Elementary Cellular Automata for Reservoir Computing with Modest Resources

Kazuhiro Yokota
Ion Technology Center Co., Ltd
Hirakata, Osaka, 573-0128, Japan
yokota@iontc.co.jp

Several elementary cellular automaton (ECA) rules have been found useful for reservoir computing (RC). In this paper, we investigate how effectively ECA rules operate with reduced computational resources. We examine successful rules with the 5-bit memory task and nonlinear autoregressive moving-average (NARMA) benchmarks. Our model is a slightly modified version of one previously reported that is optimized for modest resources. We find that the features produced by a cellular automaton (CA) at each timestep vary greatly in their contributions to the computation. The performance on the NARMA task is improved, especially in a rule from Wolfram's class 2. The result demonstrates the feasibility of implementing RC with modest resources, such as those in embedded systems, and helps elucidate the role of cellular automata in RC.

Keywords: reservoir computing; elementary cellular automaton

1. Introduction

Artificial intelligence (AI) has recently become deeply involved in our daily lives. The study of neural networks (NNs) was one of the triggers in the development of this field. Actually, convolutional NNs (CNNs) are successful in spatial pattern recognition (e.g., [1]), while recurrent NNs (RNNs) are useful for processing time-series data such as natural languages (e.g., [2]). Their applications are not restricted to the scope of their original concepts (e.g., [3]).

Such deep NNs often require computational resources for learning, as in backpropagation. As AI handles bigger data and more precise responses are expected, larger computational systems are required, bringing about huge energy consumption by data servers.

On the other hand, there is a motivation to implement AI with low computational resources: even if it sacrifices the accuracy of results, an immediate response is regarded as more important. That is to say, using edge AI/on-device AI. In this case, a deep NN may be excessive or does not meet processing speed requirements. Then, another

https://doi.org/10.25088/ComplexSystems.35.2.161

approach should be considered to complete a task with low computational resources, which will coexist with high-computational systems.

Reservoir computing (RC) has attracted attention as a new way to tackle the problem of reducing computational resources. The concept stems from the echo state network (ESN), which is a particular case of RNNs [4]. The liquid state machine (LSM), proposed on the basis of a spiking neural network (NN), is also regarded as having helped form the idea of RC [5]. The schematics of a conventional NN and the simplest case of RC are shown in Figure 1. Both of them have input and output layers. In the conventional NN, all the connections between nodes have weights to be trained. On the other hand, in RC, the reservoir is a sparse network that is composed of fixed nodes, that is, the fixed weights of connections between the nodes, $\mathbf{W}$. As the weights between the input layer and the reservoir $\mathbf{W}_{\text{in}}$ are also fixed, all the parameters to be trained are only in the connections to the output layer, $\mathbf{W}_{\text{out}}$. Therefore, compared with NNs, fewer parameters are trained in RC, which can be achieved with low computational cost. Given the input vector of $\mathbf{u}$, the output vector $\mathbf{y}$ is formally expressed as

$$\begin{aligned} \mathbf{x}(t) &= f(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t-1)) \\ \mathbf{y}(t) &= \mathbf{W}_{\text{out}}\mathbf{x}(t), \end{aligned} \tag{1}$$

where $\mathbf{x}$ and f represent the state vector of the reservoir and a nonlinear function, respectively, and the time t is supposed to be discrete. As shown, the reservoir transforms an input vector nonlinearly, which is mapped into a higher-dimensional space to be linearly separable. RC is suitable for processing time-series data due to the echo state property (ESP): Irrespective of the initial state of the reservoir, the state after time has passed is significantly determined by the near-past input data, and the output finally converges. Therefore, the reservoir should have appropriate memory capacity.

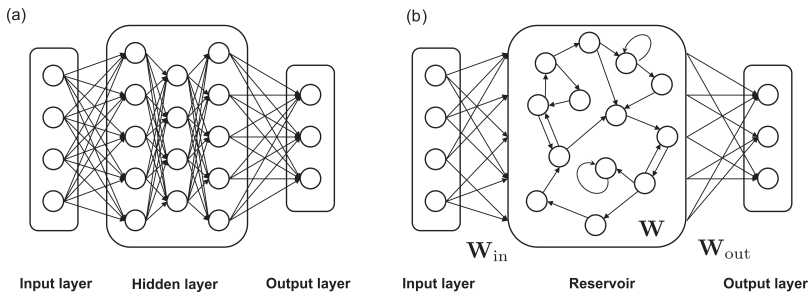


Figure 1. Schematics of (a) a conventional neural network and (b) reservoir computing.

Many proposals have been reported that implement RC with the aid of dynamics in various physical systems, which is often called physical reservoir computing (for a review, see e.g., [6]). The cellular automaton (CA) has been studied as one of the simplest systems for RC.

The first work on RC using a CA was reported in [7, 8], in which the memory task was used as the benchmark. While the proposal handled the entire input and output data at once, methods of sequential processing were also proposed, as in [9–11]: Given an input at a time, the output is predicted at every time. While the 5-bit memory task has often been used as a benchmark, the task was critically examined recently in [12]. Another benchmark, such as nonlinear autoregressive moving average (NARMA), has also been used, as in [9]. In [13], by refining the process through various techniques, performance on several regression tasks is demonstrated. It was also shown that a CA provided a viable solution in image classification [14–17]. As explained in Section 2, according to a rule, the state of a CA evolves in discrete timesteps and behaves in a characteristic manner. In the cited papers, several rules were found to be successful for RC, which were often clarified by exhaustive searches.

In this paper, we examine the successful rules in detail and consider their operational behaviors. It is shown that the states of the CA at each timestep produce the features, which vary greatly in their contributions to the computation. This suggests the possibility of reducing computational resources by selecting the features appropriately. For the purpose of practical implementation with modest resources, the model we use is a slightly modified version of the one previously reported in [10], which has a simple structure for processing binary data. Compared with previous reports, our modification improves the performance on the NARMA task, where the model is tailored for 8-bit input data. Our result shows that RC using a CA is compatible with a tiny system such as an embedded system.

2. Elementary Cellular Automata

A CA is composed of cells arranged in a grid. According to a rule, the state of each cell undergoes a discrete time evolution. The simplest CA is known as an elementary CA (ECA), which is given by cells arranged in one dimension, and each cell takes on a binary state [18]. The state of a cell in the next timestep is determined by the current state of the cell itself and the states of its closest neighboring cells, namely, the right cell and the left cell. The edges of the ECA are treated as cyclic in this paper: The far-right (far-left) cell is considered

https://doi.org/10.25088/ComplexSystems.35.2.161

to be to the left (right) of the far-left (far-right) cell. As shown in Figure 2, the number of possible states for the three cells is eight, according to which the state of the center cell is fixed in the next timestep. Then, $2^8 = 256$ patterns can be considered for the state of the center cell, which are numbered by interpreting the states as binary numbers: In the case of Figure 2, as the bit string of 01 011 010 is converted to its decimal value of 90, this rule is called rule 90.

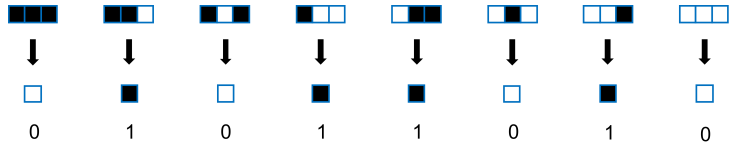
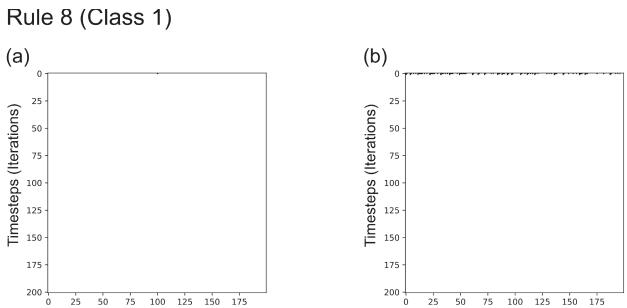


Figure 2. Rule 90. A binary state is represented by black and white cells that correspond to one and zero, respectively.

Depending on the rule, the ECA exhibits various behaviors. Several examples are shown in Figure 3, which contains all the rules discussed in this paper. Considering mirror and complement transformation, only 88 out of 256 rules are not equivalent. (For example, the list can be seen in [12].) While quantitatively evaluating the behaviors has been proposed (e.g., [19]), the rules are often classified into four classes [18]: The ECA settles into a uniform state in class 1, and a rule in class 2 makes the ECA converge to a periodic state. Random patterns are found in class 3. In class 4, the ECA exhibits a behavior that is a mixture of randomness and order, in other words, "edge of chaos." These properties have been studied from the viewpoint of computational resources. For example, rule 110 in class 4 is capable of universal computation [20].



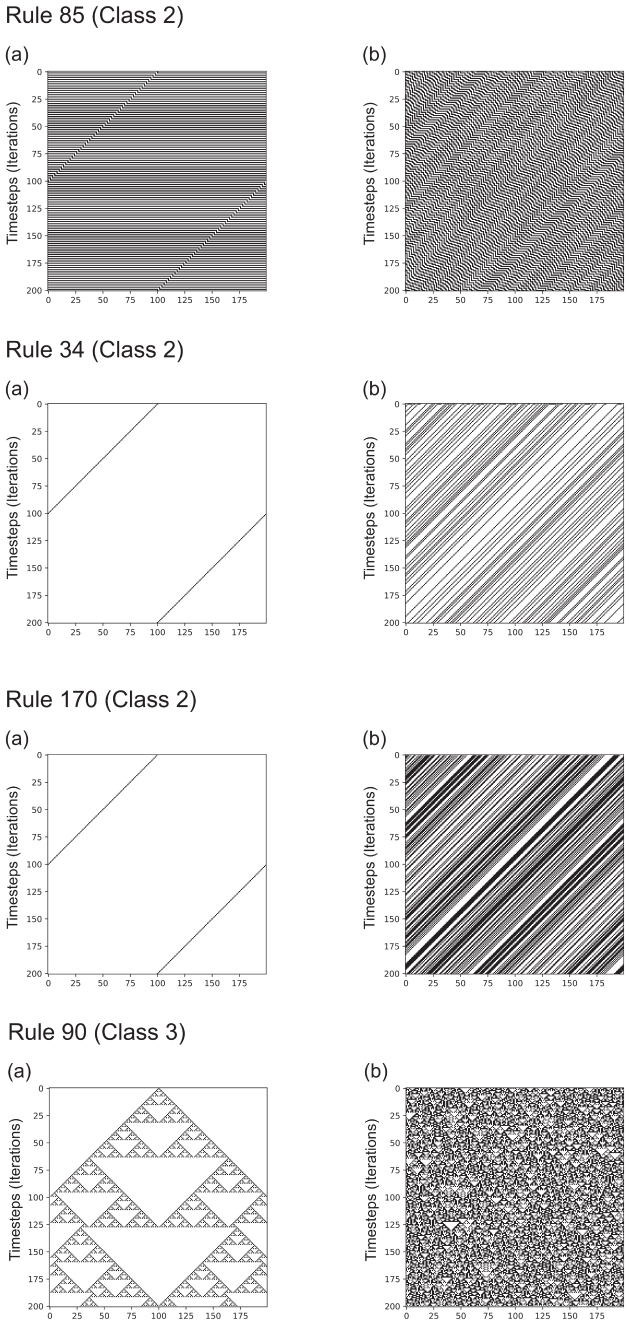


Figure 3. (*continues*).

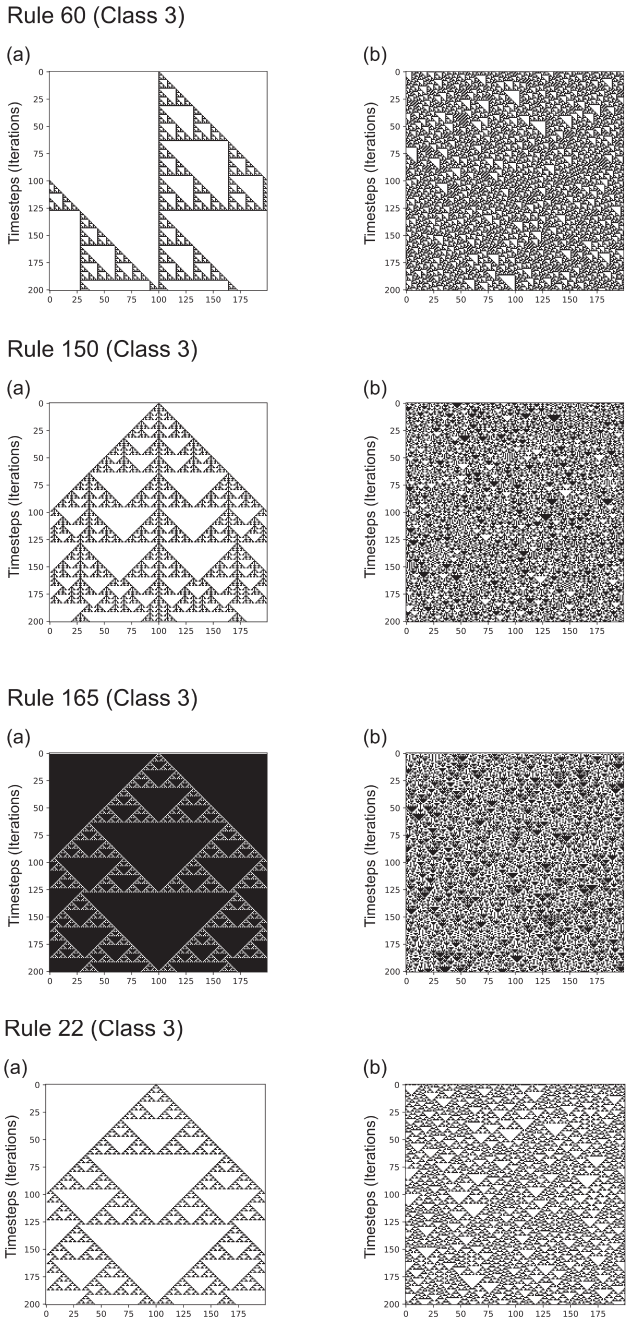


Figure 3. (*continues*).

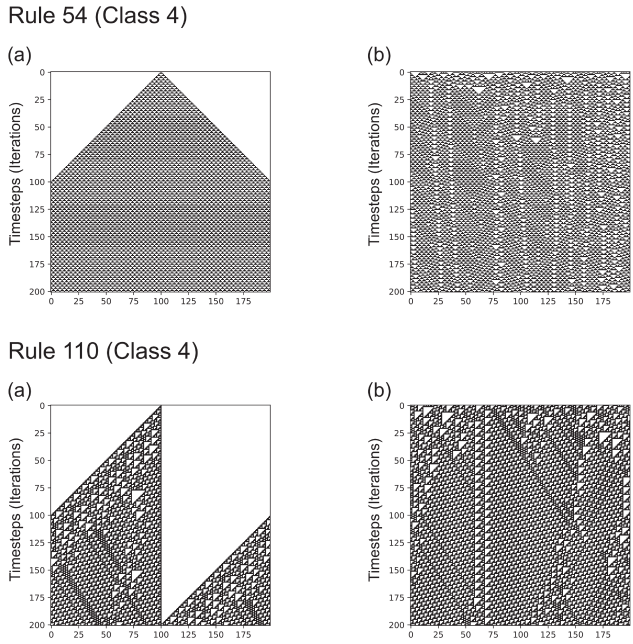


Figure 3. Time evolutions of the ECA by various rules. (a) Only the cell at the center is set to one initially. (b) Several cells are set to one initially.

3. Reservoir Computing Using Elementary Cellular Automata

In RC, making use of the ECA, several architectures have been proposed, as explained in Section 1. Referring to the model previously reported in [10] and making a minor change, we constructed a recurrent reservoir architecture as shown in Figure 4: A binary vector is input to the reservoir in sequence, and the output comes out in response to it.

The details are as follows: First, "padding" is added to an initial input vector (cell), whose ratio is given by C, that is, the size of input is increased by C times. Then, we randomly permute the cells with the number of permutation patterns given by R. The same patterns are used throughout the processing. The permuted vectors are concatenated into one vector, which corresponds to the initial state of the ECA. According to the rule, the ECA evolves in discrete timesteps, which are iterated I times. While the states of the ECA in all the timesteps are sent to the output layer (classifier/regressor) conventionally, it is also possible to select some of them.

https://doi.org/10.25088/ComplexSystems.35.2.161

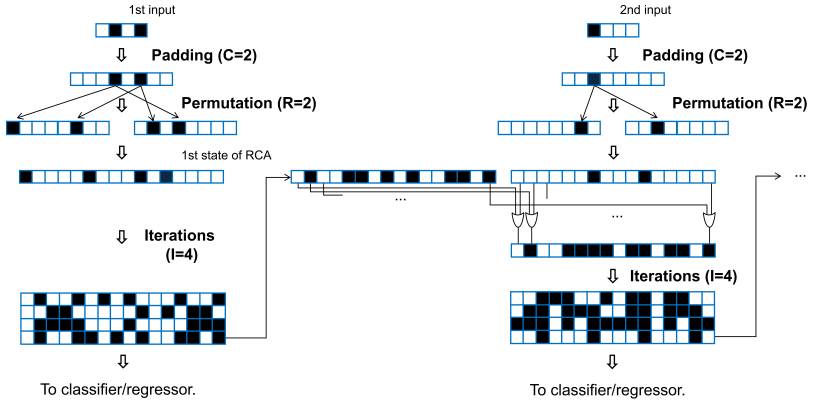


Figure 4. Schematics of RC using an ECA. This figure shows the case of $C = 2$, $R = 2$ and $I = 4$ with rule 90.

When the next input comes, padding and permutations are performed in the same way. The pre-processed input vector is merged into the ECA, namely, the state after I timesteps for the previous input. How to merge the input vector into the ECA is optional: we can use XOR, permutation transition or others [9–11]. We adopt OR, because it improves performance on the NARMA task, as seen in Section 5. After the merging, the ECA evolves for I timesteps. The process is repeated until all the inputs are provided in sequence.

We used Python to write the code for our experiment and scikit-learn [21] for learning and testing.

4. The 5-bit Memory Task

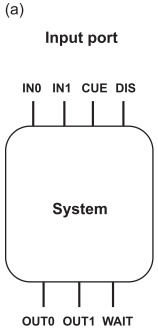
4.1 Methodology

First, we evaluate the model by the 5-bit memory task benchmark, which is often applied for estimating memory capacity [22]. As we will discuss, there is a critical view on this task to evaluate the CA-based reservoir [12]. However, it is worthwhile to verify whether our model shows similar results to those previously reported. Considering how it performs will also provide insights for a deeper understanding.

In this task, the system has the input port and the output port with four pins and three pins, respectively, as shown in Figure 5(a), where each pin takes a binary value. Additionally, exactly one pin is at logic one, meaning that both ports are always in a one-hot state. The aim of this task is to memorize temporal 2-bit patterns given via IN0 and IN1 and to reproduce the patterns via OUT0 and OUT1 later. The details are as follows: At the first five timesteps, 2-bit data is given via (IN0, IN1). As we have mentioned, their possible bit patterns are (0, 1) or (1, 0), during which the other pins CUE and DIS are zero.

Consequently, all the possible temporal bit patterns to be memorized are $2^5 = 32$ patterns corresponding to 5-bit. Then, only the DIS pin is set to one during the period of T, which gives a meaningless bit pattern to disturb the state (memory) of the system. After the distractor period, setting CUE to one gives a cue for the system to reproduce the 2-bit patterns in the first five timesteps, which should be output through OUT0 and OUT1 in the last five timesteps. Note that in the output port, only the WAIT pin should be set to one before the CUE. The series of timesteps is shown in Figure 5(b).

In this task, given the distractor period, all of the 32 patterns are learned, and the training scores themselves are used for comparison. A trial is regarded as successful on each pattern when the output port generates the correct bit stream at all the timesteps. In our experiment, the distractor period was set as $T = 200$.



(b)

Time step	Input port				Output port			
	IN0	IN1	CUE	DIS	OUT0	OUT1	WAIT	
1	0	1	0	0	0	0	1	Input a bit-pattern to be memorized.
2	1	0	0	0	0	0	1	
3	1	0	0	0	0	0	1	
4	0	1	0	0	0	0	1	
5	0	1	0	0	0	0	1	
6	0	0	0	1	0	0	1	Distractor period.
7	0	0	0	1	0	0	1	
...	0	0	0	1	0	0	1	
204	0	0	0	1	0	0	1	
205	0	0	0	1	0	0	1	
206	0	0	1	0	0	0	1	← Cue.
207	0	0	0	1	0	1	0	Reproduce the bit-pattern.
208	0	0	0	1	1	0	0	
209	0	0	0	1	1	0	0	
210	0	0	0	1	0	1	0	
211	0	0	0	1	0	1	0	

Figure 5. (a) Schematic of the 5-bit memory task. (b) Sequence of timesteps in the 5-bit memory task.

4.2 Result and Discussion

We focused our investigation mainly on the successful rules in previous studies [7–11]. As mentioned in Section 2, only 88 rules are not equivalent, and we checked at least one rule out of the 88 except for class 1. To improve readability, the results of the representative rules are shown here. We examine the case in which training was performed using only the state of the ECA after the I^{th} iteration for each input. Depending on the rules, several characteristics were found in the way the performance changed as I changed. Figure 6 presents the rates of failed trials in the 5-bit memory task on the representative rules. The choices for C and R give appropriate examples to show the characteristics on I, while we examine several cases of the parameters. In Figure 7, we also show the result when all the states of the ECA

after each iteration were used for training, as is often done. We used scikit-learn for training a logistic regression classifier; The strength of regularization was reduced (the inverse of regularization strength was 100) because the training scores were compared.

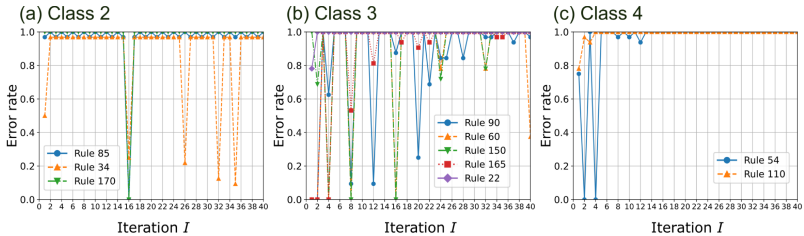


Figure 6. Error rates of the 5-bit memory task. Training was performed using only the state of the ECA after the I^{th} iteration ($C = 2$, $R = 16$).

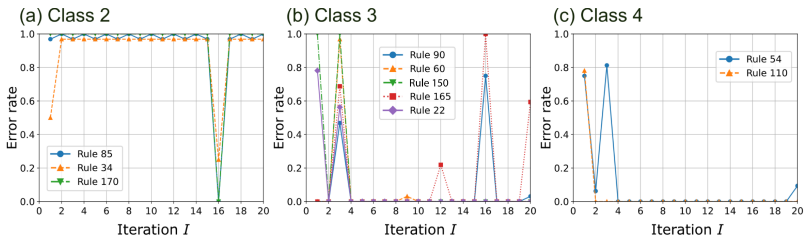


Figure 7. Error rates of the 5-bit memory task. Training was performed using all the states of the ECA after each iteration ($C = 2$, $R = 16$).

Our result was consistent with previous studies despite differences between the models, as rules in class 3 seemed to show high performances as in [7]. We review our result in detail for each class as follows.

In class 3, particular iterations have good performance in rules 90, 60, 150 and 165, as shown in Figure 6(b). Actually, it was pointed out that for rule 90, the classification error was reduced at $I = 2^m$ ($m = 0, 1, 2\ldots$) in the demonstration of pattern recognition of handwritten numbers (the MNIST database) [16]. This regularity comes from the self-similar structure, as seen in Figure 3(a) for rule 90: The iterations of $I = 2^m$ correspond to the vertices of the triangles, in which the ECA becomes sparse. Our result suggests a possibility of selecting and discarding features, namely, the states after some iterations without performance degradation, which must also contribute to avoiding overfitting in general uses. Relatively good scores can be seen in iterations other than $I = 2^m$, for example, $I = 12$ and $I = 20$, in which the state of the cells is sparse. Although such iterations may also provide useful features, it will be a reasonable strategy to pay

attention to only $I = 2^m$, as discussed in Section 5. Due to the contribution of the beneficial features provided by $I = 2^m$, these rules also tend to show high scores even in the case where all the iterations are involved, as shown in Figure 7(b).

