Evolving Reservoirs for Meta Reinforcement Learning

Corentin Léger[1,2](✉), Gautier Hamon[1], Eleni Nisioti[1], Xavier Hinaut[2], and Clément Moulin-Frier[1]

[1] Flowers Team, Inria and Ensta ParisTech, Bordeaux, France
`corentin.leger@inria.fr`

[2] Mnemosyne Team, Inria; LaBRI, Univ. Bordeaux, Bordeaux INP, CNRS UMR 5800; Univ. Bordeaux, CNRS, IMN, UMR 5293, Bordeaux, France

Abstract. Animals often demonstrate a remarkable ability to adapt to their environments during their lifetime. They do so partly due to the evolution of morphological and neural structures. These structures capture features of environments shared between generations to bias and speed up lifetime learning. In this work, we propose a computational model for studying a mechanism that can enable such a process. We adopt a computational framework based on meta reinforcement learning as a model of the interplay between evolution and development. At the evolutionary scale, we evolve reservoirs, a family of recurrent neural networks that differ from conventional networks in that one optimizes not the synaptic weights, but hyperparameters controlling macro-level properties of the resulting network architecture. At the developmental scale, we employ these evolved reservoirs to facilitate the learning of a behavioral policy through Reinforcement Learning (RL). Within an RL agent, a reservoir encodes the environment state before providing it to an action policy. We evaluate our approach on several 2D and 3D simulated environments. Our results show that the evolution of reservoirs can improve the learning of diverse challenging tasks. We study in particular three hypotheses: the use of an architecture combining reservoirs and reinforcement learning could enable (1) solving tasks with partial observability, (2) generating oscillatory dynamics that facilitate the learning of locomotion tasks, and (3) facilitating the generalization of learned behaviors to new tasks unknown during the evolution phase.

Keywords: Meta Reinforcement Learning · Reservoir Computing · Evolutionary Computation

1 Introduction

Animals demonstrate remarkable adaptability to their environments, a trait honed through the evolution of their morphological and neural structures [30,46].

C. Léger and G. Hamon—Equal first authors.

X. Hinaut and C. Moulin-Frier—Equal last authors.

C. Léger—Work done as intern at Flowers and Mnemosyne.

They are born equipped with both hard-wired behavioral routines (e.g. breathing, motor babbling) and learning capabilities for adapting based on their own experience. The costs and benefits of evolving hard-wired behaviors vs. learning capabilities depend on different factors, a central one being the level of unpredictability of environmental conditions across generations [17,42]. Environmental challenges that are shared across many generations favor the evolution of hard-wired behavior (e.g. breathing). On the other hand, traits whose utility can hardly be predicted from its utility in previous generations are likely to be learned through individual development (e.g. learning a specific language). Some brain regions might have evolved to generically facilitate the learning of diverse behaviors. For example, central pattern generators (CPGs) enable limb bambling, which may facilitate locomotion, pointing and vocalizations in humans [24]. Another example is the prefrontal cortex (PFC), a brain region that maps inputs within a high-dimensional non-linear space from which they can be decoded by other brain regions, acting as a reservoir for computations [14,23].

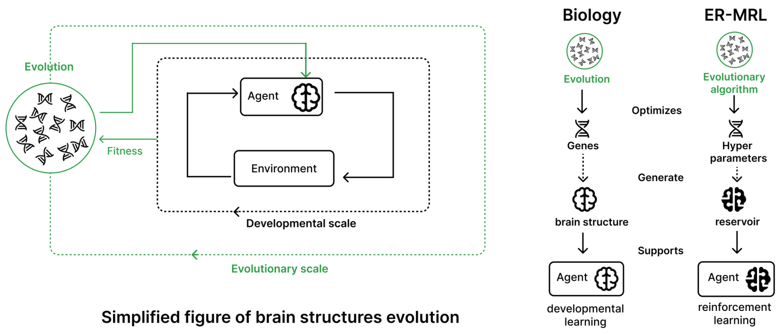


Fig. 1. (left) A simplified view of the evolution of brain structures. The generating parameters of neural structures are modified at an evolutionary loop. In the developmental loop, agents equipped with these neural structures learn to interact with their environment (right) Parallel to our computational approach. We propose a computational framework where an evolutionary algorithm optimizes hyperparameters that generate neural structures called reservoirs. These reservoirs are then integrated into RL agents that learn an action policy to maximize their reward in an environment

This prompts an intriguing question: How can neural structures, optimized at an evolutionary scale, enhance the capabilities of agents to learn complex tasks at a developmental scale? To address this question, we propose to model the interplay between evolution and development as two nested adaptive loops: neural structures are optimized through natural selection over generations (i.e. at an evolutionary scale), while learning specific behaviors occurs during an agent's lifetime (i.e. at a developmental scale). Figure 1 illustrates the interactions between evolutionary-scale and developmental-scale optimization. This model agrees with recent views on evolution that emphasize the importance of both scales for evolving complex skills [19,20]. It is also compatible with the biological principle of a *genomic bottleneck*, i.e. the fact that the information contained in the genome of most organisms is not sufficient to fully describe their morphology [52]. In consequence, genomes must instead encode macro-level properties of morphological features such as synaptic connection patterns.

In line with these biological principles, we propose a novel computational approach, called Evolving Reservoirs for Meta Reinforcement Learning (ER-MRL), integrating mechanisms from Reservoir Computing (RC), Meta Reinforcement Learning (Meta-RL) and Evolutionary Algorithms (EAs). We use RL as a model of learning at a developmental scale [9,29]. In RL, an agent interacts with a simulated environment through actions and observations, receiving rewards according to the task at hand. The objective is to learn an action policy from experience, mapping the observations perceived by the agent to actions in order to maximize cumulative reward over time. The policy is usually modeled as a deep neural network which is iteratively optimized through gradient descent. We use RC as a model of how a genome can encode macro properties of the agent's neural structure. In RC, the connection weights of a recurrent neural network (RNN) are generated from a handful of hyperparameters (HPs) controlling macro-level properties of the network related to connectivity, memory and sensitivity. Our choice of using RC relies on its parallels with biological brain structures such as CPGs and the PFC [15,50], as well as on the fact that its indirect encoding of a neural network in global hyperparameters makes it compatible with the genomic bottleneck principle mentioned above. Being a cheap and versatile computational paradigm, RCs may have been favored by evolution [39].

We use Meta-RL to model how evolution shapes development [8,32]. Meta-RL considers an outer loop, akin to evolution, optimizing HPs of an inner loop, akin to development. At the evolutionary scale, we use an evolutionary algorithm to optimize a genome specifying HPs of reservoirs. At a developmental scale, an agent equipped with a generated reservoir learns an action policy to maximize cumulative reward in a simulated environment. Thus, the objective of the outer evolutionary loop is to optimize hyperparameters of reservoirs in order to facilitate the learning of an action policy in the inner developmental loop.

Using this computational model, we run experiments in diverse simulated environments, e.g. 2D environments where the agent learns how to balance a pendulum and 3D environments where the agent learns how to control complex morphologies. These experiments provide support to three main hypotheses for how evolved reservoirs can affect development. First, they can facilitate solving partially-observable tasks, where the agent lacks access to all the information necessary to solve the task. In this case, we test the hypothesis that the recurrent nature of the reservoir will enable inferring the unobservable information. Second, it can generate oscillatory dynamics useful for solving locomotion tasks. In this case, the reservoir acts as a meta-learned CPG. Third, it can facilitate the generalization of learned behaviors to new tasks unknown during the evolution phase, a core hypothesis in meta-learning. In our case, our expectation is that HPs of reservoirs evolved across different environments will capture some abstract properties useful for adaptation.

In Sect. 2, we detail the methods underlying our proposed model, including RL (Sect. 2.1), Meta-RL (Sect. 2.2), RC (Sect. 2.3) and EAs (Sect. 2.4). We then explain their integration into our ER-MRL architecture (Sect. 3). Our results, aligned with the three hypotheses, are presented in Sect. 4. Computational specifics and supplementary experiments can be found in the appendix. The source code and videos are accessible at this link.

