Evaluating the Impact of Leak Rate in Parallel Reservoir Echo State Networks

Oswald Dai
Angel Navarro
Ujwala Nettam
Katie Nguyen
Octavio Pescador

University of California, Santa Cruz

June 2025

Code Repository:
Evaluating the Impact of Leak Rate in Parallel Reservoir Echo State Networks

Contents

1 Abstract **3**

2 Introduction **3**

3 Model and Methods **3**
3.1 Data Generation 3
3.1.1 Rössler System 3
3.1.2 Kuramoto-Sivashinsky System 5
3.2 Framework and Methodology 6
3.2.1 Input Partitioning and Training 6
3.2.2 Output Weight Regression 7

4 Evaluation Metrics **8**
4.1 Decorrelation Timescale 8
4.2 NRMSE and Absolute Value Sum Problem 9
4.3 Prediction Analysis 10
4.3.1 Lorenz 11
4.3.2 Rössler 13
4.3.3 Kuramoto-Sivashinsky 15

5 Caveats and Discussion **16**

6 Conclusion **17**

1 Abstract

Abstract

Echo State Networks (ESNs) are a class of recurrent neural networks that is particularly effective for predicting chaotic time series. In this work, we investigate the effect of the leak rate in Parallel Reservoir ESNs, we evaluate performance on three benchmark chaotic systems: Lorenz, Rössler, and Kuramoto–Sivashinsky. By analyzing decorrelation timescales, normalized root mean squared error (NRMSE), and absolute error, we find that lower leak rates enhance memory retention and yield more stable predictions. These results suggest that tuning the leak rate within parallel reservoir structures improves long-term forecasting by balancing persistence of information and dynamic adaptability.

2 Introduction

Predicting the behavior of chaotic systems is a longstanding challenge due to their sensitivity to initial conditions and nonlinear dynamics. Traditional modeling approaches often struggle to accurately capture this complexity. Echo State Networks (ESNs), a subset of recurrent neural networks, offer a promising alternative due to their efficient training procedures and strong capacity for modeling temporal dependencies.

This paper investigates a modified Echo State Network (ESN) architecture known as the Parallel Reservoir ESN, which processes inputs through multiple independently operating reservoirs. A key parameter in these networks is the leak rate, which determines the influence of past internal states on current updates effectively controlling memory decay within each reservoir. We systematically explore how varying the leak rate impacts memory retention and predictive accuracy across three canonical chaotic systems: the Lorenz system, the Rössler attractor, and the Kuramoto–Sivashinsky equation. Using evaluation metrics such as cosine similarity (to assess decorrelation timescales), normalized root mean squared error (NRMSE), and total absolute error, we examine how the network's performance responds to the dynamics of each system. The aim is to identify leak rate values that balance memory retention with the ability to respond to changing input dynamics.

3 Model and Methods

3.1 Data Generation

3.1.1 Rössler System

To generate data for the Rössler system, we numerically integrated the system's differential equations using the traditional chaotic form:

$$\frac{dx}{dt} = -y - z$$
$$\frac{dy}{dt} = x + ay$$
$$\frac{dz}{dt} = b + z(x - c)$$

We used the standard chaotic parameters $a = 0.2$, $b = 0.2$, and $c = 5.7$. The simulation begins at time t_0 and spans a total of $t_{\text{train}} + t_{\text{pred}}$ steps, where t_{train} is the number of training timesteps and t_{pred} is the number of prediction timesteps.

The initial condition was set to $\mathbf{x}_0 = [0.1,\ 0.1,\ 0.1]$, and the system was solved with the Runge–Kutta 45 integration method and relative tolerance of 10^{-6}. The timestep size Δt was calculated as:

$$\Delta t = \frac{t_f - t_0}{t_{\text{train}}}$$

The resulting output is a time series array of shape $(t_{\text{train}} + t_{\text{pred}},\ 3)$, which corresponds to the temporal evolution of the system variables $x(t)$, $y(t)$, and $z(t)$. This dataset was saved as a NumPy array and later partitioned into training and prediction windows for use in our experiments.

