A Generalized Reinforcement Learning Controller for Metaheuristic Hyperparameter Control

Victor-Deian Balutoiu [1], Roxana Teodora Mafteiu-Scai [1], Liviu Octavian Mafteiu-Scai [1]

[1] *Faculty of Informatics, West University of Timisoara, Romania*

Abstract **- It is well known that the performance of population-based metaheuristics depends strongly on how their hyperparameters are tuned. Among the various techniques available, one of the most prominent is Reinforcement Learning (RL). In general, existing RL approaches are specialized and tightly coupled with a single optimization algorithm. This paper proposes a generalized hyperparameter control method for metaheuristics, hereinafter referred to as PPO-based RL agent. The agent operates through an abstraction layer that maps generic control actions to algorithm-specific parameters. The model was trained exclusively on Differential Evolution (DE) using a set of six benchmark functions. It was evaluated on both the problems used for training and unseen problems, over 30 independent runs each. The experiments showed that the RL agent learned a generalized tuning policy that consistently outperformed static parameters on deceptive landscapes (such as the Schwefel and Rastrigin functions) and achieved comparable results to LSHADE, a specialized, state-of-the-art variant of DE. Another objective of this work was to evaluate the zero-shot transfer potential to other metaheuristics, such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA).**

Keywords **- reinforcing learning, metaheuristics, hyperparameters, tuning.**

DOI: 10.18421/SAR92-01
https://doi.org/10.18421/SAR92-01

Corresponding author: Liviu Octavian Mafteiu-Scai,
Faculty of Informatics, West University of Timisoara, Romania
Email: liviu.mafteiu@e-uvt.ro

Received: 29 April 2026.
Revised: 03 June 2026.
Accepted: 10 June 2026
Published: 27 June 2026.

1. Introduction

Mathematical optimization is a critical component of many technical domains, including training neural networks and designing complex systems. While gradient-based methods are mostly sufficient for simple optimization problems, they typically fail when applied to high-dimensional or non-convex landscapes. For such complex scenarios, population-based metaheuristics are widely deployed. Algorithms such as Differential Evolution (DE), Particle Swarm Optimization (PSO), and Genetic Algorithms (GA) utilize nature-inspired mechanisms to approximate optimal solutions within reasonable computational timeframes.

The success of metaheuristics relies heavily on the tuning of their hyperparameters. These parameters dictate the algorithmic behavior, and their optimal configuration is essential for achieving high-quality optimization results. Before advanced control methods were introduced, these parameters were typically fixed at the start of the run based on trial-and-error and human expertise. A primary limitation of static parameter configurations is that the algorithm can easily become trapped in local optima.

1.1. Real-World Applications

Adaptive hyperparameter control of metaheuristics is vital in many practical domains, particularly Machine Learning (ML). Since the hyperparameter optimization of neural networks is computationally expensive, metaheuristics are increasingly utilized to streamline this process [1].

For example, PSO has been applied to optimize the kernel sizes, strides, and filter counts of Convolutional Neural Networks (CNNs) for tasks such as mammography breast cancer classification [2].

Another notable real-world application is the dynamic scheduling of smart grids in modern power systems. The Economic Load Dispatch (ELD) problem requires algorithms to adapt to constantly changing variables.

Specifically, [3] describes the usage of PSO to dynamically distribute power demand among generators.

The field of bioinformatics also leverages metaheuristics for tasks such as protein structure prediction. Predicting the 3D structure of a protein from its amino acid sequence involves finding the conformation with the lowest free energy. In this context, [4] proposes a DE method to navigate this energy landscape effectively without becoming trapped in local minima.

Furthermore, using a real dataset from Europe and extensive experimentation, [5] demonstrates the benefits of employing metaheuristics for parameter tuning in Deep Learning (DL) models applied to energy load forecasting.

1.2. State of the Art

1.2.1. Classical Approaches to Hyper-parameter Tuning and Control

Prior to the usage of more modern techniques, researchers used offline tuning to improve the performance of metaheuristics. Such an example is proposed in the work [6] where an upper level metaheuristic (Genetic Algorithm) is used to determine the most appropriate set of parameters for a low level metaheuristic (Ant Colony System) to solve the Set Covering Problem.

These static parameterization methods are limited, especially on more complex problems, so the evolution toward adaptive control methods was a natural one.

1.2.2. Adaptive Hyperparameter Control in Specific Metaheuristics

Adaptive methods use feedback from the metaheuristic to adjust parameters at runtime. These methods are usually more successful than static algorithms and are usually tightly coupled with specific metaheuristics.

In year 2013, Tanabe and Fukunga proposed such an algorithm for DE called Success History Based Parameter Adaptation (SHADE) [7]. This algorithm keeps a history of parameter combinations that yielded powerful offspring and uses them to adjust scaling factor and crossover rate values for next step.

The algorithm proposed in [7] was improved in [8] under the name L-SHADE (SHADE with Linear Population Size Reduction). This new proposed method controls population size (dynamically resizing) using a linear function, so as the runtime progresses, the population gets smaller and such, the algorithm starts to focus on exploitation. For population Linear Population Size Reduction (LPSR) was used, which reduce the population linearly as a function of the number of fitness evaluations.

In paper [9], it is proposed a method that adjusts the inertia weight parameter in the case of Particle Swarm Optimization (PSO) metaheuristic. This parameter is adjusted based on the success rate of the swarm.

1.2.3. RL-Based Hyper-parameter Control

Hyper-parameter control is a sequence of decisions. This makes the use of Reinforcement Learning (RL) algorithms a promising research direction. The advantage of the RL is that it allows the controller to learn a certain behavior, without relying on any predetermined rules. It should be able to learn control policies based on the current state of the optimization process, such as encouraging exploration when the algorithm gets stuck, or focusing on exploitation when a potential solution is found.