2 Background

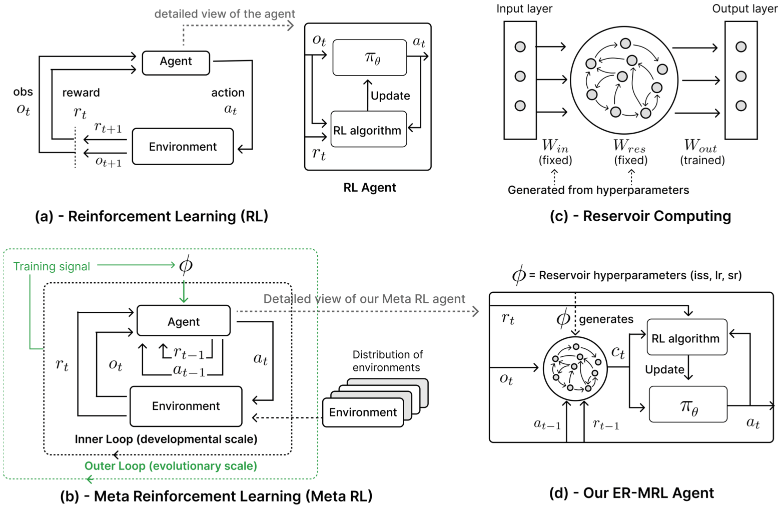


Fig. 2. Our proposed architecture, called ER-MRL, integrates several ML paradigms. We consider an RL agent learning an action policy (a), having access to a reservoir (c). We consider two nested adaptive loops in the spirit of Meta-RL (b). Our proposed architecture (d) consists in evolving HPs ϕ for the generation of reservoirs in an outer loop. In an inner loop, the agent learns an action policy, that takes as input the neural activation of the reservoir. The policy is trained using RL in order to maximize episodic return. Section 2 provides the computational details of each ML paradigm.

2.1 Reinforcement Learning as a Model of Development

Reinforcement Learning (RL) involves an agent that interacts with an environment by taking actions, receiving rewards, and learning an action policy in order to maximize its accumulated rewards (Fig. 2a). This interaction is formalized as a Markov Decision Process (MDP) [33]. An MDP is represented as a tuple (S, A, P, p_0, R), where S is the space of possible states of the environment, A is the space of available actions to the agent, $P(s_{t+1}|s_t, a_t)$ is the transition function specifying how the state at time $t+1$ is determined by the current state and action at time t, p_0 represents the initial state distribution, and $R(s_t, a_t)$ defines the reward received by the agent for a specific state-action pair. At each time step of an episode lasting T time steps, the agent observes the environment's state s_t, takes an action a_t, and receives a reward r_t. The environment then transitions to the next step according to $P(s_{t+1}|s_t, a_t)$. The objective of RL is to learn a policy $\pi_\theta(a|s)$ that maps observed states to actions in order to maximize the cumulative discounted reward G over time, where $G = \sum_{t=0}^{T} \gamma^t r_t$ [44]. The parameter $\gamma < 1$ discounts future rewards during decision making.

In Deep RL [21], the policy is implemented as an artificial neural network, whose connection weights are iteratively updated as the agent interacts with the environment. In all conducted experiments, we employ the Proximal Policy Optimization (PPO) RL algorithm [38] (see details in Sect. 6.1).

2.2 Meta Reinforcement Learning as a Model of the Interplay Between Evolution and Development

While RL has led to impressive applications [4,25,40], it suffers from several limitations: the learned policy is specific to the task at hand and does not necessarily generalize well to variations of the environment while requiring a large amount of data to converge. To address these issues Meta Reinforcement Learning (Meta-RL) [3] aims at training agents that *learn how to learn*, i.e. agents that can quickly adapt to new tasks or environments unknown during training. It is based on two nested adaptive loops: an outer loop, analogous to evolution, optimizes the HPs of an inner loop, analogous to development (Fig. 2b) [31,32]. The objective of the outer loop is to maximize the average performance of the inner loop on a distribution of environments. Formally, a set of HPs Φ are meta-optimized in the outer loop, with the objective of maximizing the average performance of a population of RL agents conditioned by Φ. In this paper, we leverage the RC framework where Φ corresponds to HPs encoding macro-level properties of a RNN, as explained in the next subsection.

2.3 Reservoir Computing as a Model of Neural Structure Generation

Meta-RL algorithms often directly optimize the weights of a RNN through backpropagation in the outer loop [10,11]. While this technique has demonstrated remarkable efficacy, it is ill-suited for addressing the research question outlined in the introduction. This is due to its lack of biological plausibility in two main aspects: (1) evolutionary-scale adaptation cannot rely on backpropagation mechanisms [43] and (2) the notion that evolution directly fine-tunes neural network weights contradicts the genomic bottleneck principle mentioned in the introduction [52]. Instead our method evolves RNNs based on the Reservoir Computing

(RC) paradigm. Instead of directly optimizing the neural network weights at the evolutionary scale, it optimizes HPs encoding macro-level properties of randomly generated recurrent networks.

The fundamental idea behind RC is to create a dynamic 'reservoir' of computation, where inputs are nonlinearly and recurrently recombined over time [22]. This provides a set of dynamic features from which a linear 'readout' can be easily trained: such training equivalent to selecting and combining interesting features to solve the given task (Fig. 2c).

A reservoir is generated from a few HPs which play a crucial role in shaping the efficiency of the reservoir dynamics. This includes the number of neurons in the reservoir, the spectral radius sr (controlling the level of recurrence in the generated network), input scaling iss (controlling the strength of the network's inputs), and leak rate lr (controlling how much the neurons retain past information); we explain reservoir HPs in more details in Appendix 6.1. In this paper, we propose to meta-optimize reservoir's HPs $\Phi = (sr, iss, lr)$ in a Meta-RL outer loop, using evolutionary algorithms explained in the next subsection. We will then explain how we propose to integrate RC with RL in Sect. 3.

2.4 Evolutionary Algorithms as a Model of Evolution

Evolutionary Algorithms (EAs) draw inspiration from the fundamental principles of biological evolution [2,36], where species improve their fitness through the selection and variation of their genomes. EAs iteratively enhance a population of candidate parameterized solutions to a given optimization problem, iteratively selecting those with higher fitness levels (i.e. higher performance of the solution) and mutating their parameters for the next generation.

In our approach, we utilize the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [13] as our designated evolutionary algorithm in order to meta-optimize HPs Φ of reservoirs. In CMA-ES, a population of HPs candidates is sampled from a multivariate Gaussian distribution, with mean μ and covariance matrix V. The fitness of each sample Φ_i of the population is evaluated (see Sect. 3 for how we do it in our proposed method). The Gaussian distribution is then updated by weighting each sample proportionally to its fitness; resulting in a new mean and covariance matrix that are biased toward solutions with higher fitness. This process continues iteratively until either convergence towards sufficiently high fitness values of the generated HPs is achieved, or until a predefined threshold of candidates is reached.

3 Evolving Reservoirs for Meta Reinforcement Learning (ER-MRL)

General Approach. Our objective is to devise a computational framework to address a fundamental question: How can neural structures adapt at an evolutionary scale, enabling agents to better adapt to their environment at a developmental scale? For this aim, we aim to integrate the Machine Learning paradigms presented above. The architecture is illustrated in Fig. 2d and the optimization procedure in Fig. 3. We call our method ER-MRL, for “Evolving Reservoirs for Meta Reinforcement Learning".

The ER-MRL method encompasses two nested optimization loops (as in Meta-RL, Sect. 2.2). In the outer loop, operating at an evolutionary scale, HPs Φ for generating a reservoir (Sect. 2.3) are optimized using an evolutionary algorithm (Sect. 2.4). In the inner loop, focused on a developmental scale, a RL algorithm (Sect. 2.1) learns an action policy π_θ using the reservoir state as inputs. In other words, the outer loop meta-learns HPs able to generate reservoirs resulting in maximal averages performance on multiple inner loops. The whole process is illustrated in Fig. 3 and detailed below.