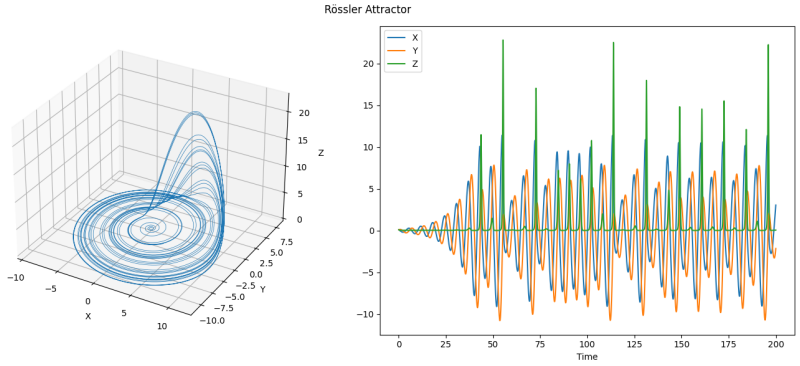


Figure 1: Visualization of the Rössler attractor dynamics used to train the ESN. The system was integrated over the time interval $[0,\ 200]$ using 100,000 evenly spaced timesteps. The simulation used standard chaotic parameters $a = 0.2$, $b = 0.2$, and $c = 5.7$, with an initial condition of $[0.1,\ 0.1,\ 0.1]$. Time integration was performed using the Runge-Kutta method with relative tolerance 10^{-6} The left panel shows the 3D trajectory in phase space, revealing the characteristic spiral structure of chaotic motion. The right panel displays the temporal evolution of each state variable.

3.1.2 Kuramoto-Sivashinsky System

To model the spatiotemporal dynamics of the Kuramoto-Sivashinsky (KS) equation, we implemented a spectral numerical solver based on the exponential time-differencing Runge-Kutta of the Fourth-Order (ETDRK4) scheme. The KS equation describes the evolution of a one-dimensional scalar field $u(x,t)$ on a periodic domain and is given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^4 u}{\partial x^4} = \mu \cos\left(\frac{2\pi x}{\lambda}\right)$$

Here, μ represents the amplitude of an inhomogeneous forcing term with spatial frequency determined by λ. When $\mu = 0$, the system exhibits translational symmetry; nonzero μ breaks this symmetry and introduces spatial localization.

We discretized the spatial domain into N grid points over the interval $[0, L]$, enabling the use of fast Fourier transforms (FFT) to efficiently compute spatial derivatives. The linear terms in the KS equation were handled entirely in Fourier space using the linear operator:

$$\hat{L}(k) = k^2 - k^4$$

where k denotes the Fourier wavenumber vector. The nonlinear advection term was computed in real space and transformed back into Fourier space at each timestep. The inhomogeneous term was precomputed and added at every stage of the time-stepping loop.

Time integration was performed over a total interval long enough to cover both the training and prediction windows. Similar to our Rössler generation, the timestep size was chosen such that the training interval $[t_0, t_f]$ was evenly discretized into t_{train} steps, giving:

$$\Delta t = \frac{t_f - t_0}{t_{\text{train}}}$$

The initial condition was generated as a low-mode cosine perturbation modulated by a sine envelope to ensure smooth periodicity. The ETDRK4 method was used to evolve the system forward in time, and the solution $u(x,t)$ was recorded at each timestep.

The resulting dataset captures the full space-time evolution of the KS field and was later segmented into training and prediction phases.

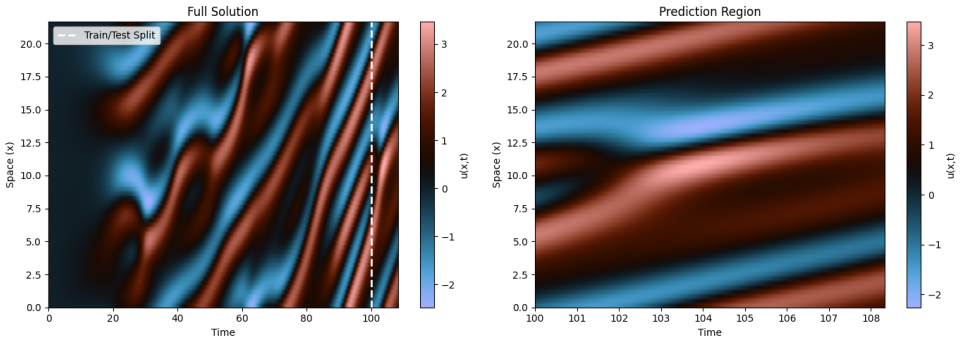


Figure 2: Spatiotemporal solution of the Kuramoto–Sivashinsky system with inhomogeneity $\mu = 0.01$, wavelength $\lambda = 100$, and domain length $L = 22$. The system was discretized using $N = 64$ spatial grid points and integrated over a time interval $[t_0, t_f] = [0, 100]$ with $t_{\text{train}} = 30000$ and $t_{\text{pred}} = 2500$ timesteps. The left panel shows the full evolution of $u(x, t)$, with the vertical dashed line marking the split between training and prediction phases. The right panel zooms into the prediction region, revealing the chaotic dynamics that the ESN aims to forecast.

