The Journal of Computer Science and Its Applications. An International Journal of the Nigeria Computer Society (NCS)
Vol.32, No.1, April, 2026

Differential Evolution-Augmented Random Search Reinforcement Learning Algorithm

B. A. Sawyerr*[1] bsawyerr@unilag.edu.ng
E. P. Fasina[1] efasina@unilag.edu.ng
A. B. Enenim[2] enenim2000@gmail.com
C. P. Ojiako[1] cojiako@unilag.edu.ng

[1]Department of Artificial Intelligence and Robotics
Faculty of Computing and Informatics
University of Lagos
[2]Department of Computer Sciences
Faculty of Computing and Informatics
University of Lagos

Abstract
Reinforcement learning (RL) has successfully solved many complex tasks; however, it often suffers from three key issues: insufficient exploration, challenging credit assignment under sparse rewards, and unstable convergence that is highly sensitive to hyperparameters. These limitations reduce the range of problems RL can reliably address. Evolutionary algorithms (EAs) can mitigate these issues, but they typically require many samples and struggle with high-dimensional parameter optimisation. In this work, a hybrid method, Differential Evolution–Augmented Random Search (DE-ARS), is proposed. DE-ARS maintains a population of EA agents and periodically injects an RL agent to replace the weakest individual, thereby supplying gradient-related guidance to the evolutionary search. A comparative analysis was conducted on the performance of DE-ARS and Augmented Random Search (ARS) on continuous control OpenAI Gym benchmarks. The results obtained suggest DE-ARS is a better alternative to ARS, with better stability and improved efficiency.

Keywords: Differential Evolution, Augmented Random Search, Reinforcement Learning, Evolutionary Algorithm

1 Introduction

Many engineering and scientific applications involve control problems with continuous actions. Although these problems are often difficult to solve, combining evolutionary algorithms (EAs) with Reinforcement Learning (RL) has produced very good results, with ongoing improvements. This study aims to take advantage of the complementary strengths of EAs and RL by integrating them to achieve better performance. Specifically, Differential Evolution (DE) was hybridised with Augmented Random Search (ARS) to form an evolutionary reinforcement learning (ERL) method called DE-ARS.

Evolutionary algorithms are probabilistic, population-based methods for searching for optimal solutions. They are inspired by natural evolution, and examples include Evolutionary Strategies, Genetic Algorithms (Holland 1984), Genetic Programming, Evolutionary Programming and Differential evolution (Engelbrecht 2007; Price, Storn, and Lampinen 2005). Swarm-intelligence methods such as particle swarm optimisation, ant colony optimisation, bacterial foraging optimisation, and the Bees algorithm are also biologically inspired approaches (Yang 2010). Broader EA classifications may further include self-organising systems, artificial life (digital organisms), and memetic and cultural algorithms (Yang 2010). In this paper, DE was chosen because it can handle non-differentiable, multimodal, and nonlinear objective functions, and it is simple to configure, requiring only a few control parameters.

Differential evolution is a stochastic, meta-heuristic optimisation method grounded in Darwinian evolution and, like many EAs, follows four main phases: initialisation, mutation, recombination, and selection (Price et al. 2014).

Machine learning consists of supervised learning, reinforcement learning, and unsupervised learning. RL methods are often classified into value-based approaches (e.g., Q-learning) and policy-gradient approaches. In RL, an agent learns directly through interaction with its environment—without a labelled dataset—by receiving feedback and discovering which actions yield positive rewards and those that lead to negative outcomes. The strength of RL lies in this trial-and-error learning process, which optimises sequences of actions to maximise cumulative reward (Sutton and Barto 2020).

Reinforcement learning has been successful on complex tasks in both discrete and continuous action spaces, but still struggles with sparse rewards, weak exploration, and unstable convergence, making it unsuitable for many complex real-world problems. These challenges motivated the augmentation of RL with metaheuristics that explore complex search spaces more effectively. EAs are very good at exploring parameter spaces of complex problems. When EAs are combined with RL methods that exploit gradient-related information in the action space, they can yield a more powerful and efficient approach called Evolutionary Reinforcement Learning (EvoRL). This motivation underpins this work. To this end, Differential Evolution–Augmented Random Search (DE-ARS), an EvoRL hybrid that combines differential evolution with a population of ARS policies and selects the best policy each episode, is proposed. The hybrid method will combine the complementary strengths of DE and ARS to develop an effective method for classic continuous-state, continuous-action control problems under practical constraints. The combination of the exploratory capability of EAs with RL's use of gradient-related information results in an approach that can search effectively in both parameter and action spaces, thereby improving its ability to find high-quality solutions in complex landscapes. In DE-ARS, ARS serves as the RL component and DE as the EA component. DE-ARS is benchmarked by comparing it with ARS on selected classic control tasks in the OpenAI Gym environment (Towers et al. 2025).

