EvoRainbow: Combining Improvements in Evolutionary Reinforcement Learning for Policy Search

Pengyi Li [1] Yan Zheng [1] Hongyao Tang [1] Xian Fu [1] Jianye Hao [1]

Abstract

Both Evolutionary Algorithms (EAs) and Reinforcement Learning (RL) have demonstrated powerful capabilities in policy search with different principles. A promising direction is to combine the respective strengths of both for efficient policy optimization. To this end, many works have proposed various mechanisms to integrate EAs and RL. However, it is still unclear which of these mechanisms are complementary and can be fully combined. In this paper, we revisit different mechanisms from five perspectives: 1) Interaction Mode, 2) Individual Architecture, 3) EAs and Operators, 4) Impact of EA on RL, and 5) Fitness Surrogate and Usage. We evaluate the effectiveness of each mechanism and experimentally analyze the reasons for the more effective mechanisms. Using the most effective mechanisms, we develop EvoRainbow and EvoRainbow-Exp, which outperform strong baselines and provide state-of-the-art performance across various tasks with distinct characteristics. To promote community development, we release the code on https://github.com/yeshenpy/EvoRainbow.

1. Introduction

Policy search is a crucial research direction in the field of machine learning, aimed at finding feasible solutions to sequential decision problems. Evolutionary Algorithms (EAs) (Bäck & Schwefel, 1993) and Reinforcement Learning (RL) (Sutton & Barto, 1998) are two widely used methods in this context, with applications in domains such as robot control (Johannink et al., 2019), game AI (Vinyals et al., 2019), and recommender systems (Zou et al., 2019). EAs are gradient-free optimization algorithms that rely on

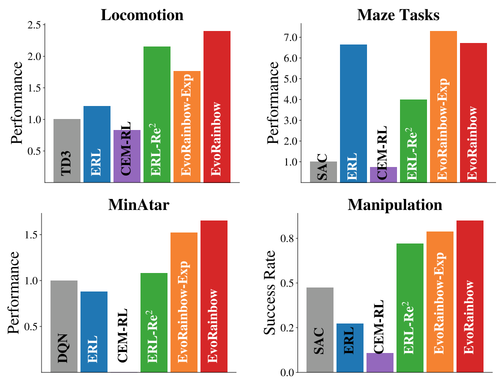


Figure 1. **Overview**. EvoRainbow and EvoRainbow-Exp compare favorably to existing ERL methods across various tasks.

a population of individuals and mimic the process of biological evolution through selection, mutation, and reproduction (Bäck & Schwefel, 1993; Chalumeau et al., 2022). EAs excel in exploration, convergence, and robustness but suffer from sample inefficiency due to the iterative population evaluation (Such et al., 2017; Salimans et al., 2017). In contrast, RL maintains a single individual and employs value function approximation to provide reliable gradients for policy search (Cobbe et al., 2021). Besides, RL exhibits high sample efficiency (Fujimoto et al., 2018; Haarnoja et al., 2018), especially for off-policy RL, but they often struggle with exploration and convergence (Duan et al., 2016; Haarnoja et al., 2018; Fortunato et al., 2018; Raileanu & Fergus, 2021; Li et al., 2022; Hao et al., 2023b; Liu et al., 2024; Li et al., 2024). Building upon their distinct characteristics, the integration of EAs and RL for policy search has emerged as a promising research direction. This integration leverages their complementary strengths, leading to outstanding performance on various tasks (Sigaud, 2022; Li et al., 2024).

ERL (Khadka & Tumer, 2018) stands as the earliest and most influential work that combines EAs and RL for policy search. ERL utilizes the Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2016) in combination with Genetic Algorithm (GA), where GA maintains a population of policy networks and employs the cumulative reward of individuals interacting with the environment as a fitness measure for population evaluation. By leveraging the

[1]College of Intelligence and Computing, Tianjin University, China. Correspondence to: Yan Zheng <yanzheng@tju.edu.cn>, Jianye Hao <jianye.hao@tju.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

diverse experiences generated during the population evaluation process, ERL addresses the challenge of the poor exploration capability of RL. Furthermore, ERL periodically injects the RL policy into the population to participate in the evolution process of GA. Notably, ERL demonstrates significant improvements over DDPG on MUJOCO tasks. Subsequently, several mechanisms are proposed to further integrate the strengths of EAs and RL. These mechanisms include distillation-based crossover and safety mutation operators in PDERL (Bodnar et al., 2020), the use of the RL Critic for population evaluation, and two additional mechanisms to enhance sample efficiency in SC (Wang et al., 2022), the adoption of Cross-Entropy Method and TD3 in CEM-RL (Pourchot & Sigaud, 2019), using EA to improve RL learning efficiency with Genetic Soft Update in Supe-RL (Marchesini et al., 2021), the variant of CEM-RL scheme and Elite Policy Guide in PGPS (Kim et al., 2021), and the efficient knowledge transfer and policy search through shared state representation and independent policy representation in ERL-Re^2 (Hao et al., 2023a). We revisit these mechanisms in Section 3. For hybrid algorithms related to policy search, we recommend readers to refer to the following surveys (Sigaud, 2022; Zhu et al., 2023; Li et al., 2024) for a more comprehensive understanding.

Previous researches propose various mechanisms to integrate EA and RL from different perspectives (Li et al., 2024). However, these mechanisms often lack systematic analysis and a comprehensive understanding of their complementarity and potential for further performance improvement. Additionally, most previous researches primarily focus on MUJOCO tasks with dense rewards, limiting the generalizability of their conclusions. To address these limitations and provide more insights, we revisit numerous mechanisms from five perspectives: 1) Interaction Mode, 2) Individual Architecture, 3) EAs and Operators, 4) Impact of EA on RL, and 5) Fitness Surrogate and Usage. We systematically compare these different mechanisms on three categories of tasks: dense reward, deceptive reward, and sparse reward. This evaluation approach allows us to gain a comprehensive understanding of the mechanisms' effectiveness across various types of tasks. Subsequently, we fuse complementary mechanisms to build EvoRainbow and EvoRainbow-Exp, incorporating the best mechanisms from different perspectives. EvoRainbow excels in handling tasks with standard rewards, whereas EvoRainbow-Exp is designed for exploration tasks with low-quality rewards (e.g., sparse or deceptive rewards). The results summarized in Figure 1 demonstrate EvoRainbow and EvoRainbow-Exp outperform the current state-of-the-art baselines on 20 tasks, including MinAtar (Young & Tian, 2019), MUJOCO (Todorov et al., 2012), Trap tasks, Maze tasks, and Robot manipulation tasks (Yu et al., 2019a).

Our contributions can be summarized as follows: i) We conduct a systematic comparison of the mechanisms proposed by previous ERL-related methods from different perspectives and provide in-depth analysis to offer more insights. ii) We propose EvoRainbow and EvoRainbow-Exp as the best frameworks by integrating the most effective mechanisms from different perspectives. iii) We demonstrate the superiority of EvoRainbow and EvoRainbow-Exp over all existing ERL-related algorithms on a wide range of tasks, including MinAtar, MuJoCo, PointMaze, AntMaze, and Robot manipulation tasks, thereby demonstrating their efficiency.

2. Background

Reinforcement Learning can be formalized as a Markov Decision Process (MDP) (Puterman, 1990) which can be defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, T, \rho \rangle$, where $\mathcal{S}$ is the state set, $\mathcal{A}$ is the action set, $\mathcal{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the transition function, $\mathcal{R}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, $\gamma \in [0, 1)$ is the discounted factor and T is the horizon. ρ represents the distribution of the initial state. The agent interacts with the environment by performing its policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$. RL (Sutton & Barto, 1998) aims to optimize the policy to maximize the expected discounted cumulative reward $J(\pi) = \mathbb{E}_{a_t \sim \pi(s_t), s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)}[\sum_{t=0}^{T} \gamma^t r_t]$, where $r_t = \mathcal{R}(s_t, a_t)$ and $s_0 \sim \rho$. The state-action value function Q^π is defined as $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t \mid s_0 = s, a_0 = a\right]$.

Evolutionary Algorithms (EAs) (Bäck & Schwefel, 1993) are gradient-free optimization methods that utilize a population of individuals (policies) $\mathbb{P} = \{\pi_1, \pi_2, ..., \pi_n\}$ to explore the policy space. EAs evaluate individuals based on their fitness $\{f(\pi_1), f(\pi_2), ..., f(\pi_n)\}$, typically defined as the average cumulative reward obtained by interacting with the environment $f(\pi) = \frac{1}{e}\sum_{i=1}^{e}[\sum_{t=0}^{T} r_t \mid \pi]$. Two commonly used EAs in this context are Genetic Algorithm (GA) (Mitchell, 1998; Such et al., 2017) and Cross-Entropy Method (CEM) (Boer et al., 2005). GA is a classic evolutionary algorithm that employs genetic operators such as crossover and mutation to explore the policy search space. The individuals (policies) in the population are selected based on their fitness, and new offspring are created through crossover and mutation operations. k-point crossover and Gaussian mutation are commonly used operators. The k-point crossover randomly exchanges segment-wise (network) parameters of parents, whereas the Gaussian mutation adds Gaussian noises to the parameters to randomly generate offspring. CEM is an Estimation of Distribution Algorithm (EDA) (Larrañaga & Lozano, 2001). It represents the population as a distribution using a covariance matrix. CEM iteratively updates the distribution by sampling individuals from it $x_i \sim \mathcal{N}(\mu, \Sigma)$, evaluating their fitness, and using the top-performing individuals $\{z_1, ..., z_{\frac{n}{2}}\}$ to update the distribution parameters. The process aims to converge toward the optimal solution by gradually shifting the distribution. We follow the CEM-RL (Pourchot & Sigaud, 2019) scheme and update CEM distribution as follows: $\mu_{new} = \sum_{i=1}^{|\mathbb{P}|/2} \lambda_i z_i$, $\mathbf{\Sigma}_{new} = \sum_{i=1}^{K_e} \lambda_i \left(z_i - \mu_{old}\right)^2 + \epsilon\mathcal{I}$., where λ_i is the weight given to the individual i.

3. Revisiting ERL Literature

3.1. High-level Overview from Five Perspectives

Many works integrate EAs and RL to design efficient hybrid policy search algorithms (Li et al., 2024). However, due to the complexity of algorithms, i.e., an algorithm involves multiple mechanisms, coupled with the lack of a comprehensive and systematic comparison, it remains unclear which mechanisms are more efficient for different types of tasks. To facilitate further analysis, we revisit these mechanisms. However, the number of possible combinations for different mechanisms exceeds one thousand, making it impractical to consider analyzing all of them in experiments. Therefore, we decouple the algorithms into combinations of multiple mechanisms and categorize them into the following five perspectives:

- ☆ **Interaction Mode**: The foundational framework that determines the dominance between EA and RL in policy search.
- ☆ **Individual Architecture**: The basic policy architecture for individuals in the population and RL agent.
- ☆ **Evolutionary Algorithms and Operator Selection**: This determines the EA used to improve the population.
- ☆ **How EA impacts RL**: This entails the mechanisms through which EA facilitates RL.
- ☆ **Fitness Surrogate Selection and Usage**: These mechanisms are employed for population evaluation, aiming to reduce the sample cost.