3.2 Framework and Methodology

3.2.1 Input Partitioning and Training

The model discussed follows the architecture described in Pathak et al. (2018) but with the minor addition of a leaky rate. Though our focus will be on a parallel model, it shares the same principles as a typical single reservoir echo-state network where an input $\boldsymbol{u}(t)$ is passed into a recurrent reservoir $\mathbf{r}$ through the I/R coupler, where the internal states are linearly combined and trained to approximate the target signal over the time interval T_{train}.

$$\mathbf{r}(t+1) = (1-\alpha)\mathbf{r}(t) + \alpha\, G[\mathbf{A}\mathbf{r}(t) + \mathbf{W}_{\text{in},i}\mathbf{u}(t)] \tag{1}$$

In the parallel model, instead of using a single input and reservoir, $\boldsymbol{u}(t)$ is split into multiple overlapping partitions, $\boldsymbol{v}_i(t)$, each assigned its own reservoir $\mathbf{r}_i$. The size of each partition is denoted by $\mathbf{D}_{\text{in}}$, which accounts for the additional length introduced by the overlap, o. As a result, most components in the standard reservoir training function (1), must be replicated for the total number of reservoirs, r_c, with each instance denoted by the subscript i as shown in equation (2):

$$\mathbf{r}_i(t+1) = (1-\alpha)\mathbf{r}_i(t) + \alpha\, G[A_i\mathbf{r}_i(t) + \mathbf{W}_{\text{in}}\mathbf{v}_i(t)] \tag{2}$$

Proceeding from left to right in equation (2), the variables are defined as follows:

- Reservoir State: $\mathbf{r}_i(t)$:
  A column vector of size D_r containing the reservoir states at time t. The initial states were set to zero in this model.

- Leak Rate: α :
  A single constant value from $[0, 1]$ that determines the weight of the previous and newly updated reservoir values when training. This value could vary between reservoirs, but was kept constant in our model.

- Nonlinear Activation Function: G :
  The function that is applied element-wise during the internal update step of the reservoirs to enable nonlinear dynamics in the input. Our model used tanh() which keeps the output bounded between $[-1, 1]$ while maintaining smooth gradients between reservoir states.

- Adjacency Matrix: $\mathbf{A}$:
  A constant [D_r x D_r] matrix which represents the internal connection weights between 'neurons' in the reservoir states. Although the matrix is random, it is scaled by setting a max spectral radius ρ and sparsity of degree k which were set to .58 and 3 respectively.

- Input Weight: $\mathbf{W}_{\text{in},i}$:
  A constant [D_r x D_{in}] matrix initialized with random values from $[-\sigma, \sigma]$ which governs the initial influence of each input value on the updating reservoirs. The value set was quite small, 0.01, but worked quite well for our model.

- Input Partition: $\mathbf{v}_i(t)$:
  The input partition assigned to the corresponding reservoir i. Each partition is of equal size based on the specified number of partitions and overlap length o.

3.2.2 Output Weight Regression

After training the reservoirs over T_{train} time steps, the reservoirs are passed into the R/O coupler (3). Inside, each time step of the reservoir states from $t \in [0, T_{\text{training}}]$ is used to construct a system of equations for solving the output weights $\mathbf{P}_{1,i}$ and $\mathbf{P}_{2,i}$ via Ridge regression (4) with a regularization parameter: $\beta = 10^{-4}$.

$$\mathbf{v}_i(t+1) = \mathbf{P}_{1,i}\,\mathbf{r}_i(t) + \mathbf{P}_{2,i}\,\mathbf{r}_i^2(t) \tag{3}$$

$$\mathbf{W}_{\text{out},\,i} = \mathbf{T}_i\,\mathbf{R}_i^{Transposed}(\mathbf{R}_i\,\mathbf{R}_i^{Transposed} + \beta\mathbf{I})^{-1} \tag{4}$$

Since there are two sets of output weights to solve, the shapes of the regression variables (4) are as follows:

- Reservoir Features Matrix: $\mathbf{R}_i$:
  A stacked [(D_r*2) x T_{train}] matrix containing the reservoir features. The upper half consists of the linear dynamics represented by $\mathbf{r}_i$, while the lower half captures the nonlinear dynamics with $\mathbf{r}_i^2$.

- Target Output: $\mathbf{T}_i$:
  The target input partition for the corresponding reservoir of size D_{in}

- Output Weights: $\mathbf{W}_{\text{out},\,i}$
  A constant matrix storing the solved output weights with the dimensions D_{in} x D_r structured such that the linear dynamics occupy the left half and the nonlinear dynamics in the right half. This results in $\mathbf{P}_{1,\,i} = \mathbf{W}_{\text{out},i}[:, \text{start} : Dr]$ and $\mathbf{P}_{2,\,i} = \mathbf{W}_{\text{out},i}[:, Dr : \text{end}]$.