There are two main approaches detailed in the literature: operator selection and continuous parameter control.

In the operator selection approaches, the agent has to choose the best course of action from some predetermined strategies. In this case, the agent does not change the actual values of the parameters, but only chooses the best search operator for a specific situation (such as mutation strategies, crossover schemes or neighborhood perturbation operators). Such an approach is proposed in the paper [10]. The authors proposed an RL-based operator selection mechanism that incorporates the problem state into its decision-making process. The RL algorithm has been developed based on Q-Learning merged with Hard-C-Means clustering algorithm. It showed that operator selection can be improved by treating it as a control problem and applying a RL-based solution. Using public test data, the results of the proposed method have been comparatively tested with other adaptive methods and the results published show that the proposed approach produces much better performance.

In the case of second approach, continuous parameter control, the actual numerical values of the hyper-parameters are adjusted at runtime, by the agent. Paper [11] proposes two parameter controllers for evolutionary algorithms using dynamic discretization of the parameter range. The proposed method improves the parameter value during the whole optimization process contrary to the other methods. The proposed approach demonstrates that RL can be used for adjusting parameter values during runtime and outperform other methods of hyper-parameter selection.

In summary, several conclusions can be drawn. Firstly, the performance of population-based metaheuristics is very sensitive to the hyper-parameter selection.

Secondly, adaptive control methods usually outperform static methods, this being the most easily observed on complex problems. Finally, it is clear that RL is a valid option for online parameter control.

2. Proposed Method

Most approaches are specific to one single algorithm type and are not applicable to any other metaheuristic.

A possible research gap in algorithm-agnostic hyper-parameter control arises here. This work addresses this gap by designing an abstraction layer between the solver and the controller. The goal of this work is to show that control behaviors can be abstracted and applied on different metaheuristics, such as Differential Evolution (DE), Particle Swarm Optimization (PSO) and Genetic Algorithms (GA).

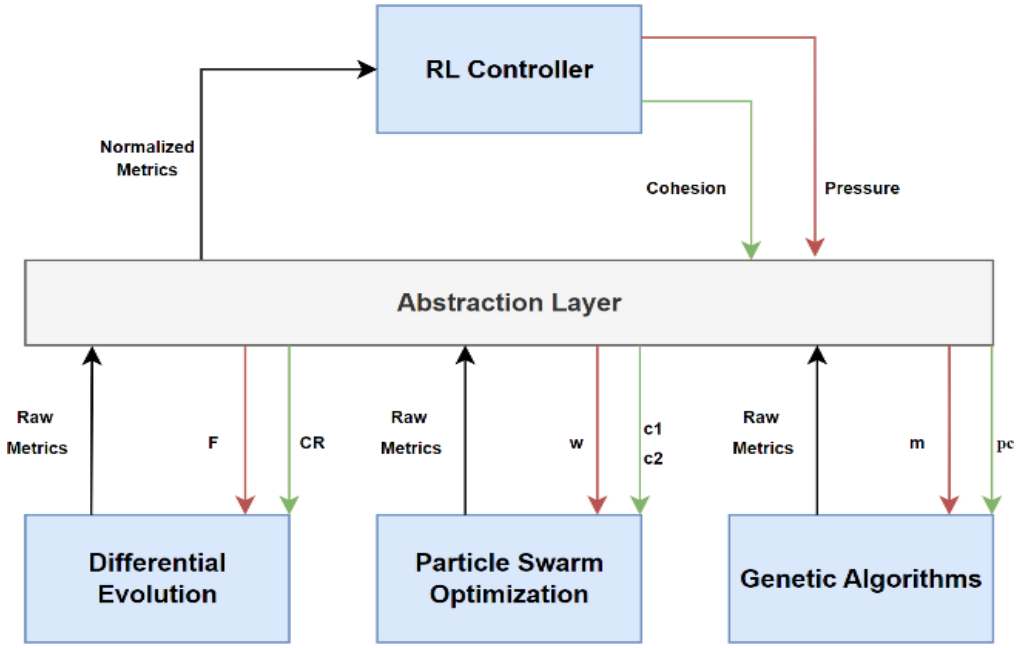

Figure 1. The System Architecture

2.1. Pseudocode of the Proposed Approach

The pseudocode of the proposed approach is shown below:

Input:
Metaheuristic algorithm M
Trained RL agent C
Abstraction layer A
Maximum number of generations max_generations
Target fitness target_fitness

Output:
Best solution found best_solution

Initialize population randomly
Evaluate fitness of population
Initialize best_solution from population
Set generation counter generation ← 0

while (generation < max_generations) and (best fitness > target_fitness) do
    Compute population diversity
    Compute stagnation
    Compute state from diversity and stagnation

    Pass state to agent C
    Receive actions pressure and cohesion from C

    Map pressure and cohesion to algorithm parameters using A
    Update parameters of M

    Execute one generation of M
    Evaluate fitness of population

    if better solution found then
        Update best_solution
    end if
    generation ← generation + 1
end while

return best_solution

2.2. System Architecture and Theoretical Description

This paper proposes a reinforcement learning framework that is designed for the control of population-based metaheuristics. The idea is that different optimization algorithms use similar high-level logic for their search behavior, even though they are based different mathematical rules.

To be able to control this general behavior, we introduced an abstraction layer. The framework was modeled as a Markov Decision Process (MDP) [12] where a RL agent acts as the controller for the optimization algorithm.

The architecture of the proposed system is illustrated in Figure 1. As shown in Figure 1, the proposed system has three layers:

1. The Agent Layer represented by RL controller;
2. The Abstraction Layer serves as a bidirectional translator between the RL agent and the metaheuristics;
3. The Solver Layer is represented by the metaheuristic that is being tuned.

