty: (1) SmoothPolynomial, a continuously differentiable random polynomial; (2) FixedNN, a function generated by a fixed randomly initialized neural network, which is non-smooth but exhibits no abrupt value shifts; (3) RandomSpiky, a smooth function with sparse sharp peaks. All functions are normalized to produce outputs within the range [0,1]. We present visualizations of representative instances from each class in Fig. 5.

We provide the task definition for RL environments in Tab. 1. In MuJoCo environments, the agent gets a task reward of 1 when it satisfies all task criteria, which are determined using the following protocol. First, we train a policy to convergence using the default reward weights provided in the

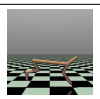
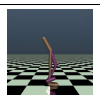
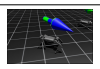
Environment	Robot	Task Criteria	Heuristic Functions	Weight
Cartpole		Final position > 0.5	Survival reward: 1 Position reward: $\mathbf{1}_{\text{pos}>0.5}$ Interference reward: $\mathbf{1}_{\text{action}_\pi=\text{action}_{\text{PID}}}$	– – –
HalfCheetah		Final position > 5 control cost ≥ -0.6 for each step	Forward reward: $\frac{dx}{dt}$ Control cost: $-\|\|action\|\|_2^2$	0.4 1
Hopper		Final position > 0.8 control cost ≥ -0.004 for each step	Survival reward: 1 Forward reward: $\frac{dx}{dt}$ Control cost: $-\|\|action\|\|_2^2$	1 1 0.002
Walker2d		Final position > 0.8 control cost ≥ -0.006 for each step	Survival reward: 1 Forward reward: $\frac{dx}{dt}$ Control cost: $-\|\|action\|\|_2^2$	1 1 0.002
Quadcopter		L2 distance to goal < 4 linear velocity < 100 angular velocity < 10000	Position reward: $1 - \tanh(\|\|p - p^*\|\|^2/0.8)$ Linear velocity cost: $-\Sigma\|\|v\|\|^2$ Angular velocity cost: $-\Sigma\|\|\omega\|\|^2$	15 0.05 0.01
Unitree-A1-2obj		L2 tracking error < 0.22 L2 action rate < 40	Tracking reward: $e^{-(v-v^*)^2}$ Action rate penalty: $-\|\|a_t - a_{t-1}\|\|_2^2$	1.5 0.01
Unitree-A1-9obj		L2 tracking error < 0.22 L2 action rate < 40	Linear xy velocity error: $e^{-(v_{xy}-v_{xy}^*)^2}$ Angular yaw velocity error: $e^{-(v_{yaw}-v_{yaw}^*)^2}$ Linear z velocity penalty: $-\|\|v_z\|\|_2^2$ Angular-xy velocity penalty: $-\|\|\boldsymbol{\omega}_{xy}\|\|_2^2$ Joint torque penalty: $-\|\|\boldsymbol{\tau}\|\|_2^2$ Joint acceleration penalty: $-\|\|\ddot{\mathbf{q}}\|\|_2^2$ Action rate penalty: $-\|\|a_t - a_{t-1}\|\|_2^2$ Air-time reward: $\Sigma_i(A_i - \tau)F_i$, if$\|\|u_{cmd}\|\| > 0.1$ Non-flat orientation penalty: $-\|\|\mathbf{g}_{b,xy}\|\|_2^2$	1.5 0.75 -2 -0.05 -0.0002 -2.5e-07 -0.01 0.25 -2.5
Reach-Franka		Position tracking error < 0.2 Fine-grained tracking reward > 0.5 Orientation tracking error<0.6 L2 action rate < 100	Position tracking error: $\|\|\mathbf{p} - \mathbf{p}^*\|\|_2$ Fine-grained pos reward: $1 - \tanh(\|\|\mathbf{p} - \mathbf{p}^*\|\|/0.1)$ Orientation tracking error: $\mathrm{d}_{\text{quat}}(\mathbf{q}, \mathbf{q}^*)$ Action rate penalty: $-\|\|a_t - a_{t-1}\|\|_2^2$	-0.2 0.1 -0.1 -0.0001
Lift-Franka		L2 object position tracking error < 0.22 L2 action rate < 25000	Reaching object reward: $1 - \tanh\left(\frac{\|\|\mathbf{p}_{\text{obj}} - \mathbf{p}_{\text{ee}}\|\|_2}{0.1}\right)$ Lifting object reward: lift $= \mathbf{1}\{z_{\text{obj}} > 0.04\}$ Object goal tracking: lift $* \left[1 - \tanh\left(\frac{\|\|\mathbf{p}^* - \mathbf{p}_{\text{obj}}\|\|_2}{0.3}\right)\right]$ Fine-grained goal tracking: lift $* \left[1 - \tanh\left(\frac{\|\|\mathbf{p}^* - \mathbf{p}_{\text{obj}}\|\|_2}{0.05}\right)\right]$ Action rate penalty: $-\|a_t - a_{t-1}\|_2^2$	1.0 15.0 16.0 5.0 -0.0001

Table 1: Task definition for RL environments.

original environment. Then, we perform multiple rollouts and record the minimum, maximum, and mean values of each objective across episodes. Based on these statistics, we define task criteria such that the default policy achieves a success rate exceeding 90%.