Inner Loop. To represent the development of an agent, we consider a RL agent (Sect. 2.1) that interacts with an environment through observation o_t, actions a_t and rewards r_t at each time step t (Fig. 2a). In our proposed ER-MRL method, this agent is composed of three main parts: a reservoir generated by HPs $\Phi = \{iss, lr, sr\}$ (see Sect. 6.1 for more details), a feed forward action policy network π_θ and a RL algorithm. At each time step, we feed the reservoir with the current o_t, and the previous action and reward a_t and r_t (Fig. 2d). Contrarily to standard RL, the policy π_θ does not directly access the observation of the environment's state o_t, but the context c_t of the reservoir instead (i.e. the vector of all reservoir's neurons activations at time t). Because reservoirs are recurrent neural networks, c_t not only encompasses information about the current time step, but also integrates information over previous time steps. In some experiments, we also use ER-MRL with multiple reservoirs. In this case, we still generate the reservoirs from a set of HPs Φ, and the context c_t given to the policy is the concatenation of hidden states of all reservoirs. We then train our policy $\pi_\theta(a|c_t)$ using RL.

Outer Loop. The outer loop employs the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) (Sect. 2.4) to optimize reservoir HPs Φ. The objective is to generate reservoirs which, on average over multiple agents, improve learning abilities. For each set of HPs, we assess the performance of our agents in multiple inner loops (we utilize 3 in our experiments), each one with a different random seed. Using different random seeds implies that, while using the same HPs set, each agent will be initialized with different connection weights of both their reservoirs, their policies and the initial environment state. Note that while the generated reservoirs have different connection weights, they share the same macro-properties in terms of spectral radius sr, input scaling iss and leak rate lr (since they are generated from the same HPs set). In assessing an agent's fitness within its RL environment, we compute the mean episodic reward over the final 10 episodes of its training. To obtain the fitness of a reservoir HPs, we calculate the mean fitness of three agents across three different versions of the same environment. These steps are iterated until we reach a predetermined threshold of CMA-ES iterations (set at 900 in our experiments).

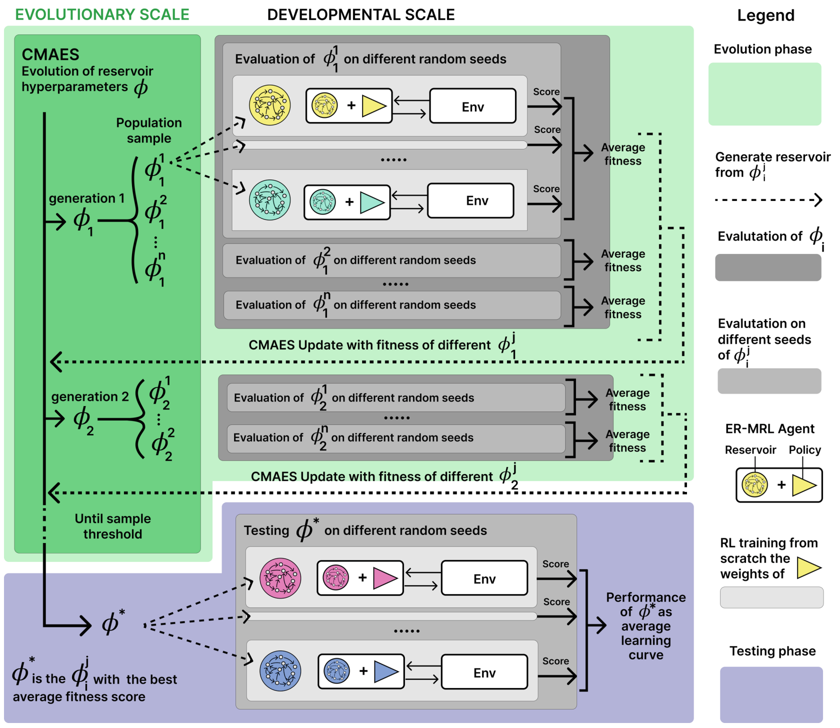

EVOLUTIONARY SCALE
DEVELOPMENTAL SCALE
Legend
CMAES
Evolution of reservoir hyperparameters ϕ
Population sample
generation 1
generation 2
Evaluation of ϕ_1^1 on different random seeds
Env
Score
Average fitness
Evaluation of ϕ_1^2 on different random seeds
Evaluation of ϕ_1^n on different random seeds
CMAES Update with fitness of different ϕ_1^j
Evaluation of ϕ_2^1 on different random seeds
Evaluation of ϕ_2^n on different random seeds
CMAES Update with fitness of different ϕ_2^j
Until sample threshold
Testing ϕ^* on different random seeds
Performance of ϕ^* as average learning curve
ϕ^* is the ϕ_i^j with the best average fitness score
Evolution phase
Generate reservoir from ϕ_i^j
Evalutation of ϕ_i
Evalutation on different seeds of ϕ_i^j
ER-MRL Agent
Reservoir
Policy
RL training from scratch the weights of
Testing phase

Fig. 3. In the evolution phase (top), CMA-ES refines Reservoir HPs Φ. At each generation i of the evolution loop (left), a population $\Phi_i : \{\Phi_i^1, \dots, \Phi_i^n\}$ of HPs is sampled from the CMA-ES Gaussian distribution. Each Φ_i^j undergoes evaluation on multiple random seeds, generating multiple reservoirs. An ER-MRL agent is created for each reservoir, with its action policy being trained from the states of that reservoir (lighter grey frames). The fitness of a sampled Φ_i^j is determined by the average score of all ER-MRL agents generated from it (mid-grey frames). The fitness values are used to update the CMA-ES distribution for the next generation (dotted arrow). This process iterates until a predetermined threshold is reached. In the Testing phase (bottom), the best set of HPs Φ^* from all CMA-ES samples is employed. Multiple reservoirs are generated within ER-MRL agents, and their performance is evaluated.

Evaluation. To evaluate our method, we select the HPs Φ^* that generated the best fitness function during the whole outer loop optimization with CMA-ES (see bottom of Fig. 3). We then generate 10 ER-MRL agents with different random seeds (with a different reservoir sampled from Φ^* for each seed, together with random initial policy weights θ) and train the action policy π_θ of each agent using RL. We report our results in the next section, comparing the performance of these agents against vanilla RL agents using a feedfoward policy.

4 Results

We designed experiments to study the following hypotheses: The ER-MRL architecture combining reservoirs and RL could enable (1) solving tasks with partial observability, (2) generating oscillatory dynamics that facilitate the learning of locomotion tasks, and (3) facilitating the generalization of learned behaviors to new tasks unseen during evolution phase.

4.1 Evolved Reservoirs Improve Learning in Highly Partially Observable Environments

In this section, we evaluate our approach on tasks with partial observability, where we purposefully remove information from the agent observations. Our hypothesis is that the evolved reservoir can help reconstructing this missing information. Partial observability is an important challenge in the field of RL, where agents have access to only a limited portion of environmental information to make decisions. This is referred to as a Partially Observable Markov Decision Process (POMDP) [26] rather than a traditional MDP. In this context, the task becomes harder to learn, or even impossible, as the agent needs to make decisions based on an incomplete observation of the environment state. To explore this issue, our experimental framework is based on control environments, such as CartPole, Pendulum, and LunarLander (see details in Fig. 9 of the appendix). We modify these environments by removing velocity-related observations, thus simulating a partially-observable task.

Let's illustrate this issue with the first environment (CartPole), where the agent's goal is to keep the pole upright on the cart while it moves laterally. If we remove velocity-related observations (both for the cart and the pole's angle), a standard feedfoward RL agent cannot effectively solve the task. The reason is straightforward: without this information, the agent doesn't know the cart's movement direction or whether the pole is falling or rising. We apply the same process to the other two environments, removing all velocity-related observations for our agents. Can the ER-MRL architecture address this challenge? To find out, we independently evolve reservoirs using ER-MRL for each task. We search for effective HPs tailored to the partial observability of each environment. To evaluate our approach, we compare the learning curves of ER-MRL agents (from the test phase, see bottom of Fig. 3) on these three partially observable environments against an agent with a feedforward policy.

