RCbench: a unified framework for benchmarking reservoir computing systems

PAPER • **OPEN ACCESS**

RCbench: a unified framework for benchmarking reservoir computing systems

To cite this article: Davide Pilati *et al* 2026 *Neuromorph. Comput. Eng.* **6** 014012

NEUROMORPHIC
Computing and Engineering

OPEN ACCESS

RECEIVED
14 November 2025

REVISED
23 January 2026

ACCEPTED FOR PUBLICATION
10 February 2026

PUBLISHED
19 February 2026

PAPER

RCbench: a unified framework for benchmarking reservoir computing systems

Davide Pilati[1,2,*], **Andrea Ceni**[3], **Fabio Michieletti**[1,2], **Claudio Gallicchio**[3], **Carlo Ricciardi**[1,*] and **Gianluca Milano**[2,*]

[1] Department of Applied Science and Technology, Politecnico di Torino, Turin 10129, Italy
[2] Advanced Materials Metrology and Life Sciences Division, Istituto Nazionale di Ricerca Metrologica, Turin 10135, Italy
[3] Department of Computer Science, University of Pisa, Largo Bruno Pontecorvo, 3, 56127 Pisa, Italy
[*] Authors to whom any correspondence should be addressed.

E-mail: davide.pilati@polito.it, carlo.ricciardi@polito.it and g.milano@inrim.it

Keywords: reservoir computing, physical reservoir computing, reservoir computing benchmarking, benchmarking physical computing systems

Supplementary material for this article is available online

Abstract

Reservoir computing (RC) is a computational framework where a fixed dynamical *reservoir* projects an input into a higher-dimensional state that is then analyzed by a *readout*, which is trained to map the reservoir state into the desired output. While the conventional RC paradigm is based on dynamics of *in-silico* implemented recurrent neural networks, this computing paradigm can be efficiently implemented in hardware by exploiting dynamics of a wide range of physical systems in a paradigm denoted as Physical RC (PRC), attracting interest from a broader research community spanning from computer scientists to physicists, and material scientists. Here, we present RCbench, an open-source RC benchmark toolkit that implements a standardized and comprehensive suite for benchmarking computational reservoir models and physical implementations of RC. RCbench integrates widely recognized metrics such as Memory capacity, Nonlinear autoregressive moving average of order N, Kernel rank , and generalization rank, along with nonlinear transformation tasks. It also allows testing and comparing different readout algorithms, the evaluation of computational capabilities with diverse accuracy metrics, and includes feature selection methods to unravel the effect of specific reservoir outputs on computational performances. In particular, the toolkit enables easy benchmarking of PRC systems, providing a comprehensive benchmark tool that can be easily integrated with experimental data acquisition processes. By standardizing performance assessments, RCbench aims to facilitate inter-study comparisons and to accelerate the exploration, characterization and optimization of RC systems.

1. Introduction

In recent years the continuous increase in power consumption, brought by the advancements of artificial intelligence and machine learning applications [1, 2], has brought to the attention of the scientific community the issue of finding alternative computing paradigms to face the ever-increasing energy costs of conventional approaches. Among emerging unconventional computing paradigms, reservoir computing (RC) has been widely explored [3–5]. The RC framework poses its foundations in transforming input data into high-dimensional space by exploiting a *reservoir*: a black-box constituted by a recurrent neural network (RNN) architecture whose internal weights are not trained [6]. The training is done on the reservoir output nodes only, using a readout layer trained with simple algorithms such as linear or ridge regression, effectively limiting the computational cost of learning in prediction, regression, and classification tasks [6]. When the reservoir is substituted by a physical system, the computing paradigm is called Physical RC (PRC). In this framework, both simulative and experimental works have shown that PRC can be successfully implemented by harnessing the dynamics of various physical system.

The state of the art for what concerns RC packages is represented by ReservoirPy and PRCPy [7, 8]. ReservoirPy is a Python package that is used to run reservoir software models and simulate different architectures. While it is a great tool for modeling and simulations, it does not cover the needs of the PRC community, which is also relying on experimental data. PRCPy is instead specifically intended for the PRC community and provides an environment for the implementation of computing tasks in PRC systems, but the pipeline structure is not always immediate for a broad audience and does not provide standardized guidelines to benchmark computing properties. Here we report on RCbench, a Python package that provides a ready-to-use and easily embeddable platform for a standardized analysis of computational capabilities of RC systems—both computational software models and PRC systems—aiming to complement currently available packages. RCbench is designed to operate as a standalone tool or to be used in conjunction with other packages, such as ReservoirPy and PRCpy. Moreover, RCbench can compute performance metrics in real-time during RC operations or operate afterward as an offline benchmarking tool, providing a complete variety of benchmarks as required for a comprehensive assessment of reservoir properties in both scenarios. Besides enabling the implementation of conventional benchmark tasks, RC bench includes ready to use functions for the assessment of various metrics to assess computing performances, different readout functions, and additional features for investigating the influence of specific reservoir outputs on the computing capabilities of the RC systems, an important aspect for the development of PRC systems.

2. RC

2.1. Background

RC is an unconventional computing paradigm that emerged in the early 2000s as an alternative to fully trained RNN. It uses a high-dimensional, nonlinear reservoir to transform the incoming input, while only a single linear readout layer is trained to match the desired output (figure 1(a)). Jaeger and Mass independently demonstrated that delegating the training to the readout layer can lead to powerful learning with minimal training cost [9, 10]. In particular, the reservoir is a dynamical system (a sparsely connected RNN or another complex system) in which the intrinsic dynamics echo past inputs. The echo-state (fading memory) property of these systems ensures that effects of arbitrarily old inputs vanish and prevents instabilities, making the system versatile and capable of universal function approximation. Crucially, the internal connection weights of the reservoir are fixed (often randomly initialized) and *not* trained. Instead, only the output layer (the *readout*) is trained to map the reservoir's state to the desired output. Training the readout typically involves a simple linear regression or ridge regression to find output weights, which is computationally lightweight [11]. This approach drastically reduces training cost compared to fully trained RNNs, since learning is confined to a single layer rather than a large number of recurrent weights, thereby eliminating the cost of intensive backpropagation through-time training.

Mathematically, an RC system can be described by equations for the reservoir state update (1) and the readout computation (2)

$$x(t+1) = f\Big(Wx(t) + W^{(\mathrm{in})}u(t)\Big) \tag{1}$$

$$y(t) = W^{(\mathrm{out})}x(t) \tag{2}$$

where $u(t)$ is the input, $y(t)$ is the reservoir output, $x(t)$ is the reservoir state, $f(\cdot)$ is an element-wise nonlinear activation function (e.g. tanh), and $W^{(\mathrm{in})}$, W, $W^{(\mathrm{out})}$ are input, reservoir, and output weight matrices respectively [11–13]. Both equations follow the standard formulation used in Echo state networks (ESNs) (bias vectors omitted for simplicity of notation). Only the output weights $W^{(\mathrm{out})}$ are adjusted during training to make $y(t)$ match the target output, typically by minimizing a mean-square error with regularization (ridge regression) [12]. The reservoir's weights remain fixed but must be chosen to ensure an echo state property or fading memory (i.e. the influence of past inputs decays) so that the system's state is stable and responsive to new inputs. Under these conditions, the reservoir's nonlinear dynamics and memory create a diverse set of basis functions of the input, analogous to a kernel transformation, that the linear readout can combine to approximate complex time-dependent mappings [11].

Key advantages of RC include fast training and low computational cost, since only the readout is learned. In practice, RC has been applied to a wide range of tasks such as time-series prediction, signal classification, speech recognition, robot control, and chaotic system forecasting, often achieving competitive performance with far less training effort than deep networks [11]. For example, an Echo State Network-based reservoir can accurately predict nonlinear time series (e.g. Nonlinear autoregressive moving average (NARMA) benchmarks) and recognize spoken digits using only a linear output learner

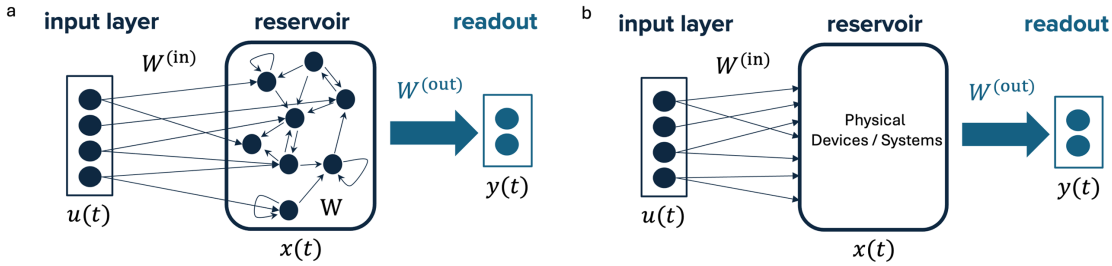

Figure 1. Conceptual schematization of RC. (a) Schematization of a software RC systems, where a recurrent neural network is used to project an input signal into a higher dimensional space that is then analyzed by a readout. (b) Schematization of a physical RC system, where the reservoir layer is be replaced by a physical device or system.