The remaining sections of this paper are as follows: Section 2 reviews related works in evolutionary reinforcement learning (EvoRL) and ARS. Section 3 presents the methodology and algorithm design, while Section 4 describes the experiments, benchmarks, and statistical analysis of the results. Finally, Section 5 concludes the research work.

2 Literature Review

2.1 Differential Evolution

Differential Evolution (DE) is a stochastic, population-based optimisation algorithm proposed by Kenneth Price and Rainer in 1995 (Price et al. 2005). As a metaheuristic, it makes no assumptions about the objective function and explores a diverse solution space to seek an optimum. For optimisation with D real variables, the population size NP must be at least 4 and remain constant during minimisation. The parameter vectors have the form:

$$X_{ij,G} = [\, X_{i1,G}\, , X_{i2,G}\, , \cdots , X_{iD-1,G}, X_{iD,G}] \qquad i = 1,2,3,\cdots,NP \qquad (1)$$

where G is the number of generations.

The initial population vector is chosen randomly based on a uniform probability distribution and should span the entire parameter space. Some predefined parameters are used in generating the initial population using a random number generator with the formula:

$$U_{ij,G} = 2 \cdot U_j \cdot \text{rand}_j(0,1) - L_j\, , i = 1,2,3,\cdots,NP \quad i = 1,2,3,\cdots,D \text{ and } G = 0 \qquad (2)$$

U_j and L_j denote upper and lower bounds, respectively, for each population vector parameter.

2.1.1 Mutation

For every target vector, $X_{i,G}$, where $i = 1,2,3,\cdots,NP$., A mutant vector can be derived with the equation,

$$U_{i,G} = X_{r1,G} + F \cdot \left(X_{r2,G} - X_{r3,G}\right) \qquad (3)$$

using mutually different random indices $r1, r2, r3 \in \{1,2,\cdots,NP\}$, and $F > 0$. The randomly selected indices $r1, r2$, and $r3$ must be different from i, so that NP is always greater than or equal to four to satisfy the minimum number required for the population size, NP ≥ 4. F is a real number and a constant factor $\in [0,2]$ that controls the amplification effect on the differential variation of $\left(X_{r2,G} - X_{r3,G}\right)$.

2.1.2 Crossover

The crossover operation is used in EA to increase the diversity of the vector parameters

$$\boldsymbol{U}_{i,G} = U_{i1,G}, U_{i2,G}, U_{i3,G}, U_{i4,G}, \cdots, U_{iD,G} \qquad (4)$$

is then formed from:

$$\boldsymbol{U}_{i,G} = u_{ij,g} = \begin{cases} v_{ij,g}, & \text{if } (\text{rand}_j \leq CR \text{ or } j = j_{rand}) \\ x_{ij,g}, & \text{otherwise} \end{cases} \quad (5)$$

CR is the crossover probability that controls the parameters received from the mutant vector. A uniform random number is compared with the CR to determine which mutant vector is inherited, and the condition ($j = j_{\text{rand}}$) is also added to ensure the trial vector is at least different from the current vector in scenarios where the first part of the if statement in equation (5) always evaluates to false.

2.1.3 Selection

To determine whether the trial vector, $U_{i,G}$ should become a member of the generation $G + 1$, it is compared to the target vector, $X_{i,G}$ element-wise based on the value of their objective function. If vector $U_{i,G}$ yields a smaller cost function value than $X_{i,G}$, then $X_{i,G+1}$ is set to $U_{i,G}$, otherwise the old value is retained.

This procedure can be described by the following formula, where $i = 1,2, \cdots, NP$.

$$\boldsymbol{X}_{i,g+1} = \begin{cases} \boldsymbol{U}_{i,g}, & \text{if } \left(f(\boldsymbol{U}_{i,g}) \leq f(\boldsymbol{X}_{i,g})\right) \\ \boldsymbol{X}_{i,g}, & \text{otherwise} \end{cases} \quad (6)$$

Once the population for the next generation is fully selected, the mutation and selection begin and the process continues until either the optimum is reached or the termination criterion is met.

2.1.4 DE termination criteria

DE termination criteria include getting the target objective, reaching a maximum number of evaluations (or generations), or loss of population diversity.

2.1.5 DE Variants

Apart from the classical DE variant, there are also other useful DE schemes (Adekanmbi, Olugbara, and Adeyemo 2014). The notation $DE/x/y/z$ is used to classify different DE variants where, x indicates the mutated vector, which can either be a randomly selected individual or the best individual from the current population, y denotes the number of individuals used for computing the difference, and z is the crossover scheme. The various ways mutation can be done, together with the two crossover schemes - Binomial (or uniform) and exponential (or two-point modulo) discussed in (Price et al. 2005), give rise to different strategies of DE called variants.