Figure 4 presents the results for the three selected partially observable tasks. We observe, as expected, that vanilla RL agents cannot learn how to solve the task under partial observability (for the reasons mentioned above). In comparison, our approach leads to performance scores close to those obtained by a RL algorithm with full observability. This indicates that the evolved reservoir is able to reconstruct missing information related to velocities from its own internal recurrent dynamics. This confirms the hypothesis that an agent with a reservoir can solve partially observable tasks by using the internal reservoir state to reconstruct missing information. We explain with more details why this method

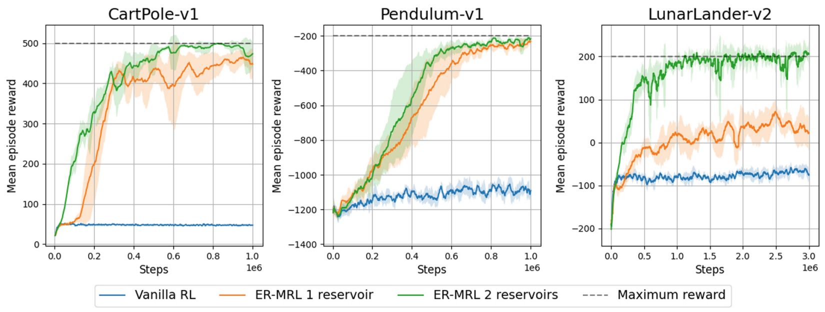


Fig. 4. Learning curves for partially observable tasks. The x-axis represents the number of timesteps during the training and the y-axis the mean episodic reward. Learning curves of our ER-MRL methods correspond to the testing phase described in the bottom of Fig. 3. Vanilla RL corresponds to a feedforward policy RL agent. The curves and the shaded areas represent the mean and the standard deviation of the reward for 10 random seeds. See Sect. 6.3 for a comparison with another method.

could work in Sect. 6.3 of the appendix. The difference in results between the model with 2 reservoirs on LunarLander environment suggests that solving it requires encoding at least two different timescales dynamics. Our interpretation here is that solving LunarLander requires to deal with both an “approaching" and “landing" phase, unlike the two other environments.

4.2 Evolved Reservoirs Could Generate Oscillatory Dynamics that Facilitate the Learning of Locomotion Tasks

In this section, we evaluate our approach on agents with 3D morphology having to learn locomotion tasks shown in Fig. 10. We postulate that the integration of an evolved reservoir can engender oscillatory patterns that aid in coordinating body movements, akin to Central Pattern Generators (CPGs). CPGs, rooted in neurobiology, denote an interconnected network of neurons responsible for generating intricate and repetitive rhythmic patterns that govern movements or behaviors [24] such as walking, swimming, or other cyclical movements. Existing scientific literature hypothesizes that reservoirs, possessing significant rhythmic components, share direct connections with CPGs [37]. We propose to study this hypothesis using motor tasks involving rhythmic movements.

We employed 3D MuJoCo environments (detailed in Fig. 10 of the appendix), where the goal is to exert forces on various rotors of creatures to propel them forward. Notably, while the ultimate goal across these tasks remains constant (forward movement), the creatures exhibit diverse morphologies, including humanoids, insects, worms, bipeds, and more. Furthermore, the action and observation spaces vary for each morphology. We individually evaluate our ER-MRL architecture on each of these tasks.

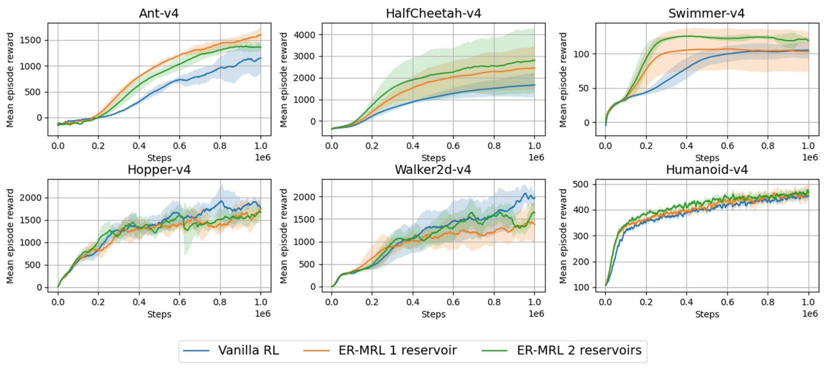


Fig. 5. Learning curves for locomotion tasks. Same conventions as Fig. 4

Our approach demonstrates improved performance in some tasks (Ant, HalfCheetah, and Swimmer) compared to a standard RL baseline, particularly noticeable in the early stages of learning, as illustrated in Fig. 5. This suggests that the evolved reservoir may generate beneficial oscillatory patterns, facilitating the learning of locomotion tasks, in line with the notion that reservoirs could potentially function as CPGs, aiding in solving motor tasks. Although carefully testing this hypotheses would require more analysis, we present in Sect. 6.4 in the appendix preliminary data suggesting that the evolved reservoir is able to generate oscillatory dynamics that could facilitate learning in the Swimmer environment. However, as shown in Fig. 5, performance enhancement was not observed in the Walker and Hopper environments compared to the RL baseline. Locomotion in both environments demands precise closed-loop control strategies to maintain an agent's equilibrium. In such cases, generated oscillatory patterns may not be as beneficial.

4.3 Evolved Reservoirs Improve Generalization on New Tasks Unseen During Evolution Phase

In this section, we address a key aspect of our study: the ability of evolved reservoirs to facilitate adaptation to novel environments. This inquiry is crucial in assessing the potential of evolved neural structures to generalize and enhance an agent's adaptability beyond the evolution phase. Building on the promising results of ER-MRL with two reservoirs in previous experiments, we focus exclusively on this configuration for comparison with the RL baseline.

Generalizing Across Different Morphologies with Similar Tasks. In prior experiments, ER-MRL demonstrated effectiveness in environments like Ant, HalfCheetah, and Swimmer. This success led us to explore whether reservoirs evolved for two of these tasks could be adaptable to the third, indicating potential generalization across different morphologies. However, due to variations in environments, including differences in morphology, observation and action spaces, and reward functions, generalization from one set of tasks to another presents a complex challenge. To ensure fair task representation of each environment in the final fitness, we employ the normalization formula detailed in Sect. 6.6. Subsequently, we select the reservoir HPs Φ^* that yielded the highest fitness and evaluate them in a distinct environment. For instance, if we evolve reservoirs on Ant and HalfCheetah, we test them in the Swimmer task.

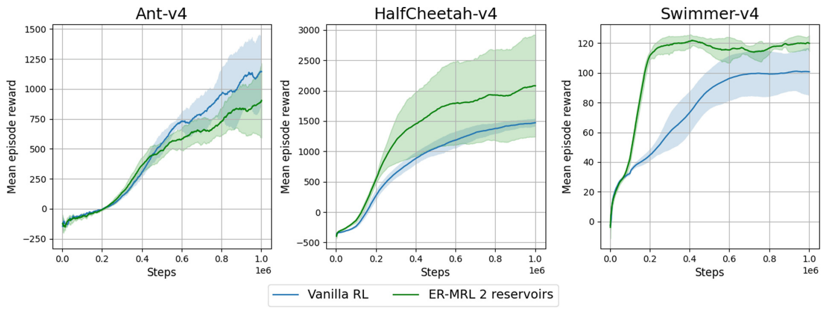


Fig. 6. Learning curves for generalization on similar locomotion tasks with different morphologies. The curves evaluate the performance of ER-MRL on an environment that was unseen during the evolution phase. For instance, the left plot shows performance of an agent on Ant, using reservoirs evolved on only HalfCheetah and Swimmer.

In Fig. 6, we observed a notable improvement in the performance of ER-MRL agents with reservoirs evolved for different tasks, particularly in HalfCheetah and Swimmer environments. This substantiates the capacity of evolved reservoirs to generalize to new tasks and encode diverse dynamics from environments with distinct morphologies. However, it's worth noting that this improvement wasn't replicated in the Ant task. This could be attributed to the unique characteristics of the Ant environment, with its stable four legged structure, in contrast to the simpler anatomies of Swimmer and HalfCheetah. For a detailed analysis, please refer to Sect. 6.7 in the appendix.