[9]. This efficiency makes RC attractive for real-time and low-power applications where conventional backpropagation-trained networks would be too slow or energy-intensive like in the case of edge AI [14].

2.2. PRC

While RC was originally conceived and implemented in software (simulated neural networks), a different approach is represented by PRC, where the reservoir is a physical system (figure 1(b)) [15]. In PRC, one leverages the inherent physics of a device or material to perform the nonlinear mapping of inputs to high-dimensional states. In other words, the physical device itself serves as the reservoir, and we only need to train a readout that processes the response of the physical system. This concept was inspired by the observation that many natural and engineered systems exhibit rich dynamical behavior (nonlinearity and memory) that can be harnessed for computation. The readout can then be implemented in software or hardware to produce the final output. Notably, because no training is required within the physical reservoir, a wide variety of analog systems can be used without the need for complex weight tuning [15]. It has to be underlined that in PRC time is continuous, although sampling the internal reservoir state necessarily involves time discretization related to sampling.

The term PRC was coined as these approaches gained popularity in the 2010s, but one early proof-of-concept dates back to 2003: Fernando and Sojakka's 'bucket of water' experiment [16]. In that work, the surface waves in a bucket of water were used as a reservoir for pattern recognition: input signals drove small motors to stir the water, and a camera recorded the resulting wave patterns as the reservoir state. A simple linear classifier then learned to distinguish patterns (such as an XOR temporal signal and spoken digit audio) from the water's response. This example demonstrates that a physical medium can compute, effectively 'outsourcing' computation to physics [16]. Using physical reservoirs offers potential advantages in speed and energy efficiency since computation is done implicitly by the physics rather than by digital number-crunching [11]. In this context, a wide range of physical systems have been explored as reservoirs, either through experiments or by simulating the response of a physical system to input data, as detailed below.

Photonic reservoirs: Optical systems have demonstrated excellent performance as reservoirs, offering ultra-fast parallel processing and low energy per operation. For example, a photonic chip with an array of 16 semiconductor optical amplifiers was experimentally implemented to serve as a reservoir, achieving tasks like 2-bit XOR logic, packet-header recognition, and spoken digit classification at high speed [17]. Photonic reservoirs can be realized either as spatial networks of interconnected optical nodes or using a *time-delay* scheme where a single nonlinear node is rapidly time-multiplexed to create a virtual network [18, 19]. Early works proposed designs in 2008 and showed their feasibility via numerical simulations, and since then, multiple experimental photonic RC systems have been built, including fiber-optic feedback loops, integrated waveguide circuits, and diffractive optical setups, all leveraging light's speed to perform computation [11]. Photonic RC is a vibrant subfield because of its promise for real-time processing of signals (e.g. RF or telecom signals) with minimal latency and power.

Memristive & electronic reservoirs: Memristors and other nonlinear electronic components provide another physical substrate for RC. Memristors naturally combine a resistive nonlinearity with memory (their resistance depends on past electrical stimuli), which can be used to mimic synapses or create complex dynamic circuits [20]. Researchers have shown that a network of memristive devices can act as a reservoir, transforming input voltage signals into a high-dimensional response space. The first

memristor-based reservoir models (around 2012) were demonstrated in simulation for tasks like waveform classification [21]. Physical implementations have followed: for instance, a 2017 study fabricated a reservoir device using 88 memristors arranged in a crossbar array and demonstrated successful classification of handwritten digit images and prediction of nonlinear time series [22]. In that system, each memristor received part of the input signal (split across devices in time) and the collective states were read out for training. Another approach is to use self-organized memristive networks made of nanoelements (e.g. nanoparticles or nanowires (NW)) [23–25]. Given the self-assembled nature of these networks, it can be challenging to study their spatiotemporal dynamics, and benchmarks often represent a viable solution to understand their computing properties [26]. In other cases, the response of a physical reservoir has been used to tune simulation models of reservoirs, bridging the gap between simulations and experiments [27]. Recently, memcapacitance and meminductance have played major roles in electronic physical reservoirs, because their intrinsic history-dependent capacitance or inductance naturally provides fading memory and nonlinear dynamics at the device level, enabling compact, energy-efficient reservoirs without explicit recurrent circuitry [28, 29]. Memristive and electronic reservoirs thus bridge the gap between neuroscience-inspired dynamics and modern computing hardware.

Mechanical reservoirs: Complex mechanical systems and unconventional materials can serve as reservoirs by virtue of their rich dynamics. The concept of morphological computation in robotics is closely related: a soft or compliant physical body can perform computational functions through its natural response to inputs [30]. In RC, this means a mechanical structure's vibrations, deformations, or waves can encode information. The water bucket example described earlier is one case of a fluidic/mechanical reservoir. Similarly, soft robots and metamaterials have been studied as reservoirs. For instance, networks of springs and masses can produce complex oscillatory patterns that were used to approximate nonlinear functions and even generate robot control signals [31]. Simulations of a mass-spring reservoir showed it could learn time series and emulated a robot arm's dynamics. In another study, a *tensegrity robot* (a structure of rods and cables) was used as a reservoir to generate stable locomotion gaits and classify terrain by the robot's sensor readings [32]. An extreme example is the octopus-inspired soft arm: a silicone robotic arm with virtually infinite degrees of freedom, whose complex movements were harnessed by an RC controller to learn dynamic tasks [33]. These works suggest that rather than fighting the complexity of flexible bodies, we can exploit it for computation. While many mechanical reservoir studies are based on numerical models, some physical prototypes have been built (e.g. the tensegrity *ReCTeR* robot reservoir), validating that real-world mechanical chaos can be tapped for computing [32]. Mechanical reservoirs are attractive for integrating directly with physical environments, allowing computation to happen within sensors, structures, or robots.

Chemical and biological reservoirs: Even chemical reaction dynamics and biological systems have been explored as reservoirs. Chemical reservoirs use reactions with inherent oscillatory or nonlinear behavior to process inputs. A notable proposal used coupled DNA-based oscillators in a microfluidic reactor as a reservoir computer [34]. In simulations of that system, the concentrations of different DNA species (which oscillate and interact) provided a set of time-varying states that a readout mapped to a desired output, successfully performing a temporal signal tracking task. This demonstrates that molecular reactions can compute, opening the door to biochemical computing devices. Similarly, the Belousov–Zhabotinsky reaction (a classic chemical oscillator) and other reagents have been considered for RC in analog chemical processors [35]. On the biological side, living neural networks and cultures of cells have reservoir-like properties. For example, populations of *E. coli* bacteria responding to chemical stimuli have been conceptually treated as a form of reservoir that processes temporal input conditions [36]. In neuroscience, it has even been speculated that certain brain circuits (like cortical microcircuits) operate on the RC principle to process temporal information [37]. While biological RC is still largely theoretical, *in vitro* experiments with cultured neurons or brain slices have started to probe their computational capacity in the RC framework [38]. Recent works have demonstrated that biomolecular systems based on ion-channel-doped lipid membranes and adaptive memimpedance elements can function as physical reservoirs, exploiting intrinsic memristive and memcapacitive dynamics for computation and learning in bio-inspired PRC platforms [39, 40]. Overall, chemical and biological reservoirs underscore how universal the RC principle is, since any system with the requisite nonlinearity and fading memory can, in theory, perform computations on time-series inputs.

2.3. Readout and training

One of the defining features of RC is the simplicity of training. As described, the internal reservoir weights are fixed *a priori* (e.g. by random initialization or by physical device properties), and learning

is applied only at the readout stage [11]. The readout takes the reservoir's state vector (which can be the set of neuron activations in a software RC, or a set of measured quantities in a physical RC) and produces the resulting output. Training the readout typically involves a linear model: e.g. solving for a weight vector $W^{(\text{out})}$ in $y(t) = W^{(\text{out})}x(t)$ that best fits the target output [12]. This is often done by linear regression with Tikhonov regularization (ridge regression), yielding an analytic solution for $W^{(\text{out})}$ [12]. For classification tasks, a logistic or softmax readout can be used, or even a simple perceptron rule in spiking implementations [11]. The key point is that training is one-shot or very rapid, with no backpropagation through time required in the reservoir. This enables RC systems to be trained on limited hardware (or even hardware-in-the-loop) and updated quickly as needed.

In physical RC implementations, the readout can be realized in different ways. Frequently, the reservoir's outputs (e.g. voltages, light intensities, strains, etc.) are digitized and a linear model is trained in software, since this is convenient for research proof-of-concepts [11]. However, also the readout process can be performed in hardware. For example, it has been demonstrated that computing can be performed all in hardware by coupling a memristive reservoir with a memristive readout [24], and FPGA/ASIC reservoir can include on-chip readouts [12]. Regardless of implementation, the limited training complexity is a major factor that makes RC attractive.

