Journal of Artificial Intelligence and Autonomous Intelligence Received 26-08-2025; Accepted 2-12-2025; Published 9-12-2025

Whether, How and When Modern Evolutionary Strategies Can Be Improved

Paolo Pagliuca paolo.pagliuca@istc.cnr.it
Institute of Cognitive Sciences and Technologies (ISTC)
National Research Council (CNR)
Via Giandomenico Romagnosi 18A, 00196, Roma, Italy

Corresponding Author: Paolo Pagliuca

Abstract

Recent advances in the development of effective Evolutionary Strategies (ESs) have paved the way to their usage in a broad variety of problems. Among the others, the OpenAI Evolutionary Strategy (OpenAI-ES) has emerged as a notable method to discover solutions for robotic problems like locomotion and aggregation, or for classic tasks like pole balancing. The main advantage of OpenAI-ES over its counterparts is the usage of momentum vectors storing information about the relationship between parameter variations and performance, with no need for computational demanding covariance matrices. However, previous studies show that OpenAI-ES may achieve low performance in rather complex scenarios or when the function used to evaluate performance is not designed for ESs. This makes us hypothesize that there is room for enhancements of OpenAI-ES. This work delves into the design and analysis of three variants of OpenAI-ES and performs a thorough comparison of OpenAI-ES, its variants and the Stochastic Steady State with Hill Climbing (SSSHC) on a broad set of problems, including several benchmarks from the literature. Our results prove that the variants introduced in this work are competitive with OpenAI-ES in many cases. Moreover, SSSHC outperforms the other methods in most of the considered problems, while OpenAI-ES excels in robot locomotion. This work contributes to shed the light on both the pitfalls OpenAI-ES can encounter and new perspectives to further improve it.

Keywords: Evolutionary Strategies, OpenAI-ES, SSSHC, Benchmarking

1. INTRODUCTION

Evolutionary Strategies (ESs) [1, 2] are search techniques belonging to the broad family of Evolutionary Algorithms (EAs) [3]. Their operation draws inspiration from biological evolution: a population of individuals, sometimes referred as “genotypes”, is randomly initialized and evolved according to the Darwin’s theory [4]. Individuals represent possible solutions to a given problem and are generally defined as vectors of bit, integer or floating-point values called “genes”, although more sophisticated encoding techniques can be used [5–9]. Evolution is typically organized as a sequence of iterations, also called generations. At each iteration, individuals are evaluated and

Citation: Paolo Pagliuca Whether, How and When Modern Evolutionary Strategies Can Be Improved. Journal of Artificial Intelligence and Autonomous Intelligence ; Research 2025;2(2):25.

Page No.- 425-449.

receive a value, defined as fitness or performance, assessing their capability to solve the considered problem. Biological mechanisms like selection, random mutations and crossover are then used to update the individuals and create the new population. Evolution stops when a termination criterion is met (e.g., the total number of iterations or a specific level of performance has been reached). A schematic is provided in FIGURE 1.

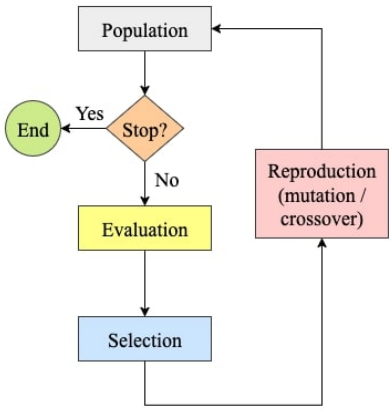


Figure 1: Illustration of the operation of an ES: a population is evolved until a stopping condition occurs. At each iteration, the population is evaluated. Selection takes place by identifying the best individuals. Finally, population is updated through mutation and/or crossover operators.

Over the last decades, we observed a shift from the biological principles inspiring early versions of ESs to more elaborate methods. For example, the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [10] computes a covariance matrix storing the mutual relationship among genes/parameters. The matrix is used to drive the search process towards the directions in the search space that should correspond to higher performance. CMA-ES proved successful in domains like pole balancing [11, 12], test functions [13], robot locomotion [14] and swarm robotics [12, 15, 16]. A similar method is the Exponential Natural Evolution Strategies (xNES) [17, 18], which has found application in a broad set of problems, such as robot locomotion [14], robot navigation [19], pole balancing [12, 18], test functions [17, 18] and swarm robotics [12, 15, 16]. Despite the good performance, both CMA-ES and xNES are computationally demanding, since storing and updating the covariance matrix is expensive and can become unfeasible when the number of genes/parameters is high [14, 20]. The Separable Natural Evolution Strategies (sNES) [18, 21] collects gene/parameter variances separately (i.e., a vector variance is stored) and uses them to drive the search towards the expected most promising regions of the search space. A major advantage is the significant computational cost reduction, since no covariance matrix has to be stored. Furthermore, the usage of a vector variance allows sNES to be applied to high dimensional problems [18, 21]. The sNES algorithm has been used in tasks like test functions [21], pole balancing [12, 14, 21], robot locomotion [14] and Multi-Objective Optimization (MOO) [22]. A common trait of these techniques is the evolution of a single individual, often termed “centroid”, which is iteratively updated based on the aforedescribed procedures.

One of the most recent and successful ESs is the OpenAI Evolutionary Strategy (OpenAI-ES) [23], which demonstrated excellent capabilities of solving rather difficult problems like robot locomotion [14, 24], swarm robotics [16, 25, 26], pole balancing [14], Atari games [23], MOO [22, 27, 28] and competitive co-evolution [29]. Differently from the previously mentioned ESs, a key feature of OpenAI-ES is the usage of two momentum vectors storing historical information about the relationship between gene/parameter variations and performance. This enables the algorithm to discover the regions of the search space corresponding to higher expected fitness and drive the search process towards those regions. As for the previous algorithms, OpenAI-ES works by evolving a single individual/centroid: at each iteration, a pool of samples (i.e., parameter variations) is randomly extracted from a normal distribution. To capture the actual effect of the variation, each sample is either added or subtracted and a pair of symmetric samples is derived [30]. All the individuals are evaluated and receive a fitness score. A ranking of the samples (guided by fitness) is performed to compute the gradient of the expected fitness. Lastly, the Adam optimizer [31] is employed to update the two momentum vectors and the centroid. An illustration of the OpenAI-ES algorithm is provided in FIGURE 2.

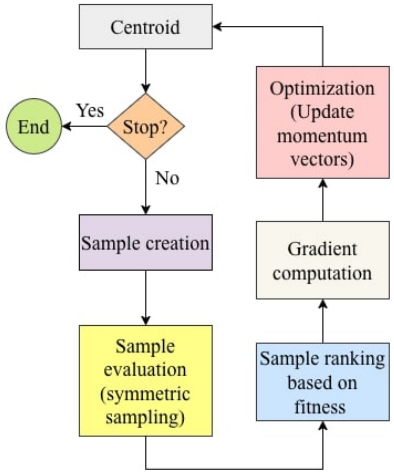


Figure 2: Schematic of OpenAI-ES: the centroid is evolved until a termination condition occurs. At each iteration, a set of samples is extracted from a normal distribution $\mathcal{N}(0, 1)$. Samples are either added to or subtracted from the centroid and these individuals are evaluated. Based on the obtained fitness, samples are ranked and are assigned weights to compute the fitness function gradient. Lastly, the two momentum vectors and the centroid are update through Adam.

Despite its success in a variety of problems (see also [32]), there are cases in which OpenAI-ES fails to reach a good performance. As pointed out in [14], the fitness function definition highly affects the chance of OpenAI-ES to discover effective solutions. In their study, the authors compared a Reinforcement Learning (RL) [33] algorithm like Proximal Policy Optimization (PPO) [34] and OpenAI-ES on a series of Pybullet locomotion problems [35]. The authors showed that OpenAI-ES falls short to PPO when the fitness function is specifically tailored for a RL algorithm, whereas the opposite is true when the fitness function is designed for an Evolutionary Algorithm (EA). In [27, 28], the authors tested the capability of OpenAI-ES to evolve solutions for a challenging

collective MOO scenario in which a group of 5 AntBullet robots [35] has to aggregate and locomote. The results achieved in both studies reveal that OpenAI-ES manages to address effectively the locomotion objective, but fails with respect to the aggregation objective. Moreover, a recent study demonstrated that a relatively simple EA like the Stochastic Steady State with Hill Climbing (SSSHC) [36] is competitive with the OpenAI-ES in a particular MOO scenario [22]. On one side, this implies that increasing the complexity of ESs does not necessarily translate into enhanced performance. On the other side, the aforementioned studies indicate that there is room for improving OpenAI-ES, particularly in those domains in which its performance is sub-optimal.

This works seeks to fill a gap in the improvement of a modern ES like OpenAI-ES by introducing three variants: (i) Population OpenAI-ES (PopOpenAI-ES), which evolves a population of centroids independently, with the possibility to replace at each iteration the worst performing centroid with the best one; (ii) Population OpenAI-ES with refinement (PopOpenAI-ES+HC), which combines PopOpenAI-ES with the Hill-Climbing (HC) algorithm [37] seeking to improve solutions through single-gene mutations as in SSSHC [22, 38], and (iii) OpenAI-ES with refinement (OpenAI-ES+HC), which combines OpenAI-ES and HC. Specifically, we compared OpenAI-ES and its three variants on a series of benchmark problems:

- a subset of 33 test functions derived from [39];
- the N-bit parity problem [40], with $N \in [5, 6, 7, 8]$;
- the standard and long-poles versions of the double pole balancing problem [14, 41, 42];
- the mountain car continuous problem [43, 44];
- the pendulum problem [43, 44];
- a grid navigation problem [44];
- a single-robot version of the foraging with poison problem [45];
- the MuJoCo pusher, reacher and swimmer problems [46];
- the Pybullet halfcheetah, hopper and walker2D problems [35];
- the recently introduced halfcheetah2D problem [47].