To accelerate training, we shorten the episode length and modify the Unitree-A1 task from 9 objectives to 2 objectives. The maximum episode lengths for the environments are 100, 100, 100, 100, 200, 250, respectively.

B.2 Hyperparameter

We use REINFORCE as the inner-loop optimizer for the CartPole experiments in Sec.4, and PPO for the MuJoCo environments and the Isaac Sim environments in Sec.6. Most hyperparameters follow the defaults of their respective implementations. Key hyperparameters are summarized in Table 2.

For CartPole, we use a single environment and set the maximum buffer size to 10,000. The inner-loop REINFORCE actor uses a learning rate of 0.001 and an L2 regularization coefficient of 0.25. The outer loop employs 5-step Neumann approximation with a learning rate of 0.001.

For MuJoCo, we use 32 parallel environments. Each PPO update uses a buffer size of 16,384 environment steps. The actor learning rate is 0.0003, value function coefficient is 0.5, maximum gradient is capped at 1.0, and the entropy coefficient is set to 0.

For Isaac Sim, we use 50 parallel environments. We use adaptive learning schedule for inner-loop policy training. The value function coefficient is 1.0, kl coefficient is 0.01, and the entropy coeffi-

Environment	Inner Loop n_epochs	Outer Loop gradient frequency	Outer Loop reset frequency
Cartpole	15	1	–
HalfCheetah	10	5	25
Hopper	10	5	25
Walker2d	20	15	50
Quadcopter	5	20	100
Unitree-A1-2obj	20	20	250
Unitree-A1-9obj	20	20	250
Reach-Franka	20	20	250
Lift-Franka	20	50	500

Table 2: Hyperparameters for RL environments.

cient is set to 0.01. The learning rate for Unitree-A1 is 1e-3, and the learning rate for Quadcopter is 5e-4.

The outer loop applies a softmax-based sampling strategy with temperature $\tau = 10$, where the sampling probability is given by $P(V_{\text{perf}_i}) = \frac{e^{V_{\text{perf}_i} * \tau}}{\sum_j e^{V_{\text{perf}_j} * \tau}}$. To preserve human-designed reward heuristics, we set the learning rate for the reward-weight network to 0.0005 and the coefficient learning rate to 0.0025. The number of gradient updates for reward weights is adaptive, with a minimum of 3 and a maximum of 10 iterations per outer-loop step, terminating early upon convergence. Similarly, the prediction network is trained using a dynamic number of iterations, from 1 to 1000, to allow it to overfit known data points during each update cycle.

Each run takes less than 1.5 hour on a 4090 GPU.

C Additional Experiments and Analysis

C.1 Preliminary Validation of Outer Loop Exploration

We validate the effect of outer loop in a simplified environment. Focusing only on the outer loop update and ignoring the inner loop RL training, we treat reward shaping as an optimization problem in a multi-dimensional space. Each reward weight corresponds to a point in the space, and the optimization objective is task performance, represented as a scalar value. For simplicity, we assume the reward weight is 2D, $x \in [-1, 1]^2$, and task performance is determined by the function $f(x) \in [0, 1]$. The optimization goal is to find the input value that maximizes $f(x)$ within 100 steps.

We design three types of functions with different characteristics to construct varying optimization landscapes: a smooth polynomial function with multiple local optima (SmoothPolynomial), a complex fixed neural network function (FixedNN), and a function with sparse spike regions (RandomSpiky). These functions are normalized to ensure the output is always within the $[0, 1]$ range. For each of the three function types, we randomly generate 10 parameter sets, resulting in a total of 30 functions. We conduct 10 independent optimization runs for each function, starting from different random initial points.

C.1.1 Timing of Exploration

We evaluate several strategies for resetting the reward weight. **No Exploration** updates the reward weights only by the gradient of the outer loop with no exploration. **Periodic Exploration** samples new reward weights based on novelty scores and resets the reward weights after every 10 gradient updates. **MORSE** reset the reward weight with the novelty metric and performance criteria after every 10 gradient updates.