3. Introduction to RCbench

RCbench is a Python package that aims at standardizing the benchmarking procedures in PRC systems. As illustrated in figure 2 the package aims at covering a variety of benchmarks to evaluate the computational capabilities of a physical reservoir, from general properties to memory and nonlinearity. It is to be underlined that algorithms like grid-search for reservoir parameters optimization have not been included in RCbench, since the nature of the toolkit is not to simulate reservoirs, but to benchmark their computational capabilities.

The complete project directory structure has been reported in supplementary information S1.

3.1. Benchmarks

The capability of RC systems to effectively cast the input data to a higher-dimensional space that can be easily analyzed by a readout relies mainly on their memory and nonlinearity, which represent two traditionally competing computational features of reservoir systems [41–43]. Based on that, RCbench includes a set of tasks and analyses that have been proposed as benchmarks to assess computational properties of a RC system [44]. Concerning reservoir memory, RCbench includes the main two tasks for the evaluation: memory capacity (MC) [18, 45–47] for the evaluation of linear memory of the system, and NARMA [48–51] for what concerns the nonlinear memory. Concerning reservoir nonlinearity, RCbench includes the non-linear transformation (NLT) tasks, in which an input must be transformed into another function through a NLT [25, 26, 52]. In addition, RCbench includes also other parameters such as Kernel rank (KR) and generalization rank (GR), also considered as benchmarks for RC systems [53]. In the following, these benchmark tasks are introduced. It is important to underline that some of the following benchmarks come for free with the same applied input, e.g. MC, NARMA-*N*, KR and GR can all be performed with the same uniformly distributed signal used as input. This means that, when considering an experimental PRC system, a wide range of benchmarks can be evaluated by performing a limited number of experiments.

3.1.1. Memory capacity

MC is a standard benchmark that measures how well the system can reconstruct past inputs from its current state. It is obtained by predicting delayed versions of the input over multiple time lags and summing the resulting squared Pearson correlation scores.

The MC evaluation algorithm follows the literature definition and is implemented as follows [18, 54–56].

The evaluation is based on an experiment in which the reservoir has been fed an input $u(t)$ such that:

$$u(t) \sim U([V_{\min}, V_{\max}]) \tag{3}$$

where $U([V_{\min}, V_{\max}])$ is a uniform random distributed signal in the interval $[V_{\min}, V_{\max}]$ and the reservoir state $x(t)$ is measured or simulated for each timestep.

The goal of this task is to reconstruct a series of delayed versions of the input for a range of k-delays. For each k, the reservoir's output, $y_k(t)$, is trained to match the target waveform, $u(t-k)$, i.e. the input

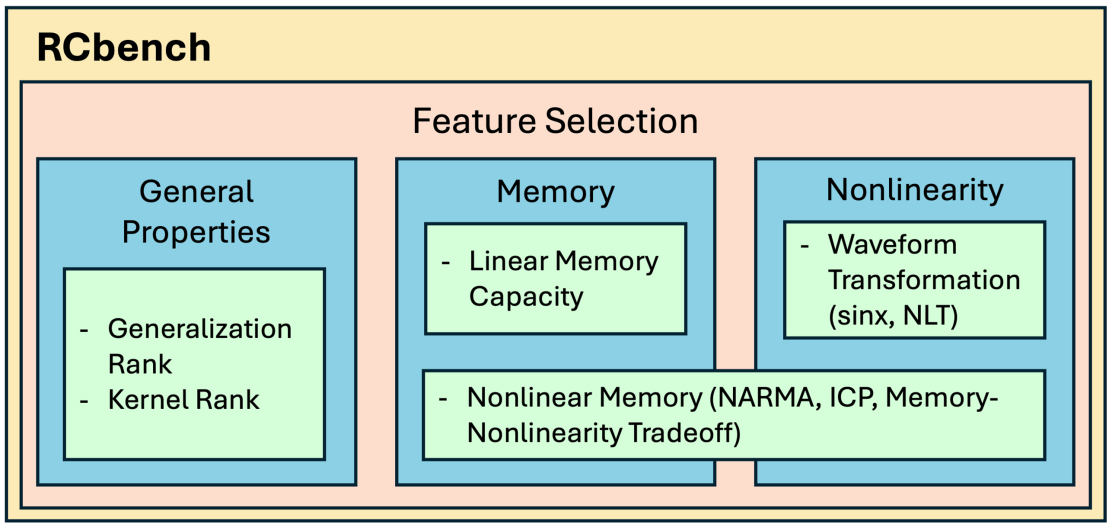

Figure 2. RCbench benchmarks. Benchmarks vary from general properties, as well as reservoir linear memory, nonlinearity, and nonlinear memory. On all these benchmarks a feature selection algorithm can be applied to evaluate the properties on a smaller subset of output nodes.

delayed of k timesteps. The MC_k is evaluated for each k following [44].

$$\mathrm{MC}_k = \frac{\mathrm{cov}^2\left(u\left(t-k\right), y_k\left(t\right)\right)}{\sigma^2\left(u\left(t\right)\right)\sigma^2\left(y_k\left(t\right)\right)}. \tag{4}$$

Then the total value of MC is evaluated as follows:

$$\mathrm{MC} = \sum_{k=1}^{\infty} \mathrm{MC}_k. \tag{5}$$

The MC value is defined as the sum of an infinite series of delay terms. In practice, the contribution of higher-order delays becomes negligible, so the sum is truncated at a finite value. Following common practice in the literature, this cutoff is often set to $2n$, where n is the number of reservoir output nodes [50]. This reflects the fact that, beyond a certain delay, the reservoir carries no meaningful memory, and the remaining terms are dominated by noise. In RCbench, the default cutoff is $k = 30$, although this parameter can be customized by the user.

3.1.2. NARMA-N

NARMA tasks require a model to predict the next value of a nonlinear autoregressive moving-average process based on a history of past inputs and outputs. They benchmark a system's ability to combine memory with nonlinear computation. It should be underlined that NARMA is not a pure metric, but a task of tunable complexity that is often used as benchmark to evaluate combined nonlinearity and memory properties. The order (N) of the NARMA task refers to the maximum delay that the reservoir has to remember in order to correctly predict the target. RCbench embeds two different algorithms for computing NARMA-N targets, depending on the order N. The NARMA-2 equation [50, 51] differs from the generalized NARMA-N [46], and this distinction is automatically addressed when selecting the NARMA order.

This must be evaluated considering a random uniformly distributed input signal, according to equation (3).

For computing NARMA-2, the equation used is

$$y\left(t+1\right) = \alpha y\left(t\right) + \beta y\left(t\right) y\left(t-1\right) + \gamma u^3\left(t\right) + \delta, \tag{6}$$

where $y(t)$ is the target function and $u(t)$ is the input at timestep t. The coefficients are arbitrary, but for literature consistency RCbench default values are $\alpha = 0.4$, $\beta = 0.4$, $\gamma = 0.6$, and $\delta = 0.1$.

For what concerns NARMA-N tasks with $N > 2$, the used equation has been standardized [46] and widely used in literature [23, 57, 58]. The target is computed as follows:

$$y(t+1) = \alpha y(t) + \beta y(t)\left(\sum_{i=0}^{N-1} y(t-i)\right) + \gamma u(t)\,u(t-N+1) + \delta. \tag{7}$$

RCbench NARMA-N coefficients can be customized by the user, but default values are consistent with the NARMA-10 literature equations, i.e. $\alpha = 0.3$, $\beta = 0.05$, $\gamma = 1.5$, and $\delta = 0.1$. [23]

3.1.3. Kernel rank

KR provides a measure of the linear separability of the reservoir's internal dynamics in terms of its effective dimensionality [59]. It characterizes the number of independent state dimensions the reservoir generates in response to the input, reflecting its ability to produce rich internal representations. The RCbench package implements the definition reported by Dale *et al* [47]. KR is computed by obtaining the rank r of a mxn matrix M. The matrix is obtained by feeding the reservoir an input u for m timesteps. The input signal is compatible with MC and NARMA experiments, since it is a uniformly distributed signal in a voltage interval. The state of the reservoir $x(t)$ is evaluated for each timestep and is made of n features (n output nodes). The matrix is filled by assigning a column for the reservoir state at a certain timestep. The rank of the matrix is evaluated via singular value decomposition [47].

3.1.4. Generalization rank

GR [60] provides a measurement of the reservoir resilience to noise or, more in general, how stable its internal dynamics are when the input is slightly perturbed. A stable reservoir should provide similar output states if stimulated with similar input sequences. If the reservoir can generalize two similar inputs, that results in a low GR value (indicating good performance). The mathematical algorithm to compute the GR is identical to the one used to obtain KR, the only difference is in the input signal fed to the reservoir, which is a random uniform distribution within a smaller range w.r.t. the one used for analyzing KR [47]. Despite being similar tasks, KR and GR have very different meanings: KR tests if the reservoir expands the input space, while GR tests if it remains consistent under small perturbations. These complementary measures jointly reveal the balance between computational richness and stability, with an optimal reservoir achieving high KR and low GR.