As a baseline method, we considered SSSHC, because it represents an effective technique in some of the considered domains [36, 38] and has proved competitive with OpenAI-ES in a MOO scenario [22].

Our outcomes demonstrate that all the proposed variants outperform or equal OpenAI-ES with respect to test functions, *N* bit-parity, double pole balancing, foraging and swimmer problems. Surprisingly, SSSHC is notably more effective than OpenAI-ES and its variants in a wide range of problems. Lastly, OpenAI-ES bests the other methods with regard to locomotion problems. Overall, this underscores that the considered variants are competitive in most of the cases, yet not as highly performing as OpenAI-ES in complex locomotion problems.

2. MATERIALS AND METHODS

In this section, we briefly review the tasks selected and we provide a thorough illustration of the experimental settings. As a simulation tool, we employed *evorobotpy3* [32].

2.1 Problems

The five algorithms (i.e., OpenAI-ES, PopOpenAI-ES, PopOpenAI-ES+HC, OpenAI-ES+HC and SSSHC) have been evaluated on 50 problems of different nature, ranging from test function optimization to robot locomotion. For each task, given a fitness function $F \in \mathbb{R}$, the objective is formulated in terms of function maximization (see Eq. 1):

$$\max F \tag{1}$$

Regarding test function optimization, function maximization is achieved by optimizing the values of the individual's genes (see Eqs. 2 - 3). Conversely, in the other problems, function maximization depends on the fitness function definitions reported in Eqs. 4 - 17.

2.1.1 Test functions

Test functions constitute widespread benchmark tasks to assess algorithmic performance [39, 48]. Given a vector x of length d, the general formulation of the problem is shown in Eq. 2:

$$\min F(x) \tag{2}$$

However, in order to be compliant with our problem definition, we considered the following formulation (see Eq. 3):

$$\max -F(x) \tag{3}$$

We used 33 test functions derived from [39], which constitute widespread problems in Evolutionary Computation (EC). In particular, we considered the functions reported in TABLE 1.

2.1.2 N bit-parity

The family of N-bit parity problems are another example of benchmark tasks largely employed in the literature [40, 49, 50]. Given as input a N-bit string, the task is to count the number of 1-bits and returning 1 when the sum is even. Because the number of N-bit strings is 2^N, the problem involves a full evaluation over all the possible inputs. Therefore, the fitness function F is defined by Eq. 4:

Table 1: List of test functions derived from [39]. Each function takes as input a vector x of length d. As concerns the Schewefel function, we set $\alpha = 5$.

Ackley $F = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + \exp(1)$	Alpine no 1 $F = \sum_{i=1}^{d} \lvert x_i sin(x_i) + 0.1x_i \rvert$	Chung-Reynolds $F = (\sum_{i=1}^{d} x_i^2)^2$
Csendes $F = \sum_{i=1}^{d} x_i^6 (2 + \sin\frac{1}{x_i})$	Deb 1 $F = -\frac{1}{d}\sum_{i=1}^{d} \sin^6(5\pi x_i)$	Dixon-Price $F = (x_1 - 1)^2 + \sum_{i=2}^{d} i(2x_i^2 - x_{i-1})^2$
Egg Holder $F = \sum_{i=1}^{d-1}(-(x_{i+} + 47)\sin(\sqrt{\lvert x_{i+1} + \frac{x_i}{2} + 47\rvert}) - x_i \sin(\sqrt{\lvert x_i - (x_{i+1} + 47)\rvert}))$	Exponential $F = -\exp(-0.5\sum_{i=1}^{d} x_i^2)$	Griewank $F = 1 + \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)$
Michalewicz $F = -\sum_{i=1}^{d} \sin(x_i)\sin^{2d}(\frac{ix_i^2}{\pi})$	Qing $F = \sum_{i=1}^{d}(x_i - i)^2$	Quintic $F = \sum_{i=1}^{d} \lvert x_i^5 - 3x_i^4 + 4x_i^3 + 2x_i^2 - 10x_i - 4\rvert$
Rana $F = \sum_{i=1}^{d-1}(x_{i+1} + 1)\cos(t_2)\sin(t_1) + x_i\cos(t_1)\sin(t_2)$ $t_1 = \sqrt{\lvert x_{i+1} + x_i + 1\rvert}$ $t_2 = \sqrt{\lvert x_{i+1} - x_i + 1\rvert}$	Rastrigin $F = \sum_{i=1}^{d}(x_i^2 - 10\cos(2\pi x_i) + 10)$	Rosenbrock $F = \sum_{i=1}^{d-1}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$
Salomon $F = 1 - \cos(2\pi\sqrt{\sum_{i=1}^{d} x_i^2}) + 0.1\sqrt{\sum_{i=1}^{d} x_i^2}$	Schaffer F6 $F = \sum_{i=1}^{d-1} 0.5 + \frac{\sin^2(\sqrt{x_i^2 + x_{i+1}^2}) - 0.5}{(1+0.001(x_i^2 + x_{i+1}^2))^2}$	Schwefel $F = (\sum_{i=1}^{d} x_i^2)^{\alpha}$
Schwefel 1.2 $F = \sum_{i=1}^{d}(\sum_{j=1}^{i} x_j)^2$	Schwefel 2.4 $F = \sum_{i=1}^{d}(x_i - 1)^2 + (x_1 - x_i^2)^2$	Schwefel 2.20 $F = -\sum_{i=1}^{d} \lvert x_i \rvert$
Schwefel 2.22 $F = \sum_{i=1}^{d} \lvert x_i \rvert + \prod_{i=1}^{d} \lvert x_i \rvert$	Schwefel 2.23 $F = \sum_{i=1}^{d} x_i^{10}$	Schwefel 2.26 $F = -\frac{1}{d}\sum_{i=1}^{d} x_i \sin(\sqrt{\lvert x_i \rvert})$
Shubert 3 $F = \sum_{i=1}^{d}\sum_{j=1}^{5} j\sin((j+1)x_i + j)$	Shubert 4 $F = \sum_{i=1}^{d}\sum_{j=1}^{5} j\cos((j+1)x_i + j)$	Sphere $F = \sum_{i=1}^{d} x_i^2$
Styblinski-Tang $F = \frac{1}{2}\sum_{i=1}^{d} x_i^4 - 16x_i^2 + 5x_i$	Sum squares $F = \sum_{i=1}^{d} ix_i^2$	Trid $F = \sum_{i=1}^{d}(x_i - 1)^2 - \sum_{i=2}^{d} x_i x_{i-1}$
Trigonometric 2 $F = 1 + \sum_{i=1}^{d} 8\sin^2(7(x_i - 0.9)^2) + 6\sin^2(14(x_i - 0.9)^2) + (x_i - 0.9)^2$	Whitley $F = \sum_{i}^{d}\sum_{j=1}^{d}(\frac{(100(x_i^2 - x_j)^2 + (1 - x_j)^2)^2}{4000} - cos(100(x_i^2 - x_j)^2 + (1 - x_j)^2)^2 + 1)$	Zakharov $F = \sum_{i=1}^{d} x_i^2 + (\frac{1}{2}\sum_{i=1}^{d} ix_i)^2 + (\frac{1}{2}\sum_{i=1}^{d} ix_i)^4$

$$F = [1 - \frac{1}{2^N}\sum_{i=1}^{2^N}(o_i - \hat{o}_i)] \tag{4}$$

where o_i is the expected output for the $i - th$ bit string and $\hat{o}_i$ represents the output.

In this work, we considered the N-bit parity problem with $N \in [5, 6, 7, 8]$, which constitutes a challenging task [40].

2.1.3 Double pole balancing

Pole balancing [42] is a widely recognized benchmark task in EC [7, 11, 18, 41]. The problem consists in controlling a system in which a cart has to maintain two poles upright. The cart can move only horizontally (see FIGURE 3).

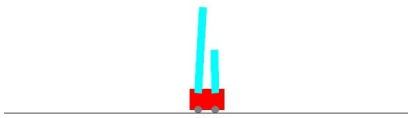


Figure 3: Double pole balancing problem: a cart (displayed in red) has to maintain two poles (displayed in light blue) upright by moving horizontally on a flat surface.

The goal is to maintain the poles upright by moving the cart within the allowed track (black line in FIGURE 3). Eq. 5 illustrates the computation of the fitness function F.

$$F = \frac{1}{N_E} \sum_{e=1}^{N_E} N_S(e) \tag{5}$$

The symbols N_E and $N_S(e)$ denote, respectively, the number of episodes and the number of steps the cart manages to maintain the poles upright without leaving the track during the $e-th$ episode. Episodes might end prematurely if either the cart goes out of the track or when at least one pole falls.

In this work we considered both the standard version of the problem [42] and the long-poles version [41, 42]. The latter problem is remarkably more challenging than the former due to the different ratio between pole dimensions (respectively $\frac{1}{2}$ and $\frac{1}{10}$). Furthermore, we adopted the "fixed initial states condition" initialization introduced in [41] (see also [22, 36]).