As shown in Tab. 3, the results show that **MORSE** significantly outperforms all baseline methods. **No Exploration** converges to suboptimal solutions on complex optimization landscapes, such as RandomSpiky. Meanwhile, **Periodic Exploration** reset the reward weights periodically, disrupting the gradient-based optimization process, which leads to worse performance compared to the simple

	RandomSpiky	SmoothPolynomial	FixedNN	Average
No Exploration	0.200 (0.422)	**0.832** (0.213)	0.945 (0.141)	0.659 (0.432)
Periodic Exploration	0.102 (0.316)	0.690 (0.214)	0.695 (0.306)	0.496 (0.393)
MORSE	**0.873** (0.319)	0.799 (0.273)	**0.946** (0.140)	**0.873** (0.254)

Table 3: Average performance of different exploration timing across 10 seeds. The results show that **MORSE** outperforms other baselines in all three optimization landscapes.

gradient-based method, **No Exploration**. In contrast, **MORSE** can terminate random exploration based on performance criteria, achieving a balance between exploration and gradient optimization.

C.1.2 Sampling Algorithm for Exploration

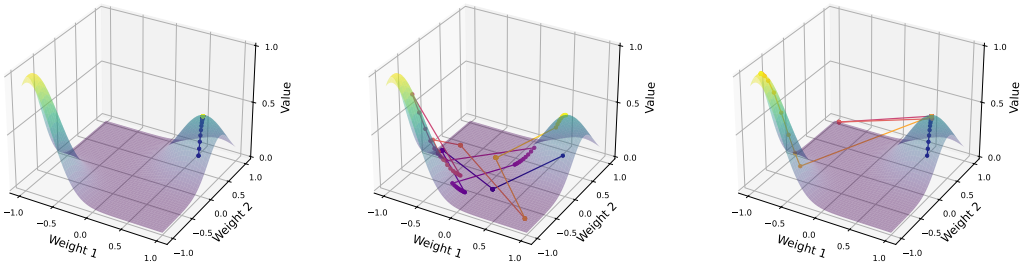
In this set of experiments, we replace RND-based sampling with alternative strategies while keeping all other settings fixed. The baselines include random sampling (random) and two canonical sample-based optimization methods, the cross-entropy method (CEM) [Rubinstein, 1999] and covariance matrix adaptation (CMA) [Hansen et al., 2003]. For CEM and CMA, each batch samples five candidate weight vectors, and the top 40% (two elites) are retained to guide the next round.

	RandomSpiky	SmoothPolynomial	FixedNN	Average
random	0.701 (0.483)	**0.872** (0.204)	0.946 (0.140)	0.840 (0.320)
CEM	0.302 (0.482)	0.711 (0.273)	0.770 (0.186)	0.594 (0.388)
CMA	0.801 (0.422)	0.698 (0.329)	0.932 (0.185)	0.810 (0.330)
RND(ours)	**0.873** (0.319)	0.799 (0.273)	**0.946** (0.140)	**0.873** (0.254)

Table 4: Average performance of different sampling strategies across 10 seeds. Using RND to inject noise outperforms other baselines in two of the three optimization landscapes.

We present the results in Tab. 4. Compared with random, RND discovers more novel reward weights within a limited number of exploration steps. The performance of CEM and CMA degrades significantly when the number of samples is restricted, as in our setting, or when the optimization landscape is highly non-smooth, as in RandomSpiky. Both methods generate multiple candidates per round and update their sampling strategies based on batch performance. In contrast, our approach samples weights sequentially, one at a time, which substantially reduces the total number of samples required to reach a feasible solution. Since each sample corresponds to a full RL training process, this results in markedly higher computational efficiency.

C.1.3 Visualization of Optimization Path



(a) No Exploration (b) Periodic Exploration (c) Ours (MORSE)

Figure 6: Visualization of optimization path. MORSE optimizes towards the global optimum on the lower left.

We visualize the optimization path of each experiment in Fig. 6. The results show that **MORSE** can quickly find the most novel reward weight and perform a stable gradient update near the reward weight that performs well, resulting in optimal performance.

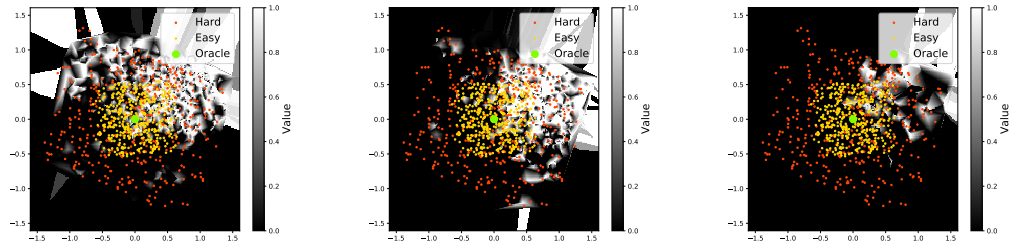


Figure 7: Visualization of the optimization planes of MuJoCo environments using PCA. The color denotes task success rate. The brighter, the higher. From left to right: HalfCheetah, Hopper, Walker2d.