2.2 Reinforcement Learning

Reinforcement learning (RL) trains an agent through trial and error, using rewards and penalties to learn a policy that maximizes long-term return. Rather than relying on labelled or unlabelled datasets, the agent learns directly from environment feedback. Examples include Q-learning and policy-gradient methods, as well as evolutionary approaches such as Augmented Random Search (ARS) and Evolutionary Reinforcement Learning (EvoRL), which combines gradient-based RL with evolutionary population search.

In RL, the agent observes the environment, takes actions, and updates its policy based on the feedback received. Over time, it learns which actions to repeat and which to avoid, balancing exploration (trying new actions) and exploitation (using what it has learned) to reduce negative rewards and improve performance.

Historically, RL evolved from two largely independent threads —trial-and-error learning and optimal-control methods—now unified in modern RL. The first thread is trial-and-error learning, inspired by human and animal learning. The second uses dynamic programming and value functions for optimal control, including Markov decision processes (MDPs) and the Bellman equation (Sutton and Barto 2020).

Dynamic programming (DP) remains a core approach for stochastic optimal control, but it scales poorly: as the number of state variables grows, computation can increase exponentially. The trial-and-error thread also led to Monte Carlo methods and temporal-difference (TD) learning; TD combines sampling-based learning with ideas from dynamic programming (Sutton and Barto 2020).

RL algorithms vary in how they explore and represent decision-making, using either policy-based methods (directly choosing actions) or value-based methods (estimating state/action values). Common examples include SARSA, Q-learning, and Deep Q-Networks.

Despite its successes, RL faces practical challenges, including limited training samples, the difficulty of building realistic environments where agents can safely fail and learn, and sparse reward signals. RL has been applied in many real-world domains, such as finance, robotics, natural sciences, transportation, engineering, and healthcare.

2.2.1 Augmented Random Search

Augmented Random Search (ARS) is a derivative-free method that explores directly in parameter space, unlike many RL approaches that search in action

space. It relies on noisy function evaluations: ARS samples random directions in parameter space and then improves the policy by optimising performance along those directions.
Many RL methods use deep networks with multiple hidden layers (and therefore require substantial computation), while ARS uses a simple perceptron policy, which makes it comparatively fast. Experiments on MuJoCo tasks in OpenAI Gym report that ARS can outperform several traditional RL algorithms and run at least 15 times faster than competing model-free methods on these benchmarks (Mania, Guy, and Recht 2018). ARS extends basic random search by evaluating perturbed parameter vectors and using the results to update the policy. Further details on random search and ARS are provided in (Mania et al. 2018).

2.3 Related Works

2.3.1 Evolutionary Reinforcement Learning

This section focuses on RL algorithms that adopt the evolution strategy in the learning process, or reinforcement learning that is guided by any of the evolutionary algorithms (EAs).
Some traditional RL techniques, such as policy gradients, can get trapped in a local optimum and, as a result, will not get the optimal policy. But this weakness can be overcome by leveraging the explorative property of evolutionary algorithms. The concept behind evolutionary reinforcement learning (EvoRL) is to combine the strength of population-based search that explores in parameter space with the gradient-based methods that explore in action space to effectively train RL agents.

Lin et al. (2025) reviewed approximately 100 evolutionary reinforcement learning (EvoRL) studies, and they categorise them into five primary mechanisms based on the type of EAs they incorporate to optimise the reinforcement learning process. The five primary mechanisms are Evolutionary Strategy (ES)-based variants, Genetic Algorithm (GA)-based, Cross-Entropy Method (CEM)-based, Population-Based Training (PBT)-based, and Other Specialised variants.

The Evolutionary Strategy (ES)-based variants used direct policy optimisation through mutation to enhance the performance of RLs. Examples include Adaptive Ensemble (Ajani and Mallipeddi 2022) and R-R1-ES (Chen et al. 2019). (Khadka and Tumer 2018) gave a general architecture of EvoRL showing the incorporation of EA's population-based learning with RL's gradient-based method. The Genetic Algorithm (GA)-based variants, such as Proximal Distilled Evolutionary RL (Bodnar, Day, and Lió 2020), leverage genetic diversity via crossover and mutation to find stable policies in highly complex, dynamic environments. Cross-Entropy Method (CEM)-based variants, including CEM-RL (Pourchot and Sigaud 2019), utilise elite selection and statistical iteration for exploration, whereas Population-Based Training (PBT)-based frameworks like EARL (Wang et al. 2023) dynamically adjust hyperparameters and actions to improve sample efficiency.