The control loop consists of two paths. One path sends raw metrics from the solver to the abstraction layer, where they are normalized, to avoid overfitting to a specific mathematical scale. The other path sends the two abstracted parameters to the abstraction layer, where they are translated to the specific hyper-parameters for the metaheuristic that is being tuned.

In Figure 1, the notations denote the specific hyperparameters translated by the Abstraction Layer for each metaheuristic. For Differential Evolution (DE), F represents the Mutation Factor and CR denotes the Crossover Rate. For Particle Swarm Optimization (PSO), w indicates the Inertia Weight, while c1 and c2 represent the cognitive and social coefficients, respectively. Finally, for Genetic Algorithms (GA), m stands for the Polynomial Mutation Index and pc represents the Crossover Probability.

Additionally, Figure 1 illustrates the bidirectional flow of observation data. Specifically, these consist of the raw spatial diversity of the population and the raw stagnation counter. Normalized Metrics are the scale-invariant, solver-agnostic state variables (Relative Diversity and Normalized Stagnation) produced by the Abstraction Layer.

The description of the system components and how they work is given below.

2.3. Reinforcement Learning

The RL agent uses the Proximal Policy Optimization algorithm, which was implemented using Stable-Baselines3 [13]. This agent looks at the normalized state and outputs continuous control actions at each generation.

- State Space

The RL agent receives a continuous state vector which is bounded in the interval [0,1] from the Abstraction layer. This vector contains the relative diversity, which is the population spatial variance divided by the initial population diversity.

The vector also contains the normalized stagnation. This is calculated as the current number of generations without any fitness improvements, divided by a maximum threshold set at 100 generations.

- Action Space

The RL agent outputs two abstract behavioral parameters:

Pressure: controls the intensity of the search. High pressure makes the population explore more aggressively, while low pressure creates exploitation. It was named "Pressure" as it is a good visual of the population being pressured to explore;

Cohesion: controls how much information is shared by candidate solutions. High cohesion means that the solutions are more tightly linked and more strongly influence each other, while low cohesion promotes more independent searching. This was named "Cohesion" because it describes how closely the population interacts.

- Reward Function

The reward function is based on logarithmic fitness improvement. If the current best fitness (ft) is better than the fitness of the previous generation (ft-1) a positive reward is given. If not, a small penalty is applied. The relation to compute this value is:

$$R_t = \begin{cases} 2.0 \cdot (1.0 + \ln(1.0 + \Delta f)) & if\ \Delta f > 0 \\ -0.2 & if\ \Delta f \leq 0 \end{cases} \quad (1)$$

where $\Delta f = f_{t-1} - f_t$

This function is designed to encourage fitness improvements between runs. Since ln (1) = 0, the 1.0 offset is added to ensure that even the smallest improvement is rewarded. The times 2 multiplier is used to even more strongly encourage fitness improvement. The penalty that is applied when the fitness does not improve (-0.2) discourages the agent from wandering aimlessly or using stagnant parameters, by making the reward smaller with each loss of fitness.

The reward function is based on the idea of logarithmic fitness improvement. With this type of optimization problems, the fitness difference between steps might drop drastically at the beginning, and then slow down to a more fine-grained evolution when getting closer to a solution. This difference of orders of magnitude would teach the agent to make large jumps that do not correspond to the actual behavior of a metaheuristic. Applying a logarithm here brings the difference to much more manageable scale, allowing the agent to be able to control both the bigger steps at the start, as well as the smaller ones at the end. This type of reward formulation is consistent with prior work in reinforcement learning for optimization, where relative and logarithmic improvement signals are used to provide a more stable and scale-invariant learning objective [14].

2.4. Abstraction Layer and Zero-Shot Translation

The abstraction layer is the main contribution in this paper. The output of the RL model is always between [-1, 1]. Since not all metaheuristic parameters use this scale, the abstraction layer translates it depending on the metaheuristic that is being controlled. A_0 is the pressure and A1 is the Cohesion.

- Differential Evolution

Pressure → Mutation Factor (F): A piecewise linear mapping is used. Negative pressure maps to a micro-exploitation range (0.01 to 0.5), while positive pressure maps to an exploration range (0.5 to 1.9).

$$F = \begin{cases} 0.49 \cdot (A_0 + 1.0) + 0.01 & if\ A_0 < 0 \\ 1.4 \cdot A_0 + 0.5 & if\ A_0 \geq 0 \end{cases} \quad (2)$$

If the agent outputs a pressure value that is smaller than 0, it means that it wants to exploit the current area. The formula maps the value from 0.01 to 0.5, to ensure that the mutation factor stays low. On the other hand, if the pressure is positive, the range is mapped from 0.5 to 1.9, signaling a need to explore.

Cohesion → Crossover Rate (CR): Mapped linearly from 0.1 to 1.0.

$$CR = 0.45 \cdot A_1 + 0.55 \quad (3)$$

In the case of the cohesion, the logic is simpler. If the agent outputs a Cohesion of -1, then the CR becomes 0.1 (which means that the search is very independent, with little information sharing), while a Cohesion of 1 means a CR of 1, which denotes maximum information sharing.

- Particle Swarm Optimization

Pressure → Inertia Weight (w): Mapped linearly from 0.1 to 0.9.

$$w = 0.4 \cdot A_0 + 0.5 \quad (4)$$

Similarly to the logic implemented for DE, a Pressure value of -1 means an inertia of 0.1 and a value of 1 meaning an inertia of 0.9. A small pressure indicates a need to exploit, while a big one encourages exploration.