On the other hand, rule 22 in class 3 did not show a good score in Figure 6(b), although it has a self-similar structure, as seen in Figure 3(a). The difference between rules 90 and 22 can be easily visualized, as in Figure 8, by comparing time evolutions as follows. Several cells are initially set to one at random, and the timesteps are iterated I times. At every I^{th} iteration, a constant input vector (such as distractor) is merged into the ECA, which differs from the initial state of the ECA. Only the states at every I^{th} iteration are extracted, the results of which are presented in Figure 8. It is verified that when $I = 4$, 8 ($I = 2^m$), the state of the ECA does not change much in rule 90. At the iterations other than $I = 2^m$, the ECA would have poorly

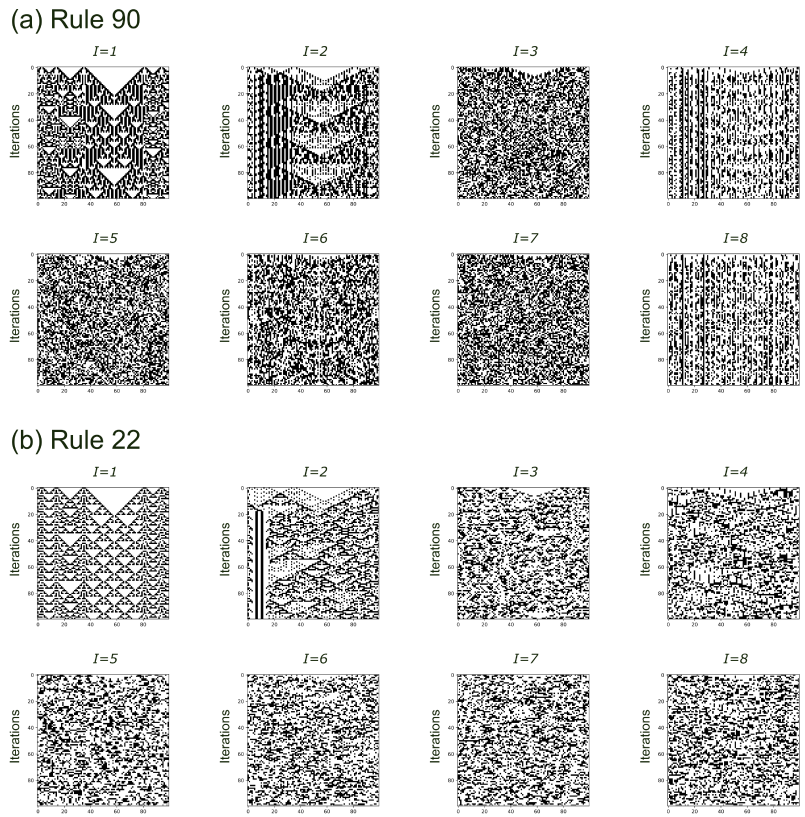


Figure 8. Extracted states of the ECA after every I^{th} iteration in (a) rule 90 and (b) rule 22.

separable representation. On the other hand, in rule 22, random patterns are always found regardless of I. As a result, the spread of disturbance during the distractor period is suppressed at the particular iteration $I = 2^m$ in rule 90.

However, rule 22 had a good performance when all the iterations were involved in training, as in Figure 7(b). In addition, class 4 rules 54 and 110 also showed such a characteristic, as seen in Figures 6(c) and 7(c). These results indicate that useful features are embedded in the trajectory. Actually, for rules 54 and 110, such structures can be clearly seen in Figure 3(b). This is consistent with the previous report [12], in which rule 54 is discussed.

On shift-based rules in class 2 as seen in Figure 3, several iterations showed good scores, as seen in Figure 6(a). However, this cannot be related to any underlying regularity such as the self-similar structure in rule 90. In Figure 9, we show the results of rules 85 and 90 when changing C, namely, the size of the ECA. While the iterations with high scores are almost the same as $I = 2^m$ in rule 90, in rule 85 iterations with high scores depend on C. Therefore, it is not practical to select the successful iterations in the shift-based rules unless there is at least some knowledge of the input data. To make matters worse, it seems useless to accumulate iterations, as shown in Figure 7(a).

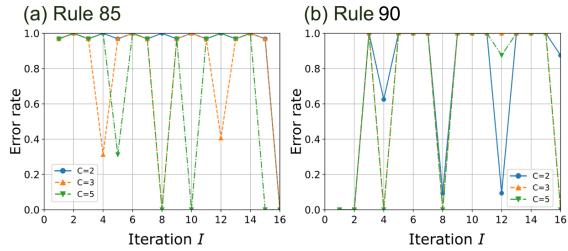


Figure 9. Error rates of the 5-bit memory task in changing C on (a) rule 85 and (b) rule 90. Only the I^{th} iterations were involved in training, as in Figure 6 ($R = 16$).

In Figure 9(a), it can be also found that in rule 85, as C becomes larger, the number of successful iterations increases. The findings in Figure 9 would come from the characteristic of the 5-bit memory task: The vectors input to the ECA are relatively sparse, and the inputs during the distractor period are constant. Therefore, as the size of the ECA becomes larger, the possibility for the ECA to go through the disturbance by distractors would increase in shift-based rules. In this sense, it would be disputable to apply the 5-bit memory task to an ECA for the purpose of evaluating the memory capacity, as pointed out in [12].

Regardless of this discussion, using shift-based rules would not be suitable for tasks that need relatively long-term memory, as the memory capacity directly depends on the size of the ECA [13]. Nevertheless, as shown in Section 5, a reasonable strategy is to increase the size of the ECA for the task with relatively short-term memory such as NARMA, which requires at most a few dozen timesteps of memory.

5. Nonlinear Autoregressive Moving Average

5.1 Methodology

NARMA has been often used as a benchmark for evaluating nonlinearity and memory capacity. For brief review, see, for example, [23]. Given a value $u(t)$, the goal of this task is to imitate the value of $y(t+1)$, which is derived from the following:

$$y(t+1) = \alpha y(t) + \beta y(t)\left(\sum_{i=0}^{N-1} y(t-i)\right) + \gamma u(t-N+1)u(t) + \delta, \tag{2}$$

where t represents discrete time, and α, β, γ and δ are parameters of real numbers. The larger N becomes, the harder it is to complete the task, because the values of $u(t-N+1)$ and $y(t-i)$ in the previous $i \le N-1$ timesteps are needed to infer $y(t+1)$.