Generalizing Across Different Tasks with Similar Morphologies. We have seen how reservoirs facilitated ER-MRL agent's ability to generalize across locomotion tasks with different morphologies. Now, we shift our focus to tasks with consistent morphologies but distinct objectives. To delve into this, we turn to the Humanoid and HumanoidStandup environments (shown in Fig. 12 of the appendix), both presenting tasks within the realm of humanoid movement. One task involves learning to walk as far as possible, while the other centers around the challenge of standing up from the ground. As in our previous study, we follow

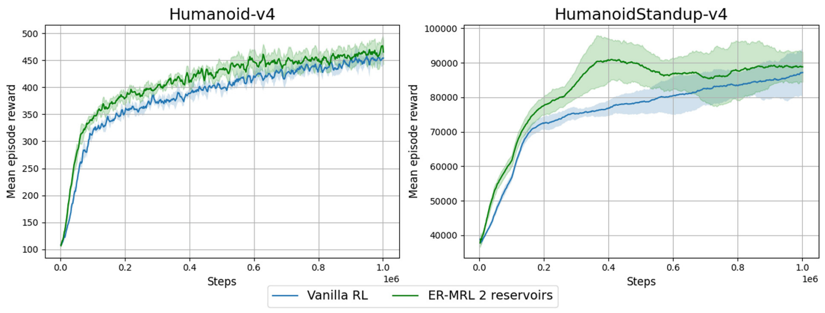


Fig. 7. Learning curves for generalization on different locomotion tasks with similar morphologies. The reservoirs are evolved on one task and tested on the other one.

the procedure of evolving reservoir-generating HPs on one task and evaluating their performance on the other.

Figure 7 provides a visual representation of our findings. While the performance improvement may not be dramatic, it underscores the generalization capabilities of reservoirs across tasks with similar morphologies but differing objectives. This observation, though promising, invites further investigation, given the limited number of experiments conducted in this context. This aspect represents an avenue for future research.

5 Discussion

In this paper, we have addressed the compelling question of whether reservoir-like neural structures can be evolved at an evolutionary time scale, to facilitate the learning of agents on a multitude of sensorimotor tasks at a developmental scale. Our results demonstrate the effectiveness of employing evolutionary algorithms to optimize these reservoirs, especially on Reinforcement Learning tasks involving partial observability, locomotion, and generalization of evolved reservoirs to unseen tasks.

Our ER-MRL approach has parallels to previous algorithms in RL that employ an indirect encoding for mapping a genome to a particular neural network architecture [12,28,41]. Our choice of employing reservoirs comes with the benefit of a very small genomic size (reservoirs are parameterised by a handful of parameters that we show in Appendix 6.1) without reducing the complexity of the phenotype (the number of weights of the reservoir policy is independent of the number of hyper-parameters). Moreover, our approach clearly distinguish neural structures optimized at the evolutionary scale (the reservoirs) vs. at the developmental scale (the RL action policy).

Nonetheless, some limitations persist within our methodology. The combination of reservoir computing and reinforcement learning remains underexplored in the existing literature [6,7], leaving substantial room for refining the algorithmic framework for improved performance. Moreover, our generalization experiments and quantitative analyses warrant further extensive testing to gain deeper insights. Notably, our approach does incur a computational cost due to the time required to train a new policy with RL for each generated reservoir. Future studies could devise more efficient evolutionary strategies or employ alternative optimization techniques.

However, because our method remains agnostic to specific environment and agent's characteristics (a reservoir architecture being independent of the shape of its inputs and outputs), we could in theory evolve reservoirs across a very wide range of environments and agent's morphologies. Such evolved generalist reservoirs could then result in highly reduced computational cost at the developmental scale, as our results suggest, compared to training recurrent architectures from scratch.

Moving forward, there are several promising avenues for exploration. Firstly, a more comprehensive understanding of the interaction between RL and RC could significantly improve the performance of such methods on developmental learning tasks. Secondly, integrating our approach with more sophisticated Meta-RL algorithms could offer a mean to initialize RL policy weights with purposefully selected values rather than random ones. Additionally, a broader framework allowing for the evolution of neural structures with greater flexibility, such as varying HPs and neuron counts, could yield more intricate patterns during the evolution phase, potentially resulting in substantial improvements in agent performance across developmental tasks [28,41].

Our research bridges the gap between evolutionary algorithms, reservoir computing and meta-reinforcement learning, creating a robust framework for modelling neural architecture evolution. We believe that this integrative approach opens up exciting perspectives for future research in RC and Meta-RL to propose new paradigms of computations. It also provides a computational framework to study the complex interplay between evolution and development, a central issue in modern biology [16,18,27,49].

Acknowledgments. Financial support was received from: the University of Bordeaux's France 2030 program/RRI PHDS framework, French National Research Agency (ANR) grants: ECOCURL ANR-20-CE23-0006 and DEEPPOOL ANR-21-CE23-0009-01. We benefited HPC resources: IDRIS under the allocation A0091011996 made by GENCI, using the Jean Zay supercomputer, and Curta from the University of Bordeaux.

6 Appendix

In this appendix, we provide comprehensive insights and clarifications on the methodologies employed in our study. Specifically, we elaborate on aspects such as the parameters governing our experiments, including the RL (PPO), the RC and the evolutionary (CMA-ES) algorithms we used. Furthermore, we furnish a detailed exposition of the environments utilized in our research. Lastly, we conduct supplementary analyses aimed at enhancing our understanding of some observed phenomena in the obtained results. In addition, we present results from experiments that were not featured in the main text to offer a more comprehensive view of our findings.

6.1 Methods

Proximal Policy Optimization (PPO). PPO, categorized as a policy gradient technique [45], undertakes exploration of diverse policies through stochastic gradient ascent. This process involves assigning elevated probabilities to actions correlated with high rewards, subsequently adjusting the policy to aim for higher expected returns. The adoption of PPO stems from its well-established reputation as a highly efficient and stable algorithm in the scientific literature, although its use does not have major theoretical implications for this particular project.

Reservoir Hyperparameters. In Reservoir Computing, the spectral radius controls the trade-off between stability and chaoticity of reservoir dynamics: in general "edge of chaos" dynamics are often desired [5]. Input scaling determines the strength of input signals, and the leak rate governs the memory capacity of reservoir neurons over time. These HPs specify the generation of the reservoir weights. Once the reservoir is generated, its weights are kept fixed and only a readout layer, mapping the states of the reservoir neurons to the desired output of the network are learned. Other HPs exist to initialize a reservoir, but they have not been studied in the experiments of the paper (as it has been tested that they have much less influence on the results).