These five perspectives constitute the process of constructing an ERL algorithm. We study these five perspectives sequentially. By exploring these perspectives, we aim to provide a thorough understanding of the related ERL approaches and shed light on their strengths and limitations.

3.2. Mechanisms Classification

In this section, we provide a detailed description of the mechanisms involved from various perspectives to facilitate a better understanding.

Interaction Mode: The interaction modes primarily consist of three types: *Parallel* mode, *EA-Master* mode, and *RL-Master* mode. **i) In *Parallel* mode**, EAs and RL simultaneously conduct policy searches and influence each other. The algorithms typically consist of an RL policy (Actor), a value function (Critic), and a population of policy networks. EAs improve the population with evolution operators. The samples generated during the population evaluation process are provided to RL, which improves the experience diversity. During the RL process, the RL policy interacts with the environment and is optimized based on the shared experience replay buffer obtained from both the RL and population interactions. The optimized RL policy is then injected into the population for evolution. If the policy performs better than the individuals in the population, it will be selected as an elite, or it will be discarded. Many methods fall into this mode, such as ERL (Khadka & Tumer, 2018), CERL (Khadka et al., 2019), PDERL (Bodnar et al., 2020), PGPS (Kim et al., 2021), ERL-Re^2 (Hao et al., 2023a). **ii) In *EA-Master* mode**, EA is the primary approach and RL plays a secondary role. In this mode, RL no longer maintains its policy but only maintains an RL Critic. EA serves as the main loop, and RL can be seen as a component within the evolutionary loop. CEM-RL (Pourchot & Sigaud, 2019) and its subsequent works (Tang, 2021; Pretorius & Pillay, 2021; Liu & Feng, 2021) fall under this mode. In the learning process, half of the individuals in the population are used to optimize the Critic, which is then used to provide policy gradients guidance for these individuals. The optimized individuals are re-introduced into the population. Then the population interacts with the environment and evolves based on the obtained fitness. **iii) In *RL-Master* mode**, RL is the primary approach for policy search while EA plays a secondary role. The population is always led by the RL policy. RL serves as the main loop, and EA serves as an auxiliary role. The typical algorithm under this mode is Supe-RL (Marchesini et al., 2021) and its subsequent work (Marchesini & Amato, 2023), where the population is generated by adding Gaussian noise to the parameters of the RL policy.

Individual Architecture Selection. All the ERL algorithms except ERL-Re^2 (Hao et al., 2023a) maintain an independent policy network π_i parameterized as θ_i for each individual in the population. We refer to this architecture as the *Private Architecture*. Each policy can be seen as composed of independent state representation S_{ϕ_i} and policy representation W_i, i.e., $\pi_{\theta_i} = S_{\phi_i}(s)^\top W_i$. Intuitively, the *Private Architecture* leads to limited knowledge sharing among individuals. In contrast, the *Shared Architecture* (Hao et al., 2023a) is characterized by shared state representations among individuals. In this architecture, each individual can be expressed as $\pi_{\theta_i} = S_\phi(s)^\intercal W_i$, where $S_\phi(s)$ represents the shared representation network. The shared state representation is improved by maximizing the Q-values of all individuals. To reduce costs, Policy-extend Value Function (PeVFA (Tang et al., 2022)) is maintained instead of managing multiple value functions. PeVFA takes the policy representation as input and generalizes the Q-values across multiple policies. The shared representation eliminates the need for redundant learning of state representations for each policy and reduces the policy search space. Shared Architecture has also proven to efficiently combine EAs with Multi-Agent RL to enhance multi-agent collaboration (Li et al., 2023). Moreover, it can further improve the sample efficiency of Quality Diversity algorithms (Xue et al., 2023).

Evolutionary Algorithms and Operator Selection. Different ERL frameworks employ different EAs, where CEM and GA are the two most commonly used ones (Khadka & Tumer, 2018; Pourchot & Sigaud, 2019). Our focus is primarily on the commonly used EAs and their variants involved in ERL-related works, which include: 1) CEM (Pourchot & Sigaud, 2019). 2) Vanilla GA (Khadka & Tumer, 2018) using k-point crossover and Gaussian mutation. These two algorithms are introduced in Section 2. 3) PD-GA (Bodnar et al., 2020) using Q-value distillation operator and safe mutation operator. Q-value distillation clones good behaviors from the parents to the offspring based on their Q-values and safe mutation considers the sensitivity of the action outputs to the parameters to ensure safe mutations. 4) Behavior-level GA (Hao et al., 2023a) using behavior-level crossover and mutation operators. These operators perform crossover and mutation on specific action output dimensions, without affecting other behaviors. Behavior-level GA is unique in that it only modifies the last layer of the network, thus more suitable for use in conjunction with Shared Architecture.

How EA impacts RL. The impact of EA on RL can be classified into two main categories: **i)** ***Indirect impact***, in which EA provides diverse experiences to RL. This is the most commonly used approach. *Buffer Filter* (Marchesini et al., 2021) is a variant of this mechanism that only retains the experience of a subset of elite individuals, thereby ensuring the quality of the provided experience. **ii)** ***Direct impact***, in which the EA population directly influences the parameters of RL policy, including 1) *Genetic Soft Update*, 2) *Elite Guide*. *Genetic Soft Update* (Marchesini et al., 2021) involves the direct soft update of parameters of elite individuals in the population to the RL policy (If a target network is present, it is necessary to perform soft updates of target networks as well): $\theta_{rl} = \tau' \theta_{\text{Elite}} + (1-\tau')\,\theta_{\text{RL}}$. *Elite Guide* (Kim et al., 2021) utilizes imitation learning to influence the RL policy learning by encouraging the output of behaviors to be similar to elites in the population by minimizing $E_D\left[\|\pi_{rl}(s) - \pi_{\text{Elite}}(s)\|_2^2\right]$.

Fitness Surrogate Selection and Usage. EAs often suffer from low sample efficiency, mainly due to the evaluation of individuals in the population. To reduce the interaction cost of individuals, there are two types of surrogate fitness evaluation methods. The first one uses the average Q-value computed over the state samples from the replay buffer as surrogate fitness (Wang et al., 2022), which we refer to as *Critic (Buffer)*. It can be formalized as: $f_i = \frac{1}{k}\sum_{j=1}^{k} Q\left(s_j, \pi_i\left(s_j\right)\right)$, where k is the samples size. The second one is based on the H-step return with the Policy-extend Value Function Approximator (PeVFA) $\mathbb{Q}\left(s, a, W\right)$. (Hao et al., 2023a; Tang et al., 2022). PeVFA maintains the Q-values of multiple policies by taking additional policy representations as inputs, alleviating the problem of policy bias (Hao et al., 2023a) compared to Critic. We refer to this surrogate fitness as *PeVFA (H-step Bootstrap)*, which can be formalized as: $f_i = \sum_{t=0}^{H-1} \gamma^t r_t + \gamma^H \mathbb{Q}_\theta\left(s_H, \pi_i\left(s_H\right), W_i\right)$. Furthermore, we have also taken into account two additional variations: the combination of replay buffer with PeVFA, and the combination of H-step Bootstrap with Critic, which are named *PeVFA (Buffer)* and *Critic (H-step Bootstrap)* respectively.

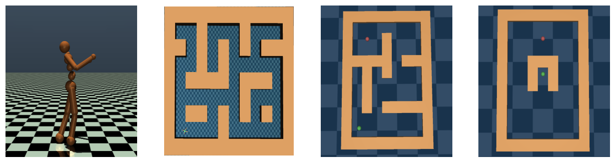


Figure 2. Task with different characteristics.

With a surrogate fitness function, there are two ways to increase the efficiency of evolution. The first approach is *Individual Control* (Wang et al., 2022; Kim et al., 2021), where a population twice the size of the actual population is maintained. The surrogate fitness is used to filter the larger population and the top half of individuals are selected to initialize the actual population. The second approach is *Generation Control* (Wang et al., 2022; Hao et al., 2023a), where the surrogate fitness function is used with a certain probability to evaluate the population in each generation. From this perspective, we investigate which fitness surrogate is more accurate and which usage scheme is more efficient.

4. Integrating the Best Mechanisms: A Comparative Analysis

In this section, we systematically revisit the five perspectives through experimental analysis. We compare mechanisms within each perspective under fair conditions and provide in-depth analysis. As the analysis progresses, we gradually integrate the most effective mechanisms for various tasks, constructing EvoRainbow and EvoRainbow-Exp.

4.1. Environments Setting

In contrast to previous studies that focus on validating in a single task type, i.e., MUJOCO (Pourchot & Sigaud,

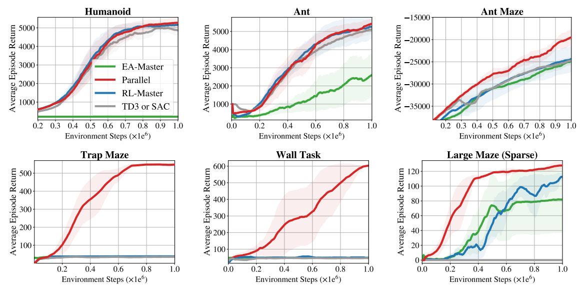
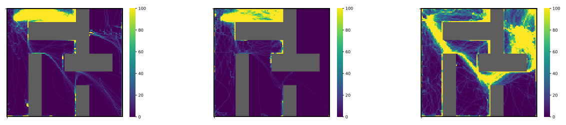


Figure 3. Performance comparison of different interaction modes



(a) EA-Master Mode (b) RL-Master Mode (c) Parallel Mode

Figure 4. Heatmap showing trajectories of different algorithms in the Trap Maze (From the top-left corner of the maze to the top-right corner). Only the parallel mode succeeds, whereas others get trapped in a local optimum.