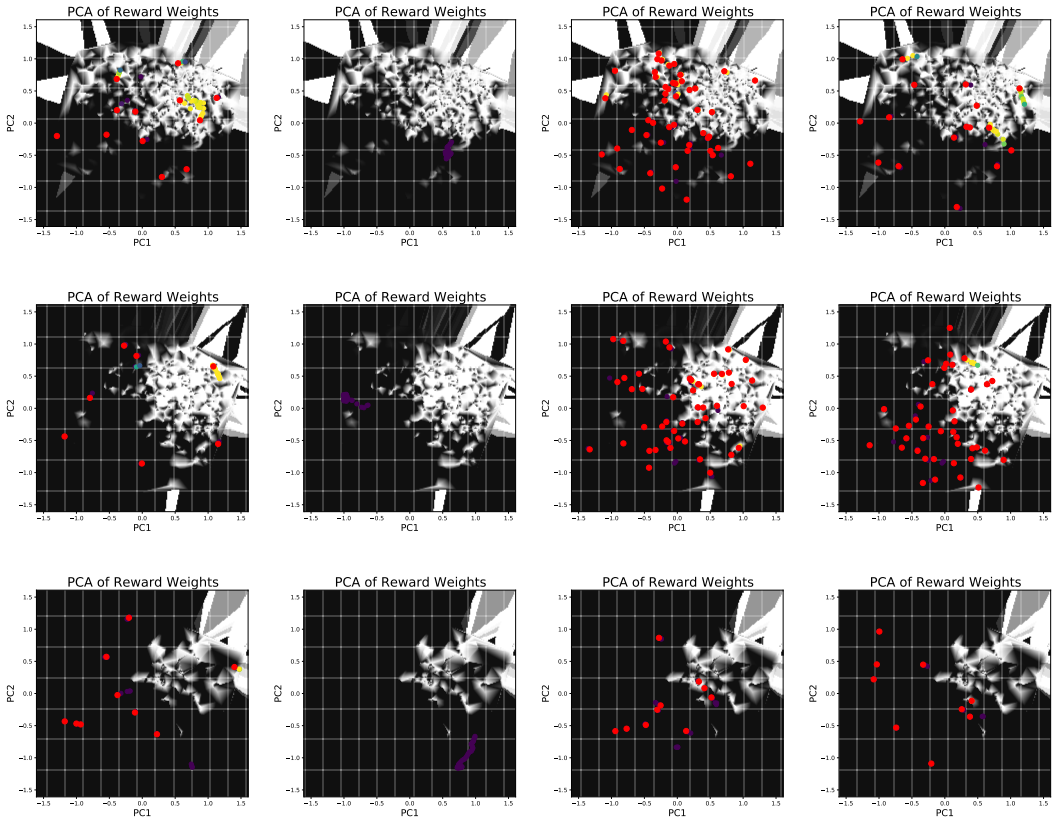


Figure 8: Evolution of reward weights in MuJoCo-Hard environments. Each row corresponds to a different environment, and each column represents a method. From top to bottom: HalfCheetah-Hard, Hopper-Hard, Walker2d-Hard. From left to right: **MORSE**, **Gradient**, **wo/ Performance Metric**, **wo/ Novelty Metric**. The color of the scatter points indicates the number of outer-loop updates, with lighter colors representing more updates. Red points denote new starting points selected when resetting the reward weight.

C.2 PCA Visualization of Weight Selection

In this section, we visualize the changes of the reward weight with each outer-loop update during the training process in MuJoCo environments. Since the reward weight is three-dimensional, we apply PCA to reduce it to two dimensions for visualization.

In Fig. 7, we illustrate the relationship between reward weight and task success rate across the entire reward weight space. We collect reward weights and success rates from all experiments and then apply interpolation and edge filling to complete the missing regions. We also highlight the human-

tuned oracle reward weight and mark the valid reward weight ranges for both easy and hard settings with 400 randomly sampled points. The resulting optimization landscape exhibits multiple local optima, consistent with our hypothesis that robotic task optimization landscapes are often rugged and therefore challenging for gradient-based optimization.

Fig. 8 presents the evolution of reward weight for MuJoCo-Hard environments. We compare four methods: **MORSE**, **Gradient**, and two ablation variants, namely, **wo/ Performance Metric** and **wo/ Novelty Metric**. All methods start optimizing from the same initial point.

Gradient is highly sensitive to the choice of initial point. When the initial point falls to undesired areas, gradient updates can be ineffective. **wo/ Performance Metric** escapes the optimal solution. **wo/ Novelty Metric** covers the reward weight space more slowly. For more challenging tasks such as Walker2d, which have lower exploration frequency and fewer reset chances, **wo/ Novelty Metric** limits the ability to find suitable reward weights.