Despite having several EvoRLs in the literature, the computational cost of using EvoRLs to solve real-life problems is prohibitively expensive. ARS has proven to be 15 times more effective than traditional RLs (Mania et al. 2018). The contribution of this paper is to optimise ARS by hybridising it with DE.

3. METHODOLOGY

The methodology employed is experimental design methodology. This involves a step-by-step process conducted in a controlled environment to discover an effect that must be well tested to establish a hypothesis.

3.1 Research Methodology

The experimental design methodology employed consists of eight phases, namely problem definition, defining objectives, design of experiment, experimentation, gathering of experimental data, analysing the data, interpreting and verifying results.

3.2 Description of the Proposed DE-ARS Algorithm

In this paper, a hybrid of DE and ARS was developed. The hybrid algorithm leverages the population-based DE to produce various experiences to train an RL agent and periodically transfers the RL policy into the DE population policies to inject RL gradient knowledge into the DE population, where ARS is the RL component. The essential takeaway from this is that any EA may be used to address the fundamental issues with RL without sacrificing the ability to use gradients for improved sampling efficiency. By using a fitness metric that aggregates the return of a full episode, evolution-based RL inherits EA's capacity to address the issue of temporal credit assignment.

In the population-based ARS called DE-ARS, DE was used as the evolutionary algorithm component and ARS as the RL component. A combination of these two techniques to obtain an improvement on ARS is shown in Algorithm 1.

Algorithm 1: DE-ARS Algorithm
Algorithm 1: Differential Evolution-Augmented Random Search

1: Initialise the hyperparameters
2: Initialise the initial population of the agent's policy
3: **for** each step in max time step **do**
4: **for** each policy π in population **do**
5: generate two other policies not equal to π, and the current policy with the highest reward
6: Perturb these three different policies using the DE mutation operation to get a trial vector u
7: Evaluate the fitness/reward of the trial vector u produced from mutation in step 6
8: Replace the current π and its reward with u and its reward
9: **end for**
10: evaluate a policy π using augmented random search (ARS)
11: sort the DE Population in order of fitness
12: **if** step mod synchronisation = 0 **then**
13: Replace the weakest member of the DE population with the ARS policy π
14: **end if**

4. EXPERIMENT AND RESULTS

To compare the performance of ARS with the DE-ARS, each model was trained on some selected classical controls and Mujoco benchmarks from the OpenAI Gym (Towers et al. 2025). Both RL models are trained on the selected benchmarks for 1,000 episodes, with each episode length limited to a maximum of 1,000. The complexity of the RL environment varies, and a detailed description of each benchmark/testbed can be found in (Towers et al. 2025). All algorithms were implemented in Python, run and tested on colab.research.google.com. The last 40 rewards are used to compute the average rewards used for plotting the analytical graph.

4.1 Experiment Design

The ARS is arguably the most effective RL model, and the results for ARS were obtained by running the official ARS implementation on Google Colab on each of the following testbeds: Pendulum-v1, MountainCarContinuous-v0, Humanoid-v4, HalfCheetahBulletEnv-v0, Reacher-v4, Ant-v4, Walker2d-v4. The average reward obtained by the agent in each episode was saved in a list as training progressed. At the end of training, the list of average rewards was then saved in a file. For each benchmark, two files are maintained to store the average rewards list; one contains the average reward for ARS, and the other contains the average reward for DE-ARS. A comparison graph that depicts the episodic progress of both RL models was plotted. These were repeated for all the benchmarks. Table 1 illustrates the parameter settings used during the experiment. A detailed analysis of the results is presented in the subsequent sections.

Table 1: Experiment parameters

S/N	Parameter	Value
1	Number of Population (NP)	10
2	Crossover rate (CR)	0.85
3	Mutation constant (F)	0.9
4	Perturbation ratio	0.2
5	Number of episodic steps	1000
6	Number of episodic lengths	1000
7	Learning rate	0.02
8	Noise	0.03

4.2 Comparison of ARS and DE-ARS Results

The performance of DE-ARS agents was evaluated on 6 Mujoco benchmarks. These are benchmarks used widely in the reinforcement learning field and are hosted through the OpenAI gym (Brockman et al. 2016). The performance of DE-ARS was compared with a standard augmented random search discussed in (Mania et al. 2018). Both algorithms were implemented in the Python programming language. Each algorithm was trained for about 1,000 steps, with an episodic length of 1,000. Table 2 tabulates the average rewards obtained by each model.