3.1.5. Waveform generation

Waveform generation is a task that allows to evaluate the nonlinearity of a reservoir by feeding as input a signal $u(t)$ which represents the argument of a target nonlinear function ($\sin(u(t))$ [61]. This can be evaluated considering an arbitrary input signal (e.g. random uniform distribution as described by equation (3)). Since the transformation does not embed any mathematical reference to past reservoir states, this nonlinearity benchmark does not require any memory.

3.1.6. Nonlinear transformation

NLT is a common benchmark to evaluate nonlinearity and memory features of a reservoir [25, 26, 52, 55]. A periodic waveform is used as input (commonly a sinewave) and the target is a periodic waveform that shares some characteristic with the input (usually phase or frequency). NLT task has been implemented in RCbench in a flexible way, to accept sinewave and triangular wave as input. The possible targets are the most common used for NLT purposes, i.e. square wave, phase-shifted input, doubled-frequency input, triangular/sine wave. An advantage of RCbench is that in experimental applications such as NLT, the target generation algorithm automatically matches the phase and the frequency of the target to the input signal: this feature avoids the user to manually detect the input phase and match the target wave starting point accordingly, besides enabling the code to fetch a large number of experimental measurements and analyzing NLT metrics automatically.

3.1.7. Memory-nonlinearity trade-off

The existence of a trade-off between the short-term memory and the nonlinearity response of a dynamical reservoir is well-documented in literature [43]. This trade-off can be measured with a task proposed by Inubushi and Yoshimura, which consists of extracting from a uniformly random input signal $u(t)$ (equation (3)) a target signal $y(t) = \sin(\nu \cdot u(t-\tau))$ [41]. In this task, ν quantifies the nonlinearity strength, and τ quantifies the memory depth [62]. Being the input given to the reservoir compatible with MC and NARMA tasks, the memory-nonlinearity benchmark can be performed together with the other two without any extra experimental measurement.

3.1.8. Information processing capacity (IPC)

IPC has been investigated by Dambre *et al* in previous works, and here implemented following their method [43]. The IPC framework quantifies how well a dynamical system encodes functions of its past inputs in its instantaneous state using a linear readout. To probe different computational features in a systematic way, the input–output tasks are constructed from products of orthogonal Legendre polynomials evaluated on delayed inputs, with each task corresponding to a specific combination of memory and nonlinearity. This can be evaluated considering an arbitrary input signal (e.g. random uniform distribution as described by equation (3)). Because these tasks are orthogonal with respect to the input distribution, their capacities measure independent contributions to the system's computation. For systems with fading memory and linearly independent states, the sum of all capacities is bounded by the number of observable states, while their distribution reveals the trade-off between memory and nonlinear processing.

3.1.9. Additional time-series repositories

While RCbench provides a dedicated set of tasks designed to assess the intrinsic computational properties of RC systems, it can be advantageous to complement these evaluations with broader time-series datasets. Such repositories offer diverse, real-world scenarios that extend the benchmarking scope beyond synthetic or controlled tasks, enabling the assessment of the generalization and robustness of RC models in applied contexts.

To this end, we recommend the following well-established repositories for time-series analysis:

- AEON classification collection [63]: A comprehensive repository encompassing a wide range of univariate and multivariate time-series classification datasets, suitable for evaluating classification performance across diverse domains.
- UCR time series classification archive [64]: One of the most widely used benchmarks in time-series research, providing standardized datasets and protocols for reproducible evaluation of classification algorithms.
- Time series extrinsic regression dataset collection [65]: A curated collection of regression-oriented time-series datasets, enabling the evaluation of models on tasks where the output is a continuous variable derived from temporal dynamics.

By combining RCbench with these external repositories, researchers can obtain a more comprehensive and comparative understanding of RC system performance, bridging the gap between theoretical benchmarking and application-oriented evaluation.

3.2. Readout and training

The RC framework is based on a readout layer to perform tasks properly. The most common ways to train the readout layer for a regression problem are linear regression and ridge regression, which are based on gaussian error distribution and L2 regularization [44]. Both algorithms are embedded in the package and can be used to perform regression on any described benchmark. Integrating a neural network as readout is possible, but is considered a feature for an advanced user, therefore is not integrated with a single command, but requires source code customization.

3.3. Metrics and features

3.3.1. Metrics

In the literature, benchmarks are often evaluated by describing the accuracy of a prediction w.r.t. a given target. Since a variety of different metrics are used to evaluate different tasks, RCbench lets the user decide which metric for assessing the accuracy for a given task. RCbench embeds mean square error (MSE), normalized MSE (NMSE), root NMSE (RNMSE), and normalized RNMSE to cover main metrics usually used in literature [44]. Detailed formulas of these metrics have been reported in supplementary information S2.

3.3.2. Feature selection

RCbench embeds a feature selection algorithm that allow to investigate the information carried by each reservoir output. All the metrics can be evaluated using all the available output nodes, or just a selection of them. The feature selection algorithm is based on the Python SciKit-learn module and can perform principal component analysis (PCA), *k*-best or mutual information feature selection in order to reduce the subset of output nodes if desired. PCA in particular has been widely adopted for feature selection, and for dataset feature reduction by PCA loading evaluation [66, 67]. The advantage of RCbench is

the flexibility of the feature selection integration, which allows to compare the performances of different feature selection algorithms for an audience of scientists that may not be very familiar with those algorithms and, more in general, with coding.

4. Benchmarking with RCbench

4.1. Reservoir input

RCbench is responsible for the readout layer and the evaluation of a benchmark, starting from two key datasets: the input applied to the reservoir and the set of reservoir outputs. Given that some of the described benchmarks are based on specific inputs, it is crucial that the user already performs the experiment (or simulation) with the right input signal applied to the reservoir (figure 3). In fact, an incorrect preprocessing of the input may alter benchmark results. In some cases, if the input is not coherent with the benchmark that has to be performed, the benchmark may fail completely (for example, in the case of NLT, a periodic waveform—sine or triangular—must be provided as input, otherwise the benchmark cannot be considered reliable). In short, different benchmarks require different reservoir inputs and is up to the user to choose the right input according to the task that has to be performed.

4.2. Reservoir output

The second dataset that has to be provided is the collection of the responses of the output nodes of the reservoir. To collect those signals, the user must probe the internal state of the reservoir by observing how physical observables evolve over time (figure 3). For instance, in an electrical reservoir, a voltage measurement can be performed on the output nodes over time to observe the evolution of the reservoir state. To maximize flexibility, RCbench accepts only the reservoir state outputs. Users are therefore free to apply any probing methodology appropriate to their system—regardless of reservoir type or measurement modality—and subsequently run the desired benchmarking routines.

4.3. Import data

Reservoir inputs and outputs are imported in RCbench (figure 3) as follows. The user can choose to provide directly a Pandas dataframe or use a text measurement file formatted with a time column, an input column, and column for each output node, with each row representing a timestep. In case the user chooses to provide a text file, the *ReservoirDataset* object can be initialized to automatically transform the file in a compatible dataframe for RCbench. When initializing the *ReservoirDataset* object, the input name and the output nodes names can be manually specified to match the ones reported on the text file. It has to be underlined that during the initialization of the *ReservoirDataset*, RCbench can detect if the name of the provided file contains any of the available benchmark names (e.g. 'memorycapacity', 'nlt', 'kernel', etc.) and automatically set a dataset property that can be used for more advanced automation features.

4.4. Benchmarking

The benchmarks are performed through a series of *Evaluators*, one for each of the benchmarks reported in the previous sections. Evaluators are objects that can perform independently on the creation of a *ReservoirDataset* object, by directly providing an input signal vector, a nodes output matrix and a series of node names. This makes RCbench compatible with every type of reservoir, both simulated or experimental. The evaluators also have to be initialized with the necessary parameters for the creation of the desired target (e.g. for NARMA tasks alpha, beta, gamma, and delta parameters have to be provided). Each *Evaluator* features a *run_evaluation* method, which accepts a series of argument to customize the benchmark (figure 3). These arguments include the choice of the metric (MSE, NMSE, etc.), the choice of a feature selection algorithm (PCA, *k*-best, none), the maximum number *n* of features to be considered (when performing feature selection the algorithm scores the nodes and the *n*-best scoring output nodes are selected, or can be set to 'all' to include all the output nodes), the model type for the readout training (linear regression or ridge regression), and the ratio to split the dataset into training and test set. Having all these parameters as attributes of the *run_evaluation* method allows to maintain the same dataset of input and output nodes, as well as the same target, while comparing different algorithms for feature selection, regression, and number of output nodes considered.