2019), we specifically focus on three types of tasks with distinct characteristics: i) **High-dimensional locomotion tasks with dense reward:** Following the ERL literature, we select the two most high-dimensional locomotion tasks, Humanoid and Ant, for our analysis. In these tasks, agents need to control different joints to maximize forward speed while minimizing energy consumption. ii) **Trap tasks with deceptive rewards:** These tasks are commonly used in the EA literature (Chalumeau et al., 2022), where the policy controls the agent to reach a target point (refer to Figure 2(b) 2(c), 2(d)). The reward in these tasks is calculated as the exponentiation of the negative distance between the agent's current position and the target point. It is important to note that these tasks exhibit a gradient deception phenomenon, leading to unreliable policy gradients and suboptimal or unstable results, such as getting stuck on a wall and failing to reach the target point. iii) **Maze tasks with sparse reward:** Apart from the aforementioned deceptive reward maze environments, we also incorporate maze environments with sparse rewards, which have been extensively investigated in both EA and RL studies (Pugh et al., 2016; Wiering & Van Hasselt, 2008). In these tasks, rewards are only obtained upon successfully reaching the target location, highlighting the significance of exploration. More detailed information is shown in Appendix A. By evaluating the mechanisms in diverse tasks, we aim to provide comprehensive insights into their performance and effectiveness.

4.2. Interaction Mode Selection

We begin by conducting comparative experiments on the three interaction modes mentioned in Section 3, referred to as **Interaction Mode Selection**. The GA-RL interaction modes are categorized as follows: **Parallel Mode**, where GA provides experience for RL and RL injects optimized policies into GA; **EA-Master Mode**, where RL Critic is trained based on half of the individuals in the EA population, and policy gradients are injected into these individuals through Critic before reintroducing them into the population; **RL-Master Mode**, where individuals trained by RL always occupy the elite position in the population, and other individuals conduct crossover and mutation around the RL policy. To ensure fairness, we implement these modes based on the corresponding official codes. In the EA aspect, we utilize Vanilla GA. In the RL aspect, we employ SAC for maze and trap tasks and TD3 for locomotion tasks. All other hyperparameters are finetuned and the best results are provided. More implementation details and experiment settings can be found in Appendix E.

As depicted in Figure 3, for locomotion tasks, both Parallel and RL-Master modes exhibit comparable performance, surpassing the EA-Master Mode. This can be attributed to the relatively sufficient reward signals in these tasks, making RL-based methods more efficient, whereas EA-oriented approaches weaken the influence of RL, leading to less remarkable performance in such tasks. In maze and trap tasks, we observe that the Parallel Mode outperforms the EA-Master and RL-Master modes. These tasks emphasize exploration ability, which is relatively limited in RL (SAC fails in all maze tasks). Additionally, the RL-Master Mode does not maintain the elite individual for EA, compromising the exploration ability of EA. Moreover, the EA-dominant approach utilizes the Critic to maintain the Q values of multiple individuals, resulting in half of the population optimizing towards an average direction, thus weakening the exploration ability. Furthermore, we provide heatmaps based on the trajectories collected during the learning process of different algorithms on the Trap Maze in Figure 4. These heatmaps provide a visualization of the visitation frequency, with higher values indicating more frequent visits. From the heatmaps, we observe that both RL-Master and EA-Master, influenced by deceptive rewards, have become trapped in local suboptimal areas. In contrast, Parallel Mode shows a strong exploration ability and successfully guides the agent to reach the target point.

In conclusion, through experiments conducted on tasks of different natures, we find that the Parallel Mode is more effective and outperforms other modes.

4.3. Individual Architecture Selection

In this section, we delve deeper into the selection of individual architectures based on the findings from the previous section. We consider two types of architectures: **Private Architecture**, which maintains individual policies, and **Shared Architecture**, which utilizes a shared state repre-

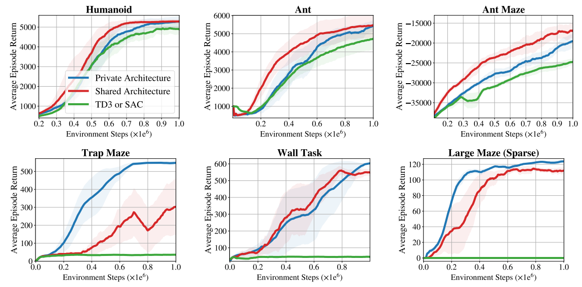


Figure 5. Performance comparison of Private Architecture and Shared Architecture.

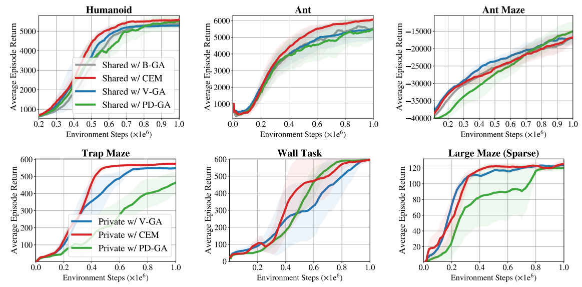


Figure 6. Performance comparison of different EAs.

sentation for all policies. These architectures are implemented based on the Parallel Mode discussed in the previous subsection.The results depicted in Figure 5 on six tasks reveal that Shared Architecture outperforms Private Architecture in locomotion tasks, whereas Private Architecture surpasses Shared Architecture in Trap Maze, Wall Task, and Large Maze (Sparse Reward). The primary reason for this phenomenon lies in the optimization process of the shared architecture, which relies on maximizing the Q-values of all individuals. However, in exploration tasks with sparse rewards and deceptive rewards, the low-quality reward signals reduce the accuracy of the value function estimates. Consequently, the expressive capacity of the shared state representation becomes limited, impeding the construction of favorable policy space, whereas Private Architecture is less affected by the reward signal quality, enabling more efficient exploration.

Based on these experimental results, we conclude that Shared Architecture tends to be more efficient on tasks with higher-quality reward signals, whereas Private Architecture exhibits better performance in exploration tasks with lower-quality reward signals. Therefore, in the subsequent analysis experiments, we utilize Shared Architecture for locomotion-related tasks and Private Architecture for exploration tasks.

4.4. Evolutionary Algorithms and Operator Selection.

In this section, we focus on the selection of Evolutionary Algorithms and operators, with a particular emphasis on

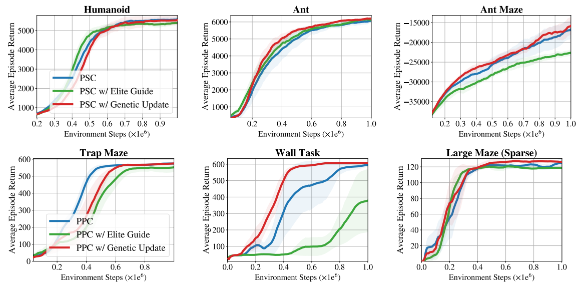


Figure 7. Performance comparison of the mechanism of impact of EA on RL.

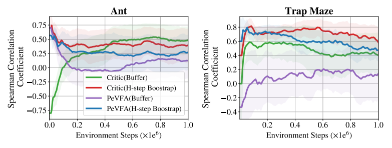
two commonly used EAs in ERL: **CEM** and **GA**. Previous works propose various GA variants using different operators, such as k-point crossover and Gaussian-weighted mutation (referred to as **V-GA**), Q-value distillation crossover and safety mutation (referred to as **PD-GA**), and crossover and mutation based on the behavioral semantic level (referred to as **B-GA**). We replace the V-GA used in Shared Architecture and Private Architecture from the previous section with these EAs to conduct comprehensive comparisons.

The experimental results, presented in Figure 6, demonstrate the superior performance of CEM compared to other operators in five out of the six tasks. Although PD-GA performs well in most tasks, it exhibits limitations in highly exploratory tasks such as Trap Maze. This limitation can be attributed to the constraints imposed by safety constraints, which restrict its exploration capability. CEM's advantage lies in its utilization of a Gaussian distribution to model the solution space and its adaptive adjustment of the covariance matrix, which efficiently guides the search process. This adaptability allows CEM to better align with the characteristics of the problem (Pourchot & Sigaud, 2019; Sigaud, 2022), resulting in improved search efficiency and enhanced exploration capability. Overall, CEM is more advantageous, so we integrate it into our algorithm.

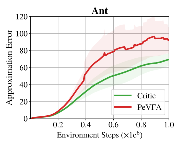
4.5. How EA impacts RL

In this section, we explore the impact of EA on RL. In the previous sections, we adopt the classic architecture in which EA provides experiences to RL, and RL updates its policy back into the population. Now, we introduce three additional mechanisms to our algorithm one by one.

We first examine the direct impact mechanisms: **Genetic Soft Update** and **Elite Policy Guide**. These mechanisms are integrated into two combinations: Parallel Mode, Shared Architecture, and CEM (referred to as PSC), and Parallel Mode, Private Architecture, and CEM (referred to as PPC). The experimental results depicted in Figure 7 demonstrate that Genetic Soft Update generally performs better, whereas Elite Policy Guide exhibits poor performance in Maze and Trap tasks. Additionally, Elite Policy Guide incurs more



(a) Ranking Correlation



(b) Error

Figure 8. Analysis on fitness surrogates.

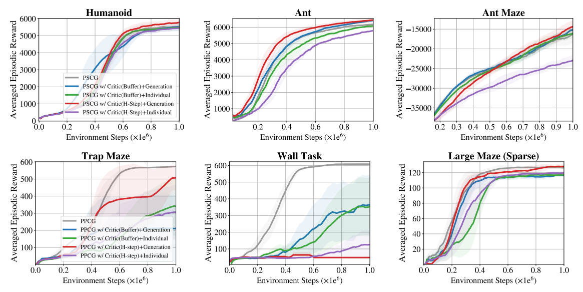


Figure 9. Performance comparison of fitness and surrogate usages.

time overhead than Genetic Soft Update as it requires supervised learning to ensure similarity between RL behavior and elite behavior. It is important to note that Genetic Soft Update does not guarantee consistent performance improvement across all tasks. For instance, there may be a slight decline in performance observed in tasks such as Trap Maze. The reason for Genetic Soft Update being more efficient may lie in its substantial modifications to the RL policy parameters. This change enhances the plasticity of the RL policy (Nikishin et al., 2022), enabling it to more fully utilize existing experiences and escape suboptimal solutions. The experiments on **Experience Filter** are presented in the Appendix D, where we discover that Experience Filter does not further enhance the algorithm's performance.

By analyzing different mechanisms, we observe that Genetic Soft Update is more effective. Therefore, we integrate it into PSC and PPC, resulting in PSCG and PPCG.

4.6. Fitness Surrogate Selection and Usage.

In this section, we investigate Fitness Surrogate Selection and Usage. We begin by examining the four fitness surrogate options: **Critic(Buffer)**, **Critic(H-step Bootstrap)**, **PeVFA(Buffer)**, and **PeVFA(H-step Bootstrap)**. These mechanisms vary in terms of whether they utilize Critic or PeVFA for fitness estimation and whether they directly use experiences from the buffer or interact with the environment for a certain number of steps before value estimation.