2.1.4 Mountain car continuous

The mountain car continuous problem [43] is a benchmark task in RL [51, 52]: a car starts from a resting point in a sinusoidal valley and has to reach a target location at the top (see FIGURE 4). In the version included in Gymnasium [44], the fitness function F is computed according to Eq. 6:

$$F = \frac{1}{N_E} \sum_{e=1}^{N_E} [B(e) - 0.1 \times a(e)^2] \tag{6}$$

$$B(e) = \begin{cases} 100 & \text{if target is reached} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

In Eq. 6, $B(e)$ is a bonus for reaching the target location at the episode e, whereas $a(e)$ denotes the force applied to move the car during the $e-th$ episode.

2.1.5 Pendulum

The pendulum problem takes inspiration from the classic task in control theory. It is characterized by low dimensionality, but high non-linearity [53]. Specifically, a pendulum has one end attached to a fixed joint, while the other end is free to move (see FIGURE 5). The pendulum is initially placed in a random position. The objective is to apply a torque making the pendulum swing, ultimately reaching an upright configuration. Eq. 8 provides the definition of the fitness function F:

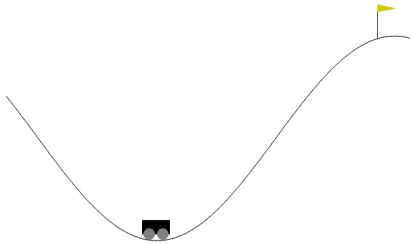


Figure 4: Mountain car continuous problem: a car (black rectangle) must reach a target location (marked with a flag) by climbing a hill.

$$F = -\frac{1}{N_E}\sum_{e=1}^{N_E}[norm(\theta(e))^2 - 0.1 * \dot{\theta}(e)^2 - 0.001 * a(e)^2] \tag{8}$$

where e indicates the evaluation episode $\theta(e)$ is the pendulum angle in radians, $\dot{\theta}(e)$ denotes the angular velocity and $a(e)$ represents the applied torque. The function $norm()$ converts the angle $\theta(e)$ into a value bounded in the range $[-\pi, \pi)$.

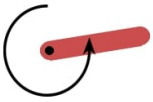


Figure 5: Pendulum task: the pendulum starts from a random position and must reach an upright configuration.

2.1.6 Grid navigation

The grid navigation problem is a commonly used task in EC literature [22, 54, 55], which involves an agent that has to navigate in a grid environment and reach a target location (see FIGURE 6). The problem is formulated according to Eq. 9:

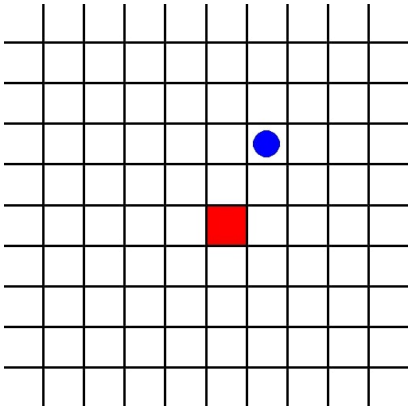


Figure 6: Grid navigation task: an agent (displayed in blue) must reach a target location (displayed in red). The agent can perform one out of four possible actions: (i) left; (ii) top; (iii) right; (iv) bottom.

$$F = \frac{1}{N_E} \sum_{e=1}^{N_E} [B(e) - P(e)] \tag{9}$$

$$B(e) = \begin{cases} 1 & \text{if target reached} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$P(e) = \begin{cases} 1 & \text{if agent out of grid} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

As shown in Eq. 9, the fitness function F is the difference between the bonus for reaching the target $B(e)$ and the penalty for leaving the grid $P(e)$, and e is the evaluation episode.

2.1.7 Foraging with poison

The foraging with poison problem typically involves a group of robots that must forage cooperatively by avoiding poisonous items [56–58]. In this work, we considered a single-agent version [45] in which a robot has to forage by avoiding food items. A snapshot of the task is presented in FIGURE 7: the environment is filled with 10 objects equally split between food sources and poisonous items. The objective is to forage as many food sources as possible by avoiding poisonous items. The fitness function F is calculated according to Eq. 12:

$$F = \frac{1}{N_E} \sum_{e=1}^{N_E} [N_F(e) - N_P(e)] \tag{12}$$

where $N_F(e)$ and $N_P(e)$ indicate, respectively, how many food sources and poisonous items the robot eats, while e denotes the evaluation episode.

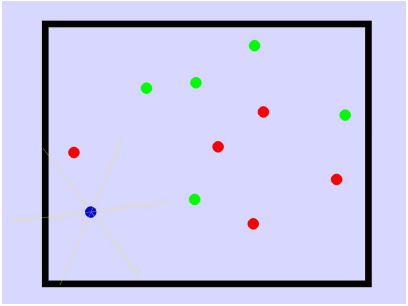


Figure 7: Foraging with poison problem: an agent (blue circle) has to forage food (green circles) and avoid poisonous items (red circles).

2.1.8 MuJoCo problems

MuJoCo [46] is a popular simulation engine containing several environments including robot locomotion, robot manipulation and control problems. A detailed description of the environments is provided at `https://gymnasium.farama.org/environments/mujoco/`. In particular, we selected three widespread problems: pusher, reacher and swimmer.

Pusher

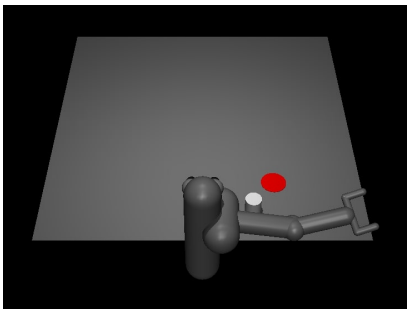


Figure 8: Pusher task: a robotic arm with two fingertips must take an object (displayed in white) and place it in a target location (displayed in red).

The pusher problem consists in a robotic arm with two fingertips that has to take an object and place it in a target location (see FIGURE 8). The fitness function F is expressed as in Eq. 13:

$$F = -\frac{1}{N_E}\sum_{e=1}^{N_E}[\|\overrightarrow{x}(e) - \overrightarrow{g}(e)\| + 0.1 \times a(e)^2 + 0.5 \times -\|\overrightarrow{x}(e) - \overrightarrow{f}(e)\|] \tag{13}$$

where e is the evaluation episode, $\overrightarrow{x}(e)$ is the center of mass of the object, $\overrightarrow{g}(e)$ is the center of mass of the goal, $a(e)$ is the applied torque to the joints and $\overrightarrow{f}(e)$ is the center of mass of the fingertips.

Reacher

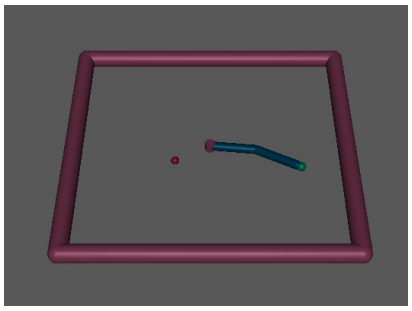


Figure 9: Reacher task: a robotic arm hinged at the center of the arena must reach a target object (displayed in red).

The reacher problem consists in a two-segments robotic arm, hinged to the center of the environment, which has to move in order to reach a target object (see FIGURE 9). The fitness function F is expressed as in Eq. 14:

$$F = -\frac{1}{N_E}\sum_{e=1}^{N_E}[\|\overrightarrow{x}(e) - \overrightarrow{f}(e)\| + a(e)^2] \tag{14}$$

where e is the evaluation episode, $\overrightarrow{x}(e)$ is the center of mass of the object, $\overrightarrow{f}(e)$ is the center of mass of the fingertip of the arm and $a(e)$ is the applied torque to the joints.

Swimmer

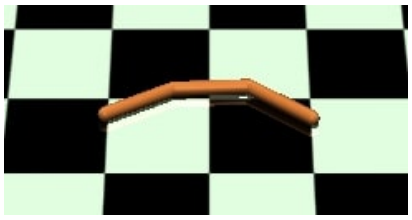


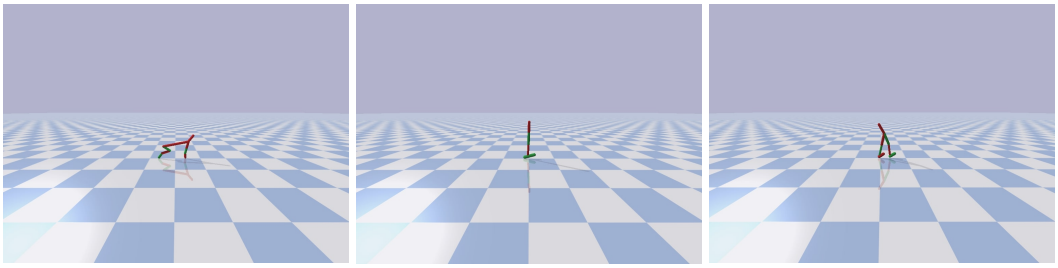
Figure 10: Swimmer task: a snake-like robot must locomote as far as possible in the environment.

The swimmer problem consists in a snake-like robot that has to travel as far as possible (see FIGURE 10). The fitness function F is expressed as in Eq. 15:

$$F = \frac{1}{N_E}\sum_{e=1}^{N_E}[v_x(e) - 0.0001 \times a(e)^2] \quad (15)$$

where e is the evaluation episode, $v_x(e)$ is the velocity along the x-axis and $a(e)$ is the applied torque to the joints.