4.5. Data visualization

Each *Evaluator* features a *PlotConfig* attribute, which allows to choose from a set of visualization presets for that particular benchmark. For instance, when performing a MC benchmark, the corresponding

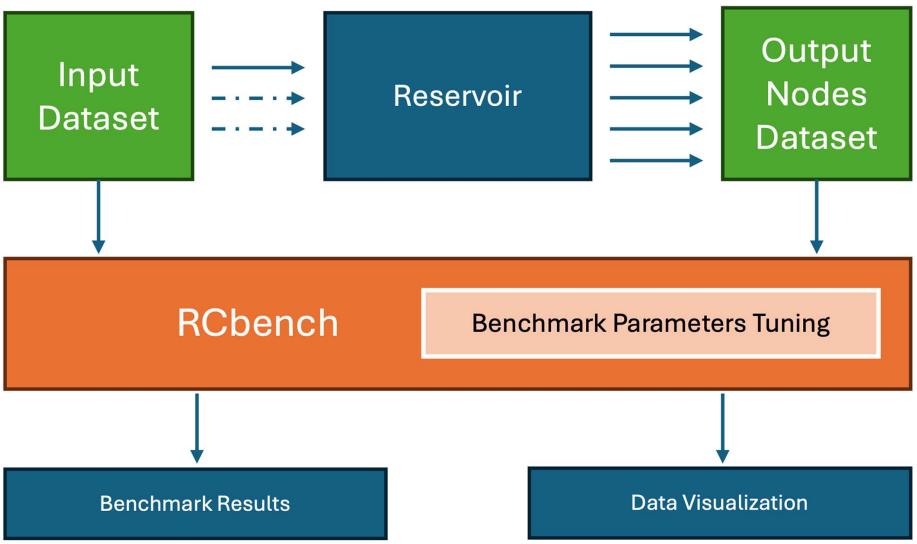

Figure 3. RCbench workflow. An input is applied to the reservoir. Reservoir outputs are collected and stored in a dataset. Input and output signals are passed to RCbench to perform the desired benchmarks. RCbench provides raw data as well as a first-investigation data visualization. Custom data visualization can be performed by accessing the raw benchmark data.

PlotConfig object can be set to automatically plot MC w.r.t. k-delay, the reservoir predictions for each k-delay, the node responses over time, the frequency analysis for each node, the nonlinearity analysis and a few other visualization presets. Each one of these plots can be individually enabled or disabled with a simple true/false flag in order to make the user experience as simple as possible (figure 3).

4.6. Computational cost

Some of the benchmarks included in RCbench may be computationally intensive, therefore a brief qualitative analysis will be here reported to help the user make more informed experiment design choices (details in supplementary information S3). Most of the benchmarks have an execution time in the 10^{-3}–10^{-2} s range, depending on the number of input samples and the number of reservoir output nodes. The longest task is the NLT task, which is heavily automated and is effectively performing four evaluations in the same task (one for each target waveform). In conclusion, the majority of the tasks have similar execution times, varying both with the number of input samples and the number of reservoir nodes.

4.7. Standardization and reproducibility

To enhance reproducibility and reduce inter-laboratory variability, RCbench adopts explicit and configurable evaluation protocols aligned with current benchmarking practice. Unless otherwise stated, benchmarks are evaluated on input sequences of at least 10^3–10^4 samples, which was found sufficient to ensure statistical convergence across memory, nonlinearity, and rank-based tasks, while an initial transient of 10%–20% of the dataset can be discarded to eliminate dependence on reservoir initialization. This percentage is of course depending on the ability of the reservoir to converge to a stable working state; therefore, it is best practice to remove the initial transient according to the specific response of the reservoir under investigation. Default train/test splits are fixed to 70/30, with training always performed on temporally contiguous data to preserve causality. While sensitivity to sampling rate, noise, and dataset length is intrinsic to physical reservoir systems, RCbench exposes these parameters explicitly and enforces consistent defaults, enabling controlled comparisons and reducing the risk of inconsistent benchmarking practices, particularly for experimental PRC users. Moreover, is recommended to report, in conjunction with benchmark results, evaluation details like train/test split, number of input samples, and reservoir nodes to ensure reproducible and standardized results.

5. Examples

5.1. Benchmarking software reservoirs

To demonstrate the interoperability of RCbench with existing software RC frameworks, we provide an example that integrates with ReservoirPy, an open-source Python library for ESNs [7]. The workflow for integration of RCbench with ReservoirPy is reported in supplementary information S4. An example

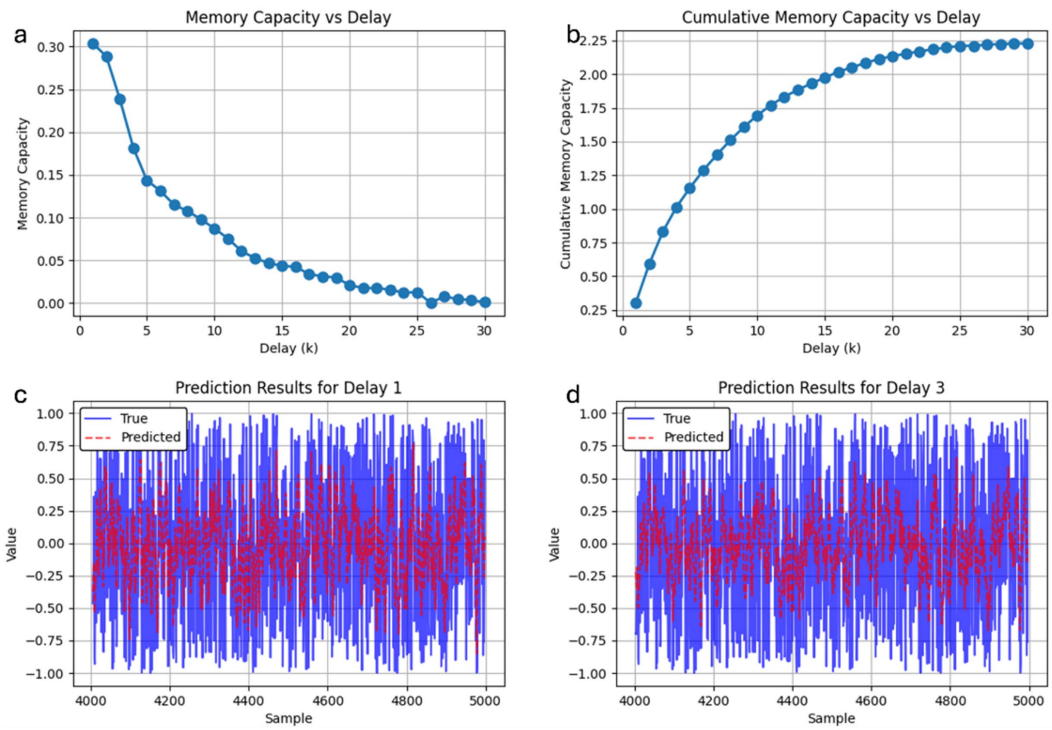

Figure 4. Example of MC benchmark of a software reservoir system. Benchmarking has been performed by interfacing RCbench (benchmark) with ReservoirPy (simulation). (a) MC vs delay (k). (b) Cumulative MC vs delay (k). (c) Prediction vs target (true) for delay $k = 1$. (d) Prediction vs target (true) for delay $k = 3$.

of results from the MC benchmark of a reservoir instantiated with ReservoirPy is reported in figure 4. Both evaluations employ Ridge regression with configurable regularization and support PCA for optional dimensionality reduction of the reservoir state space. This modular integration approach allows researchers to leverage ReservoirPy's efficient ESN simulation capabilities while benefiting from RCbench's comprehensive benchmarking suite and visualization tools, thereby facilitating systematic comparison of reservoir architectures across standardized metrics.

5.2. Benchmarking physical reservoirs

To demonstrate the compatibility of RCbench with physical reservoirs, we report in the following *i)* an example of benchmarking an experimental physical reservoir system based on memristive NW networks and *ii)* an example of integrating the benchmarking tools in a pre-existing pipeline experiment design software built with PRCpy.

5.2.1. Network-based memristive reservoir

The working principle of physical reservoir based on these networks can be found in previous works [25, 55, 68]. The measurement setup exploited to extract the dynamical response of the NW network reservoir consists in a parallel array of 16 SMUs and has been detailed in some of our previous works [26, 52]. The network is contacted through 16 gold-plated pogo pin electrodes that allow to stimulate and measure the dynamic evolution of the system in different spatial locations. One electrode serves as input node, one as ground node, and 14 electrodes are used for probing the floating voltage (physical observable of the NW network reservoir) and therefore exploited as output nodes (figure 5).

The target benchmark is the evaluation of the MC, so the network is fed with a uniformly distributed random input waveform (figure 5(a)). The output nodes' voltages are collected as well as the output current from the ground node (figure 5(b)). The MC benchmarking workflow, including dataset description and management, is reported in supplementary information S5.