The population evaluation solely focuses on individual rankings, without considering specific scores. Thus we first analyze the Spearman's rank correlation coefficient (Sedgwick, 2014). The correlation coefficient is particularly suitable for quantifying the similarity of rankings between two variables and is not influenced by the specific values of the variables. It is defined within the range of -1 to 1, and its formula is given as $r_s = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}$, where n represents the population size and d represents the differences in ranks between the two variables. During the learning process, we calculate the Spearman correlation coefficient of the four different surrogates and the true values. These true values are obtained by evaluating individual performance through 10 episodic runs in the environment. Figure 8(a) illustrates that the H-step Bootstrap methods generally exhibit higher correlation and the correlation of Critic(Buffer) gradually increases as learning progresses, especially in the Ant task. Despite the overall superiority of the H-step Bootstrap approaches over the Buffer approaches in terms of correlation in Ant and Trap tasks, it is still difficult to determine which solution is more suitable due to the additional sample overhead associated with H-step Bootstrap approaches. However, it is worth noting that the PeVFA approach consistently performs worse than the Critic approach overall, primarily due to the increased difficulty of maintaining Q-values for multiple policies. The results in Figure 8(b) indicate that PeVFA often has a larger approximation error compared to Critic, making it more challenging to provide accurate Q-values. Based on these findings, we can conclude that the Critic approach is more capable of providing more accurate rankings compared to the PeVFA approach. Thus we choose Critic as the value estimator. However, we cannot determine whether the H-step Bootstrap or Buffer approach is more advantageous for policy learning.

Next, we directly compare the performance of using Critic (H-step Bootstrap) and Critic (Buffer), which involves the two ways of using surrogates: **Individual Control** and **Generation Control**. The experimental results in Figure 9 show that Critic(H-Step) Generation Control outperforms other combinations and leads to improvements in locomotion tasks. However, in Trap Maze and Wall Task, the algorithm performance collapses. This phenomenon can be explained by the RL individual elitism rate. For example, in the Ant task, RL individuals have about a 50% probability of being selected as elites and about a 30% probability of being discarded. Even if there is a misleading fitness surrogate and the population is promoted with poor individuals being selected as elites, periodically injecting RL individuals into the population prevents performance loss or fluctuation. In contrast, in exploration tasks with low-quality reward signals, RL struggles to learn effective policies. In these cases, EAs dominate and have an approximately 80% elitism rate, making it difficult for RL individuals to be selected as elites. When the population is promoted based on inaccurate signals, the elite individuals may be removed while the RL individuals fail to reach the elite performance, resulting in a significant impact on the final performance.

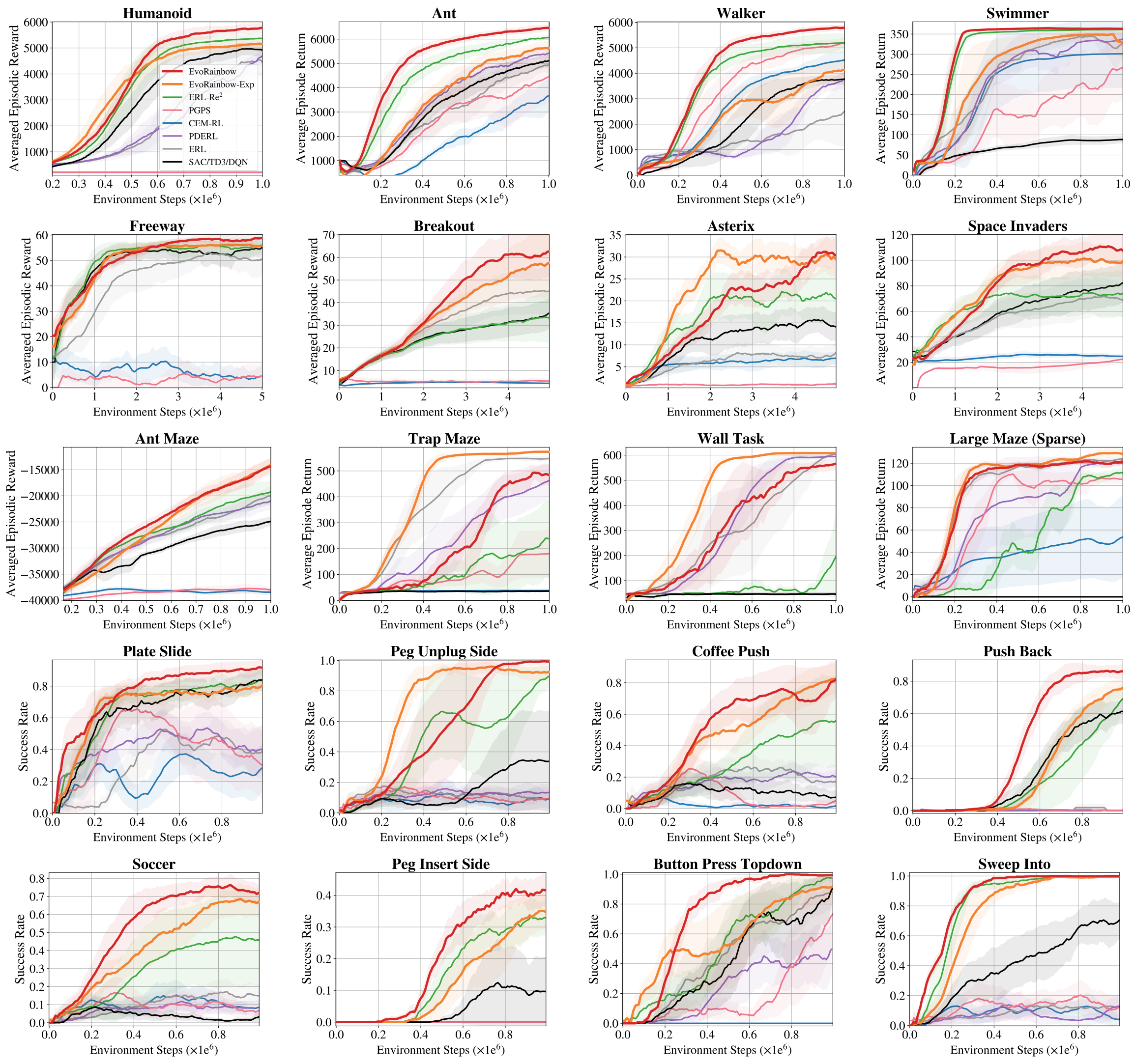


Figure 10. Performance comparison on different types of tasks.

Based on the above findings, we integrate Critic(H-Step) + Generation with PSCG to form EvoRainbow, which excels in general tasks. However, the fitness surrogate can not yield benefits in exploration tasks. Therefore, we retain the combination with the best exploration capability, PPCG, resulting in EvoRainbow-Exp.

5. Further Experimental Evaluation

Through the comparison and analysis of 18 different mechanisms, we integrate the most effective ones to form EvoRainbow and EvoRainbow-Exp, The former incorporates mechanisms Parallel mode, Shared Architecture, CEM, Genetic Soft Update, and Surrogate with Critic (H-Step Bootstrap) and is primarily utilized for general tasks, whereas the latter integrates mechanisms Parallel mode, Private Architecture, CEM, and Genetic Soft Update and is mainly suitable for exploration tasks. To provide a comprehensive experimental comparison, we further validate the efficiency of EvoRainbow and EvoRainbow-Exp and compare them with other methods, including ERL, PDERL, CERL, CEM-RL, PGPS, ERL-Re^2, SAC (Haarnoja et al., 2018), TD3 (Fujimoto et al., 2018), DQN (Mnih et al., 2013). We conduct comparisons on 20 tasks, including locomotion tasks, MinAtar tasks (Young & Tian, 2019), maze tasks (Fu et al., 2020), and manipulation tasks (Yu et al., 2019b). All compared algorithms are based on their official implementation. For the RL algorithms, we adopt TD3 for comparison in locomotion tasks, DQN in MinAtar tasks, and SAC in maze and manipulation tasks. All results are obtained from 5 independent runs. This is consistent with the setting in ERL and PDERL. We report the average with 95% confidence regions. For more implementation details, please refer to Appendix E. The results in Figure 10 reveal that EvoRainbow exhibits outstanding performance in most tasks, whereas EvoRainbow-Exp outperforms other algorithms significantly in exploration tasks. These experiments convincingly demonstrate the superiority of both algorithms.

6. Further Analysis and Validation

In Section 4, we systematically analyze different mechanisms step-by-step based on their importance, and select the best mechanisms to construct new combinations for subsequent analysis. Due to not considering all possible combinations, concerns about reliability may arise: conclusions under some combinations may differ from those under others. To solve the problem, we conduct additional experiments to revalidate the mechanisms involved in each perspective across various combinations. Due to space constraints, the experiments are shown in Appendix B. The experimental results indicate that across various combinations, the conclusions drawn from different mechanism selections align with those presented in the main text. This demonstrates the reliability of the conclusions presented in the main text.

Additionally, in Appendix C, we conduct ablation studies, component replacements, and analyses on EvoRainbow across tasks in five different domains: Ant, Breakout, Trap Maze, Push Back, and Large Maze (Sparse Reward). Readers can refer to the appendix for more details.

7. Conclusion

In this paper, we provide a comprehensive review of the mechanisms proposed in previous ERL literature from five perspectives. We conduct systematic analysis and experimental comparisons on tasks with dense, deceptive, and sparse rewards to evaluate the effectiveness of these mechanisms. Based on our experimental findings, we carefully select the most effective components from different perspectives to construct EvoRainbow and EvoRainbow-Exp. The superiority of these two algorithms is demonstrated through experiments conducted on 20 tasks.

Here, we offer insights from our experiments that may help researchers make more informed and efficient selection.

- Regarding the interaction mode, we believe that choosing the parallel mode directly for future research should become the default choice for ERL architectures.
- If the quality of the reward signal is good, the shared architecture representation should be preferred; otherwise, opt for the private architecture.
- We find that the shared architecture cannot solve the problem of poor reward signal, and we have identified the reason for this problem as the reliance on value function approximation for shared representation learning. Researchers can develop more efficient shared representations that not only facilitate efficient knowledge sharing but also construct robust policy spaces.
- More efficient EAs or operators are also necessary. Current work mostly revolves around algorithms such as CEM and GA, overlooking more advanced EAs.
- According to the analysis in Figure 8, we can also propose more efficient surrogate mechanisms to improve overall sample efficiency.

The above are some conclusions and insights we have identified in current ERL research. Researchers can use these findings to construct more efficient alternative mechanisms or to conduct further research.

However, it is important to acknowledge the limitations of our work. Firstly, we decouple and compare various mechanisms step by step based on their importance, without encompassing all possible combinations. This approach to some extent overlooks the interactions between mechanisms. Secondly, we have not identified a single framework capable of addressing all types of tasks, necessitating the selection between two proposed methods based on task characteristics. These limitations will be investigated as follow-up work.