2.1.9 Pybullet locomotion problems



(a) **(b)** **(c)**

Figure 11: Illustration of the Pybullet locomotion problems considered in this work: (a) halfcheetah; (b) hopper; (c) walker2D.

Pybullet locomotion [35] is a classic benchmark for RL and ES [14, 59, 60]. Specifically, we considered the halfcheetah (FIGURE 11-(a)), hopper (FIGURE 11-(b)) and walker2D (FIGURE 11-(c)) problems. The goal is to locomote as far as possible toward a target location at approximately 1 km away from the robot's starting position. The problem is formulated according to the following fitness function (see Eq. 16):

$$F = \frac{1}{N_E}\sum_{e=1}^{N_E}[A(e) + \Delta x(e) - 2 \times \frac{1}{N_j}\sum_{j=1}^{N_j}|a_j(e) \times v_j(e)| - 0.1 \times \frac{1}{N_j}\sum_{j=1}^{N_j} a_j(e) - 0.1 \times {N_j}^L(e)] \quad (16)$$

where e is the evaluation episode, $A(e)$ is a boolean value rewarding for the robot's survival ability (i.e., $A(e) = 1$ if robot is alive), $\Delta x(e)$ is the displacement of the robot, N_j is the number of joints, $a_j(e)$ is the torque applied to the $j-th$ joint, $v_j(e)$ indicates the velocity of the joint j and ${N_j}^L(e)$ counts the number of joints reaching their limits.

2.1.10 Halfcheetah2D

The last considered problem is the 2D version of the Pybullet halfcheetah, called "Halfcheetah2D", recently introduced in [47]. The task is illustrated in FIGURE 12: a cheetah robot has to locomote as far as possible over an almost flat surface, with only limited variations in the slope of the terrain. Eq. 17 provides the definition of the fitness function F:

$$F = \frac{1}{N_E} \sum_{e=1}^{N_E} [\Delta x(e) - 0.1 \times {N_j}^L(e)] \tag{17}$$

where e represents the evaluation episode, Δx is the distance traveled by the robot and ${N_j}^L(e)$ counts the number of joints reaching their limits.

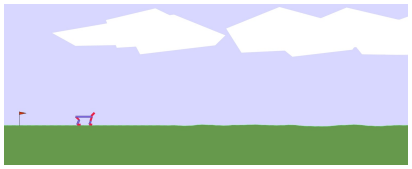


Figure 12: Halfcheetah2D problem: a cheetah robot must locomote as far as possible in the environment, which is characterized by limited variations in the slope of the terrain.

2.2 Experimental Settings

All the experiments have been replicated 10 times (i.e., $N_R = 10$). As regards OpenAI-ES, we kept the same experimental settings reported in [14, 16, 22, 24]. The length of the refinement process of PopOpenAI-ES+HC and OpenAI-ES+HC has been set to 5 (i.e., $N_{RI} = 5$) analogously to SSSHC (see also [22, 36, 38]). The population size of PopOpenAI-ES and PopOpenAI-ES+HC has been set to 10, while SSSHC evolves a population of 20 individuals. A detailed list of algorith settings is provided in TABLE 2.

Table 2: List of algorithm settings used for the experiments reported in this work. The symbols are defined as follows: N_R indicates the number of replications; N_{EE} is the number of evaluation episodes (i.e., the length of the evolution); N_{RI} denotes the number of refinement iteration; $PopSize$ is the population size; $MutRate$ represents the mutation rate; $RecRate$ indicates the recombination rate (i.e., crossover); $ReplProb$ is the probability to replace the worst centroid with a copy of the best one. The symbol "N/A" has been used to indicate that the corresponding parameter is not applicable to that algorithm.

Parameter	OpenAI-ES	PopOpenAI-ES	PopOpenAI-ES+HC	OpenAI-ES+HC	SSSHC
N_R	10				
N_{RI}	N/A		5		
$PopSize$	N/A	10		N/A	20
$MutRate$	0.02				
$RecRate$	N/A				0.05
$ReplProb$	N/A	0.2		N/A	

The description of the length of evolution N_G, the number of evaluation episodes N_E and the number of episode's steps N_S (i.e., the length of the episode) is shown in TABLE 3. Concerning test

functions, our experiments have been run for 10^6 iterations (i.e., $N_G = 10^6$) by setting the length of the input vector to $d = 50$, which is evaluated once only (i.e. $N_E = 1$, $N_S = 1$).

Table 3: Length of evolution N_G, number of episodes N_E and episode length N_S for the N-bit parity, double pole balancing, mountain car continuous, pendulum, grid navigation, foraging with poison, pusher, reacher, swimmer, halfcheetah, hopper, walker2D and halfcheetah2D problems.

Problem	N_G	N_E	N_S
5-bit parity	2.5×10^7	32	1
6-bit parity	2.5×10^7	64	1
7-bit parity	5×10^7	128	1
8-bit parity	10^8	256	1
Double-pole	5×10^7	8	1000
Long-poles	5×10^7	8	1000
Mountain car continuous	10^7	3	1000
Pendulum	5×10^7	5	200
Grid navigation	10^7	10	100
Foraging with poison	2.5×10^7	5	1000
Pusher	5×10^7	3	1000
Reacher	5×10^7	3	1000
Swimmer	5×10^7	3	1000
Halfcheetah	5×10^7	3	1000
Hopper	5×10^7	3	1000
Walker2D	5×10^7	3	1000
Halfcheetah2D	5×10^7	3	500

With the exception of test functions, we used an artificial neural network to compute the output and evaluate the performance. Regarding double pole balancing and grid navigation, we used a Recurrent Neural Network (RNN) [61, 62]. Conversely, we employed a multi-layer perceptron [63] for the other problems. The full list of network architectures and their parameters is illustrated in TABLE 4.

Finally, we adopted batch normalization [64] for the mountain car continuous, reacher, pusher, swimmer, halfcheetah, hopper, walker2D and halfcheetah2D problems similarly to previous studies [14, 23, 24, 47].

3. RESULTS

TABLE 5 summarizes the outcomes of our experiments. Surprisingly, SSSHC (i.e., our baseline method) achieves the best performance in 29 over 50 tasks, including rather difficult problems like long-poles, grid navigation, foraging and MuJoCo swimmer. Moreover, it outperforms the other algorithms in 22 test functions (Kruskal-Wallis H test, $p < 10^{-6}$). However, the performance of SSSHC is remarkably lower than the others with respect to robot locomotion (Kruskal-Wallis H test, $p < 10^{-6}$). OpenAI-ES bests other methods in 12 tasks, particularly in the Pybullet loco-

Table 4: Description of the neural network controller used for the N-bit parity, double pole balancing, mountain car continuous, pendulum, grid navigation, foraging with poison, pusher, reacher, swimmer, halfcheetah, hopper, walker2D and halfcheetah2D problems. The symbol N_L denotes the number of layers, whereas N_H represents the number of hidden/internal neurons per layer. In addition, the symbols N_I and N_O indicate the number of inputs and outputs, respectively. Finally, the symbols act_H and act_O identify the activation function of hidden and output neurons, respectively.

Problem	Network	N_L	N_H	N_I	N_O	act_H	act_O
5-bit parity	MLP	5	10	5	1	tanh	tanh
6-bit parity	MLP	5	10	6	1	tanh	tanh
7-bit parity	MLP	5	10	7	1	tanh	tanh
8-bit parity	MLP	5	10	8	1	tanh	tanh
Double-pole	RNN	1	10	3	1	tanh	tanh
Long-poles	RNN	1	10	3	1	tanh	tanh
Mountain car continuous	MLP	1	50	2	1	tanh	linear
Pendulum	MLP	1	20	3	1	tanh	linear
Grid navigation	RNN	1	10	4	1	tanh	tanh
Foraging with poison	MLP	1	50	18	2	tanh	tanh
Pusher	MLP	1	50	23	7	tanh	linear
Reacher	MLP	1	50	11	2	tanh	linear
Swimmer	MLP	1	50	8	2	tanh	linear
Halfcheetah	MLP	1	50	26	6	tanh	linear
Hopper	MLP	1	50	15	3	tanh	linear
Walker2D	MLP	1	50	22	6	tanh	linear
Halfcheetah2D	MLP	1	50	26	6	tanh	linear

motion problems and in the MuJoCo pusher and reacher tasks (Kruskal-Wallis H test, $p < 10^{-6}$). PopOpenAI-ES+HC excels in 11 tasks like the complex N-bit parity (with $N = 6, 7, 8$) and double pole balancing. This proves its effectiveness moderately difficult scenarios. Lastly, PopOpenAI-ES and OpenAI-ES+HC obtain lower performance on average with regard to their counterparts. The final ranking of the algorithms is provided in the last row of TABLE 5.

Table 5: Analysis of the fitness achieved by the different algorithms. Data is represented as $\mu \pm \sigma$ (with μ and σ denoting, respectively, mean and standard deviation) and is the average of 10 replications of the experiments. Bold values mark the best performance for each problem.

	OpenAI-ES	PopOpenAI-ES	PopOpenAI-ES+HC	OpenAI-ES+HC	SSSHC
Ackley	-2.3632 ± 0.1941	-1.8805 ± 0.2880	-0.0058 ± 0.0006	-0.0093 ± 0.0010	$-0.0005 \pm 8.2536 \times 10^{-5}$
Alpine no 1	-0.0363 ± 0.0010	-0.0353 ± 0.0011	-0.0350 ± 0.0010	-0.0367 ± 0.0009	$-0.0002 \pm 1.1271 \times 10^{-5}$
Chung-Reynolds	$-3.2760 \times 10^{-8} \pm 2.2760 \times 10^{-8}$	$-5.8923 \times 10^{-9} \pm 2.8151 \times 10^{-9}$	$-1.3522 \times 10^{-8} \pm 6.3492 \times 10^{-9}$	$-5.6439 \times 10^{-8} \pm 1.7879 \times 10^{-8}$	$-6.7720 \times 10^{-13} \pm 4.1614 \times 10^{-13}$
Csendes	$-2.2096 \times 10^{-11} \pm 8.8472 \times 10^{-12}$	$-7.0367 \times 10^{-12} \pm 2.7817 \times 10^{-12}$	$-2.2030 \times 10^{-12} \pm 5.3365 \times 10^{-12}$	$-2.5046 \times 10^{-11} \pm 8.3021 \times 10^{-12}$	$-2.7806 \times 10^{-20} \pm 3.3793 \times 10^{-20}$
Deb 1	0.9883 ± 0.0014	0.9910 ± 0.0020	0.9911 ± 0.0013	0.9899 ± 0.0014	$1.0 \pm 3.1828 \times 10^{-8}$
Dixon-Price	-2.3759 ± 0.0912	-2.0633 ± 0.2334	-0.0063 ± 0.0010	-0.0087 ± 0.0011	$-0.0005 \pm 9.2185 \times 10^{-5}$
Egg Holder	$3156.2269 \pm 1.8606 \times 10^{-6}$	3156.2247 ± 0.0025	3152.7616 ± 0.4638	$3156.2269 \pm 2.3020 \times 10^{-6}$	1507.2689 ± 0.0064
Exponential	$0.9999 \pm 1.8297 \times 10^{-5}$	$1.0 \pm 6.2709 \times 10^{-6}$	$0.9999 \pm 1.2303 \times 10^{-5}$	$0.9999 \pm 3.8119 \times 10^{-5}$	$1.0 \pm 1.2097 \times 10^{-7}$
Griewank	$-1.3189 \times 10^{-6} \pm 3.1832 \times 10^{-7}$	$-7.1438 \times 10^{-7} \pm 1.1922 \times 10^{-7}$	$-1.2748 \times 10^{-6} \pm 2.8631 \times 10^{-7}$	$-2.1797 \times 10^{-6} \pm 6.7491 \times 10^{-7}$	$-3.7365 \times 10^{-8} \pm 1.4462 \times 10^{-8}$
Michalewicz	19.4949 ± 1.0618	17.4037 ± 1.4677	34.0146 ± 0.5825	34.7250 ± 0.3684	35.7310 ± 0.0006
Qing	-0.0038 ± 0.0009	-0.0017 ± 0.0002	-0.0013 ± 0.0002	-0.0028 ± 0.0007	-40425.4119 ± 0.0731
Quintic	-1.4949 ± 0.1652	-1.2321 ± 0.1153	-1.3848 ± 0.1326	-1.5972 ± 0.1018	-0.0290 ± 0.0037
Rana	199.3569 ± 46.4177	82.9414 ± 7.0947	84.4439 ± 7.0020	120.4843 ± 25.9393	23.9979 ± 0.0528
Rastrigin	-25.2212 ± 3.2965	-22.6098 ± 3.5331	-0.0671 ± 0.0237	-0.1153 ± 0.0211	$-0.0002 \pm 4.5038 \times 10^{-5}$
Rosenbrock	-0.0733 ± 0.0204	-0.0331 ± 0.0046	-0.0472 ± 0.0100	-0.1072 ± 0.0379	-0.0046 ± 0.0010
Salomon	-0.4099 ± 0.0300	-0.3899 ± 0.3000	-0.3899 ± 0.3000	-0.4099 ± 0.0300	$-0.3999 \pm 4.2501 \times 10^{-14}$
Schaffer F6	$-0.0003 \pm 7.3073 \times 10^{-5}$	$-0.0001 \pm 3.0339 \times 10^{-5}$	$-0.0002 \pm 4.6357 \times 10^{-5}$	-0.0005 ± 0.0001	$-1.6206 \times 10^{-6} \pm 4.5289 \times 10^{-7}$
Schwefel	$-1.9177 \times 10^{-19} \pm 1.7759 \times 10^{-19}$	$-1.8632 \times 10^{-21} \pm 1.6198 \times 10^{-21}$	$-6.0948 \times 10^{-20} \pm 7.2573 \times 10^{-20}$	$-1.2840 \times 10^{-18} \pm 1.7389 \times 10^{-18}$	$-5.1907 \times 10^{-31} \pm 5.9951 \times 10^{-31}$
Schwefel 1.2	-0.2833 ± 0.0829	-0.1930 ± 0.0780	-0.0470 ± 0.0073	-0.0971 ± 0.0330	-0.0006 ± 0.0004
Schwefel 2.4	$-0.0002 \pm 5.4252 \times 10^{-5}$	$-0.0001 \pm 2.7330 \times 10^{-5}$	$-0.0002 \pm 4.7089 \times 10^{-5}$	-0.0004 ± 0.0002	$-6.1160 \times 10^{-6} \pm 1.2761 \times 10^{-6}$
Schwefel 2.20	6240.4587 ± 5.6536	679.6833 ± 1.9553	620.1269 ± 1.5509	5589.5693 ± 5.1116	49.9958 ± 0.0008
Schwefel 2.22	-0.1282 ± 0.0190	-0.0881 ± 0.0069	-0.1107 ± 0.0119	-0.1428 ± 0.0196	-0.0042 ± 0.0007
Schwefel 2.23	$-9.4544 \times 10^{-19} \pm 4.1940 \times 10^{-19}$	$-2.4257 \times 10^{-19} \pm 9.6119 \times 10^{-20}$	$-4.3998 \times 10^{-19} \pm 2.3700 \times 10^{-19}$	$-9.5818 \times 10^{-19} \pm 5.7985 \times 10^{-19}$	$-7.3094 \times 10^{-34} \pm 1.1630 \times 10^{-33}$
Schwefel 2.26	$3.9453 \pm 4.4204 \times 10^{-8}$	$3.9453 \pm 1.5574 \times 10^{-8}$	$3.9453 \pm 1.3523 \times 10^{-8}$	$3.9453 \pm 4.7073 \times 10^{-8}$	$0.8413 \pm 2.1447 \times 10^{-5}$
Shubert 3	366.8638 ± 17.4677	398.5567 ± 18.4648	741.8781 ± 0.0145	741.8208 ± 0.0265	630.7989 ± 0.0504
Shubert 4	375.9930 ± 10.4417	369.7455 ± 22.4995	429.3589 ± 3.2593	427.1516 ± 1.9956	$425.8915 \pm 1.6183 \times 10^{-5}$
Sphere	$-0.0001 \pm 3.7477 \times 10^{-5}$	$-6.3435 \times 10^{-5} \pm 1.1988 \times 10^{-5}$	$-0.0001 \pm 2.0918 \times 10^{-5}$	-0.0002 ± 0.0001	$-8.0744 \times 10^{-7} \pm 3.0828 \times 10^{-7}$
Styblinski-Tang	1654.3678 ± 54.1088	1662.8504 ± 26.4095	1894.6926 ± 20.2407	1880.5553 ± 40.6046	499.8678 ± 0.0156
Sum squares	-0.0034 ± 0.0010	-0.0013 ± 0.0003	-0.0021 ± 0.0002	-0.0036 ± 0.0015	$-1.7786 \times 10^{-5} \pm 8.3318 \times 10^{-6}$
Trid	9648.1236 ± 7.8111	1178.7363 ± 2.9439	1108.2032 ± 3.4866	8823.9405 ± 7.2847	48.9842 ± 0.0020
Trigonometric 2	-59.8702 ± 9.5667	-72.2643 ± 8.3345	-1.0009 ± 0.0001	-1.0009 ± 0.0001	$-1.0 \pm 1.8610 \times 10^{-7}$
Whitley	1706.7062 ± 105.9269	1143.3136 ± 258.3018	2435.8984 ± 4.6378	2424.8824 ± 16.3101	2481.5684 ± 2.8576
Zakharov	-0.1124 ± 0.0101	-0.0560 ± 0.0151	-0.0607 ± 0.0119	-0.1105 ± 0.0093	-0.9054 ± 0.2456
5-bit parity	0.7689 ± 0.0980	0.9281 ± 0.0420	0.9656 ± 0.0219	0.8594 ± 0.1103	0.9844 ± 0.0156
6-bit parity	0.7328 ± 0.1359	0.9172 ± 0.0681	0.9281 ± 0.0678	0.8141 ± 0.1104	0.9281 ± 0.0390
7-bit parity	0.8086 ± 0.0965	0.8758 ± 0.0519	0.9047 ± 0.0476	0.8883 ± 0.0266	0.8438 ± 0.0454
8-bit parity	0.7664 ± 0.0881	0.8305 ± 0.0894	0.8941 ± 0.0344	0.7828 ± 0.0909	0.8359 ± 0.0446
Double-pole	980.6500 ± 61.1901	975.1250 ± 78.6617	1000.0 ± 0.0	930.1880 ± 118.9012	1000.0 ± 0.0
Long-poles	975.3000 ± 78.1083	1000.0 ± 0.0	1000.0 ± 0.0	955.4000 ± 94.0298	1000.0 ± 0.0
Mountain car continuous	$99.9999 \pm 8.2639 \times 10^{-5}$	99.9527 ± 0.1106	99.9225 ± 0.1131	99.9997 ± 0.0002	21.1963 ± 37.9636
Pendulum	-439.4861 ± 193.7389	-782.7599 ± 67.4736	-654.1126 ± 45.0868	-298.8404 ± 163.0051	-581.8754 ± 75.0294
Grid navigation	0.6900 ± 0.1758	0.46000 ± 0.1356	0.4900 ± 0.2625	0.6200 ± 0.1470	0.9900 ± 0.0300
Foraging with poison	3.1000 ± 1.9905	3.000 ± 0.9209	3.2200 ± 0.4686	3.9400 ± 1.2682	4.9800 ± 0.0600
Pusher	-20.9844 ± 1.0802	-21.1583 ± 2.2109	-29.0674 ± 2.9379	-33.3659 ± 3.8585	-33.4437 ± 1.2860
Reacher	-1.5365 ± 0.3045	-2.2971 ± 0.5370	-2.9864 ± 0.8479	-3.0912 ± 0.6699	-4.2399 ± 0.3923
Swimmer	354.9453 ± 12.9441	362.0592 ± 0.3778	361.9829 ± 0.5881	362.7665 ± 0.6289	363.1246 ± 0.4265
Halfcheetah	2694.3129 ± 302.9857	2077.7122 ± 257.8364	1667.3282 ± 336.6630	1119.8957 ± 686.8173	482.1538 ± 146.5778
Hopper	2397.6716 ± 204.3949	1822.4191 ± 326.3898	1810.0932 ± 210.6096	2125.1624 ± 273.3858	1367.7663 ± 310.7625
Walker2D	1185.3637 ± 630.5734	820.8865 ± 596.2367	673.9069 ± 364.4404	247.9873 ± 305.3153	230.8398 ± 77.8970
Halfcheetah2D	211.6580 ± 15.3208	187.7617 ± 12.9251	166.5091 ± 14.2574	112.5669 ± 24.9665	41.0083 ± 30.0051
# of tasks with best performance	12	5	11	3	29
Ranking	2	4	3	5	1

If we look carefully at our outcomes, we can see that the three variants introduced in this work outperform OpenAI-ES in many of the considered problems (see TABLE 6), including several test functions, N-bit parity and MuJoCo swimmer. Furthermore, they are competitive with respect to double pole balancing, mountain car continuous and foraging problems. This demonstrates that these methods represent valid alternatives to OpenAI-ES, particularly in domains characterized by limited stochasticity or not involving jointed robots interacting with the environment. Conversely, all the three variants fall short to OpenAI-ES with regard to robot locomotion. This can be explained by considering that all these methods require an extra-budget reducing the overall number of evolutionary iterations, which has a huge impact in such scenarios. Concerning PopOpenAI-ES+HC and OpenAI-ES+HC, another drawback lies in the increase of weight size caused by HC. Especially in complex problems like Pybullet halfcheetah, hopper and walker2D, the refinement process is highly likely to fail, because finding adaptive single-gene mutations is far from trivial. In addition,

PopOpenAI-ES+HC and OpenAI-ES+HC are likely to follow completely different, eventually sub-optimal, evolutionary paths compared to OpenAI-ES, particularly if the local refinement search affects the centroid during the early stages of the evolution when performance is generally low.

Table 6: Number of problems in which the PopOpenAI-ES, PopOpenAI-ES+HC and OpenAI-ES+HC algorithms outperform or equal OpenAI-ES.

	PopOpenAI-ES	PopOpenAI-ES+HC	OpenAI-ES+HC
# of tasks with performance higher or equal than OpenAI-ES	31	37	24

A summary of the performance difference between OpenAI-ES and the three variants introduced is shown in FIGURE 13, in which we provide a detailed statistical analysis with respect to the 50 considered problems. As can be observed, PopOpenAI-ES, PopOpenAI-ES+HC and OpenAI-ES+HC are competitive with OpenAI-ES as regards test function optimization, N-bit parity, double pole balancing, grid navigation, foraging with poison and MuJoCo swimmer (FIGURE 13, blue and green squares). Instead, they are outperformed by OpenAI-ES with respect to MuJoCo pusher, MuJoCo reacher and Pybullet locomotion problems (FIGURE 13, red squares).

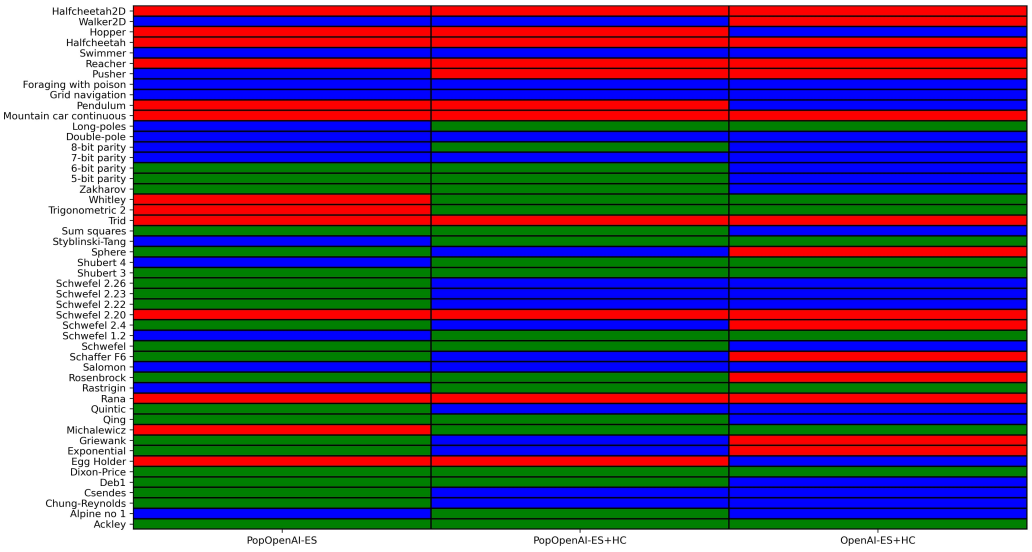


Figure 13: Statistical comparison between OpenAI-ES and its three variants on the 50 considered problems. Red squares mean that OpenAI-ES significantly outperforms its counterpart based on the Mann-Whitney U test with Bonferroni correction. Conversely, green squares indicate the statistical superiority of a variant (i.e., PopOpenAI-ES, PopOpenAI-ES+HC or OpenAI-ES+HC) over OpenAI-ES. Finally, blue squares denote no statistical difference.

4. DISCUSSION

Our results emphasize that PopOpenAI-ES, PopOpenAI-ES+HC and OpenAI-ES+HC are better alternatives than OpenAI-ES regarding 12 test functions, N-bit parity and MuJoCo swimmer, i.e. problems characterized by limited or absent stochasticity. Furthermore, all the proposed variants obtain competitive performance in general (see TABLE 6). A notable exception is the class of robot locomotion problems, in which PopOpenAI-ES, PopOpenAI-ES+HC and OpenAI-ES+HC are significantly outperformed by OpenAI-ES. As already pointed out in Section 3, the proposed variants require an extra-budget caused by either the higher number of samples evaluated in each iteration (PopOpenAI-ES), or the refinement process (OpenAI-ES+HC), or both (PopOpenAI-ES+HC), which reduces the number of evolutionary iterations and the chance to improve performance. Moreover, as pointed out in [22, 38], the refinement process could fail in discovering adaptive single-gene mutations, particularly in non-deterministic problems. Future studies should delve into the mitigation of the stochasticity impact on the local search process.

PopOpenAI-ES is the variant achieving better performance on the robot locomotion problems, although significantly inferior to OpenAI-ES. A possible drawback is the replacement of worst centroid (and the corresponding momentum vectors): on one hand, this technique increases the average centroid performance on the spot and can be beneficial to escape strong local minima. On the other hand, it could have a disruptive effect by eliminating solutions that could have later improved if retained in the population. As a consequence, avoiding the replacement of the worst centroid (i.e., evolving independent centroids) could have positive effects on the performance.

We implemented this modified version of PopOpenAI-ES, called PopOpenAI-ES_noRep, and conducted a preliminary comparison of PopOpenAI-ES and PopOpenAI-ES_noRep in the robot locomotion problems. The outcomes are shown in TABLE 7 and demonstrate that the evolution of independent centroids leads to enhanced performance in all the cases, particularly for the hopper task where we found a statistical difference between the two methods (Mann-Whitney U test, $p < 0.05$, see also FIGURE 14). However, the performance comparison of PopOpenAI-ES_noRep and OpenAI-ES indicated the superiority of the latter with respect to halfcheetah, hopper and halfcheetah2D problems (Mann-Whitney U test, $p < 0.05$), while no statistical difference has been found regarding the walker2D problem (Mann-Whitney U test, $p > 0.05$). Therefore, these results highlight that PopOpenAI-ES_noRep represents a further step toward the enhancement of modern ESs in complex robot locomotion tasks.

Table 7: Preliminary results of the PopOpenAI-ES algorithm without replacement of the worst centroid in the Halfcheetah, Hopper, Walker2D and Halfcheetah2D problems.