Cohesion → Social/Cognitive Ratio (c_1, c_2): Total acceleration is maintained at 4.0. High cohesion minimizes cognitive pull (c_1) and maximize social pull (c_2).

$$\begin{aligned} c_1 &= 4.0 \cdot (-0.4 \cdot A_1 + 0.5) \\ c_2 &= 4.0 - c_1 \end{aligned} \quad (5)$$

The RL agent outputs a Cohesion value between -1 and 1. In PSO, c_1 represents how much a particle trusts its own memory (personal best), and c_2 represents how much it trusts the swarm's collective memory (global best).

A Cohesion of -1 indicates a high independence between the particles and maps to c_1 = 3.6 and c_2 = 0.4, while a value of 1 indicates a high sociality and maps to c_1 = 0.4 and c_2 = 3.6.

- Genetic Algorithms

Pressure → Polynomial Mutation Index (η_m): Mapped inversely from 50.0 to 1.0. High pressure yields a low η_m, which injects high variance (chaos) into the population.

$$\eta m = -24.5 \cdot A0 + 25.5 \quad (6)$$

Pressure value of -1 means a mutation index of 50.0 and a value of 1 meaning a mutation index of 1.0. A small pressure indicates a need to exploit (by restricting mutations closely to the parent), while a big one encourages exploration (by injecting high variance into the population).

Cohesion → SBX Crossover Probability (p_c): Mapped linearly from 0.1 to 1.0. High cohesion results in frequent mating.

$$p_c = 0.45 \cdot A_1 + 0.55 \quad (7)$$

The RL agent outputs a Cohesion value between -1 and 1. In GA, p_c represents the probability that two selected parents will mate and share their genetic information. A Cohesion of -1 indicates a high independence between the individuals and maps to p_c = 0.1 (meaning parents mostly clone themselves), while a value of 1 indicates a high sociality and maps to p_c = 1.0 (meaning parents always mate and combine their genes).

3. Experimental Results

The evaluation of the performance of the RL controller was made by using six benchmark functions, three that were used for training and three that were unseen by the algorithm at that point. This was done to confirm that the model learned some control principles, not just how to solve a specific equation. The selection of the benchmark functions was aided by the paper [15], which describes the behavior of 175 functions, as well as the website [16], which also helps with the visualization. The used benchmarks were:

1. Sphere (Seen): A unimodal and convex landscape.
2. Rastrigin (Seen): A multimodal and deceptive landscape.
3. Rosenbrock (Seen): A non-convex landscape with a narrow and curved valley.
4. Ackley (Unseen): An almost flat outer region, with deep central hole.
5. Schwefel (Unseen): A highly deceptive landscape, with deep local minima that are far from the global optimum.
6. Griewank (Unseen): A highly multimodal landscape with heavy variable linkage.

All experiments across all algorithms share the same computational budget and configuration. The following configuration was used:

1) Dimensionality: 10 dimensions
2) Population Size: 100 individuals
3) Maximum Generations: 500 generations
4) Target Fitness: If a run reached a precision threshold of 10ˆ-6, it was terminated
5) Independent Runs: 30 runs per configuration

- Hardware Configuration

The hardware specifications:

1. CPU: Processor 11th Gen Intel(R) Core (TM) i5-1155G7 @ 2.50GHz, 2496 Mhz, 4 Core(s), 8 Logical Processor(s)
2. RAM: 16GB DDR4
3. Storage: 500 GB SSD

- Software Configuration

The following software versions were used for the implementation and the experiments:

1. Operating System: Microsoft Windows 11 Home
2. Python: Python 3.10.11
3. Libraries
   a. pymoo 0.6.1.6
   b. numpy 2.2.6
   c. stable_baselines 2.7.1
   d. gymnasium 1.2.3
   e. pyade-python 1.1

- Static Parameter Baselines

The fixed parameters used for testing each benchmark were not the default ones, but were optimized for each case in particular. Since each function has a specific landscape, certain combinations of parameters are favorable. The used parameters are highlighted in Table 1, Table 2 and Table 3.

Table 1. DE Static Parameters

Benchmark Function	Scaling Factor (F)	Crossover Rate (CR)
Sphere	0.5	0.9
Rastrigin	0.5	0.2
Rosenbrock	0.8	0.9
Ackley	0.5	0.9
Schwefel	0.9	0.9
Griewank	0.5	0.2

For Differential Evolution (Table 1), a very high Crossover Rate (CR = 0.9) is used for functions with a smoother landscape, like Sphere and Ackley to help the algorithm share information rapidly and converge as fast as possible, while a standard Mutation Factor (F = 0.5) keeps the step sizes balanced.

Because Rastrigin and Griewank are "separable" (meaning the dimensions can be solved somewhat independently) but have many local traps, the crossover rate is set to much smaller 0.2. This keeps the algorithm from mixing up the variables too much, letting it optimize different parts of the problem independently. Rosenbrock is shaped like a narrow, curved, flat valley. In this case, information sharing is kept high (CR = 0.9), but the mutation factor is increased to 0.8 to give the population the larger jump sizes it requires to be able to navigate the curve of the valley without getting stuck. Finally, Schwefel is highly deceptive with its best solutions hidden far away in the corners of the search space. To combat this, a big scaling factor (F = 0.9) is used so the population takes bigger leaps to escape the deep local traps.