We investigated the cases of $N=2$ (NARMA2) and $N=10$ (NARMA10). The parameters $\alpha=0.3$, $\beta=0.05$, $\gamma=1.5$ and $\delta=0.1$ were used for both cases. A value for $u(t)$ was chosen from uniformly distributed random numbers in the range from 0 to 0.5. Without choosing these parameters appropriately, $y(t)$ often diverges.

In our experiment, $u(t)$ is encoded in 8-bit before being input to the ECA: For example, $u(t) = 0.2305 \sim 0.23046875 = 0.00111011_2$. (Equivalently, multiply $u(t)$ by 2^8, round the value to its nearest integer and then encode it in 8-bit to obtain the same bit string.) Note that this bit string corresponds to an input vector in equation (1), namely, an eight-dimensional binary vector. From the viewpoint of actual implementation, the 8-bit encoding is reasonable and consistent with the challenge of reducing computational resources. Actually, 8-bit microcontrollers (MCUs) are currently working in tiny systems, while many major MCUs now have a 32-bit data bus.

A dataset containing 50 000 data points was prepared, corresponding to the $(u(t), y(t+1))$ values for $1 \le t \le 50\,000$ in equation (2). 75% of the data points were used for training, with the remaining used for testing. To be precise, the values for $1 \le t \le 37\,500$ were used for training, and the values for $37\,501 \le t \le 50\,000$ were used for testing. We used scikit-learn to train and test a linear regression model.

https://doi.org/10.25088/ComplexSystems.35.2.161

We evaluated the performance of the model with normalized root mean squared error (NRMSE) given by

$$\mathrm{NRMSE} = \frac{\sqrt{\sum_{t=1}^{T} \|y(t) - \hat{y}(t)\|^2}}{\sqrt{\sum_{t=1}^{T} \left\|y(t) - \overline{y(t)}\right\|^2}}, \tag{3}$$

where $\hat{y}(t)$ and $y(t)$ are the predicted and target values, respectively; the number of the data points is represented by T; and $\overline{y(t)}$ is the time average of $y(t)$, namely, $\overline{y(t)} = \sum_{t=1}^{T} y(t)/T$.

5.2 Result and Discussion

As in the 5-bit memory task, we examined the case in which only the state of the ECA after the I^{th} iteration was used for each input vector. On the representative rules, the test scores of NARMA2 are shown in Figure 10. Figure 11 shows the scores when all the iterations were used in training and testing.

Some of the class 3 rules exhibited a tendency that the iterations of $I = 2^m$ had high performances in Figure 10(b). This tendency was particularly evident in rule 90. In Figure 11(b), the result of rule 90 (2) represents the case in which only the iterations of $I = 2^m$ were used for regression. For example, when $I = 8$, the states after the first, second, fourth and eighth iterations were used. Clearly the performance was almost the same as the one in which all the iterations were involved, and it seemed that the states after the iterations other than $I = 2^m$ had little contribution to the performance. Rather, this result may suggest that the state of the ECA evolves in a nonlinear way at the iterations between $I = 2^m$ and $I = 2^{m+1}$ that are not a power of two, in which case the ECA loses the separable representation as seen in Figure 8(a). In any case, parameters to be trained can be reduced in rule 90, as suggested in Section 4, which is also effective to prevent overfitting.

Only the first few iterations of class 4 rules seem relatively useful, as shown in Figure 10(c). Although there was a slight performance improvement by using all the iterations in Figure 11(c), only the first few iterations were also helpful.

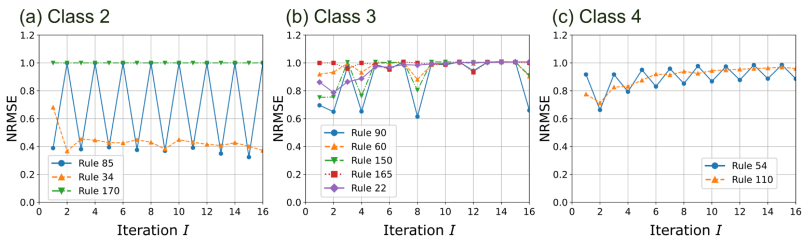


Figure 10. The test scores of NARMA2. Only the I^{th} iteration was used for training and testing ($C = 2$, $R = 32$).

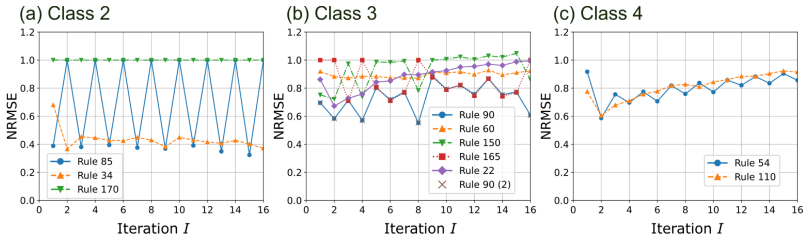


Figure 11. The test scores of NARMA2. All the iterations were used in training and testing ($C = 2$, $R = 32$).

The rule that showed the best performance in our experiment is from class 2, as shown in Figure 10(a). Although rule 170, which causes a simple shift operation, was entirely unsuccessful, rules 85 and 34 achieved low NRMSEs. In these successful rules, a unique operation is followed by a shift (see Figure 3): For example, rule 85 flips the bits before shifting them. Actually, when I was even, rule 85 did not give a good score, because it shifted the bits in the same way rule 170 did. As shown in Figure 11(a), the performances when all the iterations were used were the same as in Figure 10(a). We were also not able to confirm the performance improvement for rule 85 by selecting only the odd iterations. It indicates that a shift is needed for memorization as in [13], but the amount of the shift is not so essential to obtain useful features in this task. Particularly, in rule 85, only the first iteration seems enough for this task, by which computational resources can be drastically reduced.

Thus, we suggest that in addition to time evolution by the rule, the way an input vector is merged into the ECA plays a significant role in the nonlinearity. Actually, we have adopted OR for the merging process: Although we also examined the case of XOR, we did not see a high performance, especially in rule 85. It seems that the best choice for the merging process depends on both the rule and the type of task. If we were to explain the performance in rule 85, it would be as follows. When an input is merged into the ECA, the bits in the ECA immediately after the final iteration for the previous input have been flipped and shifted. Then, the bit values of the new input are reflected only in the cells of the ECA whose bit values are zero due to OR. The bit values in these cells were one at the time of the previous input. In this way, the rule with OR implements the interactions among the input vectors in time series.