Afterwards, the prediction loop begins using the final reservoir state from training and the prediction, $\tilde{\mathbf{v}}(t)$, as its initial state and input. Note, $\tilde{\mathbf{v}}(t)$ is the result from equation (3) using the solved output weights in $\mathbf{W}_{\text{out},\,i}$. The tilde notation indicates that the value is a predicted quantity rather than measured data. From this point forward, the reservoirs continue to update using equation (2)(5) with the same variables as in training, except the training input $\mathbf{u}(t)$ is replaced with the prediction $\tilde{\mathbf{v}}(t)$, which is generated at the end of each loop using equation (3)(6). For clarity, the modified equations of the prediction process are shown below.

$$\mathbf{r}_i(t+1) = (1-\alpha)\mathbf{r}_i(t) + \alpha\, tanh[A_i\mathbf{r}_i(t) + \mathbf{W}_{\text{in}}\tilde{\mathbf{v}}_i(t)] \tag{5}$$

$$\tilde{\mathbf{v}}_i(t+1) = \mathbf{P}_{1,i}\,\mathbf{r}_i(t) + \mathbf{P}_{2,i}\,\mathbf{r}_i^2(t) \tag{6}$$

4 Evaluation Metrics

4.1 Decorrelation Timescale

In order to evaluate how long a reservoir retains memory of its past input, we use a metric called the decorrelation timescale. This measures how long the reservoir holds onto its initial state before the similarity with future states drops to zero. In other words, we are determining how long the reservoir "remembers" its previous dynamics before forgetting. We look at the cosine similarity between the reservoir state at a fixed time t_0 and all future reservoir states to determine decorrelation time. Cosine similarity is a metric that measures how aligned two high dimensional vectors are. A value of 1 means the vectors are of identical shape, while a value of 0 means they're completely uncorrelated. Therefore in this context, a value of 1 would mean the reservoir state is the exact same as it was at t_0 and a value of 0 would mean the system had completely forgotten the original state. We compute this similarity over time using the formula:

$$similarity(t) = \frac{r_0 \cdot r(t)}{\|r(t_0)\| \cdot \|r(t)\|}$$

Here, $r(t_0)$ represents the reservoir state at some starting time. We use this starting time as a reference point and then compare it to every reservoir state at a later time. In this case, $r(t)$ represents the reservoir state at some future time t. Once the cosine similarity drops to zero, we say that the reservoir has completely decorrelated, meaning it no longer resembles its previous state at all. The timestep where this happens is what we call the decorrelation time.

We implemented this into our code by first selecting a reference state t_0. We then looped through all future timesteps and computed cosine similarity. We repeated this for each reservoir in the parallel model and stored the timestep when similarity first reached zero. This process was also done on the raw data of the Lorenz, Rössler, and Kuramoto-Sivashinsky systems to compare the model's memory to the system's own memory decay.

It is also important that timestep counts mean different things depending on what system we are looking at. That's because each system uses a different number of training steps and simulates over a different total time interval, which changes the size of one timestep. We calculate the time resolution for each system using the equation:

$$\Delta t = \frac{t_f - t_0}{t_{train}}$$

For the Lorenz system, we used the values $t_{train} = 10000$ and $t_f = 40$, giving us a value of $\Delta t = 0.004$. For the Rössler system, we used the values $t_{train} = 20000$ and $t_f = 500$, giving us a value of $\Delta t = 0.025$. For the Kuramoto-Sivashinsky system, we used the values $t_{train} = 10000$ and $t_f = 75$, giving us a value of $\Delta t = 0.0075$. Since each system uses a different Δt, the number of timesteps represents a different amount of time for every system. For example, 100 steps in Rössler represents 2.5 seconds, but only 0.75 seconds in KS and 0.4 seconds in Lorenz. To make comparisons more accurate, we convert all decorrelation step counts into real time by multiplying the timestep index by the corresponding system's Δt value.

In each of these systems, we repeat this analysis for each system across three different leak rates: $\alpha = 1.0$, $\alpha = 0.5$, and $\alpha = 0.1$, in order to directly compare how the reservoir's ability to retain memory changes as the leak rate decreases. Tracking when the cosine similarity hits zero gives us a straightforward way to measure how quickly the reservoir forgets. This helps us see how different leak rates impact memory retention and how well each model reflects the behavior of the system it's trying to learn

4.2 NRMSE and Absolute Value Sum Problem

To measure the accuracy of our model, we test performance using the NRMSE and Absolute Sum evaluation metrics. NRMSE demonstrates the reservoir's prediction accuracy and tells us when there are dramatic differences in prediction and ground truth. Absolute Value Sum validates our model by weighing small and large errors equally, thus capturing systematic bias. The error time series is modified to standardize our evaluations' scope. First, for every validation timestep, we pair the reservoir's mean prediction y^t with the matching ground-truth KS state y_t. Then, we convert each y_t and y^t from a 1-D spatial field of length M into a row vector of shape $1 \times M$. By stacking these row vectors over all N validation timesteps, we form two matrices of size $N \times M$: one for the ground truth, and one for the predictions.