Table 2. PSO Static Parameters

Benchmark Function	**Inertia Weight (w)**	**Cognitive Coeff. (c1)**	**Social Coeff. (c2)**
Sphere	0.700	2.000	2.000
Rastrigin	0.400	2.000	2.000
Rosenbrock	0.729	1.494	1.494
Ackley	0.729	1.494	1.494
Schwefel	0.900	2.000	2.000
Griewank	0.500	1.500	1.500

For Particle Swarm Optimization (Table 2), standard settings (w = 0.7, c1 = 2.0, c2 = 2.0) are used for the Sphere function, meaning the particles have moderate momentum and trust their own memory at the same rate as the swarm's memory. Rastrigin is bumpy, so the inertia weight is lowered to 0.4 so the particles lose momentum quickly. Once they find a good local minima, they stop and search it rather than flying past it. For Rosenbrock and Ackley, the standard constriction coefficients (w = 0.729, c1 = 1.494, c2 = 1.494) [17] are used because they mathematically guarantee that the swarm will stay stable and converge smoothly. For the Schwefel function, since the global optimum is so far away from the other local traps, the inertia weight is pushed to 0.9. This high momentum forces the particles to fly fast and go over local optima instead of getting stuck early on. In the case of Griewank both the momentum (0.5) and the social/cognitive pulls (1.5) are lowered to make the swarm move more carefully and exploit the area without moving around aimlessly.

Table 3. GA Static Parameters

Benchmark Function	Crossover Prob. (pc)	Mutation Prob. (pm)
Sphere	0.9	0.05
Rastrigin	0.9	0.20
Rosenbrock	0.8	0.10
Ackley	0.9	0.10
Schwefel	0.8	0.20
Griewank	0.9	0.10

For Genetic Algorithms (Table 3), a high mating probability (pc = 0.9) is used to combine good traits, and a very low mutation rate (pm = 0.05) is used because there is no need to inject much chaos to find the optimal solution, in the case of the Sphere function. Ackley and Griewank have a bit more complexity, so the mutation rate is increased to 0.10 to ensure the population does not get stuck on the minor bumps before converging into the global optima. For Rosenbrock, the crossover probability is reduced to $p_c = 0.8$, which helps preserve useful solutions and prevents them from being disrupted by excessive recombination while navigating the narrow curved valley. For Rastrigin and Schwefel the mutation rate is significantly increased to $p_m = 0.20$, introducing a constant level of randomness that helps the population avoid converging prematurely and to escape local optima.

3.1. L-SHADE Setup

The experiments also include L-SHADE (Success-History Based Adaptive Differential Evolution with Linear Population Size Reduction), a widely recognized and competitive variant of Differential Evolution for continuous optimization. L-SHADE uses adaptive parameter control by keeping a list of historical memories of successful mutation and crossover parameters, as well as a strategy of linear population reduction that improves convergence over time

For consistency, L-SHADE was initialized using the same population size and termination criteria as the other solvers in the experimental setup.

The implementation was based on the PyADE library, which provides an implementation of the L-SHADE variant of DE.

4. Discussion of Results

Overall, the RL controller achieved its most robust and consistent performance when tuning its native training algorithm, Differential Evolution. The zero-shot transfers to Particle Swarm Optimization and Genetic Algorithms yielded polarized but relevant results. The AI controller excelled at navigating complex and deceptive landscapes but struggled to exploit simpler landscapes (Sphere and Ackley) compared to static parameters

The variance bands in the convergence graphs reveal that the RL driven approach generally showed a wider variance between independent runs.

4.1. Performance on Differential Evolution

Table 4 presents the experimental data obtained with the DE algorithm for the set of six benchmark functions used in our experiments. Figure 2 graphically represents the performance of the DE algorithm, the interpretation of the results being made in the following.

The convergence results for Differential Evolution are depicted in Figure 2. On the simpler, globally structured functions (Sphere and Ackley), the AI-Controller's tendency to maintain high population diversity made the exploitation unnecessarily slow. Both the static baseline and L-SHADE aggressively exploited to achieve a perfect median and worst fitness of 1.00e-06, but the AI-Controller exhibited much higher variance. Although its median fitness on the Sphere function successfully reached the 1.00e-06 target, its tendency to explore too much resulted in a worst-case error of 7.40e+00 and a standard deviation of 2.01e+00.

Table 4. DE Results

Benchmark	Method	Median Fitness	Worst Fitness	Std Deviation
Sphere	Static DE	1.00e-06	1.00e-06	2.12e-22
Sphere	L-SHADE	1.00e-06	1.00e-06	2.12e-22
Sphere	AI Agent	1.00e-06	7.40e+00	2.01e+00
Rastrigin	Static DE	3.48e-05	8.86e-04	2.46e-04
Rastrigin	L-SHADE	1.00e-06	1.00e-06	2.12e-22
Rastrigin	AI Agent	1.00e-06	5.45e+00	1.51e+00
Rosenbrock	Static DE	2.58e+00	7.23e+00	2.34e+00
Rosenbrock	L-SHADE	1.00e-06	1.00e-06	2.12e-22
Rosenbrock	AI Agent	2.79e-06	7.17e+00	1.88e+00
Ackley	Static DE	1.00e-06	8.38e-03	1.50e-03
Ackley	L-SHADE	1.00e-06	1.00e-06	2.12e-22
Ackley	AI Agent	1.00e-06	5.39e+00	1.45e+00
Schwefel	Static DE	4.74e+02	7.12e+02	1.46e+02
Schwefel	L-SHADE	2.08e+03	4.15e+03	1.04e+03
Schwefel	AI Agent	1.00e-06	5.61e+00	1.70e+00
Griewank	Static DE	6.95e-02	1.18e-01	2.84e-02
Griewank	L-SHADE	1.00e-06	1.00e-06	2.12e-22
Griewank	AI Agent	1.00e-06	6.22e+00	1.69e+00

The advantage of this exploratory behavior is proven on the highly multimodal and deceptive landscapes, such as Rosenbrock, where the static DE baseline showed premature convergence, permanently stagnating at a bad median fitness of 2.58e+00.

The RL agent successfully controlled pressure and cohesion to push the population through the local traps, which improved the median solution down to 2.79e-06.