Impact Statement

This paper presents work whose goal is to advance the field of Machine learning. There are many potential social consequences of our work, none of which we feel must be specifically highlighted here.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant Nos. 92370132, 62106172), the National Key R&D Program of China (Grant No. 2022ZD0116402) and the Xiaomi Young Talents Program of Xiaomi Foundation.

References

Bäck, T. and Schwefel, H. An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, 1993.

Bodnar, C., Day, B., and Lió, P. Proximal distilled evolutionary reinforcement learning. In *AAAI*, 2020.

Boer, P. D., Kroese, D., Mannor, S., and Rubinstein, R. A tutorial on the cross-entropy method. *Ann Oper Res*, 2005.

Chalumeau, F., Boige, R., Lim, B., Macé, V., Allard, M., Flajolet, A., Cully, A., and Pierrot, T. Neuroevolution is a competitive alternative to reinforcement learning for skill discovery. *CoRR*, 2022.

Cobbe, K., Hilton, J., Klimov, O., and Schulman, J. Phasic policy gradient. In *ICML*, 2021.

Duan, Y., X.Chen, Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. Noisy networks for exploration. In *ICLR*, 2018.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint*, 2020.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *ICML*, 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Hao, J., Li, P., Tang, H., Zheng, Y., Fu, X., and Meng, Z. Erl-re^2: Efficient evolutionary reinforcement learning with shared state representation and individual policy representation. In *ICLR*, 2023a.

Hao, J., Yang, T., Tang, H., Bai, C., Liu, J., Meng, Z., Liu, P., and Wang, Z. Exploration in deep reinforcement learning: From single-agent to multiagent domain. *TNNLS*, 2023b.

Johannink, T., Bahl, S., Nair, A., Luo, J., Kumar, A., Loskyll, M., Ojea, J. A., Solowjow, E., and Levine, S. Residual reinforcement learning for robot control. In *ICRA*, 2019.

Khadka, S. and Tumer, K. Evolution-guided policy gradient in reinforcement learning. In *NeurIPS*, 2018.

Khadka, S., Majumdar, S., Nassar, T., Dwiel, Z., Tumer, E., Miret, S., Liu, Y., and Tumer, K. Collaborative evolutionary reinforcement learning. In *ICML*, 2019.

Kim, N., Baek, H., and Shin, H. Pgps: Coupling policy gradient with population-based search. 2021.

Larrañaga, P. and Lozano, J. A. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Science & Business Media, 2001.

Li, P., Tang, H., Yang, T., Hao, X., Sang, T., Zheng, Y., Hao, J., Taylor, M. E., Tao, W., Z. Wang, Z., et al. Pmic: Improving multi-agent reinforcement learning with progressive mutual information collaboration. *ICML*, 2022.

Li, P., Hao, J., Tang, H., Zheng, Y., and Fu, X. Race: Improve multi-agent reinforcement learning with representation asymmetry and collaborative evolution. In *ICML*, 2023.

Li, P., Hao, J., Tang, H., Fu, X., Zheng, Y., and Tang, K. Bridging evolutionary algorithms and reinforcement learning: A comprehensive survey. *arXiv preprint*, 2024.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

Liu, J. and Feng, L. Diversity evolutionary policy deep reinforcement learning. *Comput. Intell. Neurosci.*, 2021.

Liu, J., Wang, Z., Zheng, Y., Hao, J., Bai, C., Ye, Y., Wang, Z., Piao, H., and Sun, Y. Ovd-explorer: Optimism should not be the sole pursuit of exploration in noisy environments. In *AAAI*, 2024.

Marchesini, E. and Amato, C. Improving deep policy gradients with value function search. In *ICLR*, 2023.

Marchesini, E., Corsi, D., and Farinelli, A. Genetic soft updates for policy evolution in deep reinforcement learning. In *ICLR*, 2021.

Mitchell, M. *An introduction to genetic algorithms*. MIT Press, 1998.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *arXiv preprint*, 2013.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P., and Courville, A. C. The primacy bias in deep reinforcement learning. In *ICML*, 2022.

Pourchot, A. and Sigaud, O. CEM-RL: combining evolutionary and gradient-based methods for policy search. In *ICLR*, 2019.

Pretorius, K. W. and Pillay, N. Population based reinforcement learning. In *SSCI*, 2021.

Pugh, J. K., Soros, L. B., and Stanley, K. O. Quality diversity: A new frontier for evolutionary computation. *Front. Robot. AI*, 2016.

Puterman, M. L. Markov decision processes. *Handbooks in operations research and management science*, 1990.

Raileanu, R. and Fergus, R. Decoupling value and policy for generalization in reinforcement learning. In *ICML*, 2021.

Salimans, T., Ho, J., Chen, X., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint*, 2017.

Sedgwick, P. Spearman's rank correlation coefficient. *Bmj*, 2014.

Sigaud, O. Combining evolution and deep reinforcement learning for policy search: a survey. *arXiv preprint*, 2022.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. 1998.

Tang, H., Meng, Z., Hao, J., Chen, C., Graves, D., Li, D., Yu, C., Mao, H., Liu, W., Yang, Y., Tao, W., and Wang, L. What about inputting policy in value function: Policy representation and policy-extended value function approximator. In *AAAI*, 2022.

Tang, Y. Guiding evolutionary strategies with off-policy actor-critic. In *AAMAS*, 2021.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 2019.

Wang, Y., Zhang, T., Chang, Y., Liang, B., Wang, X., and Yuan, B. A surrogate-assisted controller for expensive evolutionary reinforcement learning. *Inf. Sci.*, 2022.

Wiering, M. A. and Van Hasselt, H. Ensemble algorithms in reinforcement learning. *IEEE SMC B*, 2008.

Xue, K., Wang, R., Li, P., Li, D., Jianye, H., and Qian, C. Sample-efficient quality-diversity by cooperative coevolution. In *ICLR*, 2023.

Young, K. and Tian, T. Minatar: An atari-inspired testbed for more efficient reinforcement learning experiments. *CoRR*, 2019.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *CoRL*, 2019a.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *CoRL*, 2019b.

Zhu, Q., Wu, X., Lin, Q., Ma, L., Li, J., Ming, Z., and Chen, J. A survey on evolutionary reinforcement learning algorithms. *Neurocomputing*, 2023.

Zou, L., Xia, L., Ding, Z., Song, J., Liu, W., and Yin, D. Reinforcement learning to optimize long-term user engagement in recommender systems. In *KDD*, 2019.

A. Details of Environments

We evaluate the methods on a total of 20 tasks, divided into the following categories: Maze Tasks, Locomotion Tasks, and MinAtar, Manipulation Tasks. The details of each environment are as follows:

Maze Tasks: We design four maze tasks in total according to previous EA and RL literature, including Ant Maze, Trap Maze, Wall Task, and Large Maze (Sparse Rewards). All environments are built based on the D4RL (Fu et al., 2020) code repository[1]. We provide a detailed description of each task below.

- Ant Maze: In this task, the algorithm aims to learn a policy for guiding the ant agent to reach the target point in the upper right corner of the maze. The reward signal is deceptive, represented by the negative Euclidean distance between the Ant and the target point. Due to the need for simultaneous movement and navigation control, this reward signal can often cause the Ant to collide with walls and become stuck. The task is considered successfully completed when the score reaches -20000.
- Trap Maze: The goal of this task is to navigate the point agent through a maze to reach the target point. The reward signal is deceptive, calculated as the exponentiation of the negative distance between the agent's current position and the target point. However, learning based on the reward signal gradient can lead to the agent getting trapped at multiple locations, and unable to reach the target point.
- Wall Task: In this task, the algorithm must guide the point agent to bypass a wall obstacle and reach the target position. The reward signal is the same as the Trap Maze and is deceptive. The agent often faces challenges of getting stuck inside the wall and failing to reach the target point.
- Large Maze: The objective of this task is to explore a large maze and guide the point agent to the target point. The reward signal is sparse, where the agent receives a reward of 1 upon reaching the target point. Due to the sparsity of the reward signal, effective exploration strategies are crucial in this task.

In all four types of tasks, the algorithm needs to provide sufficient exploration capability to avoid suboptimal solutions and achieve the task's goal successfully.

Lomotation Task: We include four tasks from the MUJOCO environment: Humanoid, Ant, Walker, and Swimmer. These tasks involve controlling different types of physical agents and serve as commonly used benchmarks in the field of reinforcement learning.

MinAtar: For the MinAtar benchmark, we conduct experiments on the official tasks[2] for Breakout, Asterix, Freeway, and Space Invaders with the default settings.

Manipulation Tasks: We chose Metaworld (Yu et al., 2019a) for algorithm validation on Manipulation Tasks. We conduct experiments with the default settings in the official implementation[3]. The eight tasks involved in the experiments are Plate Slide, Peg Unplug Side, Coffee Push, Push Back, Soccer, Peg Insert Side, Button Press Topdown, and Sweep Into.

B. Mechanism Comparison with Various Combination

In the previous section, we systematically compare different mechanisms based on their importance. At each step, we construct a new combination based on the mechanism chosen in the previous step for further analysis. However, this may raise concerns about the consistency of conclusions across different combinations. To address this problem, we validate the robustness of our findings by employing various combinations.

Specifically, we conduct a comprehensive re-evaluation of various mechanisms, based on different combinations, and evaluate them on four tasks with different characteristics: Ant, Trap Maze, Large Maze (Sparse Reward), and Push-Back. Among them, Ant and Push-Back are conventional tasks with relatively superior reward signals, while Trap Maze and Large Maze (Sparse Reward) are exploration tasks with deceptive and sparse reward signals. Each experiment is conducted based on five independent runs. We evaluate these combinations with two metrics: final performance and learning efficiency

[1]https://github.com/Farama-Foundation/D4RL
[2]https://github.com/kenjyoung/MinAtar
[3]https://github.com/Farama-Foundation/Metaworld

All Tasks	Private + CEM + Genetic + Surrogate	Share + CEM	Share + CEM + Surrogate	Share + CEM + Genetic + Surrogate
Parallel Mode	**0.91** | **0.78**	**0.89** | **0.72**	**0.76** | 0.61	**0.86** | **0.82**
RL-Master Mode	0.76 | 0.64	0.82 | 0.67	0.72 | **0.67**	0.72 | 0.55
EA-Master Mode	0.20 | 0.11	0.39 | 0.29	0.38 | 0.26	0.38 | 0.26

Table 1. Performance and efficiency comparison across different combinations for Interaction Mode Selection on all four tasks.

