RESERVOIR COMPUTING FOR EFFICIENT AND EFFECTIVE LEARNING

BEDİRHAN ÇELEBİ

ÖZYEĞİN UNIVERSITY

SEPTEMBER, 2025

RESERVOIR COMPUTING FOR EFFICIENT AND EFFECTIVE LEARNING

by

BEDİRHAN ÇELEBİ

Thesis
Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in
Artificial Intelligence

Advisor: Prof. Erhan Öztop

Graduate School of Science and Engineering
Özyeğin University
İstanbul

September, 2025

RESERVOIR COMPUTING FOR EFFICIENT AND EFFECTIVE LEARNING

Approved by:

Prof. Dr. Erhan Öztop, Advisor
Department of Artificial Intelligence and Data Engineering
Özyeğin University

Assoc. Prof. Dr. Emre Sefer
Department of Artificial Intelligence and Data Engineering
Özyeğin University

Assoc. Prof. Dr. Emre Uğur
Department of Computer Engineering
Boğaziçi University

Approval Date: 24/07/2025

To my mother and İpek

DECLARATION OF ORIGINALITY

I hereby declare that I am the sole author of this thesis and that this is the true copy of my thesis, including the final revisions, approved by my thesis committee. All data and information have been obtained, produced, and presented in accordance with the rules of research ethics and principles of academic honesty. As required by these rules, to the best of my knowledge I have acknowledged ideas, thoughts, and any copyrighted material in accordance with the standard referencing rules. I certify that any part of this thesis has not been submitted for a degree or diploma in another educational institution.

Bedirhan Çelebi

ABSTRACT

Resource-efficient modeling of nonlinear dynamical systems is vital for embedded sensing, edge analytics, and neuromorphic hardware. Reservoir Computing (RC) reduces training overhead in recurrent neural networks and transformers by fixing internal weights and training only a linear read-out. However, classical Echo State Networks (sRC) scale capacity by increasing reservoir size, which inflates memory and energy costs. This thesis addresses that bottleneck by improving the *quality*, rather than the *quantity*, of reservoir units. We propose the *Higher-Order Augmented Reservoir Computer* (haRC), which augments reservoir with a subset of multiplicative monomials of co-temporal activations. This polynomial feature space enhances nonlinear expressivity while keeping the reservoir compact. On chaotic benchmarks such as Lorenz and Mackey–Glass, haRC achieves the same or better accuracy with fewer parameters than baseline sRC models.We propose two improved models: **Selective haRC**, which ranks reservoir units by variance and covariance to discard redundant interactions, and **Sparse haRC**, which integrates controlled sparsity masks to reduce memory and compute load. Comprehensive experiments evaluate memorization and forecasting tasks under equal total and tunable parameter settings across different sparsity levels. haRC's consistently outperform sRC across all sparsity regimes, with the advantage growing in high sparsity. Noise robustness tests inject Gaussian noise into the reservoir. Selective haRC maintains or exceeds the performance of sRC in terms of root-mean-squared error and variance with 12% of the reservoir units sRC has. Demonstrating up to $120\times$ memory efficiency, higher-order augmentation emerges as a promising approach for ultra-lightweight sequence learners suited for microcontrollers, FPGA overlays, and in-memory compute systems.

Keywords: Reservoir Computing, Echo State Network, Higher-Order Augmentation, Sparse Neural Dynamics, Dynamic Systems

ÖZET

Doğrusal olmayan dinamik sistemlerin verimli modellenmesi, gömülü sistemler, uç sistemler ve nöromorfik donanımlar için büyük önem taşımaktadır. Rezervuar Hesaplama (RC), yinelemeli sinir ağları ve dönüştürücü modellerindeki eğitim yükünü, iç bağlantıları sabit tutarak ve yalnızca doğrusal bir çıktı katmanı eğiterek azaltır. Ancak klasik Echo State Ağları (sRC), kapasiteyi rezervuar boyutunu artırarak ölçeklendirdiğinden bellek ve enerji maliyetlerini önemli ölçüde artırır. Bu tez, bu darboğazı, rezervuar birimlerinin *sayısını* artırmak yerine *niteliğini* geliştirerek aşmayı amaçlamaktadır. Bu doğrultuda, eşzamanlı aktivasyonların çarpımsal monomilerinden seçili bir altküme ile rezervuarı zenginleştiren *Yüksek-Dereceli Artırılmış Rezervuar Bilgisayarı* (haRC) önerilmektedir. Lorenz ve Mackey–Glass gibi kaotik kıyaslama sistemleri üzerinde haRC, temel sRC modellerine göre daha verimlidir. İki yeni model daha önerilmektedir: **Seçmeli haRC**, gereksiz etkileşimleri elemek için rezervuar birimlerini varyans ve kovaryansa göre seçer; **Seyrek haRC** ise kontrollü seyreklik maskeleri uygular. Kapsamlı deneylerde, farklı seyreklik seviyelerinde eşit toplam ve öğrenilebilir parametre koşulları altında ezberleme ve kestirim görevleri değerlendirilmiştir. haRC modelleri, tüm seyreklik düzeylerinde tutarlı şekilde sRC modellerinden daha iyi performans göstermekte; bu fark yüksek seyreklik koşullarında daha da belirginleşmektedir. Gürültü dayanıklılığı testlerinde rezervuara Gauss gürültüsü enjekte edilmiştir. Seçmeli haRC, sRC'nin yalnızca %12'si kadar birimle çalışırken bile ortalama karekök hatası açısından iyi performans sergilemektedir. $120\times$'e varan bellek verimliliğiyle, yüksek-dereceli artırma yaklaşımı, mikrodenetleyiciler, FPGA tabanlı sistemler ve bellek-içi hesaplama mimarileri için ultra hafif dizisel öğreniciler adına umut verici bir yöntem olarak öne çıkmaktadır.

Anahtar Kelimeler: Rezervuar Hesaplama, Echo State Ağı, Yüksek-Dereceli Artırma, Seyrek Sinir Dinamikleri, Dinamik Sistemler

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Prof. Dr. Erhan Öztop, for his continuous guidance, encouragement, and insightful feedback throughout my thesis journey. His mentorship has been instrumental not only in the development of this work but also in shaping my approach to research and problem-solving.

I am also sincerely thankful to the esteemed jury members, Assoc. Prof. Dr. Emre Sefer and Assoc. Prof. Dr. Emre Uğur, for their valuable comments and suggestions that greatly enriched the final version of this thesis.

Special thanks go to my dear friends and colleagues at the Artificial Intelligence Lab at Özyeğin University — including Arash Mehrabi, Zahra Koulaeizadeh, Emir Arditi, Zehra Kesemen, Negin Amirshirzad, Mehmet Arda Eren, Hanne Say, and Amin D. Alamdari, among others — for their unwavering support, stimulating discussions, and the much-needed moments of levity during challenging times.

To my mother, whose strength and unconditional support have always been my foundation — thank you for standing by me in every chapter of this journey. And to İpek Binnaz Ünsal, for her endless encouragement, love, and belief in me — your presence gave me the clarity and calm to move forward, even in the most difficult moments.

Lastly, I want to acknowledge all the people and experiences that contributed, directly or indirectly, to this thesis and my academic growth. This work is a reflection of collective inspiration, persistence, and kindness, and I am truly grateful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

AR	Autoregressive
ARMA	Autoregressive Moving Average
ARIMA	Autoregressive Integrated Moving Average
ANN	Artificial Neural Networks
BPTT	Backpropagation Through Time
DL	Deep Learning
ESN	Echo State Network
ESP	Echo State Property
GRU	Gated Recurrent Unit
haRC	Higher-order Augmented Reservoir Computer
k-NN	k Nearest Neighbor
LSTM	Long Short-Term Memory
ML	Machine Learning
MA	Moving–Average
MLP	Multi-Layer Perceptron
RC	Reservoir Computing
RNN	Recurrent Neural Network
sRC	Standard Reservoir Computer
SARIMA	Seasonal Autoregressive Integrated Moving Average
SVR	Support Vector Regression

1. INTRODUCTION

Physical and informational processes of our universe are predominantly dynamic, involving continuous change with respect to time or other temporally evolving variables. Differential equations serve as the foundational mathematical tools for modeling such systems. They provide a formal structure for describing interactions in a wide array of domains [2], ranging from celestial mechanics [3] and weather forecasting [4, 5] to fluid dynamics [6], physiology [7], finance [8, 9], and robotics [10].

However, for many real-world systems, deriving explicit differential equations is a formidable task [11]. These equations can be of arbitrary order, often containing numerous unknown or partially observed parameters. For example, modeling turbulence in fluid dynamics [12], predicting asset prices in financial markets [13], or simulating blood cell production under hormonal influence [14] all require constructing and solving intricate differential systems. The process is not only analytically challenging but also computationally prohibitive in many cases [15].

This difficulty has motivated the use of data-driven [11, 16, 17] alternatives under the broader umbrella of *time series learning* [18] or *dynamical system learning* [19, 20]. Depending on whether the system under study is assumed to be stochastic [21] or deterministic [22], various modeling approaches are employed. In high-dimensional systems with low individual variable freedom—such as particle gases [23]—probabilistic models rooted in statistical mechanics are often preferred. In contrast, when dealing with systems governed by a few variables with high interaction complexity—such as hormone-driven blood cell dynamics [24, 25]—deterministic models are more appropriate, requiring methods capable of uncovering hidden nonlinear relationships and latent dynamics.

To address these needs, researchers have increasingly turned to statistical modeling [26], Machine Learning (ML) [27, 28], and Deep Learning (DL) [29] to approximate system dynamics directly from observed data. Statistical methods aim to capture

probabilistic distributions of future states [30], while classical ML models introduce nonlinearity and functional approximation tools [31, 32] to model deterministic trends. Deep learning takes this a step further by employing deep neural networks with multiple nonlinear layers to approximate complex differential relationships in high-dimensional temporal spaces [33].

Within this deep learning paradigm, (RNN) [34, 35, 36, 37, 38, 39] and their variants, such as Long Short-Term Memory (LSTM) [40] and Gated Recurrent Units (GRUs) [41], have been widely adopted for modeling temporal sequences. These models utilize Backpropagation Through Time (BPTT) [42, 43, 44] to learn temporal dependencies by unfolding network dynamics across multiple time steps. While powerful, BPTT-based training is computationally expensive and prone to gradient instability [42, 43, 44] and bifurcations [45, 46, 47], especially for long sequences and deep architectures if vanilla RNN's are employed. Several efforts have aimed to improve the Backpropagation Through Time (BPTT) algorithm by refining its structure and enhancing computational efficiency. These include approaches to reduce memory overhead [48], formulate alternative variants [49],and explore new theoretical and architectural perspectives [50, 51, 52] that offer promising directions for advancing sequential learning.

Reservoir Computing (RC) [51, 53, 54, 55] emerges as a compelling alternative, offering a lightweight, efficient training paradigm for learning dynamical systems. In RC architectures such as Echo State Networks (ESNs) [54], the recurrent hidden layers—referred to as the reservoir—are randomly initialized and fixed, requiring no gradient-based updates. Instead, only the readout layer is trained, typically using linear regression [56, 53, 54]. The reservoir acts not only as a projection mechanism but also as a nonlinear dynamical system in its own right. Its evolving internal state reflects the influence of input history through a network of recurrent interactions. Crucially, the

utility of the reservoir arises not just from expanding dimensionality, but from the hypothesis that its intrinsic dynamics may approximate, emulate, or partially synchronize with those of the target system being modeled [57, 58]. If the reservoir's internal evolution captures relevant features of the system's phase space—such as attractor geometry, memory timescales, or nonlinear transitions—then the readout layer can effectively exploit these internal representations to reconstruct the target output. In this sense, the reservoir serves as a dynamic encoder of the system, and learning occurs primarily by adjusting the readout to decode useful aspects of the reservoir's ongoing temporal evolution. RC has demonstrated promising performance in domains requiring low latency, fast adaptation, and energy efficiency [59, 60] such as edge computing [61] and neuromorphic [62] systems. The reservoir itself can be implemented via diverse means, including, quantum devices [63], fixed-weight RNNs [56, 53, 54], electronic circuits [64], optical media [65], liquid based devices [66] or even physical systems like fluid containers [67].

Despite their simplicity and speed, RC architectures suffer from a critical bottleneck: memory resource consumption [68]. While the internal reservoir does not require training, increasing its size to improve model capacity directly increases memory usage, making it impractical for deployment on low-resource or embedded devices. Fortunately, research has shown that sparsity [69, 70] in reservoir connectivity can mitigate this issue without severely compromising performance.

Building on these insights, this thesis aims to further improve the resource efficiency of reservoir computers by augmenting their expressive capacity without increasing their size. We introduce a novel approach called the *Higher-Order Augmented Reservoir Computer (haRC)* [71], which enhances the nonlinear representation power of reservoir states through multiplicative interactions—specifically, higher-order monomial terms constructed from existing same time step reservoir activations. This augmentation enables the model to capture richer dynamics and nonlinearities, improving learning performance

under tight memory constraints similarly having examples in [72, 73, 74, 75]

We further propose selective-haRC, a variant that augments reservoir states through statistically guided heuristics—such as variance and covariance analysis [76, 77] —to selectively identify and target specific reservoir units for augmentation. Thus, this method help with minimizing redundant computation while maximizing representational gain.

Empirical evaluations demonstrate that our augmented reservoir models achieve superior performance compared to standard RC configurations, especially in scenarios involving minimal readout layers, or severely limited memory resource capacity. In some configurations, haRC is able to operate successfully under constraints where standard reservoirs fail altogether, achieving strong predictive performance with effectively zero additional learnable parameters.

1.1 Thesis Organization

The remainder of this thesis is structured as follows:

- **Chapter 2** presents the theoretical background, including RNNs, ESNs, and a review of related work on reservoir design and augmentation strategies.
- **Chapter 3** details the contributions made in the conference paper and introduction of the Higher Order Augmented Reservoir Computer.
- **Chapter 4** outlines the contributions presented in the submitted journal paper, including sparsity and noise analysis in Reservoir Computing, and the introduction of the selective-haRC architecture.

2. THEORETICAL BACKGROUND AND RELATED WORK

2.1 Dynamical Systems

Understanding time-evolving behavior in physical, biological, and engineered systems requires a formal framework that captures change, feedback, and causality. Dynamical systems theory provides this foundation. It allows us to mathematically model how system states progress over time, often governed by internal laws and external forces. Whether describing planetary motion, neural activity, economic fluctuations, or data-driven AI systems, dynamical systems offer a unifying lens to capture temporal complexity and emergent behavior. In this section, we begin by revisiting the classical formulation of dynamical systems through differential equations, which remain essential for both theoretical insight and practical simulation.

2.1.1 Differential Equations and Real-World Systems

Time is one of the most foundational and complex constructs in our understanding of the universe. We first encountered it through the rhythm of our lives, then formalized it through observation and scientific abstraction. Eventually, time was elevated to a dimension of its own — not absolute, but relational, shaped by the motion and interaction of particles and fields, as described by the theory of relativity. In modern dynamics, we can think of time not merely as a passive canvas, but as an active platform — one on which the components of a system interact, synchronize, and evolve.

This complexity is perhaps best revealed not in particle accelerators but in everyday phenomena. Consider the intricate mechanism of a mechanical watch, where gears and springs oscillate in response to the stable periodicity of a vibrating quartz crystal. Or the passage of seasons, caused solely by the Earth's axial tilt — a seemingly trivial geometric feature that ends up guiding the evolutionary rhythms of ecosystems and climate

systems on a planetary scale. These examples remind us that small changes in configuration can give rise to large-scale temporal structures [78, 2]. Time, in this light, becomes a platform of interaction — enabling, regulating, and coordinating the evolution of complex systems.

A dynamical system is defined by its evolution — a process in which variables change with respect to time, or with respect to each other through time. These systems may range from low-dimensional constructs like a swinging pendulum or an LC circuit, to high-dimensional interactions like weather systems, neural activity, or stock market dynamics. The components of such systems may include mass, pressure, temperature, orientation, price or any measurable property that exhibits change over time. One intuitive example is average regional temperature, which varies throughout the year due to the planet's axial tilt — a temporal signal we all experience through the changing seasons.

Observing and measuring the relevant variables in these systems is often challenging [79, 80]. In fluid dynamics, for example, it is rarely feasible to track individual particles [81]; instead, dyes or tracer elements are used to infer motion indirectly. In economic systems, we might observe consumer purchases or price fluctuations, without ever accessing the underlying decision processes [82] that drive them. These measurement constraints imply that what we model is often not the true system, but a projected or noisy approximation of it. Moreover, we may misidentify which variables truly govern the dynamics — or whether what we observe is even relevant at all.

Our formal understanding of these systems comes through equations — particularly differential equations — which express how the rate of change in a variable relates to other aspects of the system. At their core, these equations encode how parts of a system interact through time. For instance, the first-order differential equation

$$\frac{dx(t)}{dt} = -\alpha x(t) \tag{2.1}$$

describes exponential decay [83], with a solution

$$x(t) = x_0 e^{-\alpha t} \tag{2.2}$$

where x_0 is the initial condition. Even in this simple case, we see that initial conditions strongly shape the long-term evolution of the system. These relations lie at the heart of our ability to forecast, control, and understand dynamical phenomena.

Such equations underpin some of the most significant scientific breakthroughs. Newton's second law, for example, describes how force relates to mass and acceleration — giving rise to celestial mechanics and classical gravity. The Black-Scholes-Merton [9] equation transformed financial markets by enabling the pricing of derivative securities under a set of stochastic assumptions, opening a new era in risk management and investment. Even the seemingly simple exponential decay equation plays a key role in nuclear physics [84], linking it to the probabilistic nature of radioactive processes.

2.1.2 Difficulties in Modeling Dynamical Systems

Although differential equations provide a powerful framework for understanding dynamical systems, deriving or applying them in real-world scenarios is rarely straightforward. In principle, one would hope to identify the key variables, construct a set of governing equations, and obtain an analytical or numerical solution to describe system behavior. In practice, however, several layers of complexity prevent this idealized process from being realized.

The first challenge lies in observation. In many systems, the relevant state variables are not directly measurable. Our instruments may have limited resolution, or we may observe proxy variables that are only loosely connected to the true underlying dynamics. For instance, tracing individual fluid particles in turbulence is impossible without introducing dyes or passive markers. Similarly, in economic systems, we might record

the transactions or market behavior of individual agents, but the motivations and decision-making processes behind those actions remain opaque. The observable world offers us fragments, not full views.

Even when relevant variables are known and measurable, constructing accurate equations is nontrivial. Many dynamical systems are high-dimensional, nonlinear, and exhibit delayed or coupled feedback [2, 85]. Solving such systems analytically is often impossible. The Lorenz attractor [78], a simplified model of atmospheric convection, provides a famous example: its differential equations are well-defined, yet they produce chaotic trajectories that are highly sensitive to initial conditions. Similarly, the Mackey-Glass [25] equation models delayed feedback in physiological systems and generates complex, unpredictable patterns. These models are useful not because they precisely represent reality, but because they reproduce important characteristics of observed behavior — enabling understanding through analogy, rather than perfect correspondence.

In some cases, modeling is not just difficult — it is philosophically ambiguous. In financial markets, for example, price movements emerge from the interactions of thousands of traders, institutions, and algorithms [86] — each operating at different time scales with their own objectives and beliefs. The aggregate result is a stochastic signal shaped by countless interacting components. Attempting to model such a system with deterministic equations would be misleading. Instead, stochastic processes and probabilistic models are employed [87, 88] to account for the underlying uncertainty and heterogeneity.

There are also systems where the boundary between deterministic and stochastic behavior blurs. Quantum mechanics presents a well-known example [89, 90], where subatomic particles obey probabilistic laws. While quantum systems may evolve according to Schrödinger's equation [91], their measurement outcomes are inherently stochastic. Likewise, complex biological systems may contain elements of both structured regulation and randomness [92, 93, 94], making them resistant to purely symbolic description.

A further complication arises when we simply do not know what the "right" variables are. Consider a gas in a container: modeling every molecule is intractable, but the system can still be described using pressure and temperature as macroscopic parameters. In robotics, a walking robot might adapt its gait based on passive feedback from the terrain, without relying on an explicit internal model of the environment [95]. In such cases, modeling focuses on observable inputs and outputs, not internal representations.

These limitations have led to a growing reliance on data-driven models [11, 16, 17]. When symbolic derivation is impossible or unreliable, machine learning offers an alternative path — learning the system's dynamics directly from observed sequences. By mapping data to future outcomes, these models approximate the underlying dynamics without requiring explicit equations. This shift from symbolic to statistical modeling marks a turning point in our ability to study, simulate, and control real-world dynamical systems.

2.2 Time Series and Sequence Modeling Methods

Time series [18] and sequence modeling involve learning patterns from temporally ordered data. The goal is to extract information from past observations to predict future values [96], detect meaningful structures [97], or infer underlying dynamics [98]. These models are used across disciplines including finance [99, 100], energy [101, 102], physiology [103], and control systems [95] — wherever events evolve over time.

Broadly, time series modeling tasks can be categorized as follows:

- **Forecasting** [30] : Predicting future values given past data. This includes one step prediction (next value) and multi-step prediction (next sequence of values).
- **Classification** [104]: Identifying the type or class of a time series (e.g., ECG anomaly detection, activity recognition).

- **Anomaly Detection** [105]: Detecting points or sequences that deviate from normal behavior.
- **Dynamical System Learning** [106]: Reconstructing or approximating the latent rules that govern the evolution of a system — often without an explicit equation.
- **Clustering and Similarity Learning** [107]: Grouping time series based on structural similarity.

Each of these tasks places different demands on the underlying models. Forecasting requires temporal memory and smooth extrapolation. Classification requires feature extraction and robustness to time warping. Dynamical system learning — the focus of this thesis — demands models that can uncover structure and encode complex dependencies, often with minimal supervision or domain-specific prior knowledge.

To address these challenges, a variety of modeling approaches have been developed over the decades, from statistical methods and classical machine learning models to deep neural architectures. In this section, we provide a structured overview of these approaches and highlight their strengths and limitations in the context of time series modeling.

2.2.1 Statistical Approaches

Statistical time series models [26] were among the earliest and most widely used methods for analyzing and forecasting sequential data. These models assume that the observed time series is generated by an underlying stochastic process [21, 108] with certain structural properties, such as stationarity [109] or linear dependence [110]. The goal is to estimate the parameters of this process using historical data, and then use the learned structure to make forecasts or understand the system's behavior.

The most prominent models in this family include:

- **Autoregressive (AR) Models** [111]: These assume that the current value of the time series depends linearly on its previous values. An AR model of order p is defined as:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + \epsilon_t \tag{2.3}$$

  where ϵ_t is white noise. The model learns the coefficients ϕ_i that best explain the temporal structure in the data.

- **Moving Average (MA) Models** [112]: These model the current value of the time series as a linear combination of past noise terms:

$$x_t = \mu + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t \tag{2.4}$$

  MA models are useful for capturing short-term error dependencies.

- **Autoregressive Integrated Moving Average (ARIMA)** [113]: This generalizes both AR and MA models and includes differencing to handle non-stationary time series. An ARIMA(p, d, q) model first differences the data d times to make it stationary, and then applies an ARMA(p, q) model. ARIMA is one of the most widely used statistical forecasting tools in practice.

- **Seasonal ARIMA (SARIMA)** [114] and **Exponential Smoothing** [115]: Extensions that incorporate seasonal effects and time-varying trends in the data.

These models typically rely on assumptions such as linearity, stationarity, and Gaussian noise. Stationarity implies that the statistical properties of the time series — such as mean and variance — remain constant over time. Autocovariance is typically analyzed to assess weak stationarity, since a stationary process exhibits time-invariant autocovariance across lags. However, it is not sufficient to determine whether a process is linear or nonlinear.

The autocovariance function of a time series is defined as:

$$\gamma(h) = \mathbb{E}[(x_t - \mu)(x_{t+h} - \mu)] \tag{2.5}$$

A time series $\{x_t\}$ is (weakly) stationary if:

$$\mathbb{E}[x_t] = \mu, \quad \text{Var}[x_t] = \sigma^2, \quad \gamma(h) = \text{Cov}(x_t, x_{t+h}) \tag{2.6}$$

Linearity refers to the assumption that future values are determined by a linear combination of past observations or errors.

Parameter estimation is often performed via maximum likelihood estimation [116] or the method of moments [117]. While powerful in many settings, especially when the underlying process is well-behaved and noise-driven, these models struggle with systems that exhibit strong nonlinearity, long-term memory, or chaotic dynamics [78] — making them less suited for modeling complex dynamical systems.

Nevertheless, they remain highly interpretable, computationally efficient, and often serve as strong baselines or components within hybrid forecasting systems.

2.2.2 Classical Machine Learning Approaches

Classical machine learning models offer a flexible alternative to statistical time series methods. Rather than assuming an explicit generative process (as in ARIMA), these models learn patterns directly from data, often by optimizing predictive accuracy through empirical risk minimization [118].

Empirical Risk Minimization Machine learning models often minimize the empirical risk:

$$\min_f \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i) \tag{2.7}$$

where $\mathcal{L}$ is a loss function (e.g., squared error), and f is the model. Many of these methods treat time series prediction as a supervised regression task [119], using lagged variables or hand-crafted features as input.

They can be loosely categorized into two broad families:

- **Statistical learners**: These include models such as decision trees [120], random forests [121], and ensemble methods [122]. They partition the feature space based on conditional distributions or entropy minimization [123] and are generally robust to noise and capable of modeling non-linear relationships.

- **Function approximators**: These include models like support vector regression (SVR) [124], k-nearest neighbors (k-NN) [125], and shallow neural networks (such as multilayer perceptrons, MLPs [126]). These methods attempt to fit a smooth function that maps historical inputs to future values and are well-suited for continuous-valued forecasting.

Commonly used models in time series forecasting include:

- **Support Vector Regression (SVR)** [124]: A kernel-based method that finds a non-linear mapping between input and output using margin-based optimization. The objective function for SVR is:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \tag{2.8}$$

subject to:

$$y_i - \mathbf{w}^\top\phi(x_i) - b \leq \epsilon + \xi_i, \quad \mathbf{w}^\top\phi(x_i) + b - y_i \leq \epsilon + \xi_i^*, \quad \xi_i, \xi_i^* \geq 0 \tag{2.9}$$

- **Decision Trees and Random Forests** [125]: Tree-based models that hierarchically partition the input space based on criteria such as entropy and information gain. Decision trees use these metrics to select optimal splits at each node, while Random Forests aggregate the outputs of multiple trees to improve robustness and provide estimates of feature importance.

Entropy:

$$H(S) = -\sum_{c=1}^{C} p_c \log_2(p_c) \tag{2.10}$$

Information Gain:

$$IG(S, A) = H(S) - \sum_{v \in \text{values}(A)} \frac{|S_v|}{|S|} H(S_v) \tag{2.11}$$

- **Gradient Boosted Trees [127] (e.g., XGBoost)**: Sequential ensembles of decision trees trained to iteratively minimize a predefined loss function, such as squared error or logistic loss. These models refine predictions by correcting residuals at each stage and have demonstrated strong performance in time series competitions. XGBoost further enhances efficiency and generalization through regularization, making it a popular component in hybrid modeling pipelines.

  Gradient Boosting

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + f_t(x), \quad f_t \in \mathcal{F} \tag{2.12}$$

  XGBoost Loss

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{2.13}$$

  where $\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$ is a regularization term.

- **Multilayer Perceptrons (MLPs)** [126]: Feedforward neural networks trained to learn nonlinear mappings from lagged inputs to future values. These models require careful tuning and are often used with fixed time window inputs.

  MLP with One Hidden Layer

$$\mathbf{y} = f(\mathbf{x}) = \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \tag{2.14}$$

Most classical ML models require feature engineering to encode time [119]: this is typically done by feeding lagged versions of the time series as input features, or constructing domain-specific descriptors (e.g., trends, moving averages, calendar-based features). Unlike RNNs or reservoir models, they lack internal temporal memory [128] — they must "see" past values explicitly as input. Moreover, these models struggle with capturing long-term dependencies and are limited in multi-step forecasting without error compounding or recursive training strategies.

Nonetheless, classical ML models remain popular due to their interpretability, low computational cost, and compatibility with tabular data pipelines. They are especially effective in scenarios with structured inputs, strong seasonality, or when combined with domain expertise.

2.2.3 Neural Network Approaches

Artificial Neural Networks (ANNs) [126] have become central tools for modeling complex functions across a wide range of machine learning tasks, including time series forecasting. Their success is largely attributed to their ability to approximate nonlinear relationships, their parameter efficiency when scaled correctly, and the availability of robust optimization algorithms such as stochastic gradient descent [129].

Inspired by the structure of biological neurons, artificial neurons — first formalized in the McCulloch-Pitts model [130] — compute weighted sums of inputs and apply nonlinear activation functions. When arranged in layers, these units enable the network to learn hierarchical feature representations. Deep neural networks, particularly multilayer perceptrons (MLPs), are capable of learning highly nonlinear mappings between input and output [129]. In time series applications, these networks typically receive lagged observations as input and are trained to predict the next value or a sequence of future values. However, feedforward networks lack the ability to store temporal context inherently; they

must be explicitly provided with time-encoded input features.

To address this limitation, **Recurrent Neural Networks (RNNs)** [34, 35, 36, 37, 38, 39, 131] were developed specifically for modeling sequential and time-dependent data. RNNs incorporate feedback loops, allowing the network to maintain a hidden state vector that evolves over time. At each timestep, the hidden state is updated based on both the current input and the previous hidden state, enabling the network to retain information about past inputs and model temporal dependencies.

The core RNN update can be expressed as:

$$\mathbf{h}_t = \sigma(\mathbf{W}_{\text{in}}\mathbf{x}_t + \mathbf{W}_{\text{rec}}\mathbf{h}_{t-1} + \mathbf{W}_{\text{fb}}\mathbf{y}_{t-1} + \mathbf{b}) \tag{2.15}$$

where $\mathbf{h}_t$ is the hidden state at time t, $\mathbf{x}_t$ is the input, $\mathbf{y}_{t-1}$ is the previous output of the network, $\mathbf{W}_{\text{in}}$, $\mathbf{W}_{\text{rec}}$ and $\mathbf{W}_{\text{fb}}$ are weight matrices, $\mathbf{b}$ is a bias vector, and σ is a nonlinear activation function (typically tanh or ReLU). The output can be computed as:

$$\mathbf{y}_t = \mathbf{W}_{\text{out}}\mathbf{h}_t + \mathbf{c} \tag{2.16}$$

Training RNNs involves computing gradients of a loss function with respect to all weights across multiple timesteps. This is done using **Backpropagation Through Time (BPTT)** [42, 43, 44], an extension of the standard backpropagation algorithm [129] that unfolds the network in time and computes the total loss across a sequence. While BPTT allows RNNs to learn from temporal dependencies, it suffers from major optimization issues. As gradients are propagated over many time steps, they can either diminish exponentially (vanishing gradients) or grow uncontrollably (exploding gradients) [42, 43, 44]. This makes it difficult for standard RNNs to learn long-range dependencies.

To overcome these limitations, specialized RNN architectures have been introduced. The most widely used among them are:

- **Long Short-Term Memory (LSTM)** networks [40]: Introduced by Hochreiter and

Schmidhuber, LSTMs use gated memory cells to retain information over long sequences. They control information flow using input, output, and forget gates, mitigating the vanishing gradient problem.

Long Short-Term Memory (LSTM)

$$
\begin{aligned}
f_t &= \sigma(\mathbf{W}_f x_t + \mathbf{U}_f h_{t-1} + b_f) && \text{(forget gate)} \\
i_t &= \sigma(\mathbf{W}_i x_t + \mathbf{U}_i h_{t-1} + b_i) && \text{(input gate)} \\
\tilde{c}_t &= \tanh(\mathbf{W}_c x_t + \mathbf{U}_c h_{t-1} + b_c) && \text{(cell candidate)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{(cell state update)} \\
o_t &= \sigma(\mathbf{W}_o x_t + \mathbf{U}_o h_{t-1} + b_o) && \text{(output gate)} \\
h_t &= o_t \odot \tanh(c_t) && \text{(hidden state)}
\end{aligned}
\tag{2.17}
$$

- **Gated Recurrent Units (GRUs)** [41]: A simpler alternative to LSTMs, GRUs combine the forget and input gates into a single update gate, and merge the cell and hidden states. They are computationally efficient and often perform comparably to LSTMs.

- **Attention mechanisms** [132]: More recent developments such as self-attention (popularized by the Transformer architecture) allow models to weigh the importance of different time steps when producing an output. While powerful, attention models often require large datasets and computational resources, making them less suited for lightweight, resource-constrained applications.

While these recurrent architectures enable modeling of complex temporal dynamics, they come with increased training complexity, large memory footprints, and sensitivity to hyperparameters [133]. This motivates the exploration of alternative architectures like Reservoir Computing [54, 55], which aim to capture temporal structure without requiring costly gradient-based training of recurrent weights. In the following section, we

introduce the Reservoir Computing paradigm and detail its advantages in resource efficiency, training simplicity [56], and applicability to dynamical system modeling.

2.3 Reservoir Computing Paradigm

Motivated by the challenges encountered in training Recurrent Neural Networks (RNNs), such as vanishing gradients and computational inefficiency, researchers proposed a fundamentally different paradigm for learning from temporal and dynamical systems. Rather than attempting to optimize all internal weights through backpropagation, the **Reservoir Computing** (RC) framework separates the recurrent processing stage from the learning stage. In this setup, the internal dynamics — referred to as the *reservoir* — remain untrained, and only a simple output layer is trained, typically via linear regression.

At the heart of this approach lies a key insight: the behavior of a low-dimensional dynamical system may be embedded in a higher-dimensional state space [134], where its structure becomes linearly separable or more easily approximated. Conversely, a high-dimensional dynamical system might naturally exhibit subdynamics that resemble those of the target system. In both cases, the reservoir acts as a nonlinear temporal feature extractor — mapping the input sequence into a richer dynamical trajectory [53, 55, 56]. The readout layer is then tasked with identifying useful projections of this trajectory to solve a target task, such as prediction or classification.

Reservoirs can be instantiated in a wide variety of substrates. These include analog electronic circuits [64], photonic systems [65], mechanical systems (such as buckets of water or passive robotic joints) [67, 95] and algorithmic structures like random recurrent neural networks [54, 56]. This flexibility makes RC appealing not only for its simplicity, but also for its compatibility with unconventional computing architectures such as neuromorphic [62] and in-memory processors [61].

In this thesis, we focus on one of the most widely used algorithmic implementations of reservoir computing: the **Echo State Network** (ESN) [54]. ESNs are discrete-time recurrent neural networks with randomly initialized internal connections. The reservoir state evolves according to:

$$\mathbf{r}_t = \tanh(\mathbf{W}_{\text{res}}\mathbf{r}_{t-1} + \mathbf{W}_{\text{in}}\mathbf{x}_t + \mathbf{W}_{\text{fb}}\mathbf{y}_{t-1} + \mathbf{b}) \tag{2.18}$$

$$\mathbf{y}_t = \mathbf{W}_{\text{out}}\mathbf{r}_t \tag{2.19}$$

Here, $\mathbf{r}_t$ denotes the reservoir state, $\mathbf{x}_t$ the input, $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{res}}$ are fixed input and recurrent weight matrices, and $\mathbf{W}_{\text{out}}$ is the only trainable matrix in the model. The key design constraint in ESNs is the *Echo State Property* (ESP) [54, 135, 136], which requires that the influence of initial conditions fades over time, ensuring that the reservoir state is asymptotically determined by the input history. This property is typically enforced by scaling the spectral radius of $\mathbf{W}_{\text{res}}$ to a value less than one.

In practice, the spectral radius and sparsity of the reservoir weights are tuned to match the temporal properties of the input [56]. For high-frequency signals, one may desire faster state decay (shorter memory), while for slowly evolving systems, longer memory is beneficial. The reservoir thereby serves as a flexible dynamic buffer of the recent past. Although reservoirs have shown strong performance in dynamic system learning tasks compared to alternative methods, achieving sufficient state variability in vanilla ESN architectures often requires extensive usage of fixed initialization parameters. Researchers worldwide have actively pursued enhancements to Reservoir Computing models. In the context of our work, these advancements can be categorized into four broad areas: (1) improvements in reservoir topology and structural design, (2) operational mechanism optimizations, (3) efficiency-oriented techniques, and (4) our area of contribution — augmentations based on the transformation and enrichment of reservoir states.

2.3.0.1 Reservoir Topology and Structural Design

The reservoir serves as the computational core of a Reservoir Computer, and its architecture plays a critical role in determining performance — particularly because its parameters are typically fixed and not subject to optimization via gradient descent. Consequently, substantial research has focused on designing and analyzing reservoir topologies that enhance representational power and memory dynamics.

Authors of the study [70] systematically explored a variety of ESN topologies, including small-world, scale-free, and hybrid structures, demonstrating how specific connectivity patterns can significantly impact predictive accuracy. Building on this, the *ES2N* model introduced in the work [137] combines nonlinear and orthogonal linear subreservoirs to balance short-term memory retention with nonlinear processing — effectively operating near the edge of chaos for optimal expressivity.

A theoretical study [138] offered an analytical framework for understanding how linear reservoir connectivity influences memory capacity. By reformulating the iterative ESN into a direct expression, their work enables principled tuning of reservoir topology to enhance short-term memory. Similarly, an intriguing study [139] showed that small-world topologies improve both signal propagation and memory performance, surpassing traditional random or regular structures in time series prediction tasks.

In the original study [140], authors proposed the Enhanced Echo-State Restricted Boltzmann Machine (*eERBM*) — a hybrid model that integrates RBM-based feature extraction with the dynamical richness of ESNs. By combining raw input signals with learned latent representations, the eERBM achieves improved accuracy in applications such as network traffic prediction.

Finally, authors of the study [141] proposed a hierarchical ESN composed of multiple reservoirs with staggered input delays. This architecture enhances multi-step forecasting of chaotic systems by capturing temporal dependencies at multiple scales, offering

a structurally elegant solution to long-range prediction.

2.3.0.2 Operational Mechanisms and Dynamical Enhancements

Beyond structural design, another major axis of innovation in Reservoir Computing involves improving how the reservoir operates. Since the reservoir is not typically trained, its dynamics, initialization schemes, and parameters such as the spectral radius must be carefully tuned to maximize performance and stability.

One foundational area of focus is spectral radius optimization, as explored in the practical study [142], which showed optimal spectral radius values to improve learning capacity in ESNs for different tasks, while longer settling times can degrade performance due to echo attenuation. The critical trade-off between nonlinearity and memory was rigorously examined in [68], who proposed hybrid linear-nonlinear reservoir designs that achieve a better balance between short-term memory and nonlinear transformation.

To enrich the temporal processing of recurrent dynamics, authors of the study [143] proposed a delayed reservoir architecture based on delayed differential equations, supported by detailed analysis involving Lyapunov exponents and memory metrics. Meanwhile, in [144] compared parallel versus series reservoir topologies, finding that while parallel structures yield lower prediction errors, series-connected ESNs are more effective for complex feature extraction.

From a more biologically inspired perspective, Authors of the innovative study [37] developed a training scheme that stabilizes chaotic trajectories by converting them into dynamic attractors — boosting robustness even under high-noise conditions. Expanding this idea, authors of the work [145] introduced symbolic dynamics through controlled chaotic switching and softmax-based gating, enabling structured transitions within the reservoir's behavior.

Leaky integrator reservoirs also received theoretical and practical enhancements.

A robust study [146] refined echo state theory for leaky integrator ESNs by incorporating output feedback and stability-aware optimization, leading to improved prediction on tasks like the Mackey-Glass time series.

A modular and biologically plausible novel architecture known as *reBASICS* was proposed in [147], where multiple small RNN modules with stable limit cycles are linearly combined using recursive least squares. Their follow-up work [148] introduced a localized variant of this model, using locally connected neurons to emulate cortical structure, reduce chaotic behavior, and promote self-sustained oscillations.

In the unique study [149] authors introduced chaos-aware validation techