Real-Time and Adaptive Reservoir Computing With Application to Profile Prediction in Fusion Plasma

Azarakhsh Jalalvand, *Member, IEEE*, Joseph Abbate, Rory Conlin, Geert Verdoolaege, *Member, IEEE*, and Egemen Kolemen

***Abstract*— Nuclear fusion is a promising alternative to address the problem of sustainable energy production. The tokamak is an approach to fusion based on magnetic plasma confinement, constituting a complex physical system with many control challenges. We study the characteristics and optimization of reservoir computing (RC) for real-time and adaptive prediction of plasma profiles in the DIII-D tokamak. Our experiments demonstrate that RC achieves comparable results to state-of-the-art (deep) convolutional neural networks (CNNs) and long short-term memory (LSTM) models, with a significantly easier and faster training procedure. This efficient approach allows for fast and frequent adaptation of the model to new situations, such as changing plasma conditions or different fusion devices.**

***Index Terms*— Adaptive learning, condition monitoring, nuclear fusion, reservoir computing (RC), tokamak plasma.**

I. Introduction

THE research on controlled nuclear fusion aims at the development of a source of power that is clean, safe, and as good as inexhaustible. Among the various approaches, the tokamak configuration, based on magnetic confinement of a hot hydrogen isotope plasma, is currently the most advanced one. Plasma performance is often quantified using the triple product $nT\tau_E$, where n is the density of particles, T is their temperature, and τ_E is the energy confinement time, a measure of how long it takes energetic particles to leave the plasma. To achieve ignition, the triple product must exceed 3×10^{21} keV s/m^3 (measuring temperatures in eV is standard in plasma physics, with 1 eV $\sim$ 11 606 K). In simpler terms, there must be enough particles (n) at high enough temperature (T) for a long enough time (τ_E) for enough particles to undergo fusion and make reaction self-sustaining. Although the goal is generally to increase temperature and density, steep gradients can drive instabilities that reduce confinement, and there are upper limits on how hot and dense the plasma can be before the magnetic fields can no longer contain it. Therefore, understanding and control of the physical mechanisms governing the transport of heat and particles through the plasma are essential to optimize the plasma confinement, hence fusion performance, while ensuring machine operation within the design limits. In turn, this requires careful monitoring and control of the plasma properties, such as density and temperature throughout the plasma volume. In particular, model-based prediction of plasma properties, based on measurements of the current and past state of the plasma, has the potential to greatly facilitate maintaining the plasma in the desired state, using a variety of actuators.

Physics models are currently not always sufficiently accurate or computationally too demanding to be used for plasma control, particularly when certain plasma instabilities arise, which may eventually lead to a complete loss of plasma confinement in a disruption [1], [2]. In this regard, data-driven techniques are being studied to simulate the evolution of important plasma properties and predict the plasma state on a very short time scale. In particular, artificial neural networks (ANNs) have been extensively studied for plasma evolution monitoring tasks. Researchers at TEXT [3], DIII-D [4], ASDEX [5], and JET [6], [7] have presented proof-of-concept ANN-based models to monitor the plasma state and detect disruptions by using diagnostic data available in real time as input signals. Recently, a disruption predictor combining recurrent and convolutional neural networks (CNNs) has been developed by Kates-Harbeck *et al.* [8]. The neural network has also been applied to analyze the tearing mode (TM) on JET [9]. Other machine learning methods, such as support vector machines [10], [11], discriminant analysis [12] and ensemble methods [13]–[15] have been studied as well with a view to disruption prediction.

The advanced machine-learning models that have been used so far for plasma state prediction are usually developed by training, optimizing, and testing the model with large

Manuscript received September 14, 2020; revised February 11, 2021; accepted May 18, 2021. Date of publication June 11, 2021; date of current version June 2, 2022. This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, DOE ARPA-E, using the DIII-D National Fusion Facility, a DOE Office of Science user facility, under Award DC-AC02-09CH11466, Award DE-SC0015480, Award DE-SC0015878, Award DE-FC02-04ER54698, Award DE-AR0001166, Award DE- SC0021275, and under Field Work Proposal 1903. The work of Azarakhsh Jalalvand was supported by the Ghent University through the Special Research Award under Award BOF19/PDO/134. *(Corresponding author: Azarakhsh Jalalvand.)*

Azarakhsh Jalalvand is with the Department of Electronics and Information Systems, Ghent University, 9052 Ghent, Belgium, and also with the Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: azarakhsh.jalalvand@ugent.be).

Joseph Abbate is with the Department of Astrophysical Sciences, Princeton University, Princeton, NJ 08544 USA, and also with the Princeton Plasma Physics Laboratory, Princeton, NJ 08543 USA.

Rory Conlin and Egemen Kolemen are with the Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ 08544 USA, and also with the Princeton Plasma Physics Laboratory, Princeton, NJ 08543 USA.

Geert Verdoolaege is with the Department of Applied Physics, Ghent University, 9000 Ghent, Belgium.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3085504.

Digital Object Identifier 10.1109/TNNLS.2021.3085504

datasets. This is an expensive process for complex models, such as deep neural networks [8], which requires highly skilled individuals, large datasets, huge computational resources and capabilities, and a lot of time. Unfortunately, current neural network models are not adaptable to incremental changes in the environmental conditions, despite the intense investment of time and resources. For instance, it has proven difficult to generalize the performance of disruption predictors between operational conditions or from one fusion device to another. Without adaptability, the retraining of models must be performed by submitting new training datasets, again requiring a huge investment of resources and time.

In [16], an adaptive ensemble of classification and regression trees have been proposed for disruption prediction in plasma. The adaptation strategy is based on monitoring the performance of the model after each run of tokamak and updating the training set once a classification error has occurred or an alarm has been triggered. This idea was later expanded to prevention and mitigation of disruption as well as transferring and adapting the model from one machine to another [17]. In another study, Humbird *et al.* [18] trained a five-layer fully connected neural network on simulated data and transferred the learning from simulation to experimental data by freezing the first three layers and retraining the last two. The results show that this is an effective, but slow, approach to calibrate the neural network.

To address the challenges of neural network training for plasma state prediction, in this study we employ adaptive reservoir computing (RC) [19]–[22]. To our knowledge, this is the first application of RC in the context of fusion research. RC has been shown to be very effective for a variety of tasks, e.g., in the analysis of high-dimensional data and time-evolving chaotic systems [23]–[26]. We demonstrate the performance of an RC-based model for space-resolved prediction of density and temperature in the DIII-D tokamak, given information on the present plasma state and future information on the actuator settings. We provide a perceptual understanding about the impact of the hyperparameters on the memory concept of the reservoir. This is followed by a detailed description on how to develop and tune the network to achieve the optimal performance.

Furthermore, we investigate the potential of a RC-based models in real-time adaptation to recent input. Adapting RC has already been shown to be fast and effective in noise-robust speech processing [27] and fluid turbulence analysis [28]. This is an important asset of this type of neural networks that discriminates them from alternative complex data-driven models, such as deep neural networks.

The rest of this article is organized as follows: Section II gives an overview of RC. In Section III, we explore the main hyper-parameters of the RC and study their impact on the reservoir states. Section IV describes the collected dataset followed by the experimental results in SectionV. The article ends with a brief conclusion and ideas for future work.

II. Reservoir Computing (RC)

RC is a neural network with two particular computational layers: 1) a hidden layer of recurrently interconnected nonlinear neurons, driven by inputs and by delayed feedback

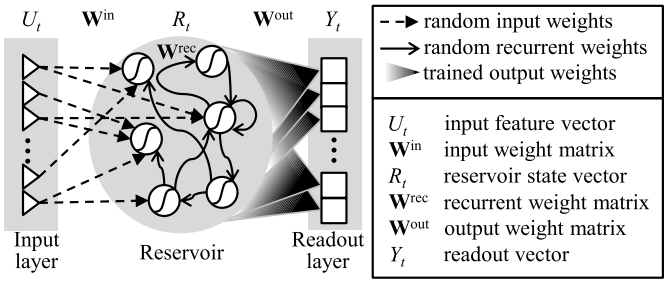


Fig. 1. Basic RC consists of a reservoir and a readout layer. The reservoir is composed of interconnected nonlinear neurons with fixed random weights. The readout layer consists of linear neurons with trained weights.

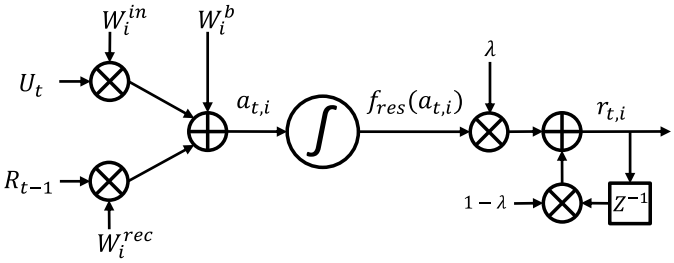


Fig. 2. Visualization of (1) describing leaky integrator neuron i.

of its activation and 2) an output layer of linear neurons, driven by the hidden neuron activation (Fig. 1). A fundamental point is that the input weights and the recurrent connection weights are initialized randomly, and only the output weights are optimized (trained) for solving the targeted problem.

The recurrently interconnected hidden neurons constitute a reservoir (a pool) of computational neurons. The reservoir can be viewed as a nonlinear dynamical system that analyzes a stream of inputs, e.g., time-series data. The outputs are usually called readouts [29], so as to differentiate them unambiguously from the reservoir outputs. If U_t, R_t, and Y_t represent the reservoir inputs, the reservoir outputs and the readouts at time t, the RC equations can be written as follows:

$$R_t = (1-\lambda)R_{t-1} + \lambda f_{\text{res}}(\mathbf{W}^{\text{in}}U_t + \mathbf{W}^{\text{rec}}R_{t-1} + \mathbf{W}^b) \quad (1)$$

$$Y_t = \mathbf{W}^{\text{out}}R_t \quad (2)$$

with λ being a leaking rate between 0 and 1, with f_{res} being the nonlinear activation function of the reservoir neurons (we used hyperbolic tangent in this work) and with $\mathbf{W}^{\text{in}}$, $\mathbf{W}^{\text{rec}}$, $\mathbf{W}^b$, and $\mathbf{W}^{\text{out}}$ being the input, recurrent, bias, and output weight matrices, respectively. Equation (1) and Fig. 2 represent a leaky integration of the neuron activation.

The weights of the hidden neurons are fixed by means of a random process that is characterized by four parameters [30]: 1) α_U, the maximal absolute eigenvalue of the input weight matrix $\mathbf{W}^{\text{in}}$; 2) ρ, also known as spectral radius, the maximal absolute eigenvalue of the recurrent weight matrix $\mathbf{W}^{\text{rec}}$; (3) K^{in}, the number of inputs driving each reservoir neuron; and 4) K^{rec}, the number of delayed reservoir outputs driving each reservoir neuron. The first two parameters control the relative importance of the inputs and the delayed reservoir outputs in the reservoir neuron activation. The latter two

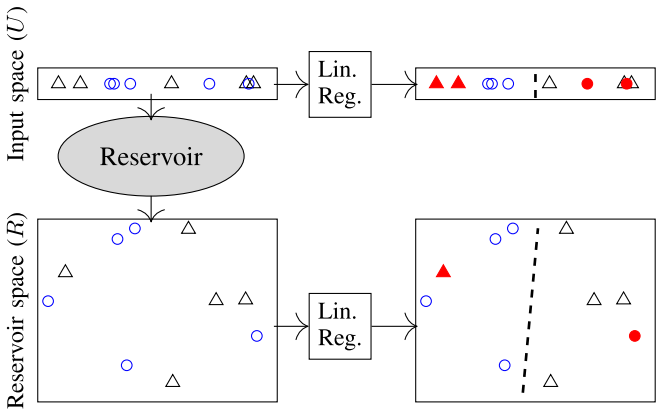


Fig. 3. Example of applying a 2-node reservoir to a 1-D dataset. Top left is the data feature space, and the top right shows the outcome of linear regression to classify the samples. Bottom left is the result of feeding the above samples to the reservoir, and bottom right shows the optimal regression on this new space. The black dashed line shows the hyperplane determined by the regression model and the miss-classified samples are marked with red.

control the sparsity of the input and the recurrent weight matrices. $\mathbf{W}^b$ is also initialized randomly and rescaled by α_b as a hyperparameter.

A. How Does RC Work?

In essence, the reservoir can be seen as a random fixed projector that nonlinearly projects the input sample, represented by an input feature vector of size N^{in}, to a (usually much) higher dimensional feature space of size N^{res}. Assuming that the complexity (e.g., the nonlinearity) of the given task and data is an obstacle to easily (e.g., linearly) separate the data samples, the hypothesis is that the reservoir projects the inputs ($\mathbf{U}$) to a new feature space $\mathbf{R}$ and facilitates the linear separation of the data samples. In Fig. 3, we provide a simple example, which can help to understand how this projection works. It shows a small dataset consisting of ten 1-D samples randomly labeled as two classes and the task is to train a linear classifier based on the available data. Obviously, there is no linear model to perfectly separate these two classes and the best model scores 6 out of 10.

By initializing a RC model of size 2, each of the 1-D samples is projected to a new 2-D feature space. Consequently, we are able to train a linear model on this space, which can better classify the data and achieve a score of 8. Although a valid concern would be the sensitivity of the performance to the random initialization of reservoir weights, studies show that the performance of the model is quite stable for sufficiently large reservoirs [31].

B. Training

The aim of the training is to find the output weights that minimize the mean squared difference between the readouts Y_t and their desired values D_t across N^{tr} available training examples. Introducing the matrices $\mathbf{R}$ and $\mathbf{D}$ with columns R_t and D_t, respectively, the output weights are the solution of a regularized Tikhonov regression problem [32]

$$\mathbf{W}^{\text{out}} = \arg\min_{\hat{\mathbf{W}}^{\text{out}}} \left(\frac{1}{N^{tr}} \left|\left| \hat{\mathbf{W}}^{\text{out}} \mathbf{R} - \mathbf{D} \right|\right|^2 + \epsilon \left|\left| \hat{\mathbf{W}}^{\text{out}} \right|\right|^2 \right) \quad (3)$$

with ϵ being the regularization parameter, which is intended to prevent overfitting to the training data. The solution is obtained in a closed-form [33] as

$$\mathbf{W}^{\text{out}} = (\mathbf{R}^\intercal \mathbf{R} + \epsilon\, \mathbf{I})^{-1} (\mathbf{R}^\intercal\, \mathbf{D}) \quad (4)$$

with $\mathbf{I}$ representing the identity matrix and $\mathbf{A}^{-1}$ the Moore-Penrose pseudo inverse of $\mathbf{A}$ [34]. Algorithm 1 presents the steps of training a RC.

Algorithm 1 Training RC

procedure INITIALIZE $\mathbf{W}^{in}$, $\mathbf{W}^{rec}$, $\mathbf{W}^{b}$
procedure EXECUTE THE RC ON THE SAMPLES
  $\mathcal{R} \leftarrow$ zero array of shape $(N^{res}+1) \times (N^{res}+1)$
  $\mathcal{D} \leftarrow$ zero array of shape $(N^{res}+1) \times N^{out}$
  $\mathcal{N}^{tr} \leftarrow 0$
  for each training shot of length T **do**
    $U \leftarrow$ feature array of shape $N^{in} \times T$
    $D \leftarrow$ target array of shape $N^{out} \times T$
    $R \leftarrow$ zero array of shape $N^{res} \times T$
    $R_0 \leftarrow$ zero array of shape $N^{res} \times 1$
    for t: 1 to T **do**
      $R_t \leftarrow \tanh(\mathbf{W}^{\text{in}} \times U_t + \mathbf{W}^{\text{rec}} \times R_t + \mathbf{W}^{b})$
      $R_t \leftarrow (1-\lambda) R_{t-1} + \lambda R_t$
    add a row of 1s to R for bias
    $\mathcal{N}^{tr} \leftarrow \mathcal{N}^{tr} + T$
    $\mathcal{R} \leftarrow \mathcal{R} + R^\intercal \times R$
    $\mathcal{D} \leftarrow \mathcal{D} + R^\intercal \times D$
procedure TRAIN THE OUTPUT WEIGHTS
  $\mathbf{W}^{\text{out}} = (\mathcal{R} + \epsilon\, \mathbf{I})^{-1} (\mathcal{D})$

C. Adaptation

Most predictive models are developed under the assumption that training and testing data are generated from a stationary process. However, this assumption often does not hold true in practice. For instance, changes in either the process configuration or machine calibration are the usual sources of concept drift in the data, which directly influences the performance of the data-driven prediction approaches [35]. In fusion devices, such drifts can arise when transitioning to a new regime of plasma operation, or when testing a model on a different machine. As a result, there is a need to retrain or adapt the deployed models to the changes of the environment. Although retraining is a very complex and expensive process in many state-of-the-art models such as deep neural networks, RC's easy one-shot training process makes it a very appealing alternative.

Because the readout nodes are linear, the linear transformation of readouts is equivalent to a linear transformation of the readout node parameters (weights). Learning the latter transformation can be formulated as training the readouts with the original training data supplemented with the adaptation data.

Based on Algorithm 1, instead of storing the whole training data we only need to store the matrices $\mathcal{R}$ (a symmetric matrix) and $\mathcal{D}$ along with the scalar $\mathcal{N}^{tr}$ after the training. By collecting $\mathcal{R}_A$, $\mathcal{D}_A$ and $\mathcal{N}_A^{tr}$ from the new (adaptation) data, the adapted readout weights can be obtained as

$$\mathbf{W}^{\text{out}} = ((1-\gamma)\mathcal{R} + \gamma\,\mathcal{R}_A + \epsilon\,\mathbf{I})^{-1}((1-\gamma)\mathcal{D} + \gamma\,\mathcal{D}_A) \tag{5}$$

where γ is a factor that controls how much the adaptation data contribute to these weights. In that regard $\gamma = 1$ is a special case in which the impact of the old samples are discarded when they become obsolete and therefore misleading. The cost of adaptation is discussed in Section V.

D. Parallel Training

Algorithm 1 suggests that the main steps of training RC are 1) initializing the input, recurrent and bias weight matrices; 2) accumulating $\mathcal{R}$ and $\mathcal{D}$ for the samples in the training set; and 3) applying ridge regression on the final accumulated $\mathcal{R}$ and $\mathcal{D}$. Assuming that executing the reservoir (lines 10–17 of the algorithm) for each training sample begins from the initial state of $R_0 = 0$, we can conclude that calculating $\mathcal{R}$ and $\mathcal{D}$ for each sample (each discharge in this dataset) is independent of the others. Therefore, it is possible to split the training dataset into batches of samples, execute each batch in a separate processor, and accumulate the aforementioned matrices afterward to calculate the output weights (see Algorithm 2).

Algorithm 2 Parallel Training of RC

1: Initialize $\mathbf{W}^{in}$, $\mathbf{W}^{rec}$, $\mathbf{W}^{b}$
2: Split the dataset to G arbitrary groups and distribute them to the available processors
3: **for** `each training group` g **do**
4: Calculate $\mathcal{R}_g$, $\mathcal{D}_g$ and $\mathcal{N}_g^{tr}$ (See Algorithm 1 procedure 2)
5: **procedure** TRAIN THE OUTPUT WEIGHTS
6: $\mathcal{N}^{tr} = \sum \mathcal{N}_g^{tr}$
7: $\mathcal{R} = \sum \mathcal{R}_g$
8: $\mathcal{D} = \sum \mathcal{D}_g$
9: $\mathbf{W}^{\text{out}} = (\mathcal{R} + \epsilon\,\mathbf{I})^{-1}(\mathcal{D})$

III. HYPERPARAMETERS OF RC

In this section, we analyze the impact of the important hyperparameters, namely the input scaling α_{U} and the spectral radius ρ along with the density of the $\mathbf{W}^{\text{in}}$ and $\mathbf{W}^{\text{rec}}$, the leakage λ, and the bias scaling α_{b}. Although there have been many studies on instructions for optimizing these hyperparameters for a specific use-case and available training data [22], [36]–[40], the aim of this section is to propose a training-free analysis that would provide a perceptual understanding of reservoir behavior independent of the use-case and quality of the training data. To that end, we feed a time-shifted unit impulse into a small RC and observe the absolute values of the impulse responses inside the reservoir. The impulse input is a multivariate time-series input U with an arbitrary input dimension of $N^{\text{in}} = 10$ and the duration of $T = 100$. The impulse occurs at $t = 5$, hence, the input U_t is a zero vector in all time steps except for time $t = 5$, where $U_5 = [1, 1, \ldots, 1]$.

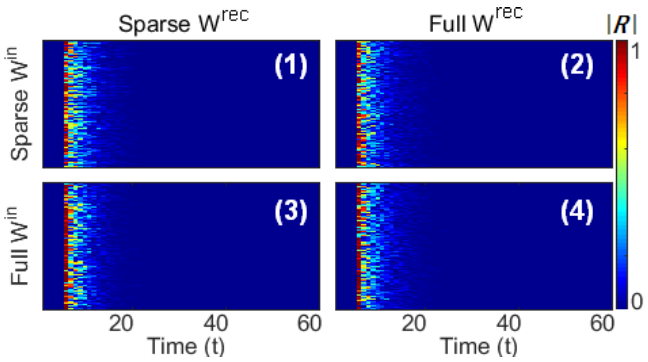


Fig. 4. Impulse response of a 100-node reservoir with (1) sparse input and recurrent connection matrices, (2) sparse input and full recurrent connection matrices, (3) full input and sparse recurrent connection matrices, and (4) full input and recurrent connection matrices.

A. Density of the Input and Recurrent Connections

Studies on different tasks show that the sparsity of the input and recurrent weights does not significantly influence the performance as long as the relative scales, α_U and ρ, are adjusted correctly [41], [42]. Nevertheless, it is of interest to verify whether the sparsity of the input and recurrent weights have noticeable impact on the reservoir activation. Fig. 4 shows the impulse response of a 100-node reservoir with combinations of sparse and dense input and recurrent weights. Although there is no clear impact on the shape and length of the activation (i.e., dynamical memory of reservoir), some arguments in favor of sparse connections are as follows:

1) In a fully connected network, all the nodes are triggered with all the input features and the difference in their activation only relies on the random connection weights given to each feature. On the contrary, in a sparsely connected network the activation of each node is different from the other not only because of the random weights, but also because each node is triggered by a different set of input features. This increases the diversity of information combination and, similar to dropout in deep learning, reduces the adverse effects of missing or noisy data, which could lead to more robustness of the model.
2) Employing sparse connections has a significant impact on the hardware efficiency of the model, especially when dealing with multidimensional inputs and/or when very large reservoirs are required. For instance, $\mathbf{W}^{\text{rec}}$ of a 16 000-node fully connected reservoir occupies around 2 GB of memory, whereas such a matrix for the same reservoir with only ten connections per node would occupy less than 2 MB.
3) Moreover, if we define the sparsity of input connections by the number of input connections to each reservoir node (K^{in}) instead of a total random selection, the total number of nonzero elements in $\mathbf{W}^{\text{in}}$ will be $N^{\text{res}} \times K^{\text{in}}$. Therefore, the size of $\mathbf{W}^{\text{in}}$, and consequently the required computational resources, will be independent of the dimension of input feature vector.

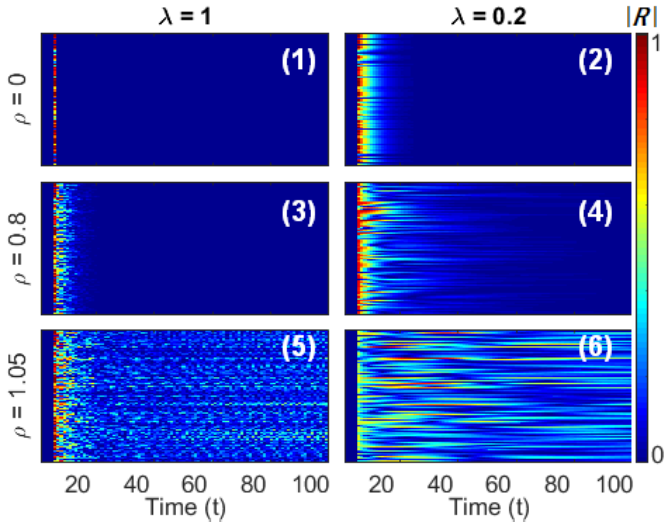


Fig. 5. Impulse response of a 100-node reservoir (1) with simple neurons and without recurrent connections, (2) with leaky integrated neurons and without recurrent connections, (3) with simple neurons and with recurrent connections, and (4) with leaky integrated neurons and with recurrent connections. (5) and (6) show the unstable regime of the reservoir when $\rho > 1$.

B. Memory Concept of RC

In a second experiment, we studied the impact of leaking rate (λ) and spectral radius (ρ) as the two parameters to control the memory of reservoirs. In theory, the leaky integration neurons (LINs) and the recurrent connections of the reservoir enable modeling of the short-term dynamics of motion. To visualize these dynamics, Fig. 5 shows the impulse response of a 100-node reservoir, with and without recurrent connections (controlled with ρ) and with and without LINs (controlled with λ). Obviously, the system without LIN and recurrent connections has no means to remember the past, hence the response is also an impulse. Such a model is known as conventional extreme learning machine (ELM) [43]. By replacing simple neurons with LINs, it is possible to provide a linear fading memory: the smaller the λ, the longer the effect of the past information but with weaker first response. In practice, this means that the state of each neuron will change faster or slower depending on the leakage λ. A reservoir with simple neurons and with recurrent connections benefits from a complex short-term dynamics. This information also fades out through time (if $\rho < 1$), but it is difficult to interpret, because of the nonlinearity of the neurons and random initialization of the recurrent weights. The reservoir with LINs and recurrent connections benefits from both of these memory concepts, and the combined information lasts longer than each individual ones.

Fig. 5 also shows the impulse responses for $\rho = 1.05$. We can see that the reservoir states are decaying very slowly, and they are oscillating with a resonance frequency. For many tasks it is necessary to keep $\rho < 1$, which preserves the echo state property of reservoir, stating that with time, the reservoir should forget the initial state it was in [29]. However, in some cases, the spectral radius can be larger than 1 [44].

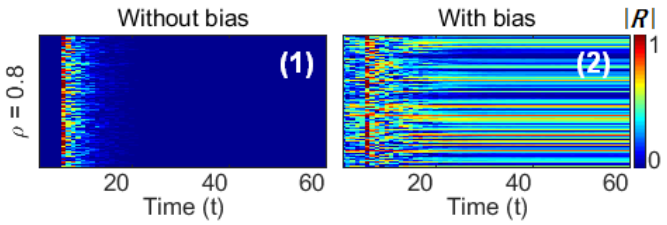


Fig. 6. Impulse response of a 100-node reservoir (1) with recurrent connections and without bias and (2) with recurrent connections and with bias. Bias helps to reach more nonlinear areas of the activation function by changing the stable state of the reservoir nodes.

C. Bias Scaling

Another hyperparameter to be studied is the bias scaling α_b. The stable state of all reservoir neurons without bias is the center value of the nonlinear activation function (tanh), i.e., 0. The bias scaling can help to reach more nonlinear areas of the activation function by changing the stable state of the reservoir nodes. This is particularly useful in the tasks when the nonlinearity of the model plays an important role on the performance.

In Fig. 6, we present the impact of bias in a reservoir without leaky integration ($\lambda = 1$) but with a spectral radius of $\rho = 0.8$. The absolute value of the stable states of the reservoir neurons is approximately distributed in the activation range and each neuron has its own stable state. When new information from the input is passed to the reservoir neurons, this is the excitation point.

IV. Dataset

In this work, data were collected from the DIII-D tokamak, from experimental campaigns covering the years 2010–2018. The MDSplus data management software was used to load the data [45], alongside the OMFIT software framework for data preprocessing [46]. A new module has been developed for this task and is publicly available within OMFIT. A tokamak is an inherently pulsed device, with the device settings and plasma conditions determined by the experimental program. A plasma pulse (discharge or "shot") broadly consists of a ramping-up phase of the plasma current, a current flat-top phase, and a current ramp-down. In this work, only data from the flat-top phase were used. DIII-D shots have a typical duration of the order of seconds.

All signals were resampled to a common 50 ms time base by averaging or nearest-neighbor interpolation. Our model considers data in 50 ms nonoverlapping frames, a spacing large enough to smooth over most irrelevant signal variations, like modulations of the injected power. The following three types of data were considered at each time step:

1) *Plasma profiles:* spatial distribution of a given quantity in the plasma, such as density and temperature. Although in general these would be functions of three real space coordinates, because of the toroidal symmetry, the plasma is homogeneous in the toroidal and poloidal directions. We only consider radial profiles, from the center of the vacuum vessel up to the plasma boundary.

TABLE I

SIGNALS COLLECTED FROM DIII-D TOKAMAK AND USED IN THE PROFILE PREDICTION NEURAL NETWORK

	Signal name	Units
Profiles	Electron Density (n_e)	$10^{19}/\text{m}^3$
	Electron Temperature (T_e)	keV
	Ion Rotation (Ω)	kHz
	Rotational Transform (ι)	-
	Plasma Pressure (P)	Pa
Actuators	Injected Power	MW
	Injected Torque	N·m
	Target Current	A
	Target Density	$10^{19}/\text{m}^3$
Scalar params	Top Triangularity	-
	Bottom Triangularity	-
	Plasma Elongation	-
	Plasma Volume	m^3
	Internal Inductance	μH
	Line Avg. Density	$10^{19}/\text{m}^3$

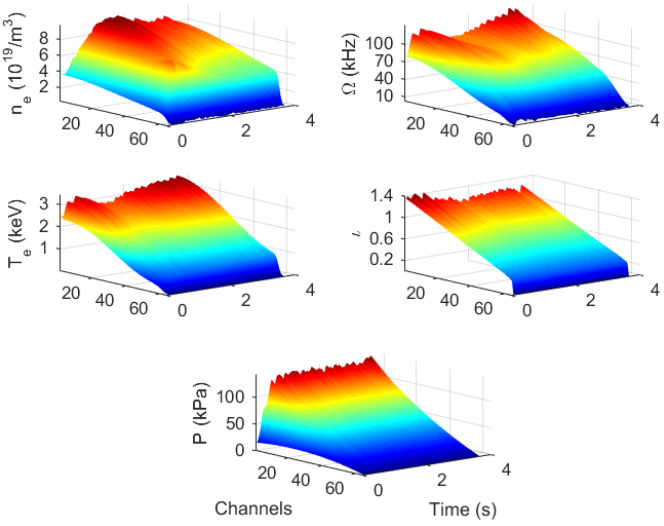


Fig. 7. Measured evolution of profiles of electron density (n_e), electron temperature (T_e), toroidal rotation (Ω), field line pitch (ι), and total pressure (P) in DIII-D shot #154971.

These profiles are discretized by their value at 65 equally spaced radial points, leading to a 65-D data vector per profile at time t.

2) Global plasma parameters, which characterize the overall plasma conditions. For instance, elongation and triangularity are the parameters that describe the shape of the plasma cross section. We also included global scalar properties, such as the total plasma volume, inductance, and average density.
3) Actuator settings, i.e., plasma and machine parameters, that allow the operator a certain amount of control over the plasma profiles.

The full list of signals used in this work is depicted in Table I.

Fig. 7 provides an example of the time evolution of the various profiles in one plasma discharge at DIII-D.

The input for the model is the plasma state at the current time step, consisting of the measured profiles and global plasma parameters. In addition, a "proposal" is given for the value of the actuators at each of the four time steps (200 ms) into the future and the algorithm predicts the change in each of the profiles 200 ms (four time steps) into the future. This 200-ms prediction window was chosen based on the typical energy confinement time (τ_E) at DIII-D, and was empirically found to be a period over which the profiles change noticeably while not so long that the future state cannot be reliably predicted. However, in Section V, we also briefly study the performance of the RC as a function of this prediction window.

All shots were sorted chronologically and split into 75%, 15%, and 10% for training, validation, and test sets. Therefore, the first 2750 shots (ca. 132 000 data-points) are used for training the models while the next 705 and 300 shots are assigned to the validation and test sets, respectively. Each signal is normalized by subtracting out the median value and dividing by the interquartile range.

It is important to note that 1) the modification of the hardware and control systems of DIII-D through years, intrinsically leads to changes in some characteristics of the shots and 2) the physicists usually experiment several shots in a row with the same setup. These consecutive shots, thus, presumably share similar characteristics.

V. EXPERIMENTAL RESULTS

We report the performance of the prediction model using root mean squared errors (RMSEs), based on the difference between the prediction of the model and the actual measurement for each signal, and normalized by the standard deviation of the actual data.

A. RC Hyperparameter Tuning

In Section III, we presented a perceptual understanding of the reservoir's hyperparameters consisting of (K^{in}, K^{rec}, α_U, ρ, λ) along with bias and the reservoir size. Setting up a suitable RC has been studied in detail in [22] for the task of speech recognition. The empirical findings of that work shows that there are simple and comprehensible rules that allow to design a reservoir in a structured manner rather than a naive and time-consuming grid search over all the parameters.

1) In order to optimize the hyperparameters, one can begin with a rather small size reservoir.
2) The input and recurrent weight matrices ($\mathbf{W}^{\text{in}}$ and $\mathbf{W}^{\text{rec}}$) can be very sparse. In particular, five to ten elements per node are enough, regardless of the size of the reservoir and the input feature vector.
3) ρ and α_U together control the relative importance of the inputs and recurrent neuron activation. Therefore, they can be tuned based on prior knowledge on the relation of these two activations (e.g., current input needs more weight than the past information) or a plain grid search.
4) The leaking rate λ can be tuned based on the minimum time (in scan steps) the reservoir output is expected to remain constant, i.e., the size of memory needed for the given task.
5) The last parameter to optimize is the size of the reservoir. The maximum size of the reservoir depends on the

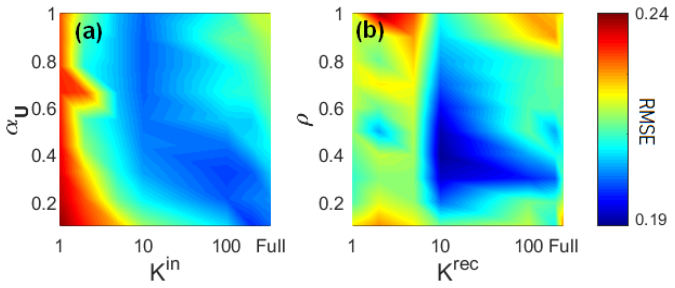


Fig. 8. RMSE on the validation set (a) as a function of the input scaling factor α_U and input connections sparsity K^{in} and (b) as a function of spectral radius ρ and recurrent connection sparsity K^{rec}.

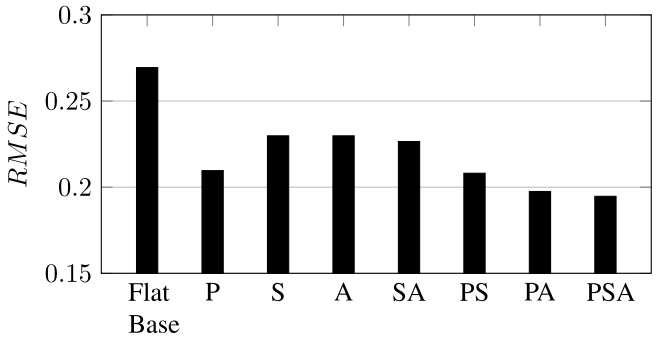


Fig. 9. Performance of the RC on the validation set as a function of the different combinations of inputs. P, S, and A refer to profiles, scalar parameters, and actuators, respectively. Flat Baseline assumes that the signal at the intended future will be exactly the same as now.