The NARMA10 Test scores are shown in Figure 12, where only the state of the ECA after the I^{th} iteration was taken. We used the same number of data points for training and testing as in the NARMA2 task. Compared with the result of NARMA2 in Figure 10, the performances decreased in all the rules as the task became more difficult. However, their qualitative trends were the same.

https://doi.org/10.25088/ComplexSystems.35.2.161

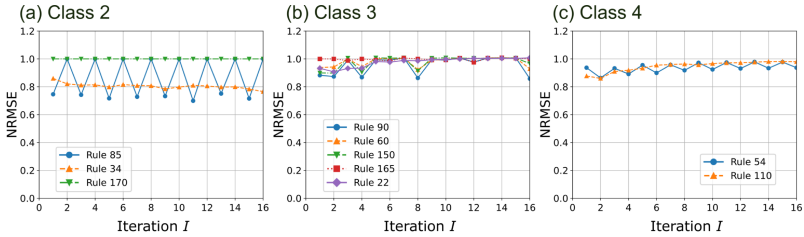


Figure 12. The test scores of NARMA10. Only the I^{th} iteration was used for training and testing ($C = 2$, $R = 32$).

We examined whether the performance of rule 90 could be improved by changing C and R. To examine as many iterations as possible without overfitting, the number of data points was increased to 200 000 only in this experiment; 75% of the data points were used for training, with the remaining used for testing. As mentioned earlier, the reasonable strategy for rule 90 is to select only the iterations of $I = 2^m$, with results as shown in Figure 13. It seems that the increase in R was more effective than the increase in C. In any case, involving a few iterations improved the performance. However, as the number of iterations increases too much, the performance appears to have slightly dropped, for example, at $I = 32$ in Figure 13(b) $R = 64$. Considering the number of data points for training, this degradation came from overfitting. Actually, due to this reason, we could not examine the case of $R = 64$ in Figure 13(c). Even if some improvement occurs by increasing the number of iterations, it is likely to be negligible; For practical implementation, this rule is not suitable for further enhancement of performance in this task. Here we would like to remark that the size of the ECA should be reconsidered in order to verify the results in more iterations: In rule 90, given the size of the ECA, $L = 2^m$ ($m \geq 2$), all the cells become zero when $I = L/2$.

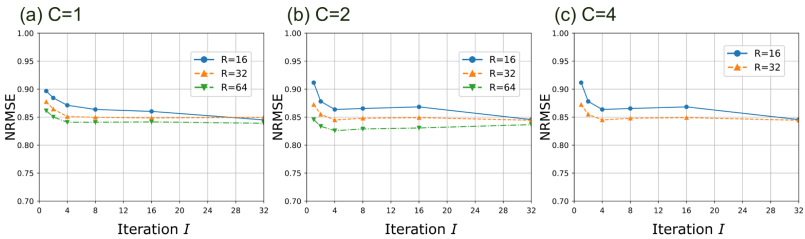


Figure 13. The test scores of NARMA10 in changing C and R for rule 90. All the iterations of $I = 2^m$ were used in training and testing.

Rule 85 was also investigated in detail. As explained previously, only the first iteration is enough to be examined. When $I = 1$, the

results in changing C and R are shown in Figure 14(a). By increasing the size of the ECA, that is, increasing C, high scores were obtained, which indicated that the memory capacity was insufficient in the case of $C = 2$ and $R = 32$ in Figure 12(a). As noted at the end of Section 4, this task does not need relatively long-term memory, unlike the 5-bit memory task, and the improvement in performance by increasing C is plausible. We also show an example of the data points in Figure 14(b). Clearly, it can be confirmed that the predicted values actually follow the target ones. In this situation, we also examined the statistical robustness across seeds for the permutation R. With 10 different seeds, the best, worst and average NRMSE scores were 0.44, 0.48 and 0.46, respectively.

(a)

	C=2	C=4	C=8	C=16
R=8	0.80	0.69	0.63	0.47
R=16	0.77	0.62	0.53	0.46
R=32	0.75	0.52	0.44	0.40
R=64	0.70	0.46	0.43	0.44

(b)

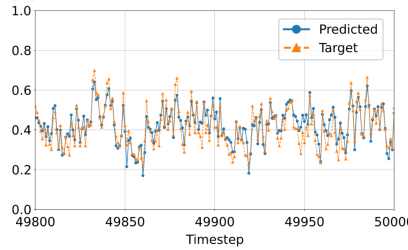


Figure 14. (a) The result of NARMA10 for rule 85 ($I = 1$). The values represent NRMSEs. (b) Target values and predicted values in using rule 85 with $C = 8$, $R = 32$ and $I = 1$.

Here, we would like to make a comparison with a previous study [9]. Their results of various rules (including rule 85) were reported using NARMA30 as a benchmark, and the NRMSE best score was 0.74. Although we examined NARMA2 and NARMA10, we can conclude our model improved the performance. Actually, we also evaluated the case of NARMA30 on rule 85 ($C = 8$, $R = 32$, $I = 1$) with the same parameters in equation (2) as in [9], and the NRMSE test score was 0.55. As mentioned, our choice of OR for the merging process effectively improved the performance. In addition, the 8-bit quantization of input data was also a successful process. In [9], the input was basically in 64-bit format (float64), although its bit width was reduced by two bits due to the restricted range of values. Consequently, an ECA enables RC with modest resources, such as those in embedded systems.

We would like to consider the computational resources quantitatively in the case of Figure 14(b). As the input is given by 8-bit data, the ECA is composed of 2048 cells ($8 \times C \times R$). Rule 85 can be implemented by NOT and bit-shift operations. After only the first iteration,

the number of features sent to a linear regressor is also 2048 ($2048 \times I$). Note that because each feature is binary, the total amount of information is 2048 bits. After training is completed, the computation (inference) is effectively done with sum only, in other words, without multiplication: There is a weight associated with each binary feature, and all the regressor has to do is select the weights to be added and calculate their sum in accordance with the binary features. If the coefficients (weights) and the intercept of the linear function are in 16-bit format, all the parameters can be stored in a memory of $16 \times 2049 = 32\,784$ bits, that is, about 4KB. As a result, all the processes—pre-processing input data, time evolutions of an ECA and linear regression—can be easily implemented with low computational resources. For example, even an entry model field-programmable gate array (FPGA) has a memory of a few hundred KB. The linear regression can be achieved by adders, without using complex logic gate combinations or a soft/hard processor core. When the regressor is implemented with sequential circuits, the calculation per 8-bit input data can be completed within at most a few clock cycles, which is typically in the MHz range in an embedded system. Then inference latency is on the order of microseconds.