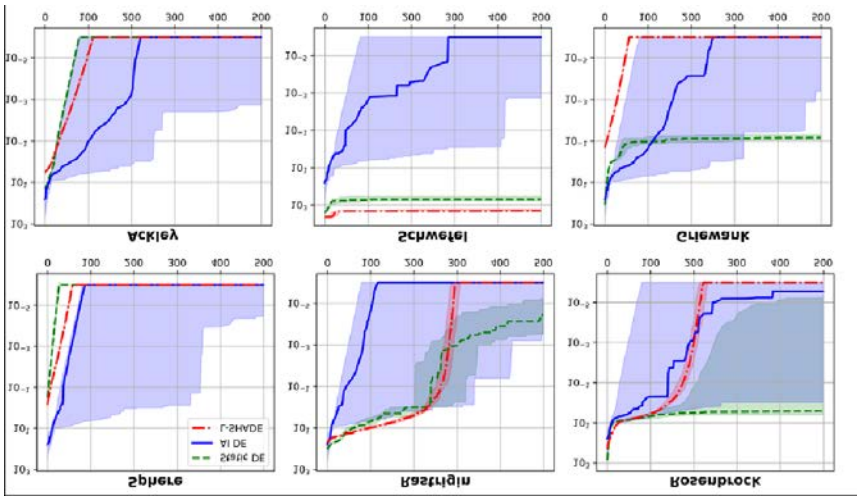

Figure 2. Graphs Depicting DE Performance

The most important result occurs on the Schwefel function, which is very deceptive, with its optimal solutions hidden at the very far edges of the search space. In this case, both the static parameters and L-SHADE failed, recording median fitness values of 4.74e+02 and 2.08e+03 respectively. The RL agent did better, achieving an ideal median fitness of 1.00e-06, with a worst-case error of 5.61e+00.

4.2. Performance on Particle Swarm Optimization - Zero-Shot Transfer

Table 5 presents the experimental data obtained with the PSO algorithm for the set of six benchmark functions used in our experiments. Figure 3 graphically represents the performance of the DE algorithm, the interpretation of the results being made in the following.

Table 5. PSO Results

Benchmark	Method	Median Fitness	Worst Fitness	Std Deviation
Sphere	Static PSO	1.00e-06	1.00e-06	2.12e-22
Sphere	AI Agent	5.98e-01	6.71e+00	2.58e+00
Rastrigin	Static PSO	1.99e+00	5.49e+00	1.10e+00
Rastrigin	AI Agent	1.29e+00	7.02e+00	2.85e+00
Rosenbrock	Static PSO	5.21e+00	9.54e+00	1.81e+00
Rosenbrock	AI Agent	6.76e-01	6.50e+00	2.67e+00
Ackley	Static PSO	1.00e-06	1.00e-06	2.12e-22
Ackley	AI Agent	5.59e-01	8.36e+00	2.98e+00
Schwefel	Static PSO	7.80e+02	1.24e+03	2.25e+02
Schwefel	AI Agent	6.71e-01	8.83e+00	2.77e+00
Griewank	Static PSO	6.88e-02	5.47e-01	9.18e-02
Griewank	AI Agent	1.89e-01	6.71e+00	2.82e+00

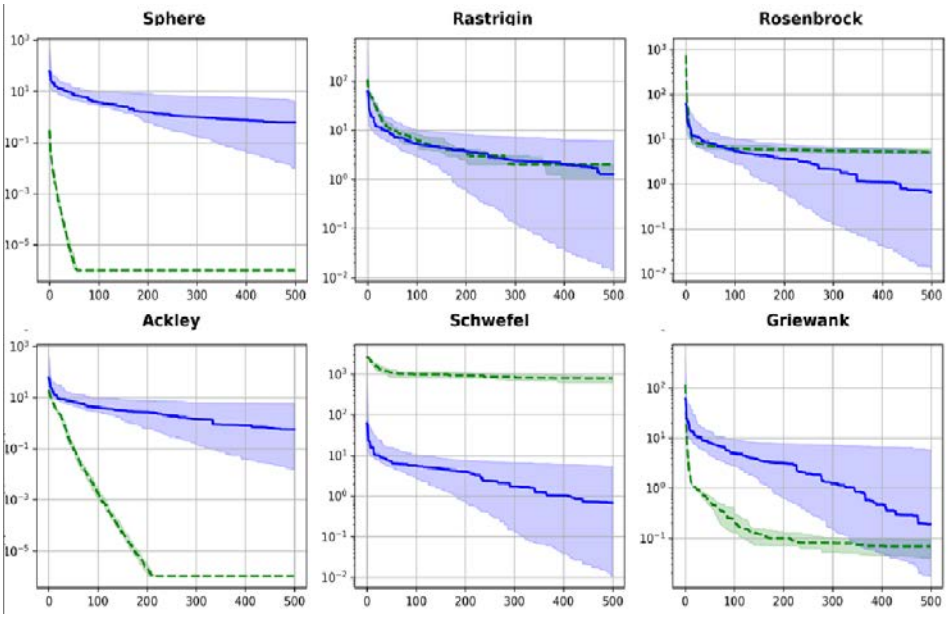

Figure 3. Graphs Depicting PSO Performance

The comparative results for Particle Swarm Optimization shown in Figure 3, show the zero-shot transfer potential. It also highlights some weaknesses of this approach.

The primary vulnerability of standard PSO is a rapid collapse of the swarm, where the particles converge onto local minima too quickly and permanently lose the momentum required to escape said minima. The static parameter baseline showed this flaw on the Schwefel function, where it permanently got stuck at a median error of 7.80e+02 and a worst-case error of 1.24e+03. The RL agent managed to correctly control the solver so that it would not get stuck in these local minima, without being trained on this specific algorithm. It had a much smaller median fitness of 6.71e-01 and a worst fitness at 8.83e+00.

On the other hand, the RL agent's performance on less complex functions was worse. On the Sphere function, the static baseline easily found the global optimum (median and worst fitness of 1.00e-06), while the RL-controlled PSO failed to properly exploit and it finished with a much higher median fitness of 5.98e-01 and a standard deviation of 2.58e+00. This shows that the RL agent makes a priority of trying to avoid getting stuck in local minima, which gets punished in a problem with little to no such minima.

4.3. Performance on Genetic Algorithms - Zero-Shot Transfer

Table 6 presents the experimental data obtained with the GA algorithm for the set of six benchmark functions used in our experiments. Figure 4 graphically represents the performance of the DE algorithm, the interpretation of the results being made in the following.

A similar comparative advantage and trade-off are observed when transferring the policy to the Genetic Algorithm, as depicted in Figure 4.

On Rosenbrock, the static parameters plateaued at a median fitness of 6.00e+00 with a worst-case error of 8.68e+00. These sharp stagnation points show a fast loss of genetic diversity. Once the population becomes homogenous the algorithm is mathematically incapable of generating the new offspring required to escape local minima. The RL agent resolved this genetic stagnation. It outperformed the static configurations on Rosenbrock, improving the median fitness to 4.97e-03. This improvement was also visible on Griewank, where the RL agent achieved a median fitness of 5.00e-03 compared to the static baseline's 5.61e-02.

On the Schwefel function, the RL agent successfully stopped catastrophic failures from occurring, with a worst-case error to 6.31e+00, while the static baseline had some bigger failures, with a worst-case of 2.37e+02.

On the simple Sphere function, the static parameters achieved a perfect median fitness of 1.00e-06. In contrast, the RL agent fell behind with a median of 4.61e-03 and a high standard deviation of 2.72e+00. This confirms that the RL struggles to quickly zoom in on the exact best answer when the problem is straightforward and lacks traps.

Table 6. GA Results

Benchmark	Method	Median Fitness	Worst Fitness	Std Deviation
Sphere	Static PSO	1.00e-06	1.00e-06	2.12e-22
Sphere	AI Agent	5.98e-01	6.71e+00	2.58e+00
Rastrigin	Static PSO	1.99e+00	5.49e+00	1.10e+00
Rastrigin	AI Agent	1.29e+00	7.02e+00	2.85e+00
Rosenbrock	Static PSO	5.21e+00	9.54e+00	1.81e+00
Rosenbrock	AI Agent	6.76e-01	6.50e+00	2.67e+00
Ackley	Static PSO	1.00e-06	1.00e-06	2.12e-22
Ackley	AI Agent	5.59e-01	8.36e+00	2.98e+00
Schwefel	Static PSO	7.80e+02	1.24e+03	2.25e+02
Schwefel	AI Agent	6.71e-01	8.83e+00	2.77e+00
Griewank	Static PSO	6.88e-02	5.47e-01	9.18e-02
Griewank	AI Agent	1.89e-01	6.71e+00	2.82e+00

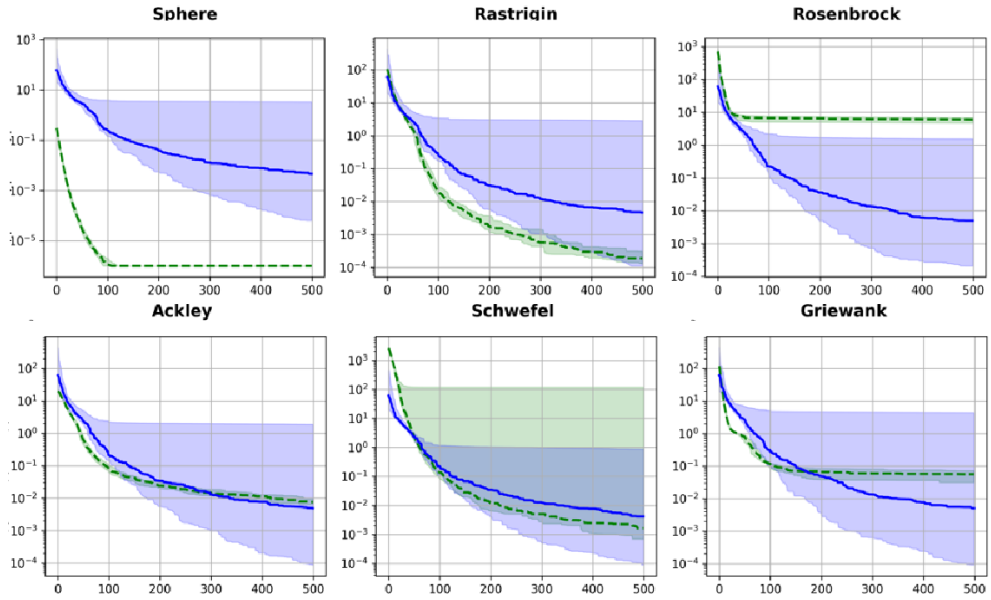

Figure 4. Graphs Depicting GA Performance

5. Conclusion

The primary objective of this paper was to investigate whether hyperparameter control in population-based metaheuristics can be abstracted and learned by an algorithm-agnostic agent, for population-based metaheuristics.

Abstracting raw parameters into two control primitives (Pressure and Cohesion), allowed a PPO agent to independently control the behavior of the metaheuristics. During its training on DE, the agent learned to avoid premature convergence on complex landscapes such as the Rastrigin and Schwefel functions.

A second important result was that the learned policy showed zero-shot transfer to PSO and GA. The model was able to control the behavior of PSO and GA without any retraining. This suggests that at least part of metaheuristic behavior can be captured through a shared control representation.