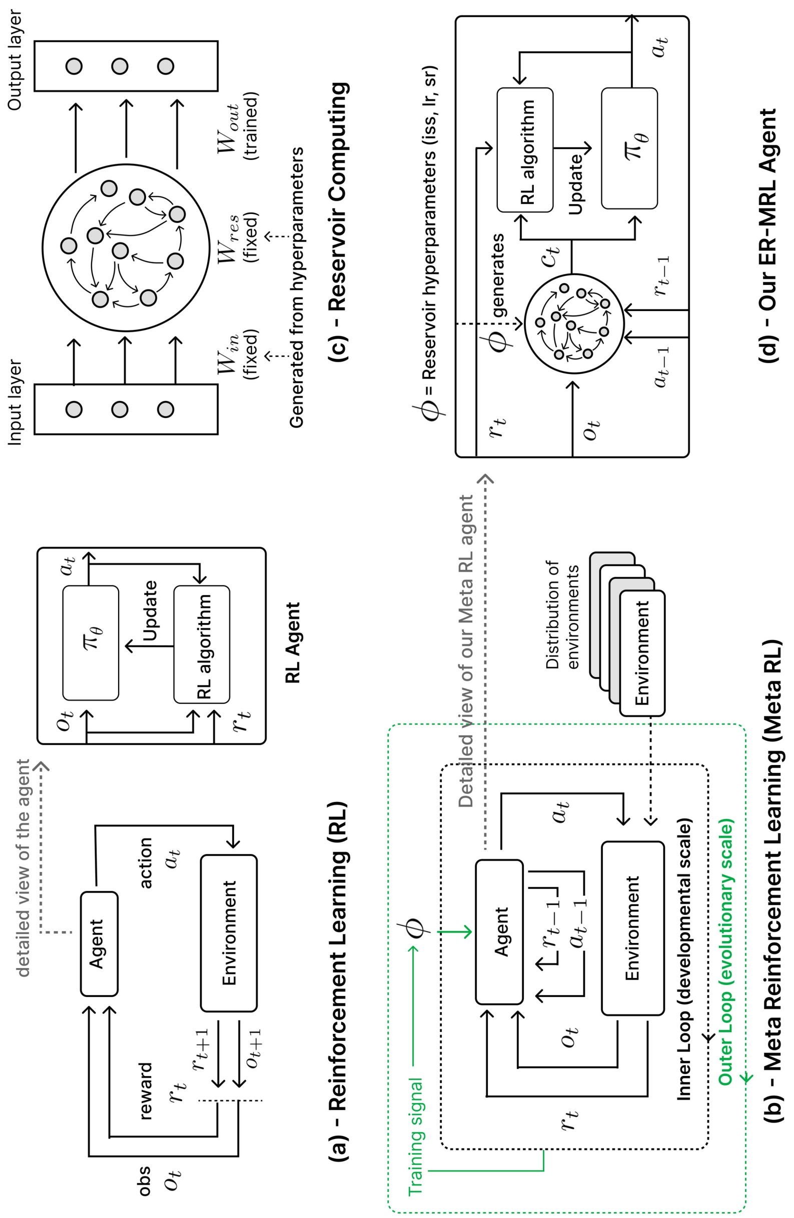
6.2 Experiment Parameters

General Parameters. In our experiments, we adapted the number of timesteps during the training phase of our ER-MRL agent in the inner loop, based on whether we were evolving the reservoir HPs or testing the best HPs set discovered during the CMA-ES evolution. For the evolution phase, which was computationally intensive, we utilized 300,000 timesteps per training. Conversely, when evaluating our agents against standard RL agents, we employed 1,000,000 timesteps. Notably, in the case of the LunarLander environment, we extended the testing to 3,000,000 timesteps, as the learning curve had not yet converged at 1,000,000 timesteps.

PPO Hyperparameters. Regarding the parameters of our RL algorithm, PPO, we used the default settings provided by the Stable Baselines3 library [35]. For tasks involving partial observability, we made a slight adjustment by setting the learning rate to 0.0001, as opposed to the standard 0.0003. This modification notably enhanced performance, potentially indicating that reservoirs contained a degree of noise, warranting a lower learning rate to stabilize RL training.

CMA-ES Hyperparameters. For the parameters of our evolutionary algorithm, CMA-ES, we adopted the default settings of the CMA-ES sampler from the Optuna library [1].

Reservoirs Hyperparameters. For the reservoirs, we only modified the parameters mentioned in Sect. 6.1 and the number of neurons. We consistently used 100 neurons per reservoirs during all experiments. All the other HPs were



detailed view of the agent
Agent
Environment
action
a_t
reward
r_t
r_{t+1}
o_{t+1}
obs
o_t
(a) - Reinforcement Learning (RL)
π_θ
Update
RL algorithm
RL Agent
Input layer
Output layer
W_{in} (fixed)
W_{res} (fixed)
W_{out} (trained)
Generated from hyperparameters
(c) - Reservoir Computing
ϕ = Reservoir hyperparameters (iss, lr, sr)
ϕ generates
c_t
r_{t-1}
a_{t-1}
(d) - Our ER-MRL Agent
Detailed view of our Meta RL agent
Distribution of environments
Training signal
ϕ
Inner Loop (developmental scale)
Outer Loop (evolutionary scale)
(b) - Meta Reinforcement Learning (Meta RL)

Fig. 8. Rotated view of Fig. 2 presenting the background methods used, and how our ER-MRL agents incorporate them

kept the same and are the default reservoir parameters used in ReservoirPy [48]. We conducted additional analyses and observed that they exerted a relatively modest influence on tasks of this nature. However, a more refined analysis of the importance of these HPs could be interesting in future works.

6.3 Partially Observable Environments

In the following section, we present the different Reinforcement Learning environments from the Gymnasium library [47], used during our experiments on partial observability.

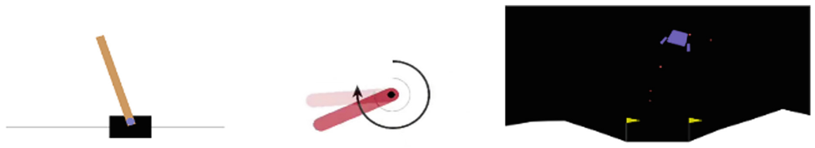


Fig. 9. Partially observable environments used, The goal of CartPole (left) is to learn how to balance the pole on the cart. The goal of Pendulum (middle) is to learn how to maintain the pendulum straight up by applying forces on it. The goal of LunarLander (right) is to learn how to land between the two flags by generating forces on the different spaceship reactors.

Results Analysis. To better understand the reservoir's capabilities on these tasks, we conducted several tests on supervised learning problems where a sequence of actions, rewards, and observations (without velocity) was provided to a reservoir with a linear readout. In one case, the model had to reconstruct full observation information (position, angle, velocity, angular velocity), and in the other, it had to reconstruct positions and angles over several time steps (doing this only for the last 2 time steps allows a PPO to achieve maximum reward later on). In both cases, this model successfully solved the tasks with very high performance. Moreover, it was also capable of predicting future observations, which can be extremely valuable to find an optimal action policy.

Benchmark Comparison. Regarding benchmarks, our approach compares favorably with the results reported in the blog post from Raffin [34] where he used another model combining a RNN (LSTM [51]) with a RL algorithm (PPO, the one we also used) on the same partially observable tasks. The performance on each environment are pretty similar, but it is the training timesteps needed to reach the maximum performance that varies the most between the methods. Indeed for the LunarLander environment, our method is able to learn in less timesteps after evolving reservoirs, but it is the contrary with CartPole and Pendulum tasks.

It is worth noting that even if both approaches have similarities, ER-MRL consists in optimizing the HPs of reservoirs at an evolutionary scale, whereas the method presented in the blog post trains a recurrent architecture from scratch. This divergence complicates direct comparisons between both methods. Indeed, our results are derived after an extensive phase of computation in a Meta-RL outer loop, but the subsequent evaluation with the final reservoir configuration is comparatively swift. as only the RL policy (linear readout) requires training. In contrast, the LSTM-PPO method does not incorporate a computationally intensive meta-learning phase, but their training process takes more time per timestep update. Indeed, each training step of the this demands more computation, due to having to train the LSTM from scratch in addition to PPO, compared to our method where only the linear readout is trained at the developmental scale.

However to ensure a fair and comprehensive comparison with other baselines, especially in tracking the time required to achieve presented results, more experiments are necessary.

6.4 MuJoCo Forward Locomotion Environments

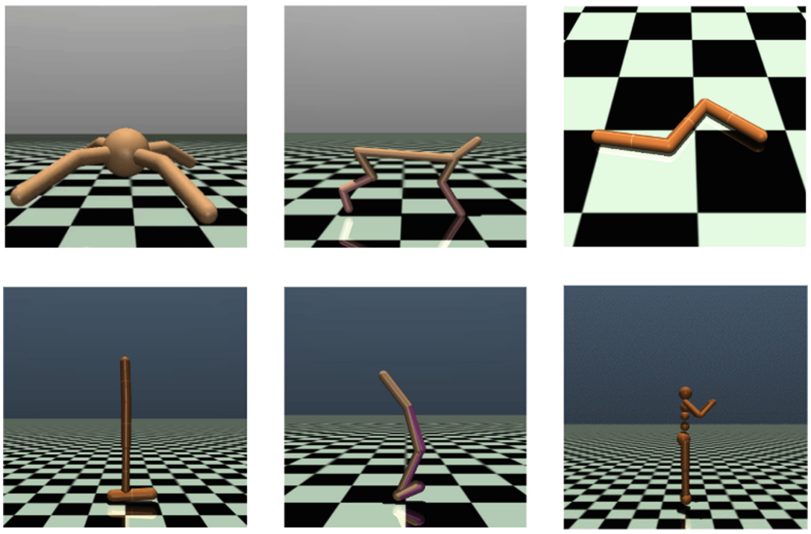


Fig. 10. MuJoCo environments, the goal of these tasks is to apply force to the rotors of the creatures to make them move forward. On the top row, we have from left to right the Ant, HalfCheetah and Swimmer environments, and on the bottom row, the Hopper, Walker and Humanoid environments. The environment observations comprise positional data of distinct body parts of the creatures, followed by the velocities of those individual components, while actions entail the torques applied to the hinge joints.