6. Summary

We examined several elementary cellular automaton (ECA) rules for reservoir computing (RC) with the 5-bit memory task and the nonlinear autoregressive moving-average (NARMA) task. We found that particular iterations provide more beneficial features, which implies the possibility of reducing computational resources. On the 5-bit memory task, successful rules such as rule 90 were discussed in relation to the self-similar structure. The sparse representation at the particular iteration prevents the spread of a disturbance during the distractor period.

On the NARMA task, high performance was obtained in the shift-based rules such as rule 85, which have an advantage in tasks that require relatively short-term memory. The memory capacity can be easily increased by changing the size of the ECA. It was found that the way an input participated in the ECA affected the performance significantly. We also showed the feasibility of implementing RC with modest resources quantitatively.

In future work, we will try to implement RC on a field-programmable gate array (FPGA) and apply it to physical phenomena. We believe such an attempt will clarify the scope of application, and the limit gives us a deeper understanding of cellular automata themselves.

References

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, **60**(6), 2017 pp. 84–90. doi:10.1145/3065386.

[2] A. Graves, A.-r. Mohamed and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, Canada, New York: IEEE, 2013 pp. 6645–6649. doi:10.1109/ICASSP.2013.6638947.

[3] T. N. Sainath, B. Kingsbury, A.-r. Mohamed, G. E. Dahl, G. Saon, H. Soltau, T. Beran, A. Y. Aravkin and B. Ramabhadran, "Improvements to Deep Convolutional Neural Networks for LVCSR." arxiv.org/abs/1309.1501.

[4] H. Jaeger, "The 'Echo State' Approach to Analysing and Training Recurrent Neural Networks—with an Erratum Note," *German National Research Center for Information Technology GMD Technical Report*, **148**(34), 2001 p. 13.

[5] W. Maass, T. Natschläger and H. Markram, "Real-Time Computing without Stable States: A New Framework for Neural Computation Based on Perturbations," *Neural Computation*, **14**(11), 2002 pp. 2531–2560. doi:10.1162/089976602760407955.

[6] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano and A. Hirose, "Recent Advances in Physical Reservoir Computing: A Review," *Neural Networks*, **115**, 2019 pp. 100–123. doi:10.1016/j.neunet.2019.03.005.

[7] O. Yilmaz, "Reservoir Computing Using Cellular Automata." arxiv.org/abs/1410.0162.

[8] O. Yilmaz, "Connectionist-Symbolic Machine Intelligence Using Cellular Automata Based Reservoir-Hyperdimensional Computing." arxiv.org/abs/1503.00851.

[9] E. T. Bye, "Investigation of Elementary Cellular Automata for Reservoir Computing," Master's thesis, Norwegian University of Science and Technology, June, 2016. hdl.handle.net/11250/2415318.

[10] S. Nichele and M. S. Gundersen, "Reservoir Computing Using Nonuniform Binary Cellular Automata," *Complex Systems*, **26**(3), 2017 pp. 225–245. doi:10.25088/ComplexSystems.26.3.225.

[11] S. Nichele and A. Molund, "Deep Learning with Cellular Automaton-Based Reservoir Computing," *Complex Systems*, **26**(4), 2017 pp. 319–339. doi:10.25088/ComplexSystems.26.4.319.

[12] T. E. Glover, P. Lind, A. Yazidi, E. Osipov and S. Nichele, "Investigating Rules and Parameters of Reservoir Computing with Elementary Cellular Automata, with a Criticism of Rule 90 and the Five-Bit Memory Benchmark," *Complex Systems*, **32**(3), 2023 pp. 309–351. doi:10.25088/ComplexSystems.32.3.309.

[13] N. McDonald, "Reservoir Computing & Extreme Learning Machines Using Pairs of Cellular Automata Rules," in *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, New York: IEEE, 2017 pp. 2429–2436. doi:10.1109/IJCNN.2017.7966151.

[14] A. Morán, C. F. Frasser and J. L. Rosselló, "Reservoir Computing Hardware with Cellular Automata." arxiv.org/abs/1806.04932.

[15] D. Kleyko, S. Khan, E. Osipov and S.-P. Yong, "Modality Classification of Medical Images with Distributed Representations Based on Cellular Automata Reservoir Computing," in *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, Melbourne, VIC, Australia, New York: IEEE, 2017 pp. 1053–1056. doi:10.1109/ISBI.2017.7950697.

[16] Á. L. García-Arias, J. Yu and M. Hashimoto, "Low-Cost Reservoir Computing Using Cellular Automata and Random Forests," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Seville, Spain, New York: IEEE, 2020 pp. 1–5. doi:10.1109/ISCAS45731.2020.9180742.

[17] D. Liang, J. Shiomi, N. Miura, M. Hashimoto and H. Awano, "A Hardware Efficient Reservoir Computing System Using Cellular Automata and Ensemble Bloom Filter," *IEICE Transactions on Information and Systems*, **E105.D**(7), 2022 pp. 1273–1282. doi:10.1587/transinf.2021EDP7203.

[18] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[19] C. G. Langton, "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation," *Physica D: Nonlinear Phenomena*, **42**(1), 1990 pp. 12–37. doi:10.1016/0167-2789(90)90064-V.

[20] M. Cook, "Universality in Elementary Cellular Automata," *Complex Systems*, **15**(1), 2004 pp. 1–40. doi:10.25088/ComplexSystems.15.1.1.

[21] F. Pedregosa, A. G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel et al., " "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, **12**(85), 2011 pp. 2825–2830. jmlr.org/papers/v12/pedregosa11a.html.

[22] H. Jaeger, "Long Short-Term Memory in Echo State Networks: Details of a Simulation Study," *Constructor University Technical Reports*, Jacobs University Bremen, 2012. opus.constructor.university/frontdoor/index/index/docId/638.

[23] C. Wringe, M. Trefzer and S. Stepney, "Reservoir Computing Benchmarks: A Tutorial Review and Critique," *International Journal of Parallel, Emergent and Distributed Systems*, **40**(4), 2025 pp. 313–351. doi:10.1080/17445760.2025.2472211.