In the reported case, the plots of MC vs k-delay, cumulative MC vs k-delay, and the predictions for $k = 1$ and $k = 3$ have been enabled. After running the benchmark, the results are plotted as reported in figure 6. With a couple of lines of code, we have been able to evaluate the linear memory capabilities of this physical reservoir, visualizing the memory timescale (figures 6(a) and (b)), as well as the accuracy deterioration of the predicted waveforms w.r.t. the targets as k increases (figures 6(c) and

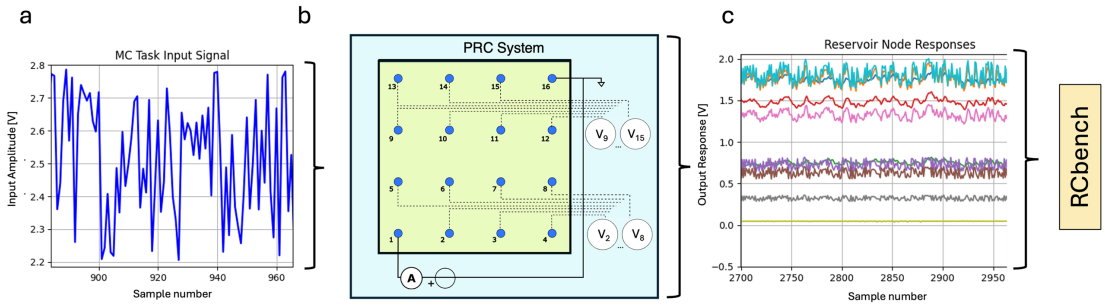

Figure 5. Electrical schematics of the experimental setup used for the RCbench reported MC benchmark example in which (a) a uniformly distributed random signal in the interval [2.2, 2.8] V is applied to (b) electrode 1 of the NW network sample. Electrode 16 of the NW network sample is set to ground, while the other 14 electrodes are used as floating output nodes. The floating voltages read from the (c) output nodes are sent to RCbench to perform the MC benchmark.

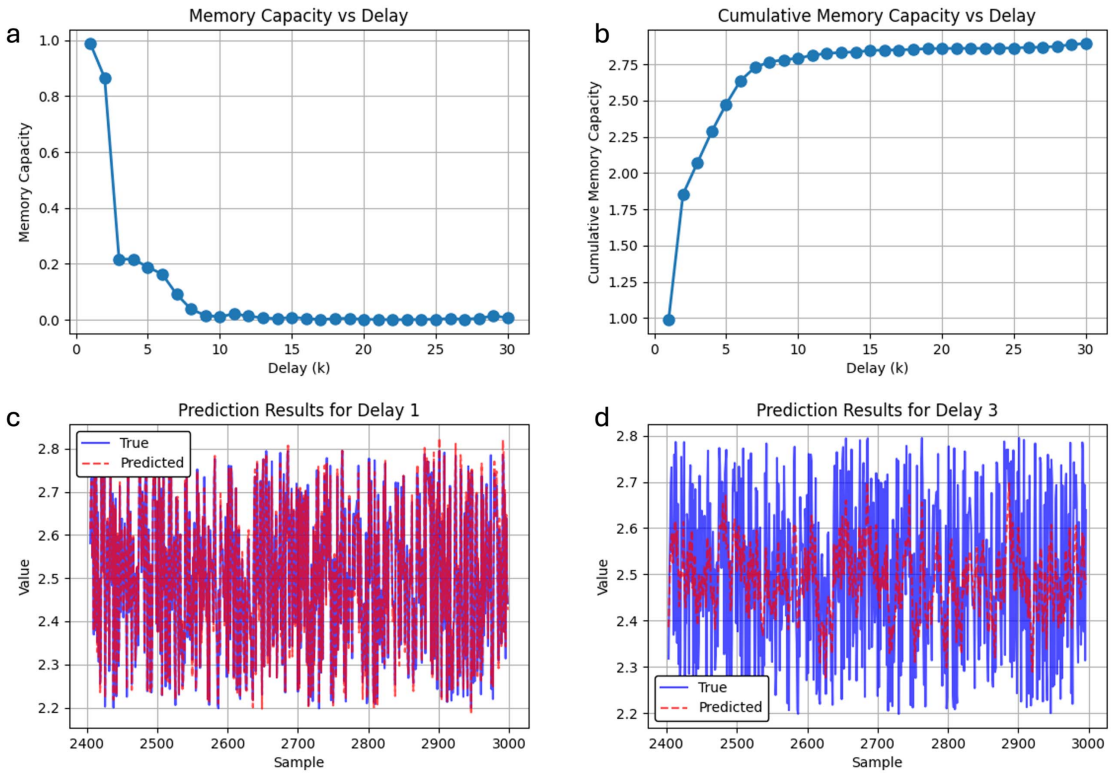

Figure 6. Example of MC benchmark on an experimental PRC system. Benchmarking has been obtained by interfacing RCbench with an experimental system based on a memristive nanowire network acting as physical reservoir. (a) MC vs delay (k). (b) Cumulative MC vs delay (k). (c) Prediction vs target (true) for delay $k = 1$. (d) Prediction vs target (true) for delay $k = 3$.

(d)). Other values can be extracted if a deeper analysis is needed, such as the selected nodes and the weights of the regressor. As shown in figure 7, a deep analysis can be conducted on the output signals of the reservoir by performing PCA. The nodes can be ranked as a function of their explained variance (figure 7(a)) and the MC benchmark can be evaluated as a function of the number of nodes selected after this ranking (figure 7(b)). Moreover, a spatial localization of the most relevant nodes can be done by plotting the explained variance as a heatmap to understand if there are more relevant areas for computing (figure 7(c)). In this experiment, we can notice that the area around the input (node 1) is more relevant for computation and that the MC value reaches saturation with only four nodes (figure 7(b)), which can be easily spotted spatially on the heatmap (figure 7(c)).

5.2.2. Integration in PRCpy experimental pipeline

To demonstrate the compatibility of RCbench with pre-existing PRC frameworks, we illustrate its integration with PRCpy, a Python library specifically designed for processing experimental data from physical reservoir devices (supplementary information S6). Unlike software-simulated reservoirs, physical

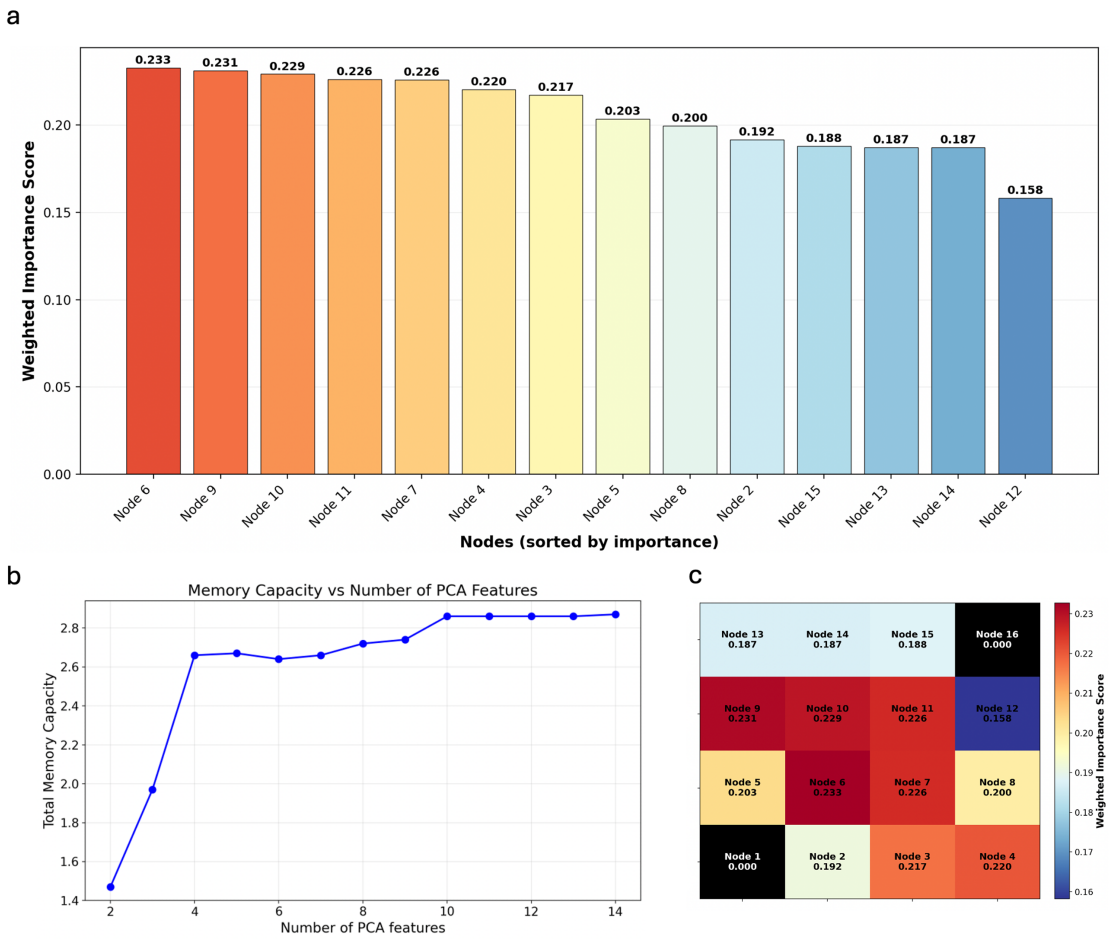

Figure 7. Example of PCA feature selection analysis. PCA feature selection has been performed on an experimental system based on a memristive nanowire network acting as physical reservoir. (a) Node ranking by explained variance (b) Total MC as a function of the number of PCA features selected. (c) Spatial disposition of the nodes as a function of the weighted variance.