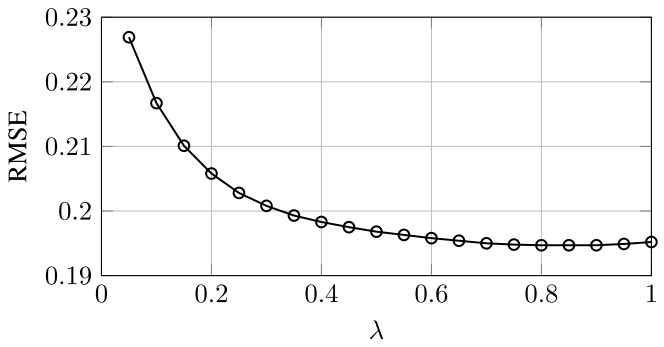


Fig. 10. Sweeping λ to find the optimal leaking memory for the reservoir shows that a marginal leakage ($\lambda = 0.85$) helps the model to better utilize the past information for predicting the future.

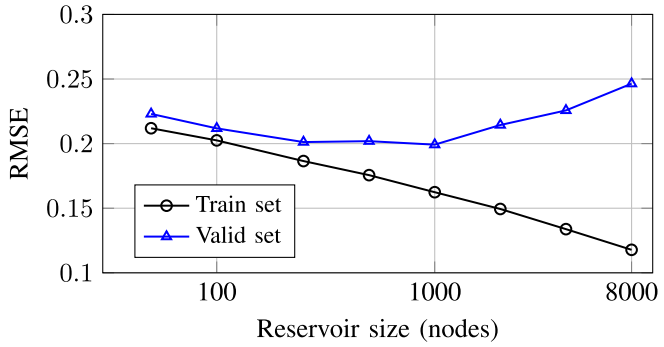


Fig. 11. Performance of RC on the training and validation set as a function of reservoir size (number of nodes). A medium-size reservoir of 1000 nodes seems to be enough to capture the most information from the training set without overfitting.

quality and quantity of the training data, the hardware limitations, and the acceptable processing time.

To optimize the input layer, we begin with a rather small reservoir of 250 nodes supplied with profiles, parameters, and actuators as inputs to predict the profiles. As a result, the input at each time step is a vector of size $N^{\text{in}} = 335$ (5 profiles of 65 each + 6 parameters + 4 actuators) to predict all five profiles ($N^{\text{out}} = 325$).

Fig. 8(a) shows the performance of an RC without recurrent connections as a function of α_U and K^{in}. The results confirm that the density of input connections is indeed not a bottleneck as long as the input weight scale is chosen correctly. Also, the linear relation of these parameters suggests that by changing one of them, the other one can be adjusted without sweeping. Fig. 8(b) depicts the outcome of a similar experiment on the recurrent weights. According to these results, 1) adding recurrent connections improves the performance of the RC and 2) a very sparse recurrent connection of only ten inputs per neuron is enough to achieve the optimal performance.

In the next experiment we study the impact of each of the three input categories (profiles, actuators, and scalar parameters) on the performance of the RC model. According to Fig. 9, although actuators and scalars fail to individually predict the profiles, actuators play a more important role in contributing to the current profiles to predict the future of the profiles. Nevertheless, because the dimension of the input vector has negligible impact on the processing complexity of RC, we feed all the available inputs to RC.

Fig. 10 presents the performance of an RC as a function of λ, and it shows that a rather small leakage ($\lambda = 0.85$) helps the reservoir to benefit from a short-term memory in predicting the profiles.

The last hyperparameter to be optimized is the size of the reservoir. According to Fig. 11, a reservoir of 200–1000 nodes is the optimal configuration. The total number of trainable parameters is in fact the size of $\mathbf{W}^{\text{out}}$ or $(N^{\text{res}}+1)\times N^{\text{out}}$. Given that the proposed model is supposed to predict five profiles of dimension 65 for each, the 1000-node RC has 325 325 parameters, which means that the overfitting starts to happen only when there are three times more model parameters compared to the training samples.

Although 200 ms has been empirically shown to be a suitable window for plasma behavior monitoring, we also investigate the performance of RC as a function of this prediction window. In Fig. 12, we compare the proposed RC model with two other references: 1) Flat Base: which assumes that the signal at the intended future will be exactly the same as now and 2) Lin Reg: which tries to predict the future by applying linear regression directly on the input features, i.e., bypassing the reservoir. This experiment shows that on average the reservoir-based model performs relatively 30% better than the Flat Baseline and also 15% better than the simple linear regression model.

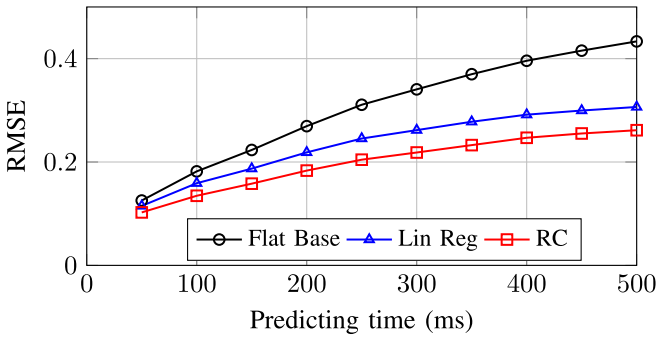


Fig. 12. RMSE of three models as a function of prediction window. RC performs relatively 30% better than the Flat Baseline and also 15% better than the simple linear regression model.

TABLE II

TIME COMPLEXITY AND ACTUAL TIMES (MEASURED ON AN INTEL CORE I7-3770) FOR THE DIFFERENT STEPS OF THE FULL TRAINING OF A RESERVOIR WITH 1000 NEURONS USING 2750 SHOTS (132 000 DATA POINTS). ITEMS TAGGED WITH (∗) CAN BE RUN IN PARALLEL

Action	Complexity	Time (s)
Compute R & D $(*)$	$\mathcal{O}(N^{tr}\,N^{res})$	25
Update $\mathcal{R}$ & $\mathcal{D}$ $(*)$	$\mathcal{O}(N^{tr}\,(N^{res})^2)$	20
Compute $\mathbf{W}^{out}$	$\mathcal{O}((N^{res})^2\,N^{out})$	0.08

TIME COMPLEXITY AND ACTUAL TIMES (MEASURED ON AN INTEL® CORE™ I7-3770) FOR THE DIFFERENT STEPS OF THE FULL TRAINING OF A RESERVOIR WITH 1000 NEURONS USING 2,750 SHOTS (132,000 DATA POINTS). ITEMS TAGGED WITH $(*)$ CAN BE RUN IN PARALLEL.

B. Training Time and Adaptation

An important asset of RC compared to state-of-the-art (deep) neural networks is its easy and fast training procedure. All the experiments in this work are conducted on a conventional CPU Intel Core i7-3770.

Table II lists the time complexity of each of the training steps as a function of data and reservoir size along with actual times in seconds to train a 1000-node RC on the given training dataset.

Collecting $\mathcal{R}$ and $\mathcal{D}$ (lines 2–17 of Algorithm 1) for the whole training dataset only takes 45 s (160 ms per shot) and training the output weights (line 19) is almost real-time (80 ms).

This suggests that by storing and updating $\mathcal{R}$ and $\mathcal{D}$, we can almost instantly adapt the RC model at the end of each shot, or even every few hundred milliseconds during a given shot.

Before investigating the adaptation ability of RC, we studied how much of the existing training data is in fact enough for prediction of the validation set. Therefore, we define three scenarios for gradually increasing the size of the training set and evaluating the trained model on the validation set described in Section IV.

1) Oldest shots: the model is trained with n_{tr} oldest shots of the training set. Given the time difference between the training and validation shots, the training set with small n_{tr} perhaps contain the least similar shots to the validation set.
2) Newest shots: the model is trained with n_{tr} most recent shots in the training set. These are perhaps the most similar shots to the validation set.

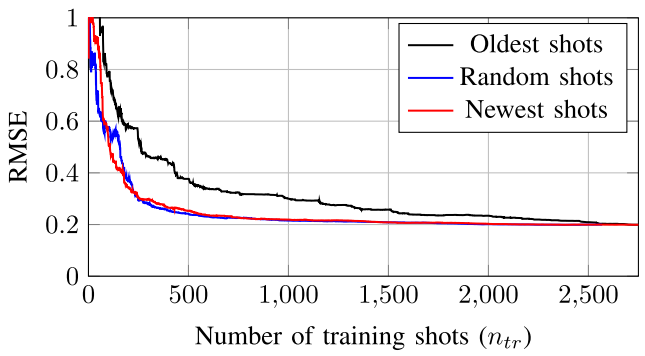


Fig. 13. Prediction performance of reservoir computing as a function of the training set consisting of n_{tr} oldest, newest, or random shots. The results suggest that training the RC with the most recent 1000 shots is almost as effective as training the RC with all 2750 available training shots.

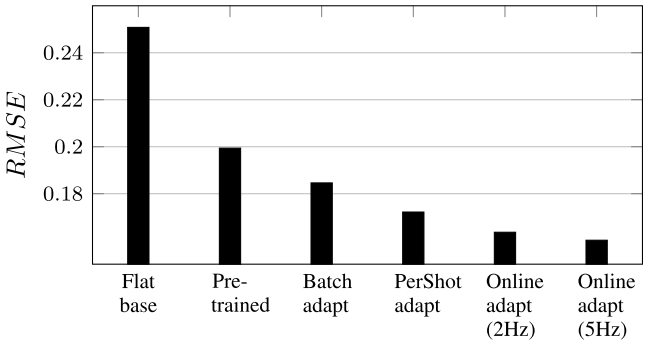


Fig. 14. Comparing the performance of adapting RC on the validation set using three approaches: batch adaptation, PerShot adaptation, and online adaptation. The latter has been studied with adaptation frequencies of 2 Hz (every 500 ms) and 5 Hz (every 200 ms).

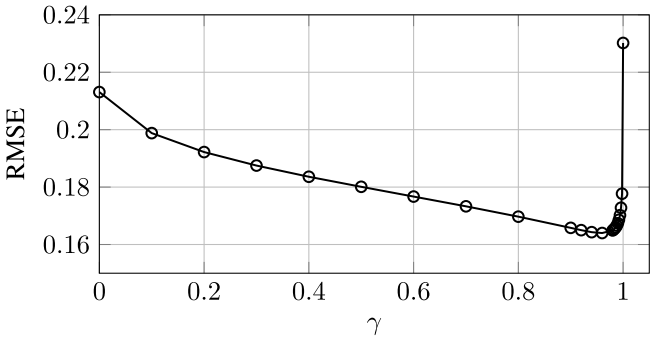


Fig. 15. Sweeping γ to find the optimal adaptation rate on the validation set. Although the optimal is rather high ($\gamma = 0.98$), discarding the previous data ($\gamma = 1$) causes overfitting.

3) Random shots: all shots are shuffled and randomly assigned to the training and validation set. The comparison of these scenarios should give an indication of how many shots, and in which order, is optimal for a pretrained RC.

In Fig. 13, we gradually increase the number of training shots in these scenarios, retrain the model using (3), and evaluate the performance of the model on the whole validation set after each step. The comparison between the Newest and Random scenarios shows that it is not necessary to train the

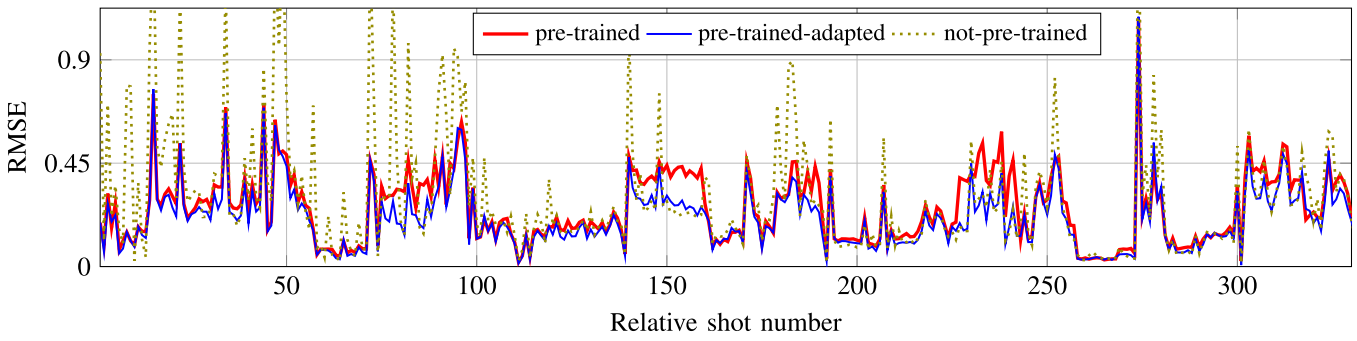


Fig. 16. Performance comparison of an RC only trained on the training data (pretrained), the same RC getting adapted to the test data (pretrained-adapted), and an RC learning the task from scratch on the test shots. The actual shot numbers in the test set start at 176 522.

RC with a very long history of shots. In fact, the model trained on the newest 1000 shots performs equally well as a model that has been trained on the oldest 3000 shots. This is important to avoid unnecessary complexity of the training procedure. Furthermore, the random scenario confirms that it is indeed redundant to train the model on all the available data. Only a quarter of the available dataset contains almost all useful information to train the prediction model.

To study the adaptation capability of RC, we first train the model on the available training set and define three scenarios to adapt the model to the new environment, which is the validation set: 1) batch adaptation, in which the output weights are updated only after every batch of ten shots; 2) PerShot adaptation, in which the output weights are updated after every shot of the validation set; and 3) online adaptation, in which the model is adapted every 500 ms "during" the shot. In all scenarios, $\mathbf{W}^{\text{out}}$ is pretrained by collecting $\mathcal{R}$ and $\mathcal{D}$ on the training data. During the adaptation, $\mathbf{W}^{\text{out}}$ is updated gradually by calculating $\mathcal{R}_A$ and $\mathcal{D}_A$ from the validation set and updating $\mathcal{R}$ and $\mathcal{D}$ [see (5)].

In Fig. 14, we evaluate these approaches and compare them with the nonadapted (pretrained) RC, as well as the Flat Baseline. As one can expect, increasing the frequency of adaptation improves the prediction performance of the model.

Fig. 15 presents the impact of the adaptation rate [γ in (3)] on the performance of the model. Because the size of the validation set is significantly smaller than the training set, it is logical to observe that the optimal adaptation rate is rather high ($\gamma = 0.98$). However, discarding the previous data ($\gamma = 1$) results in overfitting.

Table III lists the total adaptation time and the number of adaptations on the validation set for a 1000-node RC based on the three aforementioned scenarios. The numbers show that even the online adaptation, which is the most intense scenario, can be accomplished fast enough on a conventional CPU.

In the aforementioned scenarios, we assumed that there is a separated training dataset to pretrain the RC before evaluating and adapting the model under the test conditions. However, we also investigated how the RC would behave in case we begin with a nontrained RC and adapt it to the test environment as time evolves. Fig. 16 shows the performance of three RC, namely, 1) pretrained: the model is trained on the training set and evaluated on the test set; 2) pretrained-adapted: the model is pretrained on the training set and gets updated during the test; 3) not-pretrained: the model is trained on the test set gradually without any pretraining. This experiment shows that the pretrained-adapted model outperforms the other two, especially compared to the pretrained model toward the end of the test. Second, the not-trained model starts with poor prediction, as can be expected, but in the second half of the test (after around 250 shots) it gets close to the pretrained-adapted model and in some examples it even outperforms the pretrained model, which has been trained on 2 750 shots before testing.

TABLE III
TIME AND NUMBER OF ADAPTATIONS FOR THREE SCENARIOS OF ADAPTING A 1000-NODE RC TO A VALIDATION SET OF 705 SHOTS

	Batch adapt	PerShot adapt	Online adapt
Total adaptation time (s)	6	73	327
Number of adaptations	58	704	3392
Average time (ms)	103	104	96

As an example of the prediction of the proposed model, Fig. 17 depicts the five profiles in one particular shot from the test set. For each profile and at each time step, we show the average over its 65 channels of target value, along with the predictions of the pretrained RC, PerShot, and the Online adaptation. Although the pretrained RC fails to predict the second half of the signals, which exhibits a significantly different pattern compared to the first half, adapting the model just before this shot slightly helps to improve the prediction. On the contrary, adapting the model every 500 ms significantly improves the prediction.

C. Comparing RC With CNN-LSTM

CNNs in combination with Long-Short Term Memory cells have increasingly become popular approaches for many data analysis tasks involving time series [8]. However, their complex training procedure demands a large amount of training data and time. Furthermore, these systems usually achieve promising performance when the training data are rich and large enough and testing environments share common properties with training. In this section, we compare RC with a

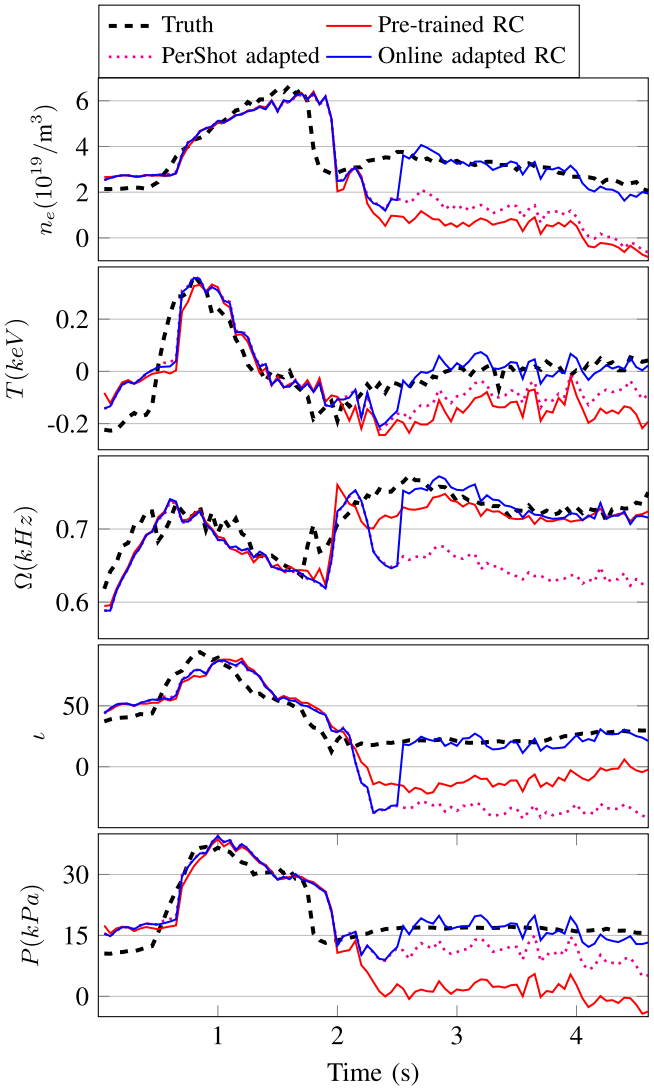


Fig. 17. Target and prediction of the spatial mean value of electron density (n_e), electron temperature (T_e), toroidal rotation (Ω), field line pitch (ι), and total pressure (P) for shot #176 796. Each data-point at time t is the spatial average of the profile at that time.

well-configured CNN-LSTM framework[1] in two scenarios: 1) when training and testing shots are randomly selected from a pool of shots recorded between 2010 and 2018 and 2) when the test shots have significantly different characteristics as they are collected from a different setup of tokamak in 2019. Fig. 18 presents the performance of RC and long short-term memory (LSTM) in these conditions. Although LSTM achieves promising scores in the matched conditions, it drastically fails on the newly recorded shots. The pretrained RC shows the same behavior, which likely means that there is a considerable difference between the new data and the training set. However, the fast adaptation ability of RC leads to 25% relative improvement. Although frequent adaptation of the CNN-LSTM model is practically very expensive, for the sake of completeness we also retrained this model on a

[1]This code is also publicly available, on GitHub (https://github.com/jabbate7/plasma-profile-predictor)

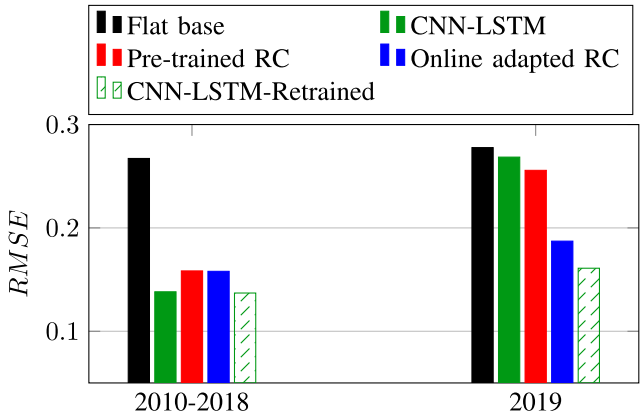


Fig. 18. Comparison of RC and CNN-LSTM when training and testing are from the same pool (collected in 2013–2018) and when the test shots are from a different tokamak setup and different year (2019).

combination of the training set and a subset of the test sets. As expected, the CNN-LSTM outperforms RC when it has already seen examples of the test data during training.

As far as the computational complexity is concerned, the training of the CNN-LSTM model took 5 h on one NVIDIA V100 GPU and eight IBM POWER9 CPU cores compared to 45 s for training the RC on a Core i7 CPU.

VI. Conclusion and Future Work

Many complex machine-learning paradigms, such as deep neural networks and CNNs, require large amount of data and time to learn the task, hence, they are usually difficult to adapt to the new conditions. We explored the potential of RC as an interesting alternative for data analysis and prediction tasks involving multidimensional time-series measured in complex physical systems. In this respect, the main assets of RC are their temporal information processing ability and yet very easy and fast training procedure.

In this study, we addressed real-time profile prediction for condition monitoring in fusion devices using RC. We laid special emphasis on model training complexity and adaptation under changing operational conditions, showing the possibility to update the model in a very efficient and near real-time fashion. In this particular setting, every adaptation takes only 100 ms, which, for instance, can be compared with the typical suggested prediction time of about 200 ms for disruption avoidance in tokamaks. We in fact plan to incorporate the proposed model in the DIII-D tokamak control system to study its application and adaptation in real world scenarios.

Moreover, we explored the main hyperparameters of RC and studied their impact on the activation and memory of the reservoir. For instance, our experiments showed that the input and recurrent connections of the model can be very sparse without compromising the model performance, which is useful for model storage. We also showed that the training of model can be accomplished in parallel. Thus, RC is suitable for pattern recognition in big data.

As a follow up to this work, we plan to utilize larger datasets covering a greater diversity of plasma conditions. In particular, although usually this is not how fusion devices

are operated, it would be extremely useful to obtain more data from shots with semirandom changes to actuators during the current flattop.

Furthermore, it would be interesting to investigate more complex architectures of RC, such as multilayer models, as well as more sophisticated topologies in connecting the reservoir neurons. For instance, a more structured input to reservoir connections would divide the reservoir neurons into segments, where each area learns specific properties of the patterns in the training data. This might lead to better understanding and control of the reservoir activation.

Finally and although in this article we focused on plasma profile prediction as a use-case, we believe that our findings can be easily generalized to other applications and across several disciplines.

ACKNOWLEDGMENT

Part of the data analysis was performed using the OMFIT integrated modeling framework.

DISCLAIMER

This report is prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

[1] F. C. Schuller, "Disruptions in tokamaks," *Plasma Phys. Controlled Fusion*, vol. 37, no. 11A, pp. A135–A162, Nov. 1995. [Online]. Available: https://doi.org/10.1088%2F0741-3335%2F37%2F11a%2F009

[2] P. de Vries *et al.*, "Survey of disruption causes at JET," *Nucl. Fusion*, vol. 51, no. 5, Apr. 2011, Art. no. 053018. [Online]. Available: https://doi.org/10.1088%2F0029-5515%2F51%2F5%2F053018

[3] J. Hernandez, A. Vannucci, T. Tajima, Z. Lin, W. Horton, and S. McCool, "Neural network prediction of some classes of tokamak disruptions," *Nucl. Fusion*, vol. 36, no. 8, p. 1009, 1996.

[4] D. Wroblewski, G. Jahns, and J. Leuer, "Tokamak disruption alarm based on a neural network model of the high-beta limit," *Nucl. Fusion*, vol. 37, no. 6, p. 725, 1997.

[5] G. Pautasso *et al.*, "Prediction and mitigation of disruptions in ASDEX upgrade," *J. Nucl. Mater.*, vols. 290–293, pp. 1045–1051, Mar. 2001.

[6] C. G. Windsor, G. Pautasso, C. Tichmann, R. J. Buttery, and T. C. Hender, "A cross-tokamak neural network disruption predictor for the JET and ASDEX upgrade tokamaks," *Nucl. Fusion*, vol. 45, no. 5, p. 337, 2005.

[7] B. Cannas *et al.*, "A prediction tool for real-time application in the disruption protection system at JET," *Nucl. Fusion*, vol. 47, no. 11, p. 1559, 2007.

[8] J. Kates-Harbeck, A. Svyatkovskiy, and W. Tang, "Predicting disruptive instabilities in controlled fusion plasmas through deep learning," *Nature*, vol. 568, no. 7753, pp. 526–531, Apr. 2019, doi: 10.1038/s41586-019-1116-4.

[9] R. J. Buttery *et al.*, "On the form of NTM onset scalings," *Nucl. Fusion*, vol. 44, p. 678, Apr. 2004.

[10] G. A. Rattá *et al.*, "An advanced disruption predictor for JET tested in a simulated real-time environment," *Nucl. Fusion*, vol. 50, no. 2, Feb. 2010, Art. no. 025005.

[11] J. Vega *et al.*, "Results of the JET real-time disruption predictor in the ITER-like wall campaigns," *Fusion Eng. Des.*, vol. 88, p. 1228, Oct. 2013.

[12] Y. Zhang, G. Pautasso, O. Kardaun, G. Tardini, and X. D. Zhang, "Prediction of disruptions on ASDEX upgrade using discriminant analysis," *Nucl. Fusion*, vol. 51, no. 6, Jun. 2011, Art. no. 063039.

[13] C. Rea *et al.*, "Disruption prediction investigations using machine learning tools on DIII-D and Alcator C-Mod," *Plasma Phys. Controlled Fusion*, vol. 60, no. 8, Aug. 2018, Art. no. 084004.

[14] K. Montes *et al.*, "Machine learning for disruption warnings on Alcator C-Mod, DIII-D, and EAST," *Nucl. Fusion*, vol. 59, no. 9, Jul. 2019, Art. no. 096015. [Online]. Available: https://doi.org/10.1088%2F1741-4326%2Fab1df4

[15] Y. Fu *et al.*, "Machine learning control for disruption and tearing mode avoidance," *Phys. Plasmas*, vol. 27, no. 2, Feb. 2020, Art. no. 022501, doi: 10.1063/1.5125581.

[16] A. Murari, M. Lungaroni, M. Gelfusa, E. Peluso, and J. V. and, "Adaptive learning for disruption prediction in non-stationary conditions," *Nucl. Fusion*, vol. 59, no. 8, Jul. 2019, Art. no. 086037. [Online]. Available: https://doi.org/10.1088%2F1741-4326%2Fab1ecc

[17] A. Murari *et al.*, "On the transfer of adaptive predictors between different devices for both mitigation and prevention of disruptions," *Nucl. Fusion*, vol. 60, no. 5, May 2020, Art. no. 056003.

[18] K. D. Humbird, J. L. Peterson, B. K. Spears, and R. G. McClarren, "Transfer learning to model inertial confinement fusion experiments," *IEEE Trans. Plasma Sci.*, vol. 48, no. 1, pp. 61–70, Jan. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8932676/

[19] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the 'echo state network' approach," German Nat. Res. Center Inf. Technol., Sankt Augustin, Germany, GMD Rep. 159, 2002.

[20] G. Tanaka *et al.*, "Recent advances in physical reservoir computing: A review," *Neural Netw.*, vol. 115, pp. 100–123, Jul. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608019300784

[21] C. Gallicchio and A. Micheli, "Richness of deep echo state network dynamics," in *Advances in Computational Intelligence*, I. Rojas, G. Joya, and A. Catala, Eds. Cham, Switzerland: Springer, 2019, pp. 480–491.

[22] A. Jalalvand, F. Triefenbach, K. Demuynck, and J.-P. Martens, "Robust continuous digit recognition using reservoir computing," *Comput. Speech Lang.*, vol. 30, no. 1, pp. 135–158, Mar. 2015.

[23] M. Xu, P. Baraldi, S. Al-Dahidi, and E. Zio, "Fault prognostics by an ensemble of echo state networks in presence of event based measurements," *Eng. Appl. Artif. Intell.*, vol. 87, Jan. 2020, Art. no. 103346. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197619302854

[24] C. Gallicchio, A. Micheli, and L. Pedrelli, "Design of deep echo state networks," *Neural Netw.*, vol. 108, pp. 33–47, Dec. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608018302223

[25] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach," *Phys. Rev. Lett.*, vol. 120, no. 2, p. 10, Jan. 2018, doi: 10.1103/PhysRevLett.120.024102.

[26] A. Jalalvand, B. Vandersmissen, W. De Neve, and E. Mannens, "Radar signal processing for human identification by means of reservoir computing networks," in *Proc. IEEE Radar Conf. (RadarConf)*, Apr. 2019, pp. 1–6.

[27] A. Jalalvand, F. Triefenbach, and J.-P. Martens, "Continuous digit recognition in noise: Reservoirs can do an excellent job!" in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, 2012, p. 644.

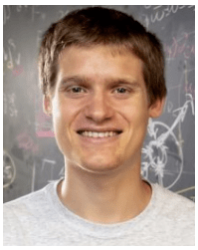
[28] M. Inubushi and S. Goto, "Transfer learning for nonlinear dynamics and its application to fluid turbulence," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 102, no. 4, Oct. 2020, Art. no. 043301, doi: 10.1103/PhysRevE.102.043301.

[29] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks—With an erratum note," German Nat. Res. Center Inf. Technol., Sankt Augustin, Germany, GMD Rep. 148, 2001. [Online]. Available: http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf

[30] A. Jalalvand, K. Demuynck, and J.-P. Martens, "Noise robust continuous digit recognition with reservoir-based acoustic models," in *Proc. Int. Symp. Intell. Signal Process. Commun. Syst.*, 2013, p. 99.
[31] D. Verstraeten, "Reservoir computing: Computation with dynamical systems," Ph.D. dissertation, Dept. Electron. Inf. Syst., Ghent Univ., Ghent, Belgium, 2009.
[32] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Comput.*, vol. 7, no. 1, pp. 108–116, Jan. 1995.
[33] D. W. Marquardt and R. D. Snee, "Ridge regression in practice," *Amer. Statistician*, vol. 29, no. 1, pp. 3–20, Feb. 1975.
[34] R. Penrose, "A generalized inverse for matrices," *Math. Proc. Cambridge Phil. Soc.*, vol. 51, no. 3, pp. 406–413, Jul. 1955.
[35] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surveys*, vol. 46, no. 4, pp. 1–37, Apr. 2014, doi: 10.1145/2523813.
[36] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE Trans. Neural Netw.*, vol. 22, no. 1, pp. 131–144, Jan. 2011.
[37] T. Strauss, W. Wustlich, and R. Labahn, "Design strategies for weight matrices of echo state networks," *Neural Comput.*, vol. 24, no. 12, pp. 3246–3276, Dec. 2012, doi: 10.1162/NECO_a_00374.
[38] A. Griffith, A. Pomerance, and D. J. Gauthier, "Forecasting chaotic systems with very low connectivity reservoir computers," *Chaos, Interdiscipl. J. Nonlinear Sci.*, vol. 29, no. 12, Dec. 2019, Art. no. 123108.
[39] T. L. Carroll and L. M. Pecora, "Network structure effects in reservoir computers," *Chaos, Interdiscipl. J. Nonlinear Sci.*, vol. 29, no. 8, Aug. 2019, Art. no. 083130, doi: 10.1063/1.5097686.
[40] L. Appeltant *et al.*, "Information processing using a single dynamical node as complex system," *Nature Commun.*, vol. 2, no. 1, pp. 1–6, Sep. 2011.
[41] C. Gallicchio, "Sparsity in reservoir computing neural networks," in *Proc. Int. Conf. Innov. Intell. Syst. Appl.*, 2020, pp. 1–7.
[42] A. Jalalvand, W. D. Neve, R. V. de Walle, and J. P. Martens, "Towards using reservoir computing networks for noise-robust image recognition," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2016, pp. 1666–1672.
[43] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 2, pp. 513–529, Apr. 2012.
[44] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade* (Lecture Notes in Computer Science), vol. 7700. Berlin, Germany: Springer, 2012, pp. 659–686.
[45] J. A. Stillerman, T. W. Fredian, K. A. Klare, and G. Manduchi, "MDSplus data acquisition system," *Rev. Sci. Instrum.*, vol. 68, no. 1, pp. 939–942, Jan. 1997.
[46] O. Meneghini *et al.*, "Integrated modeling applications for tokamak experiments with OMFIT," *Nucl. Fusion*, vol. 55, no. 8, Aug. 2015, Art. no. 083008. [Online]. Available: http://iopscience.iop.org/article/10.1088/0029-5515/55/8/083008/meta

Azarakhsh Jalalvand (Member, IEEE) is currently a Senior Data Scientist at Ghent University, Ghent, Belgium. His research focuses on data-driven discovery research tracks including audio, visual, and radar data analysis, as well as multisensor signal processing for a variety of applications, such as object recognition, surveillance, predictive maintenance, and anomaly detection.

Mr. Jalalvand received the three-year Special Postdoctoral Fellowship Award (UGent-BOF) in 2020 to investigate data-driven models for condition monitoring and plasma control in the magnetic confinement devices to produce controlled thermonuclear fusion power.



Joseph Abbate is currently a Graduate Student at Princeton University, Princeton, NJ, USA, working on model-predictive control to help operators achieve desirable plasma states in experimental fusion reactors. He is involved in running campaigns of real-time tests on the algorithm at the DIII-D tokamak in San Diego, CA, USA.

Rory Conlin is currently a Graduate Student at Princeton University, Princeton, NJ, USA, with a background in mechanical engineering, physics, and film studies. He is developing machine-learning algorithms for data-driven control of fusion plasmas to predict and avoid instabilities and achieve new confinement regimes. He has developed new methods to streamline and automate the conversion and deployment of such algorithms for real-time applications. He is also working on new numerical techniques for physics-based predictions of restive instabilities in plasmas, and developing a new stellarator equilibrium code.

Geert Verdoolaege (Member, IEEE) is currently an Associate Professor with the Department of Applied Physics, Ghent University, Ghent, Belgium, where he heads the research unit Nuclear Fusion. His research activities comprise development of data analysis techniques using methods from probability theory, machine learning and information geometry, and their application to nuclear fusion experiments.

Prof. Verdoolaege serves on the Editorial Board of *Entropy* journal and is a member of the scientific committees of several conferences and topical groups on nuclear fusion and information science.

Egemen Kolemen is currently an Assistant Professor at Princeton University, Princeton, NJ, USA, and the Andlinger Center for Energy and the Environment. His research focuses on the application of dynamics and control theory to experimental plasma physics, primarily to address the challenges of fusion reactor design. He analyzes the dynamics of complex plasma phenomena using applied mathematics and control theory with the aim of designing and implementing novel control techniques, which he then uses to build real-time control systems from the ground up.