Conventional Tasks	EA Master + CEM + Genetic + Surrogate	Parallel + CEM + Genetic + Surrogate	RL-Master + CEM + Genetic + Surrogate	RL-Master + PD-GA + Genetic + Surrogate
Shared Architecture	**0.52** | **0.26**	**0.97** | **0.92**	0.83 | **0.73**	**0.92** | 0.80
Private Architecture	0.30 | 0.16	0.85 | 0.67	**0.84** | 0.69	0.72 | **0.84**

Table 2. Performance and efficiency comparison across different combinations for Individual Architectures Selection on conventional tasks.

(area under the curve), wherein the scores have been normalized by dividing them by the maximum value. Finally, we present the average normalized scores across various tasks. Each cell in the result table represents the average scores for the corresponding combination, including the average final performance score on the left and the average sample efficiency score on the right.

B.1. Interaction Mode Selection across Various Combinations

In the main text, we explore the comparison with the combination of Private Architecture + V-GA under three modes. Here, we provide additional comparisons for four other combinations, including Private Architecture + CEM + Genetic Soft Update + H-Step Bootstrap w/ Generation (Surrogate), Share Architecture + CEM, Share Architecture + CEM + H-Step Bootstrap w/ Generation (Surrogate), Share Architecture + CEM + Genetic Soft Update + H-Step Bootstrap w/ Generation (Surrogate).

We combine the three modes with the aforementioned four combinations and compare them across four tasks. The experimental results in Table 1 indicate that the scores achieved by combining Parallel Mode with the four combinations consistently outperform those of RL-Master Mode and EA-Master Mode. This aligns with the conclusion drawn in the main text regarding the higher efficiency of Parallel Mode.

B.2. Individual Architectures Selection across Various Combinations

In the main text, we explore Individual Architectures Selection based on the combination of Parallel Mode + V-GA. Here, we present the results for four additional combinations, including EA Master Mode + CEM + Genetic Soft Update + H-Step Bootstrap w/ Generation (Surrogate), Parallel Mode + CEM + Genetic Soft Update + H-Step Bootstrap w/ Generation (Surrogate), RL-Master Mode + CEM + Genetic Soft Update + H-Step Bootstrap w/ Generation (Surrogate), RL-Master Mode + PD-GA + Genetic Soft Update + H-Step Bootstrap w/ Generation (Surrogate).

We combine the two architectures with the aforementioned four combinations and compare them. We categorized the test tasks into two groups based on the quality of the reward signals. The first group includes the conventional tasks Ant and Push-Back, while the second group includes the exploration tasks Trap Maze and Large Maze (Sparse). The experimental results are shown in Table 2 and 3. We observe that in conventional tasks, the shared architecture generally outperforms the private architecture, both in terms of final performance and sample efficiency. However, in exploratory tasks, the private architecture is typically more efficient. This is consistent with the conclusion drawn in our main text.

B.3. EAs and Operators Selection

In the main text, we primarily explore the results based on two combinations: Parallel Mode + Shared Architecture and Parallel Mode + Private Architecture. Here, we provide three additional combinations for further evaluation. The combinations include EA-Master Mode + Share Architecture + Genetic Soft Update + H-Step Bootstrap w/ Generation

Exploration Tasks	EA Master + CEM + Genetic + Surrogate	Parallel + CEM + Genetic + Surrogate	RL-Master + CEM + Genetic + Surrogate	RL-Master + PD-GA + Genetic + Surrogate
Shared Architecture	**0.23** | **0.26**	0.76 | 0.77	0.47 | 0.41	0.27 | 0.26
Private Architecture	0.09 | 0.07	**0.81** | **0.79**	**0.68** | **0.59**	**0.56** | **0.54**

Table 3. Performance and efficiency comparison across different combinations for Individual Architectures Selection on exploration tasks.

All Tasks	EA Master + Share + Genetic + Surrogate	RL Master + Share + Genetic + Surrogate	RL Master + Private + Genetic + Surrogate
CEM	**0.40** | **0.30**	**0.78** | 0.62	**0.83** | **0.73**
PD-GA	0.33 | 0.20	0.60 | 0.55	0.64 | 0.54
B-GA	0.27 | 0.19	0.70 | **0.66**	-
V-GA	0.31 | 0.24	0.75 | 0.58	0.56 | 0.45

Table 4. Performance and efficiency comparison across different combinations for EAs and Operators Selection on all tasks.

(Surrogate), RL-Master Mode + Share Architecture + Genetic Soft Update + H-Step Bootstrap w/ Generation (Surrogate) and RL-Master Mode + Private Architecture + Genetic Soft Update + H-Step Bootstrap w/ Generation (Surrogate).

We combine the four EAs with the aforementioned three combinations and compare them on all tasks. The experimental results in Table 4 indicate that across the three combinations, CEM is more efficient compared to the other three EAs. This aligns with our conclusion in the main text that CEM is more efficient.

B.4. Analysis of Genetic Soft updates

In the main text, we find that Elite Guide is more time-consuming and can even result in negative performance gains. Therefore, here we primarily evaluate the robustness of Genetic Soft Update in enhancing performance. We present four additional combinations including: Parallel Mode + Share Architecture + CEM + H-Step Bootstrap w/ Generation (Surrogate), RL-Master Mode + Share Architecture + CEM + H-Step Bootstrap w/ Generation (Surrogate), RL-Master Mode + Private Architecture + CEM + H-Step Bootstrap w/ Generation (Surrogate), and RL-Master Mode + Share Architecture + PD-GA + H-Step Bootstrap w/ Generation (Surrogate).

We combine Genetic Soft Update with the aforementioned three combinations and compare them on all tasks. The experimental results in Table 5 demonstrate that incorporating Genetic Soft Update tends to yield improvements in both final performance and sample efficiency. This finding is consistent with the conclusion drawn in the main text.

B.5. Analysis of the Surrogate Mechanism

We also analyze the robustness of the H-Step Bootstrap w/ Generation. We provided four additional combinations including: Parallel Mode + Share Architecture + CEM + Genetic Soft Update, RL-Master Mode + Share Architecture + CEM + Genetic Soft Update, RL-Master Mode + Private Architecture + CEM + Genetic Soft Update, and RL-Master Mode + Share Architecture + PD-GA + Genetic Soft Update.

We combine the H-Step Bootstrap w/ Generation with the aforementioned four combinations and compare them. In the main text, we find that the use of Surrogate varies significantly across different types of tasks. Thus we categorize the test tasks into two groups based on the quality of the reward signals. The first group includes the conventional tasks Ant and

All Tasks	Parallel + Share + CEM + Surrogate	RL Master + Share + CEM + Surrogate	RL Master + Share + CEM + Surrogate	RL Master + Share + PD-GA + Surrogate
w/ Genetic	**0.86** | **0.82**	**0.72** | 0.55	**0.77** | **0.67**	**0.60** | 0.53
w/o Genetic	0.76 | 0.61	0.72 | **0.67**	0.67 | 0.49	0.59 | **0.55**

Table 5. Performance and efficiency comparison across different combinations for Genetic Soft updates on all tasks.

Conventional Tasks	Parallel + Share + CEM + Genetic	RL Master + Share + CEM + Genetic	RL Master + Private + CEM + Genetic	RL Master + Share + PD-GA + Genetic
w/ Surrogate	**0.97** |**0.92**	**0.97** |0.70	**0.84** | **0.69**	**0.92** | **0.80**
w/o Surrogate	0.92 | 0.68	0.89 | **0.89**	0.77 | 0.52	0.81 | 0.60

Table 6. Performance and efficiency comparison across different combinations for Surrogate Mechanism on conventional tasks.

Exploration Tasks	Parallel + Share + CEM + Genetic	Parallel + Share + CEM + Genetic	RL Master + Private + CEM + Genetic	RL Master + Share + PD-GA + Genetic
w/ Surrogate	0.74 | 0.71	0.47| 0.41	0.70 | 0.65	0.27 | 0.26
w/o Surrogate	**0.87** | **0.72**	**0.69** | **0.65**	**0.74** | **0.65**	**0.51** | **0.31**

Table 7. Performance and efficiency comparison across different combinations for Surrogate Mechanism on exploration tasks.

Push-Back, while the second group includes the exploration tasks Trap Maze and Large Maze (Sparse).

The experimental results in Table 6 and Table 7 indicate that using Surrogate in conventional tasks can significantly improve both final performance and sample efficiency. However, in exploratory tasks, the use of Surrogate leads to a decrease in performance. The conclusion here is consistent with the conclusion provided in our main text.

The experiments above mainly aim to further validate different mechanisms through various combinations, thereby mitigating potential biases resulting from the single combination or the order of combinations. Additionally, evaluation based on different mechanisms for each mechanism can significantly enhance the robustness and reliability of the experimental conclusions.

C. Ablation Study and Analysis Experiments

In this section, we conduct in-depth analyses of EvoRainbow. Throughout the main text, we gradually build the entire algorithm from scratch. In this section, we perform ablation and replacement experiments on each component of EvoRainbow and test it on tasks with five different characteristics. These tasks cover five major categories mentioned in the main text, including Ant, Trap Maze, Large Maze (Sparse Reward), Breakout, and Push Back.

C.1. Ablation Study on Fitness Surrogate Selection and Usage

In this subsection, we conduct an ablation analysis on Fitness Surrogate Selection and Usage. We remove the usage of Critic(H-Step) + Generation in EvoRainbow. The experimental results in Figure 11 indicate that using fitness surrogates leads to performance improvement in general tasks, such as Breakout, Ant, and Push Back. In these tasks, the quality of reward signals is typically high. However, for tasks with poor reward signal quality, e.g., sparse rewards or deceptive rewards, fitness surrogates may result in performance loss. This is mainly because the errors in fitness surrogates can lead to the removal of population elites, and in maze tasks where the importance of the population is greater than RL, it can lead to performance loss. The conclusion obtained here is consistent with the conclusion in the main text, i.e., utilizing fitness surrogates contributes to the policy search in general tasks, but has a negative impact on exploration tasks with low-quality reward signals.

C.2. Ablation Study on How EA impacts RL

In this subsection, we conduct an ablation analysis on How EA impacts RL. We remove the Genetic Soft Update in EvoRainbow. The experimental results shown in Figure 12 demonstrate that the use of Genetic Soft Update can improve performance across different types of tasks. This improvement can be attributed to the direct influence of Genetic Soft Update on the RL parameters, potentially breaking the RL policy out of suboptimal situations. It enhances the plasticity of the RL policy, facilitating more effective utilization of experiences collected by EA, ultimately leading to improved policy performance. The conclusions drawn from experiments under EvoRainbow in this context align with those presented in the main text, i.e., Genetic Soft Update can accelerate policy learning, leading to better results.