Future Work: The first areas of expansion will be adding state-of-the-art algorithms as benchmarks for PSO and GA. Benchmarking the AI-Controlled PSO against algorithms like EPSO (Evolutionary Particle Swarm Optimization) or HPSO (Hierarchical PSO) and comparing the AI-Controlled GA against modern Adaptive Genetic Algorithms (AGA) will help with understanding the trade-offs between universality and specialization.

Another area that can be expanded is the algorithm selection. While using PSO and GA for proving zero-shot transfer was enough, it is not enough to claim universality. The performance of this controller was acceptable on the two aforementioned models, but testing on multiple other metaheuristics can strengthen the universality claim. Other algorithms that can be used are Ant Colony Optimization (ACO), Artificial Bee Colony (ABC) and Evolution Strategies (ES).

References:

[1]. R. Narayanan and N. Ganesh, "A Comprehensive Review of Metaheuristics for Hyperparameter Optimization in Machine Learning," in *Metaheuristics for Machine Learning: Algorithms and Applications*, 2024, pp. 37–72. doi: 10.1002/9781394233953.ch2.

[2]. K. Aguerchi, Y. Jabrane, M. Habba, and A. H. El Hassani, "A CNN Hyperparameters Optimization Based on Particle Swarm Optimization for Mammography Breast Cancer Classification," *Journal of Imaging*, vol. 10, no. 2, p. 30, 2024. doi: 10.3390/jimaging10020030.

[3]. R. Al-Nahhal, A. F. Naiem, and Y. G. Hegazy, "Economic Load Dispatch Problem Using Particle Swarm Optimization Technique Considering Wind Power Penetration," in *Proc. 2019 Int. Conf. Smart Energy Systems and Technologies (SEST)*, Porto, Portugal, Sep. 2019, pp. 1–6. doi: 10.1109/SEST.2019.8849005.

[4]. N. D. Jana and J. Sil, "Protein structure prediction in 2D HP lattice model using differential evolutionary algorithm," in *Proceedings of the International Conference on Information Systems Design and Intelligent Applications (INDIA 2012)*, Springer, 2012, pp. 281–290. doi: 10.1007/978-3-642-27443-5_32.

[5]. N. Bacanin, C. Stoean, M. Zivkovic, M. Rakic, R. Strulak-Wójcikiewicz, and R. Stoean, "On the Benefits of Using Metaheuristics in the Hyperparameter Tuning of Deep Learning Models for Energy Load Forecasting," *Energies*, vol. 16, no. 3, p. 1434, 2023. doi: 10.3390/en16031434.

[6]. B. Crawford, C. Valenzuela, R. Soto, E. Monfroy, and F. Paredes, "Parameter Tuning of Metaheuristics Using Metaheuristics," *Advances in Science Letters*, vol. 19, no. 12, pp. 3556–3559, 2013.

[7]. R. Tanabe and A. Fukunaga, "Success-History Based Parameter Adaptation for Differential Evolution," in *Proc. 2013 IEEE Congress on Evolutionary Computation (CEC)*, Cancun, Mexico, 2013, pp. 71–78. doi: 10.1109/CEC.2013.6557555.

[8]. R. Tanabe and A. S. Fukunaga, "Improving the Search Performance of SHADE Using Linear Population Size Reduction," in *Proc. 2014 IEEE Congress on Evolutionary Computation (CEC)*. doi: 10.1109/CEC.2014.6900380.

[9]. A. Nickabadi, M. M. Ebadzadeh, and R. Safabakhsh, "A Novel Particle Swarm Optimization Algorithm with Adaptive Inertia Weight," *Applied Soft Computing*, vol. 11, no. 4, pp. 3658–3670, 2011. doi: 10.1016/j.asoc.2011.01.037.

[10]. R. Durgut, M. E. Aydin, and I. Atli, "Adaptive Operator Selection with Reinforcement Learning," *Information Sciences*, vol. 581, pp. 773–790, Dec. 2021. doi: 10.1016/j.ins.2021.10.025.

[11]. A. Rost, I. Petrova, and A. Buzdalova, "Adaptive Parameter Selection in Evolutionary Algorithms by Reinforcement Learning with Dynamic Discretization of Parameter Range," in *Proc. Genetic and Evolutionary Computation Conference Companion (GECCO Companion)*, 2016, pp. 141–142. doi: 10.1145/2908961.2908998.

[12]. M. L. Puterman, "Markov Decision Processes," in *Handbooks in Operations Research and Management Science*, vol. 2, Elsevier, 1990, pp. 331–434.

[13]. M. Khambhayata, S. Singh, N. Bala, and A. Bhattacharyya, "Mastering Super Mario Bros.: Overcoming Implementation Challenges in Reinforcement Learning with Stable-Baselines3," in *Proc. 2024 International Conference on Advances in Computing Research on Science Engineering and Technology (ACROSET)*, Sep. 2024, pp. 1–7. doi: 10.1109/ACROSET62108.2024.10743683.

[14]. H. van Seijen, M. Fatemi, and A. Tavakoli, "Using a Logarithmic Mapping to Enable Lower Discount Factors in Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[15]. M. Jamil and X.-S. Yang, "A Literature Survey of Benchmark Functions for Global Optimisation Problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013. doi: 10.1504/IJMMNO.2013.055204.

[16]. S. Surjanovic and D. Bingham, "Optimization Test Problems," Simon Fraser University, Burnaby, BC, Canada, 2013. Available: https://www.sfu.ca/~ssurjano/optimization.html. Accessed: Apr. 25, 2026.

[17]. M. Clerc and J. Kennedy, "The Particle Swarm—Explosion, Stability, and Convergence in a Multidimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002. doi: 10.1109/4235.985692.