$$E_{t,x} = \hat{y}_{t,x} - y_{t,x}$$

$$RMSE = \sqrt{\frac{1}{NM} \sum_{t=1}^{N} \sum_{x=1}^{M} E_{t,x}^2}$$

Then, to compute the NRMSE, we first compute RMSE. In the RMSE calculations, the residuals are squared to make all errors positive and to amplify large deviations. This becomes important in detecting large divergences in a chaotic system. The double sum integrates error energy over every point in space and time. Then, we divide by NM which turns that integral into a simple arithmetic mean and because the grid is uniform, no dx weighting is required. Finally, the square root puts the result back in the original physical units, so the RMSE is an error amplitude that can be inputted directly into NRMSE.

$$NRMSE = \frac{RMSE}{\sigma}$$

In NRMSE, we compute a single overall standard deviation, σ (sigma), from all values in the ground-truth data covering N validation time steps and M spatial grid points ($N \times M$ block). Because NRMSE keeps the RMSE numerator but divides by this global σ (sigma), the averaging and square-root structure stays the same, the units cancel, and the result is a dimensionless ratio. Absolute Value Sum captures the average size of prediction errors without squaring them, making it sensitive to systematic bias—where small, consistent errors add up over space and time.

In the absolute value sum because we skip the squaring step, this metric is sensitive to systematic bias—where steady small errors accumulate. In the Absolute Sum function equation, $E_{t,x}$ is the signed residual computed as described earlier.

$$A_{t,x} = |E_{t,x}|$$

$$\text{Absolute Value Sum} = \frac{1}{NM} \sum_{t=1}^{N} \sum_{x=1}^{M} A_{t,x}$$

Double sum $\sum_{t,x}$ – integrates over space and time. $\frac{1}{NM}$ – converts the raw sum into the *mean absolute error*.

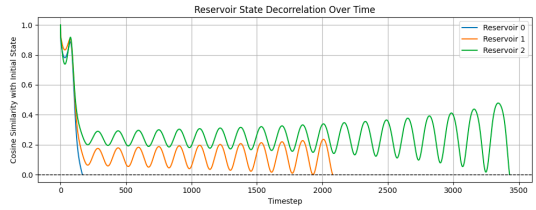
In this equation, the double sum integrates over space and time while dividing by N and M converts the raw sum into a mean absolute error.

4.3 Prediction Analysis

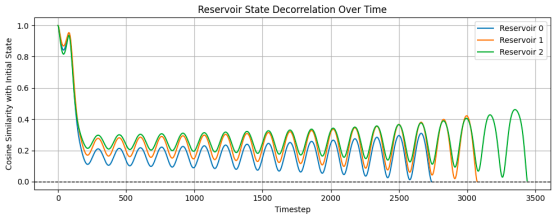
To compare performance between a single reservoir echo state network and our hybrid model parallel reservoir echo state network, we test performance in three chaotic systems: Lorenz 96, Rössler, and the KS system using NRMSE and

Absolute Sum test metrics. Additionally, we test performance between single and parallel reservoir's with varying leaky rates: $\alpha = 0.1, 0.5, 1.0$.

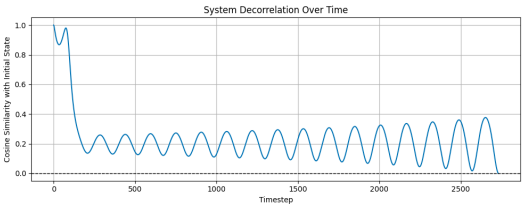
4.3.1 Lorenz



(a) $\alpha = 1.0$



(b) $\alpha = 0.1$



(c) System

Figure 3: (Decorrelation Timescale) Cosine similarity over time for the Lorenz system and reservoirs at two different leak rates.

To understand how leak rate impacts memory retention, we examine the Lorenz system at two leak rates: $\alpha = 1.0$ and $\alpha = 0.1$. Each plot shows cosine similarity over time for three parallel reservoirs, compared against the system's own decorrelation behavior. At $\alpha = 1.0$, the reservoirs forget quickly, decorrelating at steps 168, 2079, and 3430, with an average of 2179. This translates to about 8.72 seconds of memory (2179×0.004). However, at $\alpha = 0.1$, the reservoirs hold memory longer, with decorrelation steps at 2739, 3072, and 3440, averaging 3083, or about 12.33 seconds. For context, the Lorenz system itself decorrelates at step 2732, or 10.93 seconds. So, while high-leak reservoirs forget before the system does, the low-leak ones retain memory even beyond the system's own timescale, demonstrating improved retention with lower leak rates.