Table 2: Average reward obtained by each RL model

S/N	Benchmarks	ARS Reward	DE-ARS Reward
1	Reacher-v4	-113.254	-53.925
2	Pendulum-v1	-922.226	-493.575
3	HalfCheetahBulletEnv-v0	1401.280	2169.986
4	MountainCarContinuous-v0	-0.003	52.417
5	Walker2d-v4	997.722	1013.040
6	Ant-v4	-373.388	11.020

Table 2 shows that the performance of DE-ARS is better than ARS on all benchmarks. Below is the graph that illustrates the behaviour of each model on different benchmarks. A discussion on the graphical illustration and the progressive performance of each model is presented in the subsequent section.

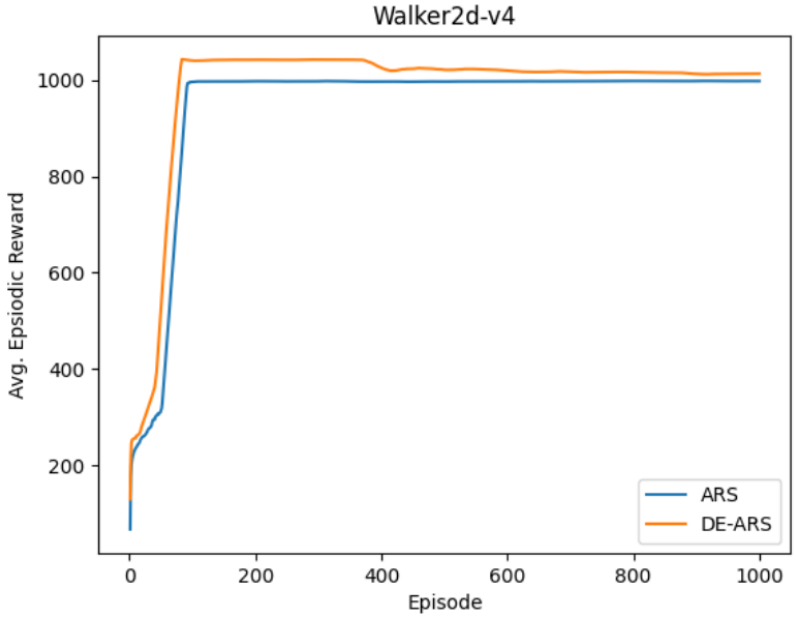


Figure. 5a: Learning curves on Walker2D benchmarks.

Figure 5a illustrates the average episodic rewards obtained by DE-ARS and ARS during each time step of the training for Walker2D benchmarks. From the graph, it is obvious that DE-ARS outperformed ARS in terms of average rewards obtained. ARS converged early, after about the 40th episode, while DE-ARS converged after the 400th episode.

Figure 5b illustrates the average episodic rewards obtained by DE-ARS and ARS during each time step of the training for HalCheetahBulletEnv-v0 benchmarks. DE-ARS outperformed ARS in terms of average rewards obtained, although ARS performed better than DE-ARS in the first 80 episodes. Eventually, DE-ARS overtook ARS until they both converged.

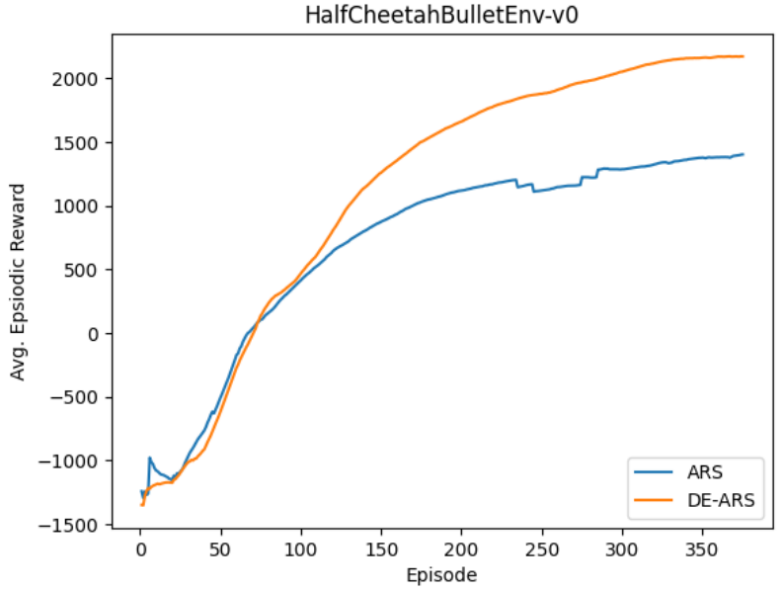


Figure. 5b: Learning curves on HalCheetahBulletEnv-v0 benchmarks.