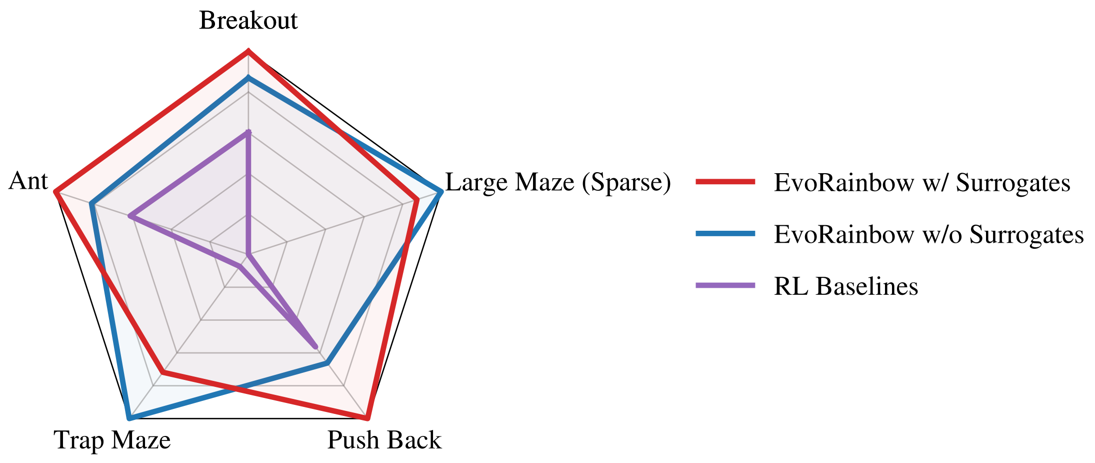


Figure 11. Ablation study on Fitness Surrogate Selection and Usage in EvoRainbow. we first average the score over five seeds up to 1 million environment steps. In most tasks, we compute the area under the score curve as it captures not only the final performance but also the amount of interaction required to achieve it. For the Maze tasks, we use the final performance since we find the area can not capture the final performance. Since absolute values vary greatly across games, we report relative quantities by dividing by the maximum value obtained in each game. **The experimental results under EvoRainbow once again confirm the conclusion in the main text, that is, Fitness Surrogate is more suitable for general tasks, whereas for tasks with high exploration requirements, using Fitness Surrogate may lead to performance loss due to the potential errors of value estimates.**

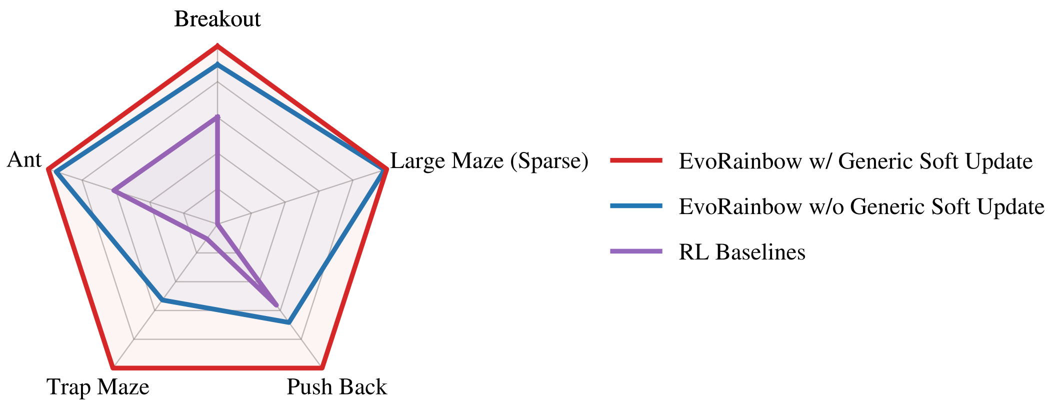


Figure 12. Ablation study on Genetic Soft Update for EvoRainbow. we first average the score over five seeds up to 1 million environment steps. In most tasks, we compute the area under the score curve as it captures not only the final performance but also the amount of interaction required to achieve it. For the Maze tasks, we use the final performance since we find the area can not capture the final performance. Since absolute values vary greatly across games, we report relative quantities by dividing by the maximum value obtained in each game. **The experimental results indicate that Genetic Soft Update can improve algorithm performance across various tasks, consistent with the conclusions presented in the main text. The primary reason for this enhancement is likely the direct impact on RL parameters, which enhances the plasticity of the RL policy. This, in turn, enables better utilization of collected experiences and the ability to break free from suboptimal situations.**

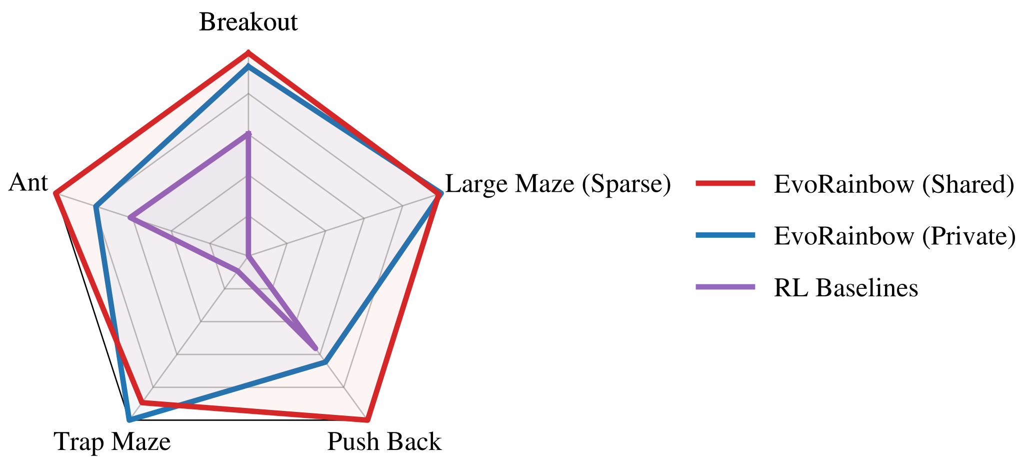


Figure 13. Comparison of shared architecture and private architecture in EvoRainbow across five tasks with different characteristics. we first average the score over five seeds up to 1 million environment steps. In most tasks, we compute the area under the score curve as it captures not only the final performance but also the amount of interaction required to achieve it. For the Maze tasks, we use the final performance since we find the area can not capture the final performance. Since absolute values vary greatly across games, we report relative quantities by dividing by the maximum value obtained in each game. **The experimental results indicate that the shared architecture is more effective for general tasks, leading to accelerated learning and superior final performance, whereas the private architecture is more suitable for exploration tasks, especially in scenarios with sparse or deceptive rewards. The relatively independent nature of individuals ensures strong exploration capabilities. The conclusions drawn from EvoRainbow regarding the shared and private architectures align with the conclusion in the main text.**

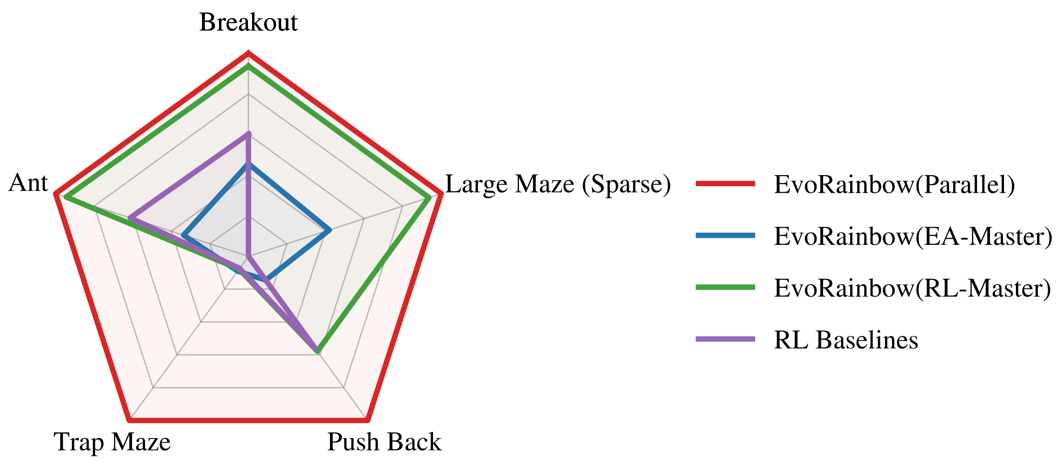


Figure 14. Comparison of Interaction Mode in EvoRainbow across five tasks with different characteristics. we first average the score over five seeds up to 1 million environment steps. In most tasks, we compute the area under the score curve as it captures not only the final performance but also the amount of interaction required to achieve it. For the Maze tasks, we use the final performance since we find the area can not capture the final performance. Since absolute values vary greatly across games, we report relative quantities by dividing by the maximum value obtained in each game. **The experimental results indicate that *Parallel Mode* is more efficient than other modes across various tasks. Since *Parallel Mode* ensures both the exploration capability of EA and the learning ability of RL. The mode replacement based on EvoRainbow further confirms the superiority of *Parallel Mode*, consistent with the conclusions drawn from the initial architecture in the main text.**

C.3. Analysis on Individual Architecture

In this subsection, we analyze the choice of individual architecture, which is one of the main differences between EvoRainbow and EvoRainbow-Exp. EvoRainbow utilizes a shared architecture and fitness surrogate, whereas EvoRainbow-Exp uses a private architecture. We compare the performance of EvoRainbow and EvoRainbow (Private). EvoRainbow (Private) replaces the shared architecture with the private architecture. The experimental results are shown in Figure 13. We can observe that the shared architecture outperforms the private architecture in various tasks, except for maze tasks. In general tasks, the shared architecture achieves better performance, thanks to the smaller policy space constructed by shared representations, which simplifies the difficulty of policy search. For exploration tasks, the performance gap between the two architectures is small in Large Maze, whereas it is significant in Trap Maze. This is mainly because Large Maze provides sparse but non-deceptive signals. Once the policy explores successful trajectories, it can rapidly construct effective shared representations, leading to performance similar to the Private architecture. In Trap Maze, where reward signals are deceptive, the shared representation space constructed based on these signals is prone to collapse, hindering policy search. Overall, the shared architecture is more suitable for regular tasks, whereas the private architecture is more suitable for exploration tasks. The conclusions drawn from EvoRainbow in this context align with the main findings in the main text.

C.4. Analysis on Interaction Mode

In the main text, we analyze the choice of Interaction Mode, which is the cornerstone of EvoRainbow, and its selection directly determines the dominant role of EA and RL in the entire algorithm framework. In the main text, based on the most basic version, the Parallel mode outperforms EA-Master mode and RL-Master mode on various types of tasks. Here, we reanalyze based on EvoRainbow to verify whether the conclusion is consistent with the initial findings.