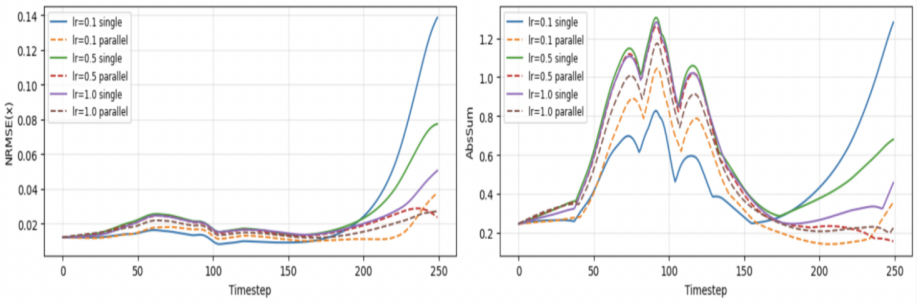


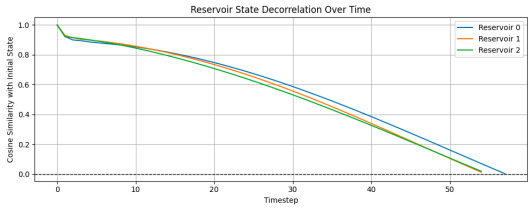
Figure 4: NRMSE and Absolute Value Sum Calculations for Parallel vs Single Reserviors in the Lorenz System

When we compare the NRMSE curves, we track how prediction error grows relative to the Lorenz system's natural variability. Because the Echo State Network (ESN) begins with true Lorenz-Attractor data and trains its readout weights on that exact state, it initially follows the target trajectory closely. The first predicted point matches the true path, but subsequent predictions feed back the ESN's own outputs as inputs. A low leaky rate then functions as a low-pass filter: it smooths the internal state and reduces the influence of high-frequency deviations.

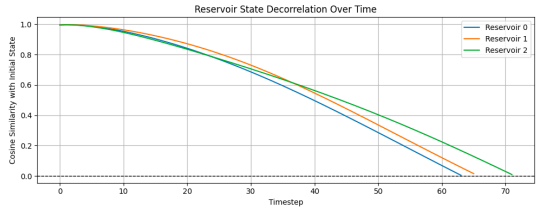
The NRMSE starts near 0.015, showing that the model tracks the reference signal well. Over time, single reservoirs (solid lines) drift sharply, whereas parallel reservoirs with varied leaky rates (dashed lines) stay below 0.04 even at later timesteps. These results indicate that the parallel reservoir approach slows error growth and sustains accuracy over longer prediction times.

In the absolute value sum graph, we see that single reservoirs quickly exceed 1.3 units of total error, while parallel reservoir remain near 1.1. Between time steps 50 and 120, errors of both single and parallel reservoirs drop toward 0.25, yet only the parallel reservoirs remain stable. Overall, the parallel reservoirs not only have smaller relative error (NRMSE) but also keeps absolute error smaller and more consistent over time.

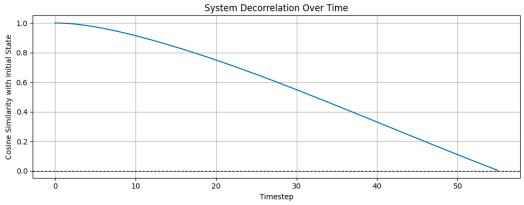
4.3.2 Rössler



(a) $\alpha = 1.0$



(b) $\alpha = 0.1$



(c) System

Figure 5: (Decorrelation Timescale) Cosine similarity over time for the Rössler system and reservoirs at two different leak rates.

To understand how leak rate impacts memory retention in the Rössler system, we compare reservoir behavior at $\alpha = 1.0$ and $\alpha = 0.1$. Each plot shows cosine similarity over time for three parallel reservoirs, alongside the system's own decorrelation curve. At $\alpha = 1.0$, the reservoirs decorrelate quickly, with steps at 58, 55, and 55, averaging just 56 timesteps, or 1.4 seconds (56×0.025). With $\alpha = 0.1$, the reservoir memory lasts longer, decorrelating at steps 64, 66, and 72, for an average of 67 timesteps, or 1.68 seconds. The Rössler system itself decorrelates at step 56, which is 1.40 seconds, matching the average of the high-leak rate reservoirs and sitting below the low-leak rate average. This indicates that with a lower leak rate, the reservoirs retain memory slightly longer than the system itself. Though the difference in timescale is smaller than in the Lorenz case, the pattern still holds that reducing the leak rate improves memory retention.