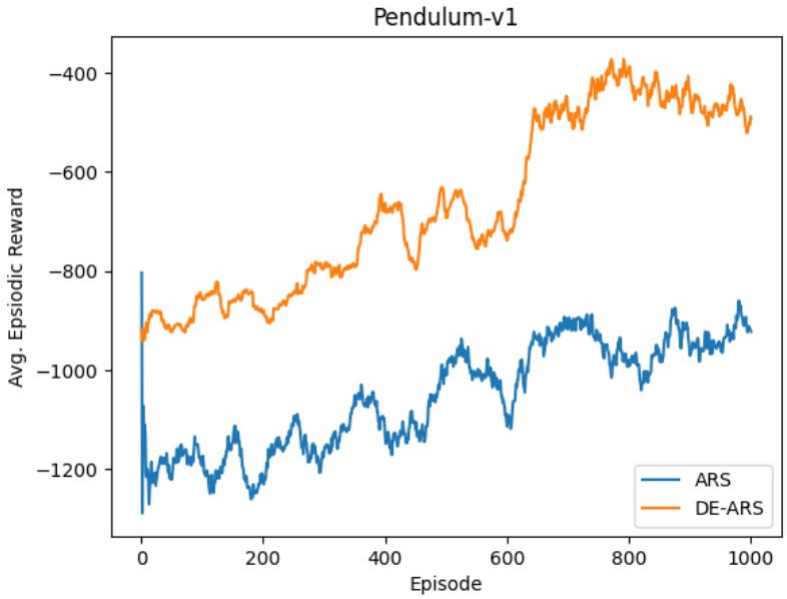


Figure. 5c: Learning curves on Pendulum-v1 benchmarks.

Figure 5c illustrates the average episodic rewards obtained by DE-ARS and ARS during each time step for Pendulum-v1 benchmarks. DE-ARS outperformed ARS in terms of average rewards obtained. However, neither of the two models converged.

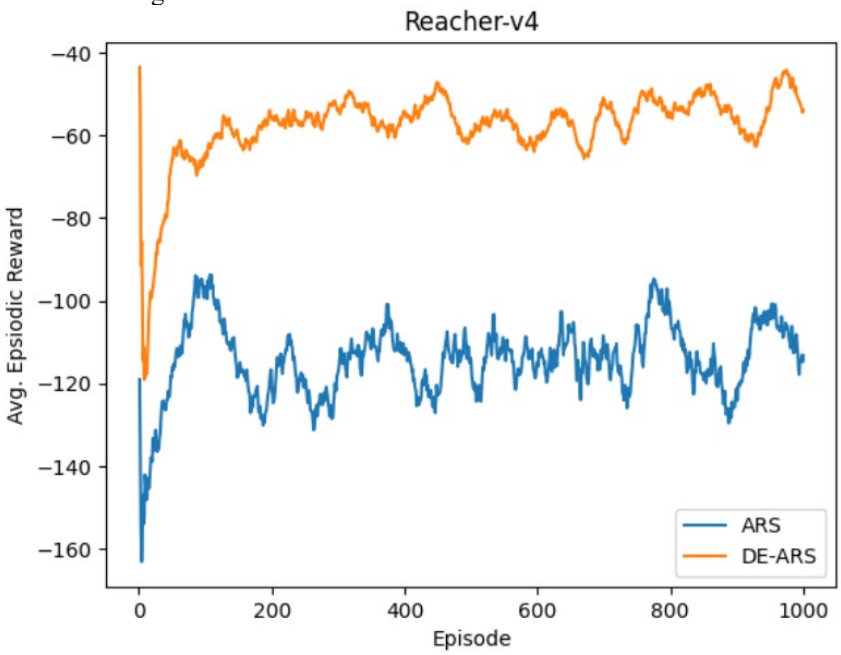


Figure. 5d: Learning curves on Reacher-v4 benchmarks.

Figure 5d shows the average episodic rewards achieved by DE-ARS and ARS during each step of training for Reacher-v4 benchmarks. It is clear from the graph that DE-ARS outperformed ARS in terms of average rewards. However, neither of the two models converged.

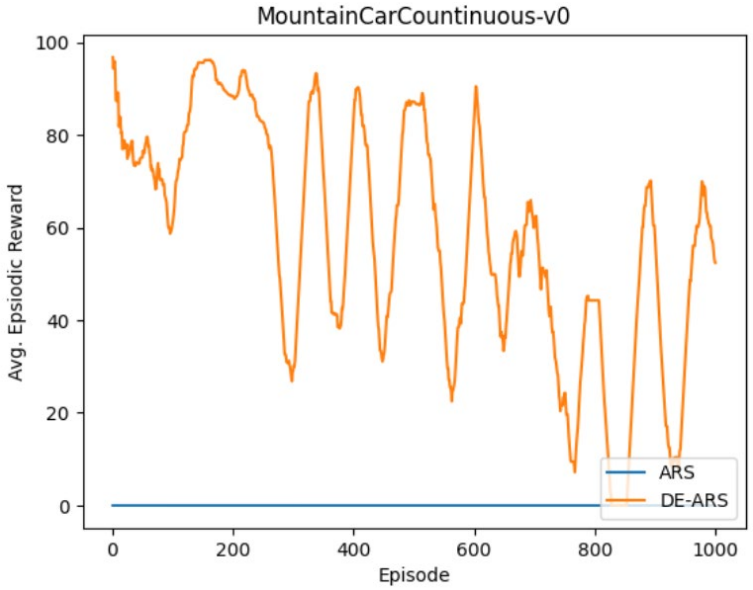


Figure. 5e: Learning curves on MountainCarContinuous-v0 benchmarks.

The graph in Figure 5e shows the average episodic rewards achieved by DE-ARS and ARS at each training step on the MountainCarContinuous-v0 benchmark. DE-ARS consistently outperformed ARS in terms of average reward. In contrast, the ARS curve remained flat throughout training, indicating that ARS failed to learn effectively in this environment.

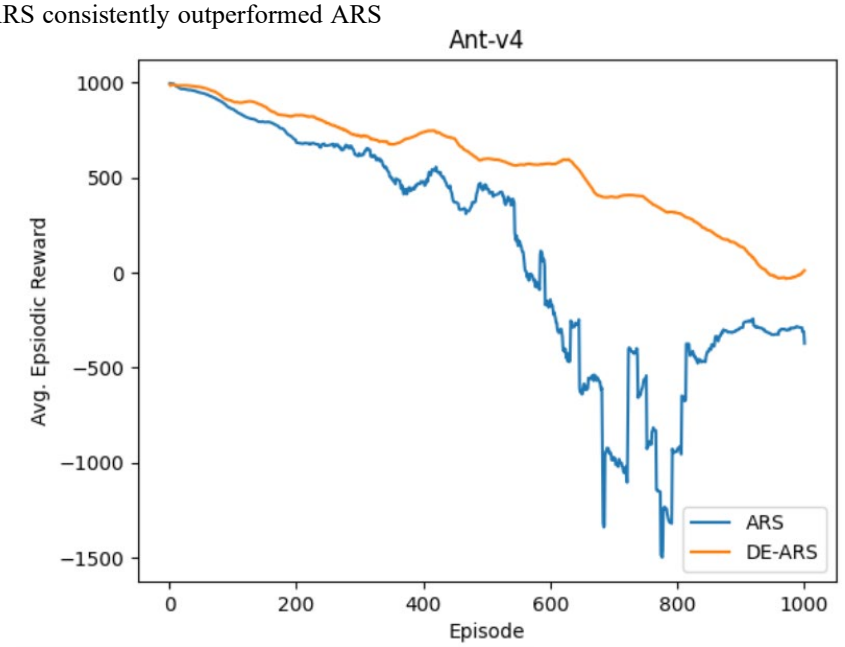


Figure. 5f: Learning curves on Ant-v4 benchmarks.

The graph in Figure 5f shows the average episodic rewards obtained by DE-ARS and ARS at each step of the training for Ant-v4 benchmarks. From the graph, it is obvious that DE-ARS outperformed ARS in terms of average rewards obtained. But neither of the two models converges. Both models also do not show any significant learning during the entire training episodes since the curve shows a downward slope, indicating that both ARS and DE-ARS were incapable of learning in such a problem environment.

4.3 Discussion

Figures 5a-f show the comparative performance of ARS and DE-ARS. DE-ARS outperforms ARS across all the benchmarks, as illustrated in both figures. DE-ARS obtained a higher average reward for each episode and maintained the lead against ARS for the following problems: HalfCheetahBulletEnv-v0,

Reacher-v4, Pendulum-v1, and Walker2d-v4. On the other hand, both ARS and DE-ARS struggle to achieve meaningful learning for Ant-v4 and MountainCarContinuous-v0, as the reward obtained by the two models keeps declining with training time; even in this scenario, DE-ARS still outperformed ARS in terms of rewards obtained per training episode.

5. CONCLUSION

Observation shows that there is a high probability that a learning model that uses population-based policies will outperform the same model using a single policy. This claim was demonstrated in this experiment, where ARS has a single policy called θ, and DE-ARS maintains many policies of different θ parameters. DE-ARS picks the best of the policies in each learning episode.