The experimental results, as shown in Figure 14, indicate that building on EvoRainbow, Parallel mode remains significantly superior to EA-Master mode and RL-Master mode across different types of tasks, RL-Master mode outperforms RL baselines, showing significant improvements in three out of the five tasks. However, both RL-Master mode and EA-Master mode fail to successfully reach the target point in the Trap Maze, which requires strong exploration. In RL-Master mode, the

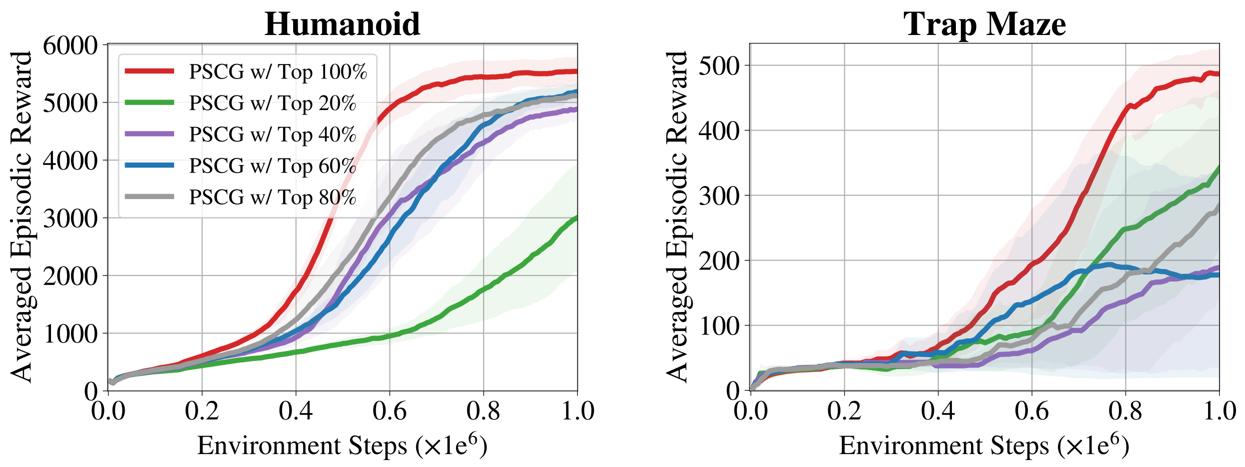


Figure 15. Analysis of Experience Filter on Humanoid and Trap Maze.

population individuals are confined to exploring around the RL policy, whereas EA-Master mode uses RL gradients to guide half of the individuals, without maintaining RL individuals. The former reduces the random exploration capability of EA, and the latter simultaneously diminishes both the exploration capability of EA and the learning ability of RL. Ultimately, both modes struggle to achieve efficient learning. Overall, Parallel mode proves to be more efficient across various tasks, ensuring both the exploration capability of EA and the learning ability of RL. This aligns with the conclusions drawn in the main text.

D. Additional Experiments

Experiments on Experience Filter: We evaluate the Experience Filter mechanism in ”How EA impacts RL”, which aims to ensure data quality by filtering the experiences provided by EA to RL. We provide RL with experiences from the top 20%, 40%, 60%, 80%, and 100% of individuals, and incorporate this mechanism into PSCG and PPCG. We conduct experiments on the Humanoid task and Trap Maze, and the results are illustrated in Figure 15. However, the results show that Experience Filter may lead to a decrease in both sample efficiency and final performance.

E. Experiment details

E.1. Implementation Details in Analysis Experiments

Our code is based on the official implementations of ERL[4], ERL-Re^2[5], PDERL[6], CEM-RL[7], and PGPS[8]. The TD3 algorithm follows the official code of ERL-Re^2. For the RL aspect, we use the PyTorch implementation of SAC[9]. The DQN algorithm uses the official implementation provided by MinAtar[10]. For the EA aspect, CEM uses the implementation from CEM-RL, Vallina-GA uses the implementation from ERL, PD-GA uses the implementation from PDERL, and Behavior-level GA uses the implementation from ERL-Re^2. Below, we provide the experimental details for five subsections of analysis experiments:

- **Interaction Mode**: We compare three modes: Parallel, RL-Master, and EA-Master. The Parallel mode follows the training architecture of ERL, PDERL, and ERL-Re^2. RL-Master follows the implementation of Supe-RL. EA-Master follows the training architecture of CEM-RL. To ensure fairness and stability in training, Vallina-GA is used for the EA aspect in all experiments. We finetune and provide the best results for all methods.

[4]https://github.com/ShawK91/Evolutionary-Reinforcement-Learning
[5]https://github.com/yeshenpy/ERL-Re2
[6]https://github.com/crisbodnar/pderl
[7]https://github.com/apourchot/CEM-RL
[8]https://github.com/NamKim88/PGPS
[9]https://github.com/denisyarats/pytorch_sac
[10]https://github.com/kenjyoung/MinAtar

- **Individual Architecture**: We compare Shared Architecture and Private Architecture. The Shared Architecture follows the implementation of ERL-Re^2 and the Private Architecture follows the implementation in ERL and PDERL. This experiment is based on the best Parallel architecture discovered in the first part for further analysis.
- **Evolutionary Algorithms and Operators**: This section compares Vallina-GA, PD-GA, Behavior-level GA, and CEM. Vallina-GA adjusts the percentage of parameter mutation to $\{0.5, 0.2, 0.1, 0.05, 0.01\}$. PD-GA adjusts the mutation magnitude hyperparameter following the original paper to $\{0.2, 0.1, 0.01, 0.001, 0.0001\}$. CEM mainly adjusts the noise magnitude hyperparameter to $\{0.1, 0.01, 0.001, 0.0001, 0.00001\}$. Behavior-level GA adjusts the percentage of mutation parameters to $\{1.0, 0.7, 0.5, 0.2\}$ following the original paper, the other hyperparameters of the evolutionary algorithm remain consistent with the original paper. In the experiments, the non-evolutionary algorithm hyperparameters are kept consistent across all the algorithms. Finally, we combine these mechanisms based on the best Parallel + Shared and Parallel + Private architectures discovered in the previous sections and report the optimal results under different combinations.
- **How EA impacts RL**: This section explores three mechanisms: Genetic Soft Update, Elite Policy Guide, and Experience Filter. Genetic Soft Update follows the implementation in Supe-RL and we adjust τ to $\{0.1, 0.3, 0.6, 1.0\}$, Elite Policy Guide follows the official code of PGPS. Experience Filter directly filters the experiences provided by EA to RL, testing different proportions of experiences, including only providing the experiences of the top 80%, 60%, 40%, and 20% individuals to the replay buffer for RL training. The above mechanisms are further tested and evaluated on the previously obtained optimal structures.
- **Fitness Surrogate Selection and Usage**: In this section, we first discuss the selection of fitness surrogates, including Critic (Buffer), Critic (H-step Bootstrap), PeVFA (Buffer), and PeVFA (H-step Bootstrap). Critic (Buffer) follows the settings in the SC paper, and the data sample size for sampling is set to 50,000. PeVFA (H-step Bootstrap) uses the settings from ERL-Re^2, where we use one-fifth of a game's episode length as the value for H. Critic (H-step Bootstrap) and PeVFA (Buffer) also follow the aforementioned settings. Regarding the usage of fitness surrogates, we explore Individual Control and Generation Control. For Individual Control, the initial population size is twice the actual population size. For Generation Control, we introduce additional probabilities to control the usage of fitness surrogates, with values of $\{0.2, 0.4, 0.6, 0.8, 1.0\}$.

E.2. Hyperparameter Settings

This section provides a comprehensive overview of the various hyperparameters used in the 20 tasks. Specifically, EvoRainbow includes the settings for noise magnitude in CEM, the τ hyperparameters for Genetic Soft Update, and the probability 1-p of utilizing Critic (H-Step Bootstrap) in fitness surrogates. EvoRainbow-Exp includes the settings for noise magnitude in CEM, the τ parameter for Genetic Soft Update. The detail hyperparameters specific to EvoRainbow and Rainbow are shown in Table 8 and Table 9, other hyperparameters are consistent with ERL-Re^2.

Table 8. Details of the hyperparameters of EvoRainbow across tasks.

Env name	Noise Magnitude	τ	p
MUJOCO *Humanoid*	1e-2	0.1	0.2
MUJOCO *Ant*	1e-4	0.1	0.2
MUJOCO *Walker*	1e-1	0.1	0.6
MUJOCO *Swimmer*	1e-3	0.3	0.4
Maze *Ant Maze*	1e-1	0.6	0.6
Maze *Trap Maze*	1e-5	0.6	1.0
Maze *Wall Task*	1e-3	1.0	1.0
Maze *Large Maze*	1e-1	0.6	1.0
MinAtar *Freeway*	1e-2	0.1	0.2
MinAtar *Space Invaders*	1e-2	0.1	0.4
MinAtar *Breakout*	1e-3	0.1	0.4
MinAtar *Asterix*	1e-2	0.3	0.8
Metaworld *Plate Slide*	1e-2	0.1	0.2
Metaworld *Peg Unplug Side*	1e-2	0.3	0.8
Metaworld *Coffee Push*	1e-4	0.3	0.2
Metaworld *Push Back*	1e-4	1.0	0.2
Metaworld *Soccer*	1e-2	0.3	0.2
Metaworld *Peg Insert Side*	1e-1	0.1	0.8
Metaworld *Sweep Into*	1e-1	0.1	0.8
Metaworld *Button Press Topdown*	1e-3	0.1	0.8

Table 9. Details of the hyperparameters of EvoRainbow-Exp across tasks.

Env name	Noise Magnitude	τ
MUJOCO *Humanoid*	1e-1	0.0
MUJOCO *Ant*	1e-3	0.1
MUJOCO *Walker*	1e-3	0.1
MUJOCO *Swimmer*	1e-3	0.1
Maze *Ant Maze*	1e-1	0.0
Maze *Trap Maze*	1e-1	0.0
Maze *Wall Task*	1e-2	0.1
Maze *Large Maze*	1e-1	0.3
MinAtar *Freeway*	1e-4	0.0
MinAtar *Space Invaders*	1e-2	0.3
MinAtar *Breakout*	1e-3	0.3
MinAtar *Asterix*	1e-3	0.1
Metaworld *Plate Slide*	1e-1	0.1
Metaworld *Peg Unplug Side*	1e-4	1.0
Metaworld *Coffee Push*	1e-5	1.0
Metaworld *Push Back*	1e-5	0.3
Metaworld *Soccer*	1e-3	1.0
Metaworld *Peg Insert Side*	1e-3	0.1
Metaworld *Sweep Into*	1e-2	1.0
Metaworld *Button Press Topdown*	1e-2	0.1