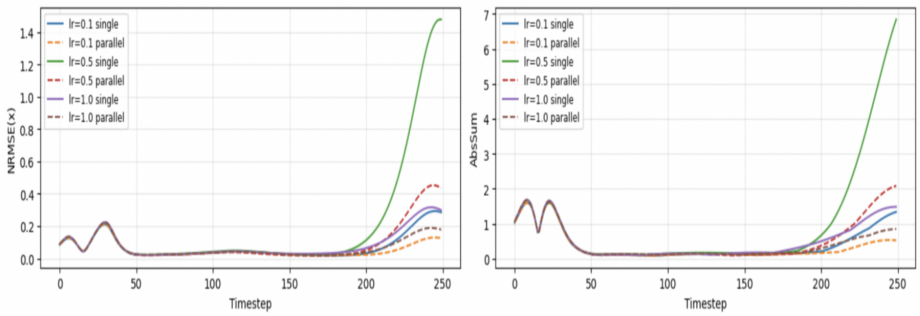


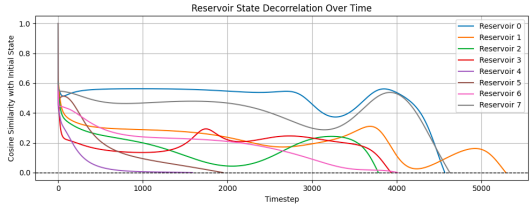
Figure 6: NRMSE and Absolute Value Sum Calculations for Parallel vs Single Reserviors in the Rossler System

In the NRMSE plot, the curve stays accurate until timestep 200. Then, towards the end, single-reservoir trajectories (solid lines) diverge far more rapidly than their parallel-reservoir trajectories (dashed lines). For instance, the single reservoir with leaky rate 0.5 (green solid) reaches an NRMSE above 1.4—an error that exceeds the data's own variability—whereas the matching parallel reservoir's error (red dashed) remains below 0.5. Hence, a single reservoir loses the true track of a chaotic Rössler dynamics much earlier, while three smaller reservoirs operating in parallel sustain a far lower normalized error.

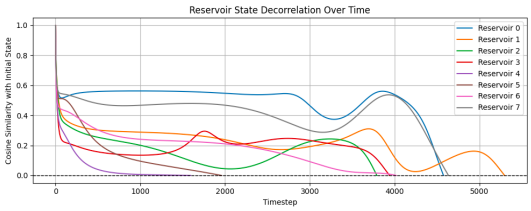
The absolute-value-sum plot shows the raw total error across the three Rössler variables. Errors peak near t=20 and reaches about 1.2 units of total error— when the system's fast oscillations make forecasts hard. By t = 250, the single model with leaky rate 0.5 accumulates nearly 7 units of total error, while the parallel reservoir's error stays below 2. Throughout the experiment, parallel reservoirs have a normalized error to < 0.5 and the absolute-sum error to < 2, whereas single reservoirs have normalized error that surpasses 1.4 and total error to 7. In both graphs, parallel reservoirs perform better compared to single reservoirs.

Splitting the task among three parallel reservoirs therefore yields more stable predictions for the Rössler system's chaotic behavior.

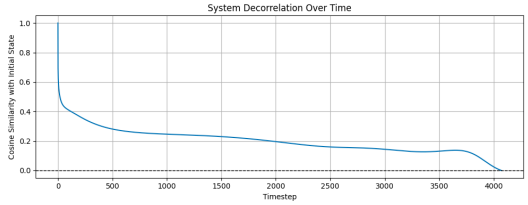
4.3.3 Kuramoto-Sivashinsky



(a) $\alpha = 1.0$



(b) $\alpha = 0.1$



(c) System

Figure 7: (Decorrelation Timescale) Cosine similarity over time for the Kuramoto-Sivashinsky system and reservoirs at two different leak rates.

To assess how leak rate influences memory in the Kuramoto–Sivashinsky system, we examine cosine similarity across eight parallel reservoirs at $\alpha = 1.0$ and $\alpha = 0.1$, and compare these results to the system's own decorrelation timescale. For $\alpha = 1.0$, the reservoirs decorrelate at steps 4568, 5287, 3783, 3932, 1589, 1948, 4011, and 4626. The average across all eight is 3718, or about 27.89 seconds 3718×0.0075, indicating that some reservoirs retain memory quite long, though a few forget relatively quickly. At $\alpha = 0.1$, the decorrelation steps shift slightly: 4577, 5297, 3790, 3941, 1594, 1958, 4018, and 4635, with an average of 3726, or 27.95 seconds. While the change in average is minimal, individual reservoir behavior becomes slightly more consistent at the lower leak rate. The system itself decorrelates at step 4072, or 30.54 seconds. Both sets of reservoirs under-approximate the system's memory, but the low-leak group ($\alpha = 0.1$) stays closer to the system's timescale overall. In this case, the benefit of a lower leak rate is less dramatic, but it still provides a slight improvement in

memory retention, particularly in aligning reservoir behavior more closely with the system.