reservoirs, such as those based on photonic, spintronic, or mechanical substrates, produce measurement data that typically require preprocessing prior to computational analysis. The workflow for integration of RCbench with PRCpy is reported in supplementary information S6. This integration bridges the gap between experimental PRCpy-based routines and standardized computational benchmarking, enabling researchers to evaluate physical reservoir systems using the same metrics employed for software reservoirs and thereby facilitating direct comparison across fundamentally different reservoir implementations.

6. Conclusions

The rapid diversification of PRC platforms has created an urgent demand for rigorous, yet accessible, benchmarking practices that allow researchers to compare devices and algorithms on equal footing. RCbench directly addresses this need by providing a unified, open-source framework that streamlines every stage of the evaluation pipeline, from data ingestion to metric computation and visualization. By incorporating the standard benchmarks for linear memory (MC), nonlinear memory (NARMA-*N*), nonlinearity (NLT, waveform generation), and capacity measures (KR, GR), the toolkit enables a holistic characterization of both software modeled and experimental physical reservoirs within a single environment. In fact, a key strength of RCbench lies in this dual orientation. The modular *ReservoirDataset* abstraction decouples data acquisition from analysis, permitting seamless integration with bespoke measurement setups, while maintaining full compatibility with in-silico studies. Advanced functionality (such as automated feature-selection routines, interchangeable read-out learners, and publication-ready plotting presets) further lowers the barrier to high-quality, reproducible experiments. The illustrative case study on a memristive NW network demonstrates that only minimal user intervention is required to extract quantitative insight into the device's linear memory horizon and the scaling behavior of prediction error with delay.

Beyond its immediate utility, RCbench establishes a reproducible protocol that can serve as a common language across the heterogeneous PRC landscape. Standardized reporting of memory depth, NLT accuracy, and rank-based complexity measures will foster more transparent cross-study comparisons, accelerate the discovery of structure-function relationships, and ultimately guide the rational co-design of reservoir substrates and read-out circuitry. Moreover, the open architecture encourages community-driven extension: additional tasks (e.g., classification, dynamical system emulation), alternative error metrics, or hardware-accelerated training back-ends can be incorporated without disrupting the existing workflow. Several other avenues for future development are evident, e.g., automated hyper-parameter optimization and meta-learning strategies could be integrated to help users exploring the often-large design space of physical reservoirs. Moreover, due to the flexible and modular design of RCbench, integration with MATLAB-based workflows can be achieved through a Python environment, for example by interfacing RCbench with MATLAB data or models via standard Python–MATLAB interoperability tools.

In summary, RCbench delivers a comprehensive, extensible, and experimentally aware benchmarking suite that unifies the evaluation of computational and physical reservoirs. By lowering the technical threshold for rigorous performance assessment and encouraging standardized reporting practices, the toolkit is poised to improve the accessibility of reliable benchmarking for PRC, toward energy-efficient, high-performance computing technologies.

Data availability statement

The Python library described in this work is released on PyPi as rcbench (version 0.1.50). The code repository, including documentation and hands-on examples, can be found on GitHub (https://github.com/nanotechdave/RCbench). The code release is available in Zenodo (https://zenodo.org/records/18607281).

All data that support the findings of this study are included within the article (and any supplementary files).

Supplementary Information available at https://doi.org/10.1088/2634-4386/ae441f/data1.

Acknowledgment

G.M. acknowledge funding by the European Union (ERC, "MEMBRAIN", No. 101160604). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. A.C., C.G., C.R., and G.M acknowledge funding by NEURONE, a project funded by the European Union—Next Generation EU, M4C1 CUP I53D23003600006, under program PRIN 2022 (prj code 20229JRTZA).

Author contributions

Davide Pilati 0009-0000-4785-0537
Conceptualization (equal), Investigation (equal), Methodology (equal), Software (lead), Writing – original draft (lead)

Andrea Ceni 0000-0002-5084-0505
Methodology (equal), Writing – review & editing (equal)

Fabio Michieletti 0009-0006-6425-8782
Methodology (equal), Writing – review & editing (equal)

Claudio Gallicchio 0000-0002-6692-2564
Funding acquisition (equal), Methodology (equal), Supervision (equal), Writing – review & editing (equal)

Carlo Ricciardi 0000-0002-4703-7949
Funding acquisition (equal), Methodology (equal), Supervision (equal), Writing – review & editing (equal)

Gianluca Milano 0000-0002-1983-6516
Conceptualization (equal), Funding acquisition (equal), Methodology (equal), Supervision (equal), Writing – original draft (equal)