Halfcheetah	Hopper	Walker2D	Halfcheetah2D
2252.7146 ± 266.8057	2135.6261 ± 194.4838	1060.9267 ± 344.8909	194.6295 ± 5.8730

5. CONCLUSIONS

Evolutionary Strategies (ESs) have undergone drastic changes since their introduction in the early 70s. In particular, a shift from biologically inspired methods toward advanced and sophisticated

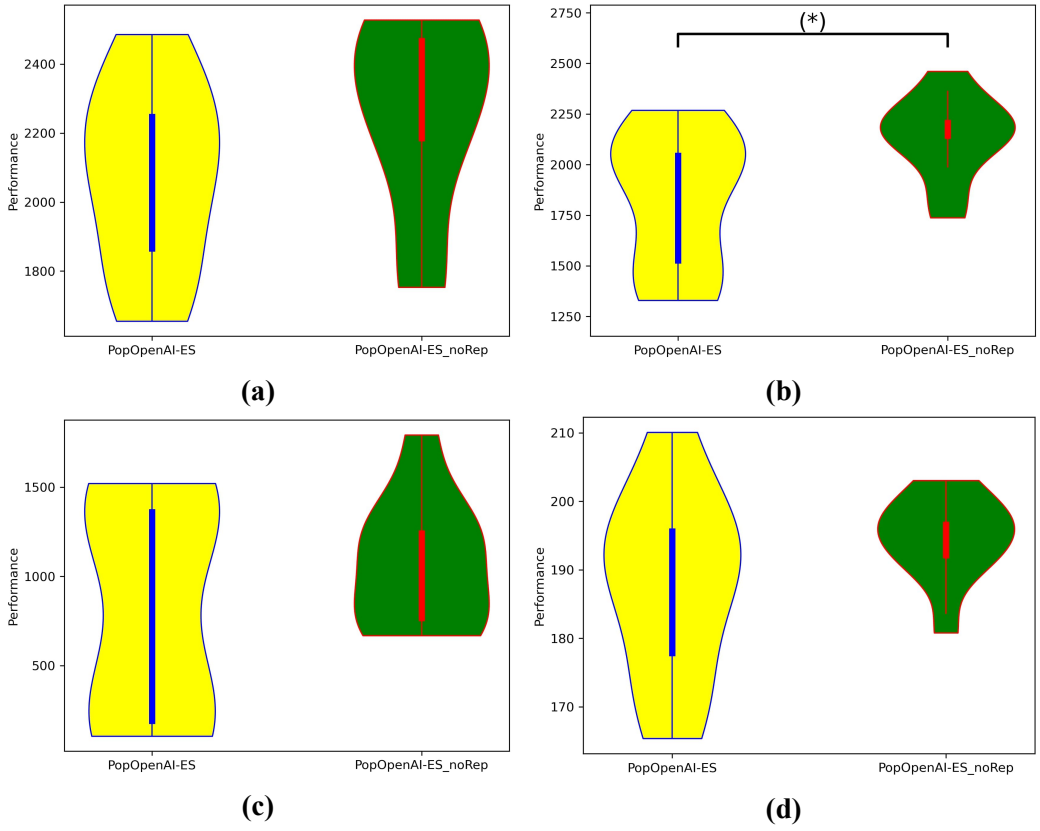


Figure 14: Comparison of the fitness obtained by the PopOpenAI-ES and PopOpenAI-ES_noRep algorithms on the locomotion problems: (a) Halfcheetah; (b) Hopper; (c) Walker2D; (d) Halfcheetah2D. The symbol $(*)$ indicates that the two methods are statistically different.

techniques has become an appealing research direction. In fact, the possibility to apply ESs in a wide range of problems (from relatively simple control tasks to challenging locomotion and manipulation problems) boosted the development of new approaches constantly aiming to enhanced performance. Among these methods, state-of-the-art techniques include CMA-ES and xNES, which compute expensive covariance matrices to store information about parameter variations that should lead to a better search space exploration. However, as the number of parameters increase dramatically, the calculation of covariance matrices becomes intractable and prevents these methods from being used in challenging locomotion problems (see [14, 20]). OpenAI-ES emerged as a valuable ES for a broad set of problems, including robot locomotion, pole balancing and swarm robotics. The main advantage of OpenAI-ES is the usage of two momentum vectors maintaining historical data about the relationship between parameter variations and performance, with no need for additional computations. However, there are cases in which OpenAI-ES generates sub-optimal performance compared to other methods like RL or EAs [14, 22].

This work introduces three variants of OpenAI-ES:

- PopOpenAI-ES: it is a version of OpenAI-ES evolving a population of independent centroids with the possibility to replace at each iteration the worst performing centroid with the best one;
- PopOpenAI-ES+HC: it combines PopOpenAI-ES with the HC algorithm, which runs a local search process seeking to improve performance;
- OpenAI-ES+HC: it combines OpenAI-ES with HC.

In particular, we delve into a comparison of OpenAI-ES, PopOpenAI-ES, PopOpenAI-ES+HC and OpenAI-ES+HC in 50 problems, including benchmark tasks like test function optimization, N-bit parity, double pole balancing and robot locomotion. Moreover, we used SSSHC as a baseline method for the comparison. The results of our investigation proves the competitiveness of the proposed variants with respect to many of the considered problems. Surprisingly, SSSHC is the best algorithm in 29 over 50 problems. This confirms that even a simple EA can perform similarly, or even outperform, a sophisticated method [22, 65]. Finally, OpenAI-ES outperforms the other methods with regard to the most complex problems like Pybullet robot locomotion. The latter outcome is mainly related to the extra-budget required by the proposed variants. Furthermore, the replacement of the worst centroid in PopOpenAI-ES and PopOpenAI-ES+HC algorithms might have disruptive effects in robot locomotion problems. In fact, the preliminary study on PopOpenAI-ES_noRep (i.e., a version of PopOpenAI-ES without replacing the worst centroid) provides better results on the halfcheetah, hopper, walker2D and halfcheetah2D problems, although still inferior to OpenAI-ES.

As new research pathways, we aim to overcome the limitations of PopOpenAI-ES, PopOpenAI-ES+HC and OpenAI-ES+HC highlighted in this work. First, the number of evolved centroids has been set to 10, but no hyperparameter analysis has been made. Future work could investigate the relationship between number of centroids and performance, and techniques to automatically determine the suitable number of centroids to be evolved, as well as to identify the optimal values of each algorithm's hyperparameters [66–69]. Furthermore, the refinement process performed by HC might fail in the enhancement of performance, especially for complex tasks like robot locomotion. In fact, although the usage of single-gene mutations is more likely to discover truly adaptive

modifications, it can have disruptive effects in stochastic scenarios such as Pybullet locomotion problems. Analyzing the effect of multiple-gene mutations could be object of future studies, as well as the implementation of techniques keeping the size of connection weights limited, which turns out to be paramount in such scenarios.

References

[1] https://gwern.net/doc/reinforcement-learning/exploration/1973-rechenberg.pdf.

[2] Schwefel HP. Numerische Optimierung von Computer-Modellen Mittels Der Evolution-sstrategie. Mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie. 1st Edition. Berlin: Springer. 1977:57.

[3] Bäck T. Evolutionary Algorithms in Theory and Practice: Evolution Strategies Evolutionary Programming Genetic Algorithms. New York: Oxford University Press. 1996.

[4] Maynard Smith J. The Theory of Evolution. Cambridge University Press. 1993:380.

[5] Drchal J, Šnorek M. Tree-Based Indirect Encodings for Evolutionary Development of Neural Networks. In International Conference on Artificial Neural Networks. Springer. 2008:839–848.

[6] Gruau F, Whitley D. Adding Learning to the Cellular Development of Neural Networks: Evolution and the Baldwin Effect. Evol Comput. 1993;1:213–233.

[7] Gruau F, Whitley D, Pyeatt L. A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks. In Proceedings of the 1st annual conference on genetic programming. 1996:81–89.

[8] Miikkulainen R, Forrest S. A Biological Perspective on Evolutionary Computation. Nat Mach Intell. 2021;3:9–15.

[9] Stanley KO, D'Ambrosio DB, Gauci J. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. Artificial life. IEEE. 2009;15:185–212.

[10] Hansen N, Ostermeier A. Completely Derandomized Self-Adaptation in Evolution Strategies. Evolutionary computation. IEEE. 2001;9:159–195.

[11] Christian Igel. Igel C. Neuroevolution for Reinforcement Learning Using Evolution Strategies. In The 2003 Congress on Evolutionary Computation. IEEE. 2003;4:2588-2595.

[12] Pagliuca P, Nolfi S. Robust Optimization Through Neuroevolution. PLOS ONE. 2019;14:e0213193.

[13] Ajani OS, Kumar A, Mallipeddi R. Covariance Matrix Adaptation Evolution Strategy Based on Correlated Evolution Paths With Application to Reinforcement Learning. Expert Syst Appl. 2024;246:123289.

[14] Pagliuca P, Milano N, Nolfi S. Efficacy of Modern Neuro-Evolutionary Strategies for Continuous Control Optimization. Front Robot AI. 2020;7:98.

[15] Pagliuca P, Vitanza A. Evolving Aggregation Behaviors in Swarms From an Evolutionary Algorithms Point of View. In Applications of Artificial Intelligence and Neural Systems to Data Science. Springer Nature. 2023:317–328.

[16] Pagliuca P, Vitanza A. A Comparative Study of Evolutionary Strategies for Aggregation Tasks in Robot Swarms: Macro- And Micro-Level Behavioral Analysis. IEEE Access. 2025;13:72721–72735.