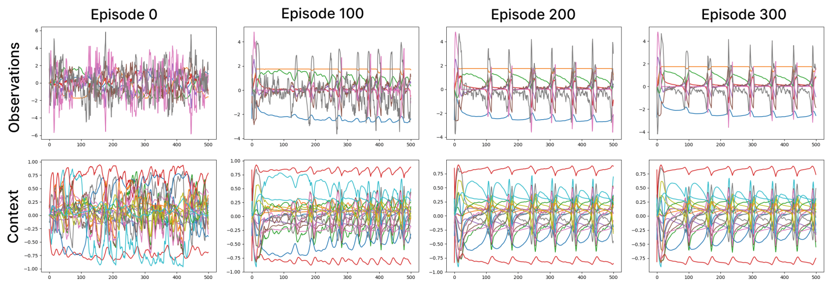


Fig. 11. Differences between the observations of a RL agent (top) with the context of an ER-MRL agent (bottom) at the same stage of training. Each episode lasts 1000 timesteps in the environment. The curves of the RL agent represent the real observation values from the environment, and the curves of the ER-MRL one part of the context given to the agent's policy: the activation values of 20 reservoir neurons (out of 100).

Results Analysis. In this section, we present how reservoirs could act as Central Pattern Generators within agents learning a locomotion task in these 3D environments.

It can be observed that the separation between the two models seems to occur starting from 100,000 timesteps at the top-right of Fig. 5. Therefore, we recorded videos of the RL and ER-MRL agents to better understand the performance difference between the two models. Furthermore, we conducted a study at the level of the input vector in the agent's policy (o_t for RL agent, and c_t for ER-MRL agent). As seen in Fig. 11, it is noticeable that very early in the learning process, the reservoir exhibits much more rhythmic dynamics than the sole observation provided by the environment. This could be due to the link between the reservoir and CPGs, potentially facilitating the acquisition and learning of motor control in these tasks.

Expanding on this, it's notable that CPGs, shared across various species, have evolved to embody common structures. Drawing parallels from nature, our investigation delves into whether generalization (results in Sect. 4.3) across a spectrum of motor tasks may mirror the principles found in biological systems.

However, further experiments, accompanied by robust quantitative analysis, are necessary to gain valuable insights into whether reservoirs can function as CPG-like structures.

6.5 MuJoCo Humanoid Environments

Interesting Reservoir Results. As seen in Sect. 2.3, one of the basic principles of RC is to project input data into a higher-dimensional space. In the case of the Humanoid tasks, where our results are displayed in Fig. 5 and Fig. 7, the initial observation and action space is larger (400 dimensions) compared to the context dimension for one or two reservoirs of 100 neurons (the dimension is equal to

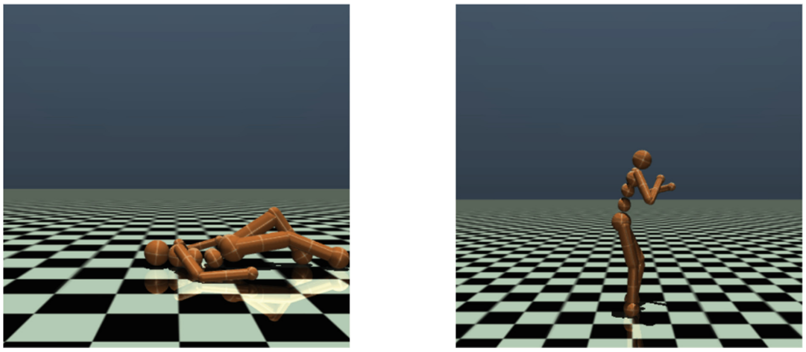


Fig. 12. MuJoCo environments with humanoid morphologies. On the left figure, the goal is to learn how to stand up, and on the right the goal is to walk forward as far as possible

the number of neurons). This means that even by reducing the input dimension in the RL policy network, the reservoir improves the quality of the data. For other morphologies, the dimension of input data is inferior to the dimension our reservoir context.

6.6 Normalized Scores for Generalization

To prevent any particular task from disproportionately influencing the fitness score due to variations in reward scales, we use a fitness function for CMA-ES that aggregates the normalized score, denoted as $nScore$, across both environments. The normalization process is defined as :

$$nScore = \frac{score - randomScore}{baselineScore - randomScore}$$

Where $randomScore$ and $baselineScore$ represent the performances of a random and of a standard PPO agent, respectively.

6.7 Reservoir Hyperparameters Analysis

In preceding sections, we observed how HPs play a pivotal role in enabling ER-MRL agents to generalize across tasks. Now, we delve deeper into understanding why some reservoirs aid in generalization for specific tasks while others do not. To gain this insight, we constructed a hyperparameter map to visualize the regions of HPs associated with the best fitness in each environment. We selected the best 30 sets of HPs, comprising the spectral radius and leak rate values of the reservoirs, out of a pool of 900 for all MuJoCo locomotion tasks (refer to Fig. 10) and plotted them on a 2D plane.

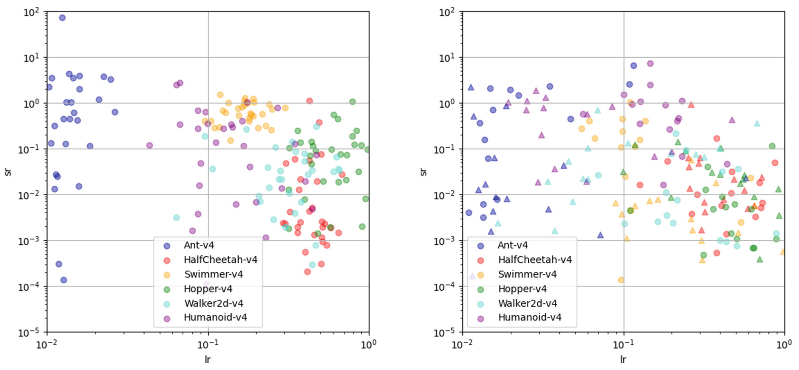


Fig. 13. The left figure represents parameters obtained with a single reservoir, while the right figure corresponds to configurations with two reservoirs (depicted as either circles or triangles).

In Fig. 13, we observe that the HPs for most environments are clustered closely together. Conversely, those for the Ant environment form a distinct cluster, characterized by notably lower leak rates. The leak rate reflects how much information a neuron retains in the reservoir, influencing its responsiveness to input data and connections with other neurons. A lower leak rate implies a more extended memory, possibly instrumental in capturing long-term dynamics. This observation aligns with the stable morphology of the Ant, potentially allowing the agent to prioritize long-term dynamics for efficient locomotion. This would partially explain why generalization wasn't successful on this environment in Sect. 4.3, when reservoirs were evolved on other types of morphologies.

7 Additional experiments

We also led other experiments that we didn't mention in the main text. As mentioned above in Sect. 6.1, we consistently employed reservoirs with a size of 100 neurons to ensure a standardized basis for result comparison. This configuration equates one reservoir to 100 neurons, two reservoirs to 200 neurons, and so forth. We conducted additional experiments to investigate the impact of varying the number of reservoirs and neurons within them. We observed that altering the number of neurons within a reservoir had a limited effect. For example, reducing the number of neurons to as low as 25 did not significantly affect performance on the partially observable environments. Increasing the size of the reservoirs didn't seem to improve the performance a lot either, except for the Humanoid environments (with a large observation space) where reservoirs equipped with a lot of neurons (1000) performed slightly better than others. While we opted for 100 neurons per reservoir in our experiments, there is surely potential for further optimization.

Furthermore, we explored experiments involving partially observable reservoirs, in which only a subset of the observation was provided to the policy. The results demonstrated that it is not always necessary to fully observe the contextual information within the reservoir to successfully accomplish tasks. On the CartPole environment, we tested 3 type of models with a reservoir of 100 fully observable neurons (the policy has access to 100 out of the 100 neurons), a reservoir of 1000 fully observable neurons, and another reservoir with only 100 partially observable neurons out of 1000. We observed that the model with 1000 fully observable neurons performed worse than the two other, who had similar results.

Regarding generalization experiments, we investigated the impact of varying the number of reservoirs. Although experiments with three reservoirs yielded intriguing insights, such as distinct memory types characterized by leak rate in the different reservoirs, the overall performance was notably lower compared to configurations with two reservoirs. This observation can likely be attributed to the increased complexity of learning due to the larger observation space, despite the potential for richer dynamics. We also noted instances where several reservoirs maintained very similar hyperparameters for specific tasks, potentially indicating the importance of capturing particular dynamics.