References

[1] OpenAI *et al* 2023 GPT-4 technical report
[2] Guo D *et al* 2025 DeepSeek-R1: incentivizing reasoning capability in LLMs via reinforcement learning
[3] Lukoševičius M, Jaeger H and Schrauwen B 2012 Reservoir computing trends *Kunstl. Intell.* **26** 365–71
[4] Gauthier D J, Bollt E, Griffith A and Barbosa W A S 2021 Next generation reservoir computing *Nat. Commun.* **12** 5564
[5] Kim J Z and Bassett D S 2023 A neural machine code and programming framework for the reservoir computer *Nat. Mach. Intell.* **5** 622–30
[6] Jaeger H 2002 *Tutorial on Training Recurrent Neural Networks, Covering BPTT, RTRL, EKF and the Echo State Network Approach* GMD-Forschungszentrum Informationstechnik vol 5
[7] Trouvain N, Pedrelli L, Dinh T T and Hinaut X 2020 ReservoirPy: an efficient and user-friendly library to design echo state networks *ICANN 2020–29th Int. Conf. on Artificial Neural Networks*
[8] Youel H, Prestwood D, Lee O, Wei T, Stenning K D, Gartside J C, Branford W R, Everschor-Sitte K and Kurebayashi H PRCpy: a python package for processing of physical reservoir computing manual (https://doi.org/10.1021/acsnano.5c02576)
[9] Jaeger H 2001 The "echo state" approach to analysing and training recurrent neural networks—with an erratum note *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* vol 148
[10] Maass W, Natschläger T and Markram H 2002 Real-time computing without stable states: a new framework for neural computation based on perturbations *Neural Comput.* **14** 2531–60
[11] Tanaka G, Yamane T, Héroux J B, Nakane R, Kanazawa N, Takeda S, Numata H, Nakano D and Hirose A 2019 Recent advances in physical reservoir computing: a review *Neural Netw.* **115** 100–23
[12] Yan M, Huang C, Bienstman P, Tino P, Lin W and Sun J 2024 Emerging opportunities and challenges for the future of reservoir computing *Nat. Commun.* **15** 2056
[13] Cucchi M, Abreu S, Ciccone G, Brunner D and Kleemann H 2022 Hands-on reservoir computing: a tutorial for practical implementation *Neuro Comput. Eng.* **2** 32002
[14] Vermesan O *et al* 2022 Internet of robotic things—converging sensing/actuating, hyperconnectivity, artificial intelligence and IoT platforms *Cognitive Hyperconnected Digital Transformation* (River Publishers) pp 97–155
[15] Lee O *et al* 2024 Task-adaptive physical reservoir computing *Nat. Mater.* **23** 79–87
[16] Fernando C and Sojakka S 2003 Pattern recognition in a bucket pp 588–97
[17] Vandoorne K, Mechet P, Van Vaerenbergh T, Fiers M, Morthier G, Verstraeten D, Schrauwen B, Dambre J and Bienstman P 2014 Experimental demonstration of reservoir computing on a silicon photonics chip *Nat. Commun.* **5** 1–6
[18] Soriano M C, Ortin S, Keuninckx L, Appeltant L, Danckaert J, Pesquera L and van der Sande G 2015 Delay-based reservoir computing: noise effects in a combined analog and digital implementation *IEEE Trans. Neural Netw. Learn. Syst.* **26** 388–93
[19] Appeltant L, Soriano M C, Van der Sande G, Danckaert J, Massar S, Dambre J, Schrauwen B, Mirasso C R and Fischer I 2011 Information processing using a single dynamical node as complex system *Nat. Commun.* **2** 468
[20] Thomas A 2013 Memristor-based neural networks *J. Phys. D: Appl. Phys.* **46** 093001
[21] Kulkarni M S and Teuscher C 2012 Memristor-based reservoir computing *Proc. 2012 IEEE/ACM Int. Symp. on Nanoscale Architectures* (ACM) pp 226–32
[22] Du C, Cai F, Zidan M A, Ma W, Lee S H and Lu W D 2017 Reservoir computing using dynamic memristors for temporal information processing *Nat. Commun.* **8** 2204
[23] Mallinson J B, Steel J K, Heywood Z E, Studholme S J, Bones P J and Brown S A 2024 Experimental demonstration of reservoir computing with self-assembled percolating networks of nanoparticles *Adv. Mater.* **36** 2402319
[24] Milano G, Pedretti G, Montano K, Ricci S, Hashemkhani S, Boarino L, Ielmini D and Ricciardi C 2021 In materia reservoir computing with a fully memristive architecture based on self-organizing nanowire networks *Nat. Mater.* **21** 195–202
[25] Hochstetter J, Zhu R, Loeffler A, Diaz-Alvarez A, Nakayama T and Kuncic Z 2021 Avalanches and edge-of-chaos learning in neuromorphic nanowire networks *Nat. Commun.* **12** 1–13
[26] Pilati D, Michieletti F, Cultrera A, Ricciardi C and Milano G 2024 Emerging spatiotemporal dynamics in multiterminal neuromorphic nanowire networks through conductance matrices and voltage maps *Adv. Electron. Mater.* **10** 2400750
[27] Milano G, Michieletti F, Pilati D, Ricciardi C and Miranda E 2025 Self-organizing neuromorphic nanowire networks as stochastic dynamical systems *Nat. Commun.* **16** 3509
[28] Mohamed A S, Armendarez N X, Hasan M S and Najem J S 2025 Memimpedance-based biomolecular device for adaptive physical reservoir computing *Adv. Intell. Syst.* **7** 2500281
[29] Pei M, Zhu Y, Liu S, Cui H, Li Y, Yan Y, Li Y, Wan C and Wan Q 2023 Power-efficient multisensory reservoir computing based on Zr-doped HfO_2 memcapacitive synapse arrays *Adv. Mater.* **35** 2305609
[30] Pfeifer R and Bongard J 2006 *How the Body Shapes the Way We Think* (The MIT Press)
[31] Hauser H, Ijspeert A J, Füchslin R M, Pfeifer R and Maass W 2011 Towards a theoretical foundation for morphological computation with compliant bodies *Biol. Cybern.* **105** 355–70
[32] Caluwaerts K, Despraz J, Işçen A, Sabelhaus A P, Bruce J, Schrauwen B and SunSpiral V 2014 Design and control of compliant tensegrity robots through simulation and hardware validation *J. R. Soc. Interface* **11** 20140520
[33] Nakajima K, Hauser H, Li T and Pfeifer R 2015 Information processing via physical soft body *Sci. Rep.* **5** 10487
[34] Goudarzi A, Lakin M R and Stefanovic D 2013 DNA reservoir computing: a novel molecular computing approach pp 76–89
[35] Yamane T, Katayama Y, Nakane R, Tanaka G and Nakano D 2015 *Wave-Based Reservoir Computing by Synchronization of Coupled Oscillators* pp 198–205
[36] Jones B, Stekel D, Rowe J and Fernando C 2007 Is there a liquid state machine in the bacterium escherichia coli? *2007 IEEE Symp. on Artificial Life* (IEEE) pp 187–91
[37] Buonomano D V and Maass W 2009 State-dependent computations: spatiotemporal processing in cortical networks *Nat. Rev. Neurosci.* **10** 113–25
[38] Dockendorf K P, Park I, He P, Príncipe J C and DeMarse T B 2009 Liquid state machines and cultured cortical networks: the separation property *Biosystems* **95** 90–97
[39] Sarles S A, Najem J S and Mohamed A S 2025 Voltage-responsive biomimetic membranes and ion channels for neuromorphic computing *npj Unconv. Comput.* **2** 26
[40] Najem J S, Hasan M S, Williams R S, Weiss R J, Rose G S, Taylor G J, Sarles S A and Collier C P 2019 Dynamical nonlinear memory capacitance in biomimetic membranes *Nat. Commun.* **10** 3239
[41] Inubushi M and Yoshimura K 2017 Reservoir computing beyond memory-nonlinearity trade-off *Sci. Rep.* **7** 10199

[42] Verstraeten D, Dambre J, Dutoit X and Schrauwen B 2010 Memory versus non-linearity in reservoirs *Proc. Int. Joint Conf. on Neural Networks*
[43] Dambre J, Verstraeten D, Schrauwen B and Massar S 2012 Information processing capacity of dynamical systems *Sci. Rep.* **2** 1–7
[44] Wringe C, Trefzer M and Stepney S 2025 Reservoir computing benchmarks: a tutorial review and critique *Int. J. Parallel Emergent Distrib. Syst.* **40** 313–51
[45] Jaeger H 2002 Short term memory in echo state networks *Technical Report 152, GMD*
[46] Rodan A and Tiňo P 2010 Simple deterministically constructed recurrent neural networks pp 267–74
[47] Dale M, Miller J F, Stepney S and Trefzer M A 2019 A substrate-independent framework to characterize reservoir computers *Proc. R. Soc. A* **475** 20180723
[48] Weigend A S 2018 *Time Series Prediction* (Routledge)
[49] Connor J T, Martin R D and Atlas L E 1994 Recurrent neural networks and robust time series prediction *IEEE Trans. Neural Netw.* **5** 240–54
[50] Fujii K and Nakajima K 2017 Harnessing disordered-ensemble quantum dynamics for machine learning *Phys. Rev. Appl.* **8** 024030
[51] Atiya A F and Parlos A G 2000 New results on recurrent network training: unifying the algorithms and accelerating convergence *IEEE Trans. Neural Netw.* **11** 697–709
[52] Michieletti F, Pilati D, Milano G and Ricciardi C 2025 Self-organized criticality in neuromorphic nanowire networks with tunable and local dynamics *Adv. Funct. Mater.* **35** 2423903
[53] Vidamour I T *et al* 2022 Quantifying the computational capability of a nanomagnetic reservoir computing platform with emergent magnetisation dynamics *Nanotechnology* **33** 485203
[54] Daniels R K, Mallinson J B, Heywood Z E, Bones P J, Arnold M D and Brown S A 2022 Reservoir computing with 3D nanowire networks *Neural Netw.* **154** 122–30
[55] Loeffler A, Zhu R, Hochstetter J, Diaz-Alvarez A, Nakayama T, Shine J M and Kuncic Z 2021 Modularity and multitasking in neuro-memristive reservoir networks *Neuro Comput. Eng.* **1** 14003
[56] Rodriguez N, Izquierdo E and Ahn Y Y 2019 Optimal modularity and memory capacity of neural reservoirs *Netw. Neurosci.* **3** 551–66
[57] Dale M 2018 Neuroevolution of hierarchical reservoir computers *Proc. Genetic and Evolutionary Computation Conf.* (ACM) pp 410–7
[58] Vinckier Q, Duport F, Smerieri A, Vandoorne K, Bienstman P, Haelterman M and Massar S 2015 High-performance photonic reservoir computer based on a coherently driven passive cavity *Optica* **2** 438
[59] Legenstein R and Maass W 2007 Edge of chaos and prediction of computational performance for neural circuit models *Neural Netw.* **20** 323–34
[60] Büsing L, Schrauwen B and Legenstein R 2010 Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons *Neural Comput.* **22** 1272–311
[61] Antonik P, Hermans M, Haelterman M and Massar S 2018 Random pattern and frequency generation using a photonic reservoir computer with output feedback *Neural Process. Lett.* **47** 1041–54
[62] Ceni A and Gallicchio C 2025 Edge of stability echo state network *IEEE Trans. Neural Netw. Learn. Syst.* **36** 7555–64
[63] Bagnall T AEON classification collection
[64] Hexagon M L UCR time series classification archive
[65] Chang W T*et al* 2020 MONASH UEA & UCR time series extrinsic regression repository (https://doi.org/10.5281/zenodo.3902651)
[66] Radeef Z M, Hashem S H and Gbashi E K 2024 New feature selection using principal component analysis *J. Soft Comput. Comput. Appl.* **1** 4
[67] Li Z and Qiu Y 2023 Feature selection based on improved principal component analysis *Proc. 2023 2nd Asia Conf. on Algorithms, Computing and Machine Learning* (ACM) pp 188–92
[68] Milano G, Montano K and Ricciardi C 2023 In materia implementation strategies of physical reservoir computing with memristive nanonetworks *J. Appl. Phys.* **56** 084005