One of the problems encountered was the lengthened time interval required for training the model to converge. There was a noticeable difference in most of the Mujoco benchmarks that were used as testbeds in the experiment. Another challenge encountered was the ability of the DE-ARS model to learn and converge efficiently for a higher value of the synchronisation period ω. And the number of Trials ξ has little effect on the learning model but will further prolong the time required for the agent to complete training. Another problem encountered was that some of the selected benchmarks seem to be unsuitable as an ideal benchmark for the competing models.

This research aimed to explore the workings of EvoRL, and how to leverage the EvoRL concept to devise a hybrid algorithm that uses population-based policies that have the gradient information injected into the population of θ for every ω to generate a diverse experience to train an RL agent. The population of θ was perturbed using differential evolution, and the gradient information was provided by augmented random search.

Future work will focus on training reinforcement learning agents with different evolutionary algorithms and comparing their performance with ARS and DE-ARS. The best-performing model could then be applied to a trading bot to support more intelligent decision-making. Further research will also examine how effectively these models can be used for asset allocation in financial portfolio management. Evolutionary reinforcement learning is a rapidly developing area of artificial intelligence with the potential to transform reinforcement learning. It offers significant promise and growing research interest, making it a compelling field for both academic and industry exploration.

References

Adekanmbi, Oluwole, Oludayo Olugbara, and Josiah Adeyemo. 2014. "An Investigation of Generalized Differential Evolution Metaheuristic for Multiobjective Optimal Crop-Mix Planning Decision." *Scientific World Journal* 2014. doi:10.1155/2014/258749.

Ajani, Oladayo S., and Rammohan Mallipeddi. 2022. "Adaptive Evolution Strategy with Ensemble of Mutations for Reinforcement Learning." *Knowledge-Based Systems* 245:108624. doi:10.1016/j.knosys.2022.108624.

Bodnar, Cristian, Ben Day, and Pietro Lió. 2020. "Proximal Distilled Evolutionary Reinforcement Learning." *Proceedings of the AAAI Conference on Artificial Intelligence* 34(04):3283–90. doi:10.1609/aaai.v34i04.5728.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. "OpenAI Gym." http://arxiv.org/abs/1606.01540.

Chen, Zefeng, Yuren Zhou, Xiao-yu He, and Siyu Jiang. 2019. "A Restart-Based Rank-1 Evolution Strategy for Reinforcement Learning." Pp. 2130–36 in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. California: International Joint Conferences on Artificial Intelligence Organization.

Engelbrecht, Andries P. 2007. *Computational Intelligence: An Introduction*. 2nd Editio. Wiley.

Holland, John H. 1984. "Genetic Algorithms and Adaptation." Pp. 317–33 in *Adaptive Control of Ill-Defined Systems*. Boston, MA: Springer US.

Khadka, Shauharda, and Kagan Tumer. 2018. "Evolution-Guided Policy Gradient in Reinforcement Learning." http://arxiv.org/abs/1805.07917.

Lin, Yuanguo, Fan Lin, Guorong Cai, Hong Chen, Linxin Zou, Yunxuan Liu, and Pengcheng Wu. 2025. "Evolutionary Reinforcement Learning: A Systematic Review and Future Directions." *Mathematics* 13(5).

Mania, Horia, Aurelia Guy, and Benjamin Recht. 2018. "Simple Random Search Provides a Competitive Approach to Reinforcement Learning."

Pourchot, Aloïs, and Olivier Sigaud. 2019. "CEM-RL: Combining Evolutionary and Gradient-Based Methods for Policy Search." http://arxiv.org/abs/1810.01222.

Price, Kenneth, Rainer M. Storn, and Jouni A. Lampinen. 2014. *Differential Evolution a Practical Approach to Global Optimization*. Springer Berlin.

Price, Kenneth V., Rainer M. Storn, and Jouni A. Lampinen. 2005. *Differential Evolution*. Berlin/Heidelberg: Springer-Verlag.

Sutton, Richard S., and Andrew G. Barto. 2020. *Reinforcement Learning: An Introduction*. The MIT Press.

Towers, Mark, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. 2025. "Gymnasium: A Standard Interface for Reinforcement Learning Environments." http://arxiv.org/abs/2407.17032.

Wang, Zhong Zheng, Kai Zhang, Guo Dong Chen, Jin Ding Zhang, Wen Dong Wang, Hao Chen Wang, Li Ming Zhang, Xia Yan, and Jun Yao. 2023. "Evolutionary-Assisted Reinforcement Learning for Reservoir Real-Time Production Optimization under Uncertainty." *Petroleum Science* 20(1):261–76. doi:10.1016/j.petsci.2022.08.016.

Yang, Xin-She. 2010. "Nature-Inspired Metaheuristic Algorithms."