Additionally, we considered the possibility of employing smaller reservoirs in greater numbers. This approach could capture a diverse range of interesting features, such as different dynamics, while keeping the total number of neurons low. This strategy would be particularly advantageous for tasks characterized by small observation and action spaces, but would also imply a wider space of reservoirs HPs search in return.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2623–2631 (2019)
2. Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. Evol. Comput. **1**(1), 1–23 (1993)
3. Beck, J., et al.: A survey of meta-reinforcement learning. arXiv preprint arXiv:2301.08028 (2023)
4. Berner, C., et al.: Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680 (2019)
5. Bertschinger, N., Natschläger, T.: Real-time computation at the edge of chaos in recurrent neural networks. Neural Comput. **16**(7), 1413–1436 (2004)
6. Chang, H., Futagami, K.: Reinforcement learning with convolutional reservoir computing. Appl. Intell. **50**, 2400–2410 (2020)
7. Chang, H.H., Song, H., Yi, Y., Zhang, J., He, H., Liu, L.: Distributive dynamic spectrum access through deep reinforcement learning: a reservoir computing-based approach. IEEE Internet Things J. **6**(2), 1938–1948 (2018)
8. Clune, J.: Ai-gas: Ai-generating algorithms, an alternate paradigm for producing general artificial intelligence. arXiv preprint arXiv:1905.10985 (2019)
9. Doya, K.: Reinforcement learning: computational theory and biological mechanisms. HFSP J. **1**(1), 30 (2007)

10. Duan, Y., Schulman, J., Chen, X., Bartlett, P.L., Sutskever, I., Abbeel, P.: Rl squared: fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779 (2016)
11. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: International Conference on Machine Learning, pp. 1126–1135. PMLR (2017)
12. Ha, D., Dai, A., Le, Q.V.: HyperNetworks (2016). http://arxiv.org/abs/1609.09106. arXiv:1609.09106 [cs]
13. Hansen, N.: The CMA evolution strategy: a tutorial. arXiv preprint arXiv:1604.00772 (2016)
14. Hinaut, X., Dominey, P.F.: A three-layered model of primate prefrontal cortex encodes identity and abstract categorical structure of behavioral sequences. J. Physiol.-Paris **105**(1–3), 16–24 (2011)
15. Hinaut, X., Dominey, P.F.: Real-time parallel processing of grammatical structure in the fronto-striatal system: a recurrent network simulation study using reservoir computing. PLoS ONE **8**(2), e52946 (2013)
16. Hougen, D.F., Shah, S.N.H.: The evolution of reinforcement learning. In: 2019 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1457–1464. IEEE (2019)
17. Johnston, T.D.: Selective costs and benefits in the evolution of learning. In: Rosenblatt, J.S., Hinde, R.A., Beer, C., Busnel, M.C. (eds.) Advances in the Study of Behavior, vol. 12, pp. 65–106. Academic Press (1982). https://doi.org/10.1016/S0065-3454(08)60046-7. http://www.sciencedirect.com/science/article/pii/S0065345408600467
18. Johnston, T.D.: Selective costs and benefits in the evolution of learning. In: Advances in the Study of Behavior, vol. 12, pp. 65–106. Elsevier (1982)
19. Kauffman, S.A.: The Origins of Order: Self Organization and Selection in Evolution. Oxford University Press, Oxford (1993)
20. Laland, K.N., et al.: The extended evolutionary synthesis: its structure, assumptions and predictions. Proc. Royal Soc. B: Biol. Sci. **282**(1813), 20151019 (2015). https://doi.org/10.1098/rspb.2015.1019. https://royalsocietypublishing.org/doi/10.1098/rspb.2015.1019
21. Li, Y.: Deep reinforcement learning: an overview. arXiv preprint arXiv:1701.07274 (2017)
22. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. **3**(3), 127–149 (2009)
23. Mante, V., Sussillo, D., Shenoy, K.V., Newsome, W.T.: Context-dependent computation by recurrent dynamics in prefrontal cortex. Nature **503**(7474), 78–84 (2013)
24. Marder, E., Bucher, D.: Central pattern generators and the control of rhythmic movements. Curr. Biol. **11**(23), R986–R996 (2001)
25. Mnih, V., et al.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
26. Monahan, G.E.: State of the art-a survey of partially observable Markov decision processes: theory, models, and algorithms. Manag. Sci. **28**(1), 1–16 (1982)
27. Moulin-Frier, C.: The ecology of open-ended skill acquisition. Ph.D. thesis, Université de Bordeaux (UB) (2022)
28. Najarro, E., Sudhakaran, S., Risi, S.: Towards self-assembling artificial neural networks through neural developmental programs. In: Artificial Life Conference Proceedings, vol. 35, p. 80. MIT Press, Cambridge (2023)

29. Nussenbaum, K., Hartley, C.A.: Reinforcement learning across development: what insights can we draw from a decade of research? Dev. Cogn. Neurosci. **40**, 100733 (2019)
30. Pearson, K.: Neural adaptation in the generation of rhythmic behavior. Ann. Rev. Physiol. **62**(1), 723–753 (2000)
31. Pedersen, J., Risi, S.: Learning to act through evolution of neural diversity in random neural networks. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1248–1256 (2023)
32. Pedersen, J.W., Risi, S.: Evolving and merging hebbian learning rules: increasing generalization by decreasing the number of rules. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 892–900 (2021)
33. Puterman, M.L.: Markov decision processes. Handb. Oper. Res. Manag. Sci. **2**, 331–434 (1990)
34. Raffin, A.: Ppo vs recurrentppo (aka ppo lstm) on environments with masked velocity (sb3 contrib). https://wandb.ai/sb3/no-vel-envs/reports/PPO-vs-RecurrentPPO-aka-PPO-LSTM-on-environments-with-masked-velocity-VmlldzoxOTI4NjE4. Accessed Nov 2023
35. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: reliable reinforcement learning implementations. J. Mach. Learn. Res. **22**(1), 12348–12355 (2021)
36. Reddy, M.J., Kumar, D.N.: Computational algorithms inspired by biological processes and evolution. Curr. Sci. 370–380 (2012)
37. Ren, G., Chen, W., Dasgupta, S., Kolodziejski, C., Wörgötter, F., Manoonpong, P.: Multiple chaotic central pattern generators with learning for legged locomotion and malfunction compensation. Inf. Sci. **294**, 666–682 (2015)
38. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
39. Seoane, L.F.: Evolutionary aspects of reservoir computing. Phil. Trans. R. Soc. B **374**(1774), 20180377 (2019)
40. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
41. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artif. Life **15**(2), 185–212 (2009). https://doi.org/10.1162/artl.2009.15.2.15202
42. Stephens, D.W.: Change, regularity, and value in the evolution of animal learning. Behav. Ecol. **2**(1), 77–89 (1991). https://doi.org/10.1093/beheco/2.1.77
43. Stork: Is backpropagation biologically plausible? In: International 1989 Joint Conference on Neural Networks, pp. 241–246. IEEE (1989)
44. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)
45. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. Adv. Neural Inf. Process. Syst. **12** (1999)
46. Tierney, A.: Evolutionary implications of neural circuit structure and function. Behav. Proc. **35**(1–3), 173–182 (1995)
47. Towers, M., et al.: Gymnasium (2023). https://doi.org/10.5281/zenodo.8127026. https://zenodo.org/record/8127025
48. Trouvain, N., Pedrelli, L., Dinh, T.T., Hinaut, X.: *ReservoirPy*: an efficient and user-friendly library to design echo state networks. In: Farkaš, I., Masulli, P., Wermter, S. (eds.) ICANN 2020. LNCS, vol. 12397, pp. 494–505. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61616-8_40

49. Watson, R.A., Szathmáry, E.: How can evolution learn? Trends Ecol. Evol. **31**(2), 147–157 (2016)
50. Wyffels, F., Schrauwen, B.: Design of a central pattern generator using reservoir computing for learning human motion. In: 2009 Advanced Technologies for Enhanced Quality of Life, pp. 118–122. IEEE (2009)
51. Yu, Y., Si, X., Hu, C., Zhang, J.: A review of recurrent neural networks: LSTM cells and network architectures. Neural Comput. **31**(7), 1235–1270 (2019)
52. Zador, A.M.: A critique of pure learning and what artificial neural networks can learn from animal brains. Nat. Commun. **10**(1), 3770 (2019)