[17] Glasmachers T, Schaul T, Yi S, Wierstra D, Schmidhuber J. Exponential Natural Evolution Strategies. In Proceedings of the 12th annual conference on Genetic and evolutionary computation. 2010:393–400.

[18] Wierstra D, Schaul T, Glasmachers T, Sun Y, Peters J, et al. Natural Evolution Strategies. J Mach Learn Res. 2014;15:949–980.

[19] Pagliuca P, Nolfi S. Integrating Learning by Experience and Demonstration in Autonomous Robots. Adapt Behav. 2015;23:300–314.

[20] Duan Y, Chen X, Houthooft R, Schulman J, Abbeel P. Benchmarking Deep Reinforcement Learning for Continuous Control. In International conference on machine learning. 2016;48:1329–1338.

[21] Schaul T, Glasmachers T, Schmidhuber J. High Dimensions and Heavy Tails for Natural Evolution Strategies. In Proceedings of the 13th annual conference on Genetic and evolutionary computation. 2011:845–852.

[22] Pagliuca P. Discovering a Single Neural Network Controller for Multiple Tasks With Evolutionary Algorithms. J Artif Intell Auton Intell. 2025;2:322–348.

[23] Salimans T, Ho J, Chen X, Sidor S, Sutskever I. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. 2017. arXiv preprint: https://arxiv.org/pdf/1703.03864.

[24] Pagliuca P, Nolfi S. The Dynamic of Body and Brain Co-Evolution. Adapt Behav. 2022;30:245–255.

[25] Pagliuca P, Vitanza A. Self-Organized Aggregation in Group of Robots With OpenAI-ES. In International Conference on Soft Computing and Pattern Recognition. Springer Nature. 2022:770–780.

[26] Jasmina Rais Martínez and Fidel Aznar Gregori. Comparison of Evolutionary Strategies for Reinforcement Learning in a Swarm Aggregation Behaviour. In 3rd International Conference on Machine Learning and Machine Intelligence. 2020:40–45.

[27] Pagliuca P, Vitanza A. Enhancing Aggregation in Locomotor Multi-Agent Systems: A Theoretical Framework. In Proceedings of the 25th Edition of the Workshop From Object to Agents. WOA. 2024:42–57.

[28] Pagliuca P, Trivisano G, Vitanza A. How to Evolve Aggregation in Robotic Multi-Agent Systems. Proceedings of the 26th Edition of the Workshop From Object to Agents. WOA. 2025.

[29] Nolfi S, Pagliuca P. Global Progress in Competitive Co-Evolution: A Systematic Comparison of Alternative Methods. Front Robot AI. 2025;11:1470886.

[30] Brockhoff D, Auger A, Hansen N, Arnold DV, Hohm T. Mirrored Sampling and Sequential Selection for Evolution Strategies. In International Conference on Parallel Problem Solving from Nature. Heidelberg: Springer Nature. 2010;11–21.

[31] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. 2014. arXiv preprint: https://arxiv.org/pdf/1412.6980v1.

[32] Pagliuca P, Nolfi S, Vitanza A. Evorobotpy3: A Flexible and Easy-To-Use Simulation Tool for Evolutionary Robotics. In Proceedings of the Genetic and Evolutionary Computation Conference. GECCO. 2025:155-158.

[33] Sutton RS, Barto AG. Reinforcement Learning: An Introduction. Cambridge: MIT press. 1998.

[34] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. 2017. arXiv preprint: https://arxiv.org/pdf/1707.06347.

[35] Coumans E, Bai Y. Pybullet a python module for physics simulation for games robotics and machine learning. 2016.

[36] Pagliuca P. Learning and Evolution: Factors Influencing an Effective Combination. AI. 2024;5:2393–2432.

[37] Rödl V, Tovey C. Multiple Optima in Local Search. Journal of Algorithms. 1987;8:250–259.

[38] Pagliuca P. Analysis of the Exploration-Exploitation Dilemma in Neutral Problems With Evolutionary Algorithms. J Artif Intell Auton Intell. 2024;1:110–121.

[39] Jamil M, Yang XS. A Literature Survey of Benchmark Functions for Global Optimisation Problems. Int J Math Model Numer Optim. 2013;4:150–194.

[40] Miller JF. An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach. In Proceedings of the genetic and evolutionary computation conference. 1999;2:1135–1142.

[41] Pagliuca P, Milano N, Nolfi S. Maximizing Adaptive Power in Neuroevolution. PLOS ONE. 2018;13: e0198788.

[42] Alexis P Wieland. Evolving Neural Network Controllers for Unstable Systems. In International Joint Conference on Neural Networks. IJCNN. IEEE. 1991;2:667–673.

[43] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, et al. OpenAI Gym. 2016. arXiv preprint: https://arxiv.org/pdf/1606.01540.

[44] Towers M, Kwiatkowski A, Terry J, Balis JU, De Cola G, et al. Gymnasium: A Standard Interface for Reinforcement Learning Environments. 2024. arXiv preprint: https://arxiv.org/pdf/2407.17032v1.

[45] Nogueira YL, de Brito CE, Vidal CA, Cavalcante-Neto JB. Towards Intrinsic Autonomy Through Evolutionary Computation. Artif Intell Rev. 2020;53:4449–4473.

[46] Todorov E, Erez T, Tassa Y. Mujoco: A Physics Engine for Model-Based Control. In 2012 IEEE/RSJ international conference on intelligent robots and systems. IEEE. 2012;5026–5033.

[47] Pagliuca P, Vitanza A. Learning Locomotion by Co-Evolution of Morphological and Neural Parameters. In Proceedings of the 2025 IEEE International Conference on Development and Learning. ICDL. IEEE. 2025:1–6.

[48] Sharma P, Raju S. Metaheuristic Optimization Algorithms: A Comprehensive Overview and Classification of Benchmark Test Functions. Soft Computing-A Fusion of Foundations Methodologies Applications. 2024;28:3123-3186.

[49] Kuyucu T, Trefzer M, Miller JF, Tyrrell A. On the Properties of Artificial Development and Its Use in Evolvable Hardware. In 2009 IEEE Symposium on Artificial Life. IEEE. 2009:108–115.

[50] James Alfred Walker and Julian Francis Miller. The Automatic Acquisition Evolution and Reuse of Modules in Cartesian Genetic Programming. In IEEE Transactions on Evolutionary Computation. IEEE. 2008;12:397–417.

[51] Immonen E, Haghbayan H. The Mountain Car Problem With a Dynamical and Finite Energy System. In 2024 10th International Conference on Mechatronics and Robotics Engineering ICMRE. IEEE. 2024:324–328.

[52] Melnikov AA, Makmal A, Briegel HJ. Benchmarking Projective Simulation in Navigation Problems. IEEE Access. 2018;6:64639–64648.

[53] file:///C:/Users/HP/Downloads/MscThesis.pdf.

[54] Heng H, Rahiman W. ACO-GA-Based Optimization to Enhance Global Path Planning for Autonomous Navigation in Grid Environments. IEEE Transactions on Evolutionary Computation. IEEE. 2025.

[55] Miglino O, Walker R. Genetic Redundancy in Evolving Populations of Simulated Robots. In Artificial Life. IEEE. 2002;8:265–277.

[56] Aldana-Franco F, Montes-González F, Nolfi S. The Improvement of Signal Communication for a Foraging Task Using Evolutionary Robotics. J Appl Res Technol. 2024;22:90–101.

[57] Haasdijk E, Eiben AE, Winfield AF. Individual Social and Evolutionary Adaptation in Collective Systems. Handb Collect Rob. 2013:413–471.

[58] Hiraga M, Wei Y, Ohkura K. Evolving Collective Cognition for Object Identification in Foraging Robotic Swarms. Artif Life Robot. 2021;26:21–28.

[59] Ajani OS, Hur SH, Mallipeddi R. Evaluating Domain Randomization in Deep Reinforcement Learning Locomotion Tasks. Mathematics. 2023;11:4744.

[60] Reda D, Tao T, van de Panne M. Learning to Locomote: Understanding How Environment Design Matters for Deep Reinforcement Learning. In Proceedings of the 13th ACM SIGGRAPH conference on motion interaction and games. 2020:1–10.

[61] Elman JL. Finding Structure in Time. Cognitive science. 1990;14:179–211.

[62] Elman JL. Distributed Representations Simple Recurrent Networks and Grammatical Structure. Machine learning. 1991;7:195–225.

[63] Frank Rosenblatt et al. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. volume 55. Spartan books Washington, DC, 1962.

[64] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. arXiv preprint: https://arxiv.org/pdf/1502.03167.

[65] El Saliby M, Nadizar G, Salvato E, Medvet E. Eventually All You Need Is a Simple Evolutionary Algorithm (For Neuroevolution of Continuous Control Policies). In Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2024:1904–1913.

[66] Eryoldaş Y, Durmuşoglu A. A Literature Survey on Offline Automatic Algorithm Configuration. Appl Sci. 2022;12:6316.

[67] Liu S, Tang K, Lei Y, Yao X. On Performance Estimation in Automatic Algorithm Configuration. In Proceedings of the AAAI Conference on Artificial Intelligence. 2020;34:2384–2391.

[68] López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T. The Irace Package: Iterated Racing for Automatic Algorithm Configuration. Oper Res Perspect. 2016;3:43–58.

[69] Rook J, López-Ibáñez M. Advanced Use of Automatic Algorithm Configuration: Single-and Multi-Objective Approaches. In Proceedings of the Genetic and Evolutionary Computation Conference Companion. 2025:1617–1642.