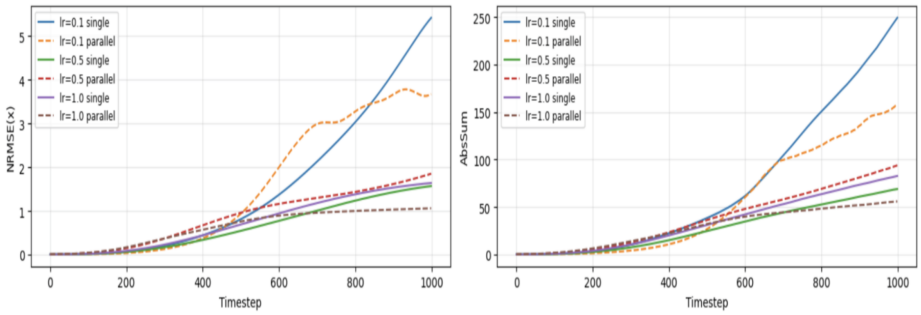


Figure 8: NRMSE and ABsolute Value Sum Calculations for Parallel vs Single Reserviors in the Kuramoto-Sivashinsky System

The NRMSE plot for the Kuramoto-Sivashinsky(KS) system illustrates the magnitude of prediction errors in relation to the natural fluctuations of the system. Initially, both single and parallel reservoirs exhibit lines that start very close to zero, indicating that they both track the true Kuramoto-Sivashinsky(KS) data effectively. However, as time progresses, the single-reservoir curves (solid lines) rise, experiencing a significant spike after approximately timestep 600. In contrast, the parallel reservoirs (dashed lines) exhibit a much slower increase. This observation demonstrates that partitioning the system into three reservoirs significantly reduces the normalized error, resulting in more accurate predictions.

In the absolute-value-sum plot, we initially observe chaotic divergence, but total error remains below 10 units. Subsequently, the total errors for the single reservoirs increase, indicating a decline in the model's accuracy towards the end. The parallel reservoir with a leaky rate of 0.1 (orange dashed line) consistently stays below 160 units, while the parallel reservoir with a leaky rate of 1.0 remains close to 60 units. Overall, across all three systems, the data clearly shows that parallel reservoirs outperform single reservoirs. They effectively control the growth of relative errors and maintain lower magnitudes of raw errors, thereby enhancing the reliability of their predictions.

5 Caveats and Discussion

Although our results suggest that lower leak rates tend to enhance memory retention and improve predictive stability, several important limitations must be acknowledged. First, the benefit of reducing the leak rate is not consistent across all systems. For example, in the Kuramoto–Sivashinsky system, lower leak rates yielded only modest gains in memory timescales. This indicates that the relationship between leak rate and memory retention is system specific and

may exhibit diminishing returns beyond a certain point.

Second, while parallel reservoirs offer improved performance compared to single-reservoir architectures, they introduce greater computational overhead and longer training times. Addressing scalability in high-dimensional settings may require additional strategies such as model compression, adaptive reservoir allocation, or more efficient training schemes. Third, our approach assumes fixed reservoir weights and does not incorporate synaptic plasticity or online learning mechanisms. In dynamic, real-world applications, models may need to adapt continuously to evolving conditions without retraining from scratch.

Finally, all experiments were conducted on synthetic chaotic systems. Extending these findings to real-world data such as meteorological or financial time series would require handling noise, partial observability, and non-stationary behavior, none of which were explicitly considered in this study.

6 Conclusion

This study examined how the leak rate influences memory retention and predictive accuracy in Parallel Reservoir Echo State Networks applied to three canonical chaotic systems. Our findings show that lower leak rates enhance memory duration and reduce prediction errors, particularly in the Lorenz and Rössler systems. Additionally, parallel reservoir architectures consistently outperformed single-reservoir setups in both NRMSE and absolute error metrics, demonstrating improved long-term stability and robustness.

These results highlight the importance of tuning the leak rate and leveraging architectural parallelism to improve ESN performance on chaotic dynamics. Future work may focus on adaptive leak rate strategies, extension to real-world datasets, and hybrid architectures that integrate plasticity and reservoir diversity to support broader generalization.

References

Chattopadhyay, A., Hassanzadeh, P., & Subramanian, D. (2020). *Data-driven predictions of a multiscale Lorenz 96 chaotic system using machine-learning methods: Reservoir computing, artificial neural network, and long short-term memory network. Nonlinear Processes in Geophysics*, *27*, 373–389. https://doi.org/10.5194/npg-27-373-2020

Pathak, J., Hunt, B., Girvan, M., Lu, Z., & Ott, E. (2018). *Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. Physical Review Letters*, *120*, 024102. https://doi.org/10.1103/PhysRevLett.120.024102