Benefits of Low-Cost Bio-Inspiration in the Age of Overparametrization

Kevin Godin-Dubois[1], Anil Yaman[1], and Anna V. Kononova[2]

[1] Vrije Universiteit Amsterdam, Netherlands
k.j.m.godin-dubois@vu.nl
[2] LIACS, Leiden Universiteit, Netherlands

Abstract. While Central Pattern Generators (CPGs) and Multi-Layer Perceptrons (MLP) are widely used paradigms in robot control, few systematic studies have been performed on the relative merits of large parameter spaces. In contexts where input and output spaces are small and performance is bounded, having more parameters to optimize may actively hinder the learning process instead of empowering it. To empirically measure this, we submit a given robot morphology, with limited proprioceptive capabilities, to controller optimization under two bio-inspired paradigms (CPGs and MLPs) with evolutionary- and reinforcement-trainer protocols. By varying parameter spaces across multiple reward functions, we observe that shallow MLPs and densely connected CPGs result in better performance when compared to deeper MLPs or Actor-Critic architectures. To account for the relationship between said performance and the number of parameters, we introduce a Parameter Impact metric which demonstrates that the additional parameters required by the reinforcement technique do not translate into better performance, thus favouring evolutionary strategies.

Keywords: Central Pattern Generator · Multi-Layers Perceptron · Evolutionary Strategies · Reinforcement Learning · Overparametrization

1 Introduction

In Evolutionary Robotics, bodies and brains co-evolve in a mutually dependent fashion, thus every change in the former has to be accommodated by the latter [7]. To address this issue, researchers have devised numerous encodings for robot morphologies, ranging from Lindenmayer Systems [17,27] to Composite Pattern Producing Networks (CPPNs), [8,41] and for controllers, such as piece-wise splines [20], Central Pattern Generators [23,40] or Artificial Neural Networks [9,26]. However, there are no clear guidelines on how to choose between different types of controllers, or how to best design an architecture for a robot with limited sensory capabilities.

To fill this knowledge gap, this work[3] is devoted to exploring the impact of high-level choices on the performance of evolved gaits in a directed locomotion

[3] Sources: https://github.com/kgd-al/apets-ariel/releases/tag/PPSN2026

task. The robot used in this study is a specific instance of modular robots [10] composed of a central control module, 8 articulations and 8 structural blocks. Each articulation possesses a single degree of freedom and provides a read out of their current position in one-dimensional local coordinates. The controller thus works with an 8-dimensional input and output space, the later being on par with existing benchmarks, while the former is much smaller. As a comparison, a similar morphology widely used as a Reinforcement Learning (RL) benchmark [32] has the same number of actuators but relies on 105 observations, including global rotation, hinge local positions and velocities and contact forces exerted on the body. This difference in input space stems directly from our focus on modular robotics, where components are low-cost and generic, as opposed to custom-built high-end alternatives found, e.g. in unitree products [4].

Thus, through systemic comparisons, this study contributes the following:

- Central Pattern Generators (CPGs) and Multi-Layer Perceptrons (MLPs) are capable of similar levels of performance but differ in their parameter-wise efficiency;
- the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) performs overall better than Proximal Policy Optimization (PPO) when considering efficiency;
- all three investigated reward functions lead to very different outcomes in terms of relative performance and transfer learning capacity.

2 Literature

Designing controllers for autonomous robots has been extensively studied in numerous fields including, but far from limited to, Evolutionary Computation [1] and Reinforcement Learning. In this work, we focus on these two specific research fields as both have been successfully used in an Evolutionary Robotics (ER) context [10]. Indeed, simultaneously optimizing both robot's body and its driving mechanism leads to many, currently unsolved, problems [7] stemming from potential mismatches and short-term performance loss. However, in ER, CPGs and MLPs stand out as the two major design principles for generic controllers with many instances of the former [23,40,21,26,25] and latter [9,26] being used both with and without morphological co-evolution.

On the one hand, CPGs have been shown to be very efficient at producing biologically plausible motion patterns for e.g. snakes [24], quadrupeds [3,43,4], hexapods [42] or swimmers [19,18]. On the other hand, Multi-Layer Perceptrons have been widely used for all sorts of morphologies, in both EC [12,13,2,34] and RL [37,14,39,29] contexts. Nonetheless, while both methodologies seem similarly competent at providing viable solutions to specific optimization tasks on locomotion, they have seldom been compared in such a restricted setting. While one such work has been done in [26], the neural architectures used many more inputs than strictly necessary and only considered a single configuration for CPGs networks. Furthermore, reproducibility is hampered by the reliance on custom implementation, whereas this work will solely rely on off-the-shelf libraries with

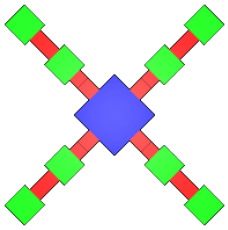


Fig. 1: Robot "spider" morphology with 8 hinges for locomotion.

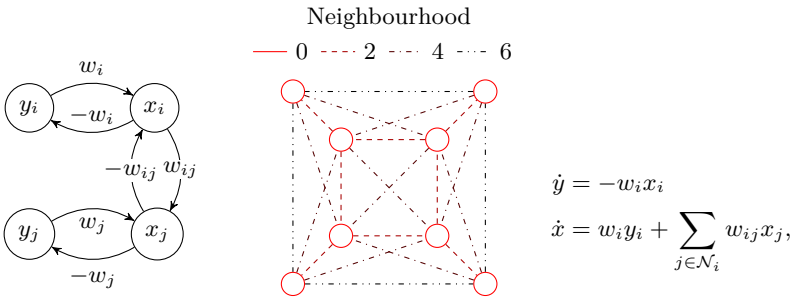
minimal changes. Another comparison has been undertaken in [40], which focuses only on CPGs with a single architecture and uses two gradient-free methods.

Another interesting dimension for comparing different architectural choices is that of the parameter space. Indeed, while current trends in Reinforcement Learning point towards using more and more parameters thanks to the observed benefits of power laws [45], this approach is far from frugal or even necessary [44]. Interestingly the wider machine learning community has recently re-invented the concept of structural bias [22] drawing attention to the fact that parameters count is only important alongside a given optimizer [30,28]. Over-parametrization, i.e. having more parameters than the number of training points, also does not apply as smoothly in robotics because the fuzzy definition of such training points in this context. A promising research direction, instead, relies on hierarchical architecture combining CPGs, for rhythmic hinge control, with MLPs, for high-level information processing [6,3,24,4,43]. Thus, parameter spaces are reduced while the intrinsic benefits of both neural paradigms can be leveraged conjointly.

3 Methods

To study the relative benefits of specific neural architectures, we use the ARIEL (formerly known as Revolve [35]) platform, which has been extensively used with both CPGs [23,40,21,26,25] and MLPs [9,26]. In this framework, robots are composed of rigid bodies, with either passive (core, brick) or active (hinges) components. The core corresponds to the real-world central processing unit containing a Raspberry Pi hardware for generic control, as well as batteries, while bricks only serve as structural components. Hinges are based on the DSS-M15 actuators augmented with a positional output. This library leverages MuJoCo [36] to handle the physical interactions in simulation to reach a high degree of robustness and reproducibility.

In this work, we focus on a traditional, so-called "spider", morphology illustrated in Figure 1. This robot has 8 active hinges, both horizontal ("hip" joints) and vertical ("knee" joints), giving it a broad range of motion up to traditional quadrupedal locomotion. It follows that the observation and action spaces have dimension 8 with values clipped in $[-1, 1]$.



(a) Paired CPGs (b) CPG Network (c) Equations system

Fig. 2: Neighbourhood configurations for a CPG network.

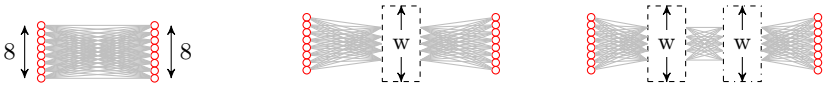
3.1 CPG controller

We rely on Central Pattern Generators (CPGs), as implemented in similar studies on the legacy version of the platform [23,40,21,26,25]. These take the form of coupled neurons x and y with recurrent connections as illustrated in Figure 2a. Traditional practices use the same weight for both connections between x and y but set one as excitatory and the other as inhibitory. This results in periodical and bounded dynamics, ideally suited for rhythmic patterns such as locomotion. However, as these neurons take no external inputs, the functions thereby produced would have very limited complexity. To alleviate this, CPGs are usually connected together to form a network where the external node (x) also receives values from neighbouring CPGs. Here, the weights are also symmetrical assigned with the connection weight w_{ji} being equal to $-w_{ij}$. The final form of the equation system for one neuron is given by Figure 2c, where $\mathcal{N}_i$ is the neighbourhood of CPG i. Usually, two neurons are said to be neighbours if their distance in the morphological tree is less than or equal to two [25]. However, in this work, we consider neighbourhood distance as a hyperparameter of the CPG network and vary it across its range, as illustrated on Figure 2b.

The simplest network, thereafter designed c_0, has no neighbourhood and thus only uses the first terms in Figure 2c. This, in turn, implies that the optimization process will have only 8 parameters to work with, reducing the search space but not necessarily the quality of the solutions. The remaining sub-architectures (c_2, c_4, c_6) have respective neighbourhood ranges of 2, 4 and 6. As only the distance between hinges is considered, c_0 and c_1 are functionally equivalent as no hinges are directly connected to one another in the spider morphology. Thus, the largest parameter space is 36 for the fully connected c_6, using all feedback pathways depicted on Figure 2b.

3.2 ANN controller

In a similar fashion to the CPG architecture, we rely on out-of-the-box implementations for the Artificial Neural Network controllers. As will be detailed later on,



(a) Perceptron m_0^0 (b) Single hidden layer m_w^1 (c) 2 hidden layers m_w^2

Fig. 3: ANN architectures with w denoting the width of hidden layers.

we use Stable Baselines3 [31] for our RL training, which also provides a default network architecture. In its baseline configuration, ANNs used in conjunction with PPO [33] have two layers of 64 neurons with hyperbolic tangent activation function except for the output layer. There, the network outputs are given by a Gaussian distribution, during training, which simplifies to the identity function during deterministic evaluation.

While, compared to CPGs, the hyperparameter space here is much richer, we focus our investigation on the direct impact of the optimized parameters. Only the depth (d) and width (w) of networks are varied with $d \in \{0, 1, 2\}$ and $w \in \{1, 2, 4, 8, 16, 32, 64, 128\}$. Furthermore, we denote a Multi-Layer Perceptron with a given depth and width as m_w^d. Note that this labelling is expedient but somewhat imprecise, when referring to the sub-architecture m_0^0 as these are functionally perceptrons, as can be seen in Figure 3a.

For $d \in \{1, 2\}$, the ANN can accurately be called an MLP with number of parameters summarized in Table 1. As will be discussed in subsection 3.4, the major difference between our optimization algorithms is the use of an actor-critic policy with Reinforcement Learning which exponentially increases the parameter space's size.

3.3 Fitness/Reward functions

In addition to varying hyperparameters of both CPGs and MLPs, we also look at the potential impact of the training regime, that is the function defining

Table 1: Parameter spaces for CPGs with neighbourhood range $\mathcal{N}$ and for MLPs with width w and depth d. Note that PPO has more parameters because of its actor and critic components.

(a) CPG

	$\mathcal{N}$	Parameters
c_0	0	8
c_2	2	18
c_4	4	30
c_6	6	36

(b) MLP

Name	d	w	Parameters	
			CMA-ES	PPO
m_0^0	0	N/A	72	89
m_w^1	1	n	$17w + 8$	$27w + 17$
m_w^2	2	n	$(18 + w)w + 8$	$(29 + 2w)w + 17$

performance. In this work, we use three methods that favour different gaits and learning trajectories, as discussed below.

Speed The most straightforward reward function is the speed along the x-axis. Given a robot at x positions $x(t)$ and $x(t+1)$ at times t and $t+1$, respectively, we define its reward R_s as:

$$R_s(t) = x_(t+1) - x(t). \tag{1}$$

The corresponding fitness function, for use with evolutionary algorithms is obtained by summing of all time steps. Thus, for a simulation of T seconds, with sampling rate of 20 Hz $R_s = \sum_{0 \leq t < T} R_s(t)$. The same methodology applies to the subsequent reward/fitness function pairs and will thus not be repeated.

Gymnasium is a collection of environments actively used for benchmarking Reinforcement Learning algorithms [38]. Through independent convergence of designs, the ant[4] morphology of this benchmark collection happens to be very similar to ARIEL's spider [23]. Thus, with minimal tweaking, it was possible to port the reward function used in this environment to train controllers that, theoretically, have a more frugal use of hinges. Given, at time t, a robot with forward velocity $V_x(t)$, applying a control signal $\boldsymbol{S}(t)$ to its hinges and receiving contact forces $\boldsymbol{F}(t)$, its so-called gymnasium reward $R_g(t)$ is defined as:

$$R_g(t) = V_x(t) - .5||S(t)||_2 - .0005||F(t)||_2, \tag{2}$$

where $||.||_2$ represents the Euclidean distance (L2 norm). Succinctly, this function promotes forward locomotion, as does R_s, but also penalizes large control forces, thereby favouring frugal locomotion, as well as contact forces, i.e. not hitting the ground hard.

Kernels-based The third and final reward function is based on Radial Basis Function (RBF) kernels as they provide a very distinct reward landscape. Given that the robot, at time t, has velocity $V(t) = (V_x(t), V_y(t), V_z(t))$ and elevation $z(t)$, we define its kernel-based reward $R_k$as:

$$\begin{aligned} K(v, \hat{v}, c) &= e^{-c(v-\hat{v})^2} \\ R_k(t) &= \frac{5}{8}K(V_x(t), 0.5, 25) + \frac{1}{8}K(V_y(t), 0, 5) \\ &\quad + \frac{1}{8}K(V_z(t), 0, 5) + \frac{1}{8}K(z(t), 0.2, .002), \end{aligned} \tag{3}$$

where $K(v, \hat{v}, c)$ defines a radial basis function with target value $\hat{v}$ and slope c. With this function, the robot is thus encouraged to reach the target velocity (50 cm/s) with no lateral or vertical speed. Furthermore, the body is expected to stay at an elevation of 20 cm, corresponding to 10 cm of distance between the bottom of the core and the ground which would penalize crawling gaits.

[4] https://gymnasium.farama.org/environments/mujoco/ant/

3.4 Optimization setup

The final variable of adjustment investigated in this work is the optimization algorithms. On the one hand, we use the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [15,16], a powerful gradient-free optimization method successfully used in conjunction with this morphological space [9]. Following usual practice with the former version of the framework, we keep all hyperparameters to their default configuration as detailed in the supplementary material [11]. As the method is gradient-free, we used it for both CPGs and MLPs optimization forming two broad groups which we will reference as **CMA/CPG** and **CMA/MLP** from now on.

Individuals were evaluated for 10 seconds each, with a control frequency of 20 Hz resulting in 200 time steps per simulation. Every run consisted of 10'000 such evaluations with 10 replicates for every condition. However, as we relied on an off-the-shelf naive implementation, the algorithm was unable to successfully explore large parameter spaces leading to prohibitive matrix size for the largest MLP architectures (more than 5Gb for m^2_{128}) and leading to premature convergence. This led to the exclusion of m^1_{128}, m^2_{64}, m^2_{128} to limit resources wastage although some of the included runs still exhibited this form of premature convergence.

On the other hand, we also used Proximal Policy Optimization [33] to train MLPs via a form of gradient descent as also done with the library [26]. As this method is inefficient with recurrent networks, no off-the-shelf solutions existed for CPGs leading to only one group for this trainer: **PPO/MLP**. The algorithm does not rely on episodic learning but, instead, works at the time step level. Thus to provide a comparable learning budget the CMA group, we let PPO run for 2'000'000 time steps of the environment, with resets every 10 s (200 time steps). With respect to the hyperparameters, we use those recommended for gymnasium ant[5] which, as already stated, has similar morphological features.

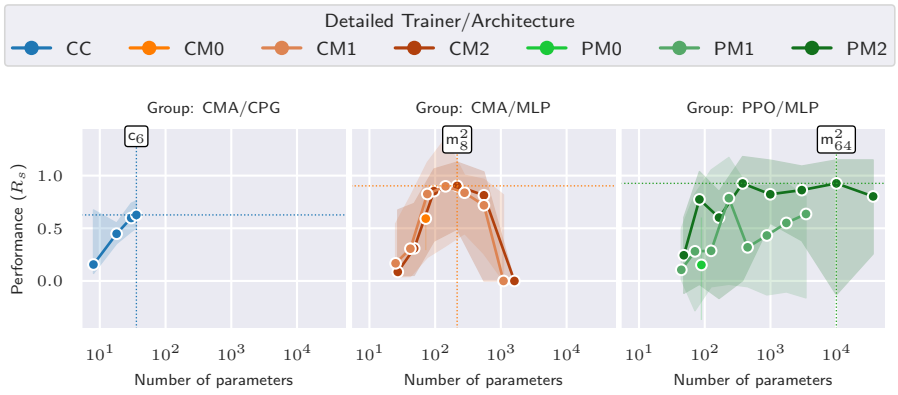
4 Results

4.1 Relative performance

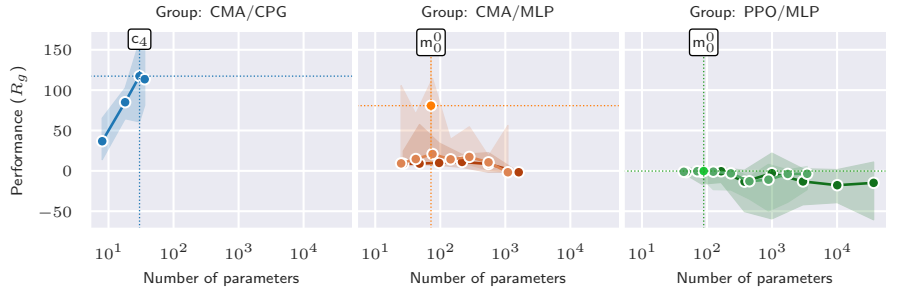
The first dimension we use to discriminate between the different sub-architectures is the raw performance on each reward function. As summarized in Figure 4, each training regime reacted very differently each scheme, with the exception of CPGs (CC), which generally benefits from having more parameters.

MLPs trained with CMA-ES, instead, exhibit a bell-shaped curve for both R_s and R_k, highlighting its inadequacy with high-dimensionality spaces. While this does not reflect on the optimal performance that could be achieved with a version of CMA-ES more suited to large search spaces, this does show what can be achieved with out-of-the-box solutions. Furthermore, as the performance of PPO is either on par (R_s) or lower (R_g, R_k) than that of CMA-ES, we can

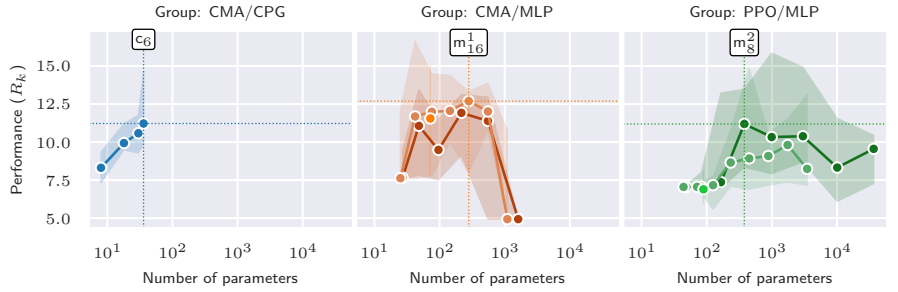
[5] https://github.com/araffin/rl-baselines-zoo/blob/master/hyperparams/ppo2.yml



(a) R_s



(b) R_g



(c) R_k

Fig. 4: Relationship between performance and number of parameters for all configurations and three reward functions. CPGs trained with CMA-ES are displayed in blue, labeled CC. MLP trained with CMA-ES and PPO are labeled CMi and PMi, respectively, for a depth of d. Sub-architectures with the highest median are annotated in each group and projections are provided for comparison. Trends show comparable performance on R_k, MLP dominance for R_s and CPG dominance for R_g. Generally, bigger CPGs and smaller MLPs perform better.

reasonably hypothesize that the parameter space of MLPs is already explored close enough to near-optimal performance.

Indeed, one can see on Figure 4a that the best performing architecture for MLP is either m_8^2 (2 hidden layers of 8 neurons each) or m_{64}^2 depending on whether the training was performed with CMA-ES or PPO. However, increasing the parameters by a factor of almost 50 only changed the performance by 3%. With respect to R_k, the general trend is similar but with an important difference on PPO saturation: with this reward function, the larger networks do not even reach the performance of the smaller ones. Given the relative straightforwardness of the task, this goes to show how overparametrization can be detrimental to the learning process, when working under a similar budget.

The case of the gymnasium-based fitness is a particularly interesting one as it should have been more beneficial to training ANNs with RL, given that it is what it was designed for. However, looking at Figure 4b, one can see that MLPs perform very poorly with only CMA-trained perceptrons achieving meaningful rewards. While this comes across as a counter-intuitive result, one must recall that R_g actively penalizes large motions, making the initial learning steps that much harder. This problem does not come into play as strongly when considering CPGs which, for better or worse, have a very hard time not generating rhythmic patterns. Thus, the initial hurdle of discovering a viable compromise between frugality and locomotion has been much easier for CPGs than for MLPs.

However, to identify what architecture/trainer combination performs best for this locomotion task we now only look at the values obtained by the best sub-architecture. Violin plots available in the supplementary material [11] summarizes these values for the three sub-architectures with the highest median, as identified previously, for each reward scheme.

As mentioned before, CPGs strongly benefit from having more parameters and, with the exception of R_g, should be used in a fully connected setting. MLPs are a different thing as CMA-ES favors small networks (m_8^2 and m_2^1 for R_s and R_k, respectively) while PPO is more dispersed (m_{64}^2 and m_8^2 for R_s and R_k). Furthermore, the lack of performance for actual Multi-Layer Perceptrons in the gymnasium case led to dominance of pure Perceptrons, clearly so for CMA-ES and more tenuously for PPO.

In terms of relative performance, MLPs are valid choices with both training methods when aiming for pure speed as they statistically outperform CPGs. Conversely, the counter-intuitive shaping of R_g made it so that frugal controllers are better modelled by CPGs or pure perceptrons. Finally, when targeting a more balanced gait, as promoted by R_k, performance is not enough to make an informed choice. All results stated here are summarized in Table 2a for easier comparison.

4.2 Parameter Impact

In the context of robotics, raw task performance may not be the only metric by which policies are compared. Indeed, efficiency may also play an important role, whether in terms of energy consumption, battery longevity or, as it interests

us here, the parameter space. As we have seen previously, a large parameter space does not correlate with better performance. This links back to the low dimensionality of our observation and actions spaces, making it possible to have prohibitively high numbers of parameters.

To better capture the relationship between parameter count and performance and to empirically measure over-parametrization instead of defining it a priori, we consider the Parameter Impact $I_F(f, p)$ as:

$$I_F(f, p) = \hat{f}/log_{10}(p) \tag{4}$$

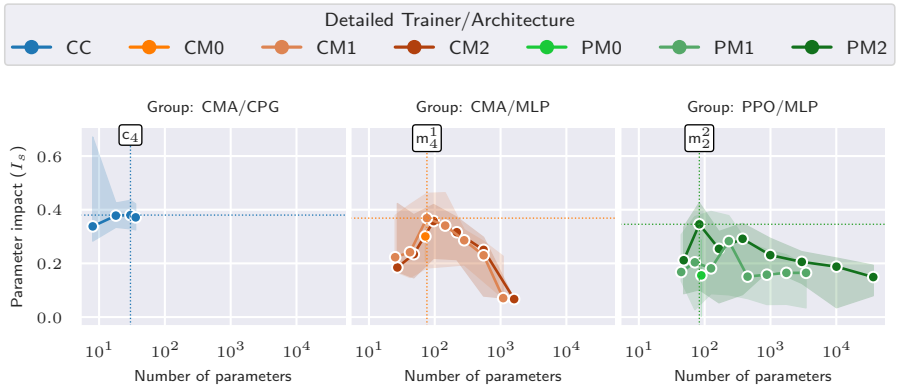
where p is the number of parameter in the policy and $\hat{f}$ is the normalized performance, obtained by uniformly scaling f with respect to the performance distribution of individuals also trained with reward function F. Given the three reward functions previously introduced, we have as many Parameter Impact metrics I_s, I_g, I_k normalizing the performance of R_s, R_g and R_k, respectively. This metric thus transforms performances into ratios, answering the question: relative to the global population, how much does the chosen parametrization help to get a good performance?

Applying this metric to our dataset results in Figure 5, where one can better compare how much increasing parameters benefits performance, even across reward functions. As the Parameter Impact only acts as a logarithmic scaling, the performances are qualitatively similar but their relationships across groups and reward schemes are not.

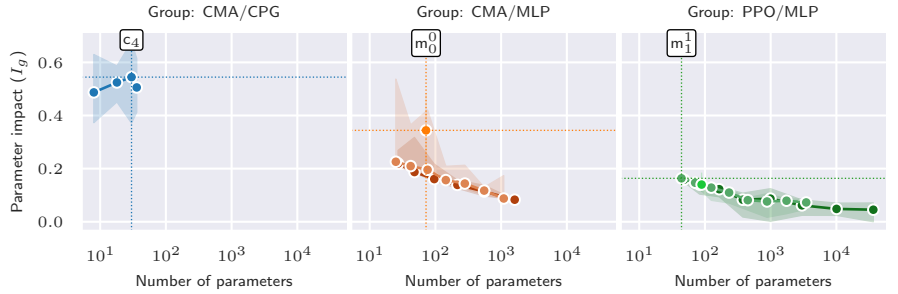
Indeed, the trends that were already visible with the raw performance are more easily accessible in this scaled down format. Notably, one can see how the 47% increase in R_s between CPGs and PPO-trained MLP is made with a parameter increase of 27958% (10065 versus 36). When scaled down with I_s, one can see that the latter was marked as less than half the efficiency of the former. Overall, this leads to changes in the best performing architectures with c_4 taking the lead for the CPGs with its almost-fully connected topology. On the one hand, MLPs trained with CMA-ES favour even smaller topologies than when looking at their raw performance, except for $I_g$which was only solved by perceptrons. On the other hand, PPO produced more efficient networks at low scales with best performing architectures being m_2^2, m_1^1, m_8^2 for I_s, $I_g$and I_k, respectively.

As before, we summarize these results in Table 2a and [11]. As opposed to results for raw performance, no architecture appears as more efficient, given the formulation of our Parameter Impact I_*. It even further magnifies the difference between CMA-ES and PPO training in case of I_g as performance is somehow invertly proportional to the number of parameters, making the former an objectively better choice. This also holds for I_k, to a lesser degree, with PPO displaying similar performance while having more parameters.

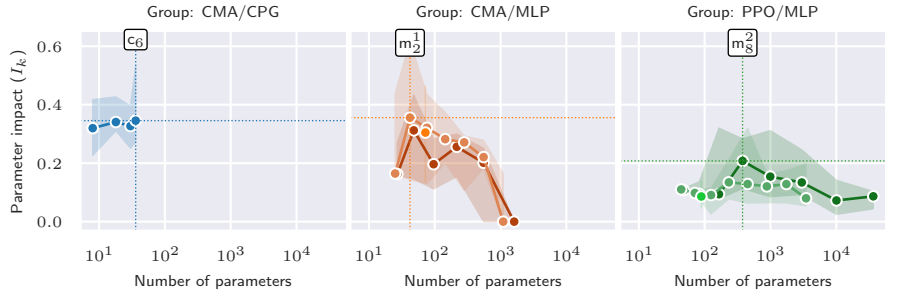
Thus, while no architecture/trainer pairing seem more efficient when aiming for pure speed, we can conclude, based on the statistical differences, that both CPGs and MLPs trained with CMA-ES perform similarly on kernel-based training. Furthermore, due to the inability of the other architectures to learn on R_g,



(a) R_s



(b) R_g



(c) R_k

Fig. 5: Parameter impact across all configurations and reward functions. Color scheme is identical as Figure 4. Sub-architectures with the highest median are also annotated. Trends show comparable performance on I_s, CPG dominance for I_g and CMA-ES dominance for I_k. CPGs with almost fully connection topologies (c_4) and very small MLPs architectures stand out.

Table 2: Overview of the best architectures for each metric in coloured bold.

(a) Performance & Parameter Impact

		R_s	R_g	R_k	I_s	I_g	I_k
CMA	**CPG**	0.63 c_6	**117.34** **c_4**	11.22 c_6	**0.38** **c_4**	**0.54** **c_4**	0.35 c_6
	MLP	0.90 m_8^2	80.78 m_0^0	**12.69** **m_{16}^1**	0.37 m_4^1	0.34 m_0^0	**0.36** **m_2^1**
PPO	**MLP**	**0.93** **m_{64}^2**	-0.23 m_0^0	11.19 m_8^2	0.35 m_2^2	0.16 m_1^1	0.21 m_8^2

(b) Cross-performance

	R_s		R_g		R_k	
	R_g	R_k	R_s	R_k	R_s	R_g
	0.64 **c_4**	0.25 c_6	**113.81** **c_6**	**42.78** **c_6**	7.81 c_2	6.99 c_0
	0.55 m_0^0	0.26 m_{32}^1	86.02 m_4^1	17.13 m_0^0	**7.89** **m_2^2**	**7.01** **m_8^1**
	0.02 m_{128}^2	**0.31** **m_8^2**	97.69 m_8^2	31.27 m_8^2	6.98 m_1^2	5.89 m_{32}^1

the former is also the better choice when optimizing for more frugal controllers. These results are numerically summarized in Table 2a.

4.3 Cross-performance

While we extensively investigate the relative performance of the various architectures on multiple reward functions, we have one more area to search for potentially meaningful differences: their cross-performance. Indeed, one might wonder whether some architectures are better at exhibiting a range of interesting behaviour, even when not explicitly selecting for them.

Table 2b provides an overview of the best performance of a given architecture when evaluated with a different reward function than the one it has been trained with. From it, we can gather that, overall, CPGs seem to perform better across the board especially with respect to R_g. Indeed, MLPs have not only failed to perform well in this specific metric but, due to the counter-productive penalty on control signals, converge towards very limited ranges of motion. Thus, even best-performing individuals end being very bad at both moving forward (R_s) or adopting a stable gait (R_k).

Complementarily, we can also observe that training directly for stable gait has interesting side effect when paired with CMA-ES, as it also provides individuals with slightly better performance over the remaining two functions. This however, is compounded by the relative proximity of CPGs on those same functions.

Thus, while CPGs do not clearly stand out in terms of pure performance, they do exhibit an overall better tendency at ranking relatively high across the board. This, as we stated before, is to be linked with built-in bias toward rhythmic patterns which, besides favouring locomotion, are indirectly rewarded in all considered fitness functions.

4.4 Diversity

Beyond performance, whether scaled or not, we finally address a potentially crucial component of the various architectures: their potential for producing

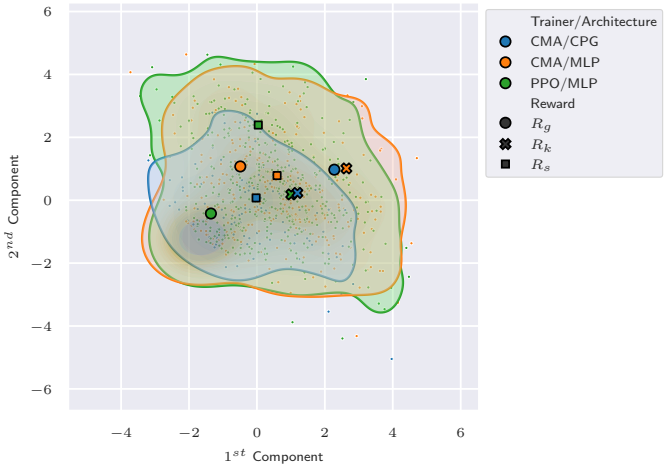


Fig. 6: PCA of the 16D diversity space obtained by fitting sinusoidals onto each of the robot's foot. Explained variance ratios are 25.6% and 22.8%. Overall best performing individuals are highlighted to show the empirical diversity.

diverse gaits. To do so while introducing as little bias as possible, we monitor the position of each of the robot's extremum brick. Assuming a smooth gait, these should also follow a rhythmic pattern with each "foot" being raised in turn to maximize locomotion, as is seen in biological gaits [5]. However, as solutions have not been constrained to follow such a rule we instead model feet independently as a sinusoidal of the form:

$$f_i(t) = A_i sin(\omega_i t + \phi_i) + c_i. \tag{5}$$

This is then fitted to the actual trajectory of foot i using the scipy library to provide estimated parameters. To remove dubious fits, we constrain $A_i < 0.5$, $c_i < 0.5$ and $|\frac{w_i}{2\pi}| < 0.25$, and set all parameters to 0 whenever any constraint is not met. This prevents immobile feet from being associated with curves with very high parameters.

Given that the robot has four feet, this makes it possible to represent every individual in a 16-dimensional space describing their intrinsic gait (or lack thereof). This space has then been reduced to 2 dimensions via Principal Component Analysis and plotted in Figure 6. While the explained variance ratios are admittedly low (25.6% and 22.8%), this provides a measure of understanding into how diverse the solutions are, given a specific architecture/trainer pair. We can see that both MLP variations occupy a larger area. While this does not come at a surprise, given that CPGs have fewer parameters and a built-in tendency towards specific patterns, this serves as an empirical demonstration of just such a behaviour. Whether or not this is a limitation, however, is open to debate. Indeed, even though CPGs are less capable of generating as wide a range of gait

as MLPs, this same limitation also causes its higher overall cross-performance ranking.

However, the curious reader can also look at videos[6] of the champions highlighted on Figure 6, to further inform their choice between performance, efficiency and diversity.

5 Discussion

In this work, we systematically compare the performance of 17 variations of different neural architectures (CPGs and MLPs), with two training algorithms (CMA-ES and PPO), on three reward functions favouring different gaits (fast, frugal, stable). These comparisons were performed using off-the-shelf state-of-the-art algorithms to provide an easily replicable context which future experimenters could compare with.

Results show that, overall, low parameter counts led to better performance especially when aiming for more balanced behaviour (frugal, stable) than exploitive ones (speed). CPGs showed a marked preference for the upper ranges of their parameter space with classes c_4 (30 parameters) and c_6 (36) outperforming loosely connected alternatives. MLPs performed best with very shallow networks as champion architectures use between 0 and 16 hidden neurons, in either one or two layers.

In terms of training algorithm, CMA-ES had better all-around outcomes, in part thanks to its gradient-free approach which made it possible to train CPGs and to save on the parameter cost of a critic. Indeed, when looking at networks' efficiency via our Parameter Impact metric, we see that shallow MLPs and CPGs strike the optimal balance between expressivity and performance. However, that very same low parameter count also make CPGs less proficient as generating diverse gaits, as observed when looking at the distribution of fitted sinusoidal parameters. While there are no current off-the-shelf solutions for CPG training with RL methods, the marked interest from the research community shows that this might change in the near future. An extension of this study to encompass this forth alternative would further demonstrate whether CPGs are an efficient, low-parameter, paradigm. Additionally, the soundness of these results should be tested on different morphologies both within the same platform and without.

Acknowledgments. This research was funded by the Hybrid Intelligence Center, a 10-year programme funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research, https://hybrid-intelligence-centre.nl, grant number 024.004.022.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

[6] https://vimeo.com/user/53652387/folder/28945427

References

1. Bäck, T.H., Kononova, A.V., van Stein, B., Wang, H., Antonov, K.A., Kalkreuth, R.T., de Nobel, J., Vermetten, D., de Winter, R., Ye, F.: Evolutionary algorithms for parameter optimization—thirty years later. Evolutionary Computation **31**(2), 81–122 (2023). https://doi.org/10.1162/evco_a_00325
2. Baldominos, A., Saez, Y., Isasi, P.: On the automated, evolutionary design of neural networks: past, present, and future. Neural Computing and Applications **32**(2), 519–545 (Jan 2020). https://doi.org/10.1007/s00521-019-04160-6
3. Bellegarda, G., Ijspeert, A.: CPG-RL: Learning Central Pattern Generators for Quadruped Locomotion (Nov 2022). https://doi.org/10.48550/arXiv.2211.00458, arXiv:2211.00458 [cs]
4. Bellegarda, G., Shafiee, M., Ijspeert, A.: Visual CPG-RL: Learning Central Pattern Generators for Visually-Guided Quadruped Locomotion (Mar 2024). https://doi.org/10.48550/arXiv.2212.14400, arXiv:2212.14400 [cs]
5. Bhattasali, N.X., Pattabiraman, V., Pinto, L., Lindsay, G.W.: Neural Circuit Architectural Priors for Quadruped Locomotion (Oct 2024). https://doi.org/10.48550/arXiv.2410.07174, arXiv:2410.07174 [q-bio]
6. Campanaro, L., Gangapurwala, S., Martini, D.D., Merkt, W., Havoutis, I.: CPG-ACTOR: Reinforcement Learning for Central Pattern Generators (Feb 2021). https://doi.org/10.48550/arXiv.2102.12891, arXiv:2102.12891 [cs]
7. Cheney, N., Bongard, J., Sunspiral, V., Lipson, H.: On the Difficulty of Co-Optimizing Morphology and Control in Evolved Virtual Creatures. Proceedings of the Artificial Life Conference 2016 (ALIFE XV) pp. 226–234 (2016). https://doi.org/10.1162/978-0-262-33936-0-ch042
8. Cheney, N., MacCurdy, R., Clune, J., Lipson, H.: Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding. Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation - GECCO '13 p. 167 (2013). https://doi.org/10.1145/2463372.2463404, iSBN: 9781450319638
9. van Diggelen, F., De Carlo, M., Cambier, N., Ferrante, E., Eiben, A.E.: Emergence of specialized Collective Behaviors in Evolving Heterogeneous Swarms. In: Parallel Problem Solving from Nature – PPSN XVIII, vol. 1 (Feb 2024). https://doi.org/10.1007/978-3-031-70068-2_4, arXiv: 2402.04763
10. Eiben, A.E.: EvoSphere: the World of Robot Evolution. In: Theory and Practice of Natural Computing, vol. 9477, pp. 3–19. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-26841-5_1, series Title: Lecture Notes in Computer Science
11. Godin-Dubois, K.: Dataset on the benefits of low-cost bio- inspiration in the age of overparametrization (Apr 2026). https://doi.org/10.5281/zenodo.19633625
12. Godin-Dubois, K., Cussat-Blanc, S., Duthen, Y.: Explaining the Neuroevolution of Fighting Creatures Through Virtual fMRI. Artificial Life **29**(1), 66–93 (2023). https://doi.org/10.1162/artl_a_00389, tex.type: journal
13. Godin-Dubois, K., Cussat-Blanc, S., Duthen, Y.: Specialization or Generalization: Investigating NeuroEvolutionary Choices via Virtual fMRI. MIT Press (Jul 2024). https://doi.org/10.1162/isal_a_00817, tex.type: conference
14. Godin-Dubois, K., Miras, K., Kononova, A.V.: AMaze: a benchmark generator for sighted maze-navigating agents. Journal of Open Source Software (2025). https://doi.org/10.21105/joss.07208, tex.type: journal

15. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE international conference on evolutionary computation. pp. 312–317 (1996). https://doi.org/10.1109/ICEC.1996.542381
16. Hansen, N., Akimoto, Y., Baudis, P.: CMA-ES/pycma on Github (Feb 2019). https://doi.org/10.5281/zenodo.2559634, tex.howpublished: Zenodo, DOI:10.5281/zenodo.2559634
17. Hornby, G.S., Pollack, J.B.: Evolving L-systems to generate virtual creatures. Computers and Graphics (Pergamon) **25**(6), 1041–1048 (2001). https://doi.org/10.1016/S0097-8493(01)00157-1, arXiv: 1011.1669v3 ISBN: 0097-8493
18. Ijspeert, A.J.: Central pattern generators for locomotion control in animals and robots: A review. Neural Networks **21**(4), 642–653 (May 2008). https://doi.org/10.1016/j.neunet.2008.03.014
19. Ijspeert, A.J., Hallam, J., Willshaw, D.: Evolving Swimming Controllers for a Simulated Lamprey with Inspiration from Neurobiology. Adaptive Behavior **7**(2), 151–172 (Mar 1999). https://doi.org/10.1177/105971239900700202
20. Jelisavcic, M., de Carlo, M., Hupkes, E., Eustratiadis, P., Orlowski, J., Haasdijk, E., Auerbach, J.E., Eiben, A.E.: Real-World Evolution of Robot Morphologies: A Proof of Concept. Artificial Life **23**(2), 206–235 (May 2017). https://doi.org/10.1162/ARTL_a_00231
21. Jelisavcic, M., Glette, K., Haasdijk, E., Eiben, A.E.: Lamarckian Evolution of Simulated Modular Robots. Frontiers in Robotics and AI **6**, 9 (Feb 2019). https://doi.org/10.3389/frobt.2019.00009
22. Kononova, A.V., Corne, D.W., De Wilde, P., Shneer, V., Caraffini, F.: Structural bias in population-based algorithms. Information Sciences **298**, 468–490 (Mar 2015). https://doi.org/10.1016/j.ins.2014.11.035
23. Lan, G., Van Hooft, M., De Carlo, M., Tomczak, J.M., Eiben, A.: Learning locomotion skills in evolvable robots. Neurocomputing **452**, 294–306 (Sep 2021). https://doi.org/10.1016/j.neucom.2021.03.030
24. Liu, X., Onal, C., Fu, J.: Reinforcement Learning of CPG-regulated Locomotion Controller for a Soft Snake Robot. IEEE Transactions on Robotics **39**(5), 3382–3401 (Oct 2023). https://doi.org/10.1109/TRO.2023.3286046, arXiv:2207.04899 [cs]
25. Luo, J., Stuurman, A.C., Tomczak, J.M., Ellers, J., Eiben, A.E.: The Effects of Learning in Morphologically Evolving Robot Systems. Frontiers in Robotics and AI **9**, 797393 (May 2022). https://doi.org/10.3389/frobt.2022.797393, arXiv:2111.09851
26. Luo, J., Tomczak, J., Miras, K., Eiben, A.E.: A comparison of controller architectures and learning mechanisms for arbitrary robot morphologies (Sep 2023). https://doi.org/10.48550/arXiv.2309.13908, arXiv:2309.13908 [cs]
27. Miras, K., Haasdijk, E., Glette, K., Eiben, A.E.: Search Space Analysis of Evolvable Robot Morphologies. In: Sim, K., Kaufmann, P. (eds.) Applications of Evolutionary Computation, vol. 10784, pp. 703–718. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-77538-8_47, series Title: Lecture Notes in Computer Science
28. Mohan, D., Scaife, A.M.M.: Natural gradient descent for improving variational inference based classification of radio galaxies (Nov 2025). https://doi.org/10.48550/arXiv.2511.13224, arXiv:2511.13224 [astro-ph]
29. OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson,

C., Pachocki, J., Petrov, M., Pinto, H.P.d.O., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., Zhang, S.: Dota 2 with Large Scale Deep Reinforcement Learning (Dec 2019), https://arxiv.org/abs/1912.06680v1, arXiv: 1912.06680

30. Pascanu, R., Lyle, C., Modoranu, I.V., Borras, N.E., Alistarh, D., Velickovic, P., Chandar, S., De, S., Martens, J.: Optimizers Qualitatively Alter Solutions And We Should Leverage This (Jul 2025). https://doi.org/10.48550/arXiv.2507.12224, arXiv:2507.12224 [cs]
31. Raffin Antonin, Hill Ashley, Gleave Adam, Kanervisto Anssi, Ernestus Maximilian, Dormann Noah: Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research **22**(268), 1–8 (2021), http://jmlr.org/papers/v22/20-1364.html
32. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-Dimensional Continuous Control Using Generalized Advantage Estimation (Oct 2018). https://doi.org/10.48550/arXiv.1506.02438, arXiv:1506.02438 [cs]
33. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms pp. 1–12 (Jul 2017), http://arxiv.org/abs/1707.06347, arXiv: 1707.06347
34. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence **1**(1), 24–35 (2019). https://doi.org/10.1038/s42256-018-0006-z
35. Stuurman, A., Weissl, O., Chiang, T.C., AndresG, Zeeuwe, D., Godin-Dubois, K., Roy: ci-group/revolve2: 1.2.3 (Nov 2024). https://doi.org/10.5281/ZENODO.14143431, tex.type: software
36. 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. pp. 5026–5033. IEEE (Oct 2012). https://doi.org/10.1109/IROS.2012.6386109
37. Tomilin, T., Dai, T., Fang, M., Pechenizkiy, M.: LevDoom: A Benchmark for Generalization on Level Difficulty in Reinforcement Learning. In: 2022 IEEE Conference on Games (CoG). pp. 72–79. IEEE (Aug 2022). https://doi.org/10.1109/CoG51982.2022.9893707
38. Towers, M., Kwiatkowski, A., Terry, J., Balis, J.U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J.J., Tan, H., Younis, O.G.: Gymnasium: A Standard Interface for Reinforcement Learning Environments (Jul 2024), http://arxiv.org/abs/2407.17032, arXiv:2407.17032 [cs]
39. Tsounis, V., Alge, M., Lee, J., Farshidian, F., Hutter, M.: DeepGait: Planning and Control of Quadrupedal Gaits using Deep Reinforcement Learning (2019). https://doi.org/10.48550/ARXIV.1909.08399, version Number: 2
40. Van Diggelen, F., Ferrante, E., Eiben, A.E.: Comparing lifetime learning methods for morphologically evolving robots. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 93–94. ACM, Lille France (Jul 2021). https://doi.org/10.1145/3449726.3459530
41. Veenstra, F., Hart, E., Buchanan, E., Li, W., De Carlo, M., Eiben, A.E.: Comparing encodings for performance and phenotypic exploration in evolving modular robots. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 127–128. ACM, Prague Czech Republic (Jul 2019). https://doi.org/10.1145/3319619.3322054

42. Wang, G., Chen, X., Han, S.K.: Central pattern generator and feedforward neural network-based self-adaptive gait control for a crab-like robot locomoting on complex terrain under two reflex mechanisms. International Journal of Advanced Robotic Systems **14**(4), 172988141772344 (Jul 2017). https://doi.org/10.1177/1729881417723440
43. Watanabe, T., Kubo, A., Tsunoda, K., Matsuba, T., Akatsuka, S., Noda, Y., Kioka, H., Izawa, J., Ishii, S., Nakamura, Y.: Hierarchical reinforcement learning with central pattern generator for enabling a quadruped robot simulator to walk on a variety of terrains. Scientific Reports **15**(1), 11262 (Apr 2025). https://doi.org/10.1038/s41598-025-94163-2
44. Wong, A., Nobel, J.d., Bäck, T., Plaat, A., Kononova, A.V.: Solving Deep Reinforcement Learning Tasks with Evolution Strategies and Linear Policy Networks (Jul 2024). https://doi.org/10.48550/arXiv.2402.06912, arXiv:2402.06912 [cs]
45. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization (Feb 2017). https://doi.org/10.48550/arXiv.1611.03530, arXiv:1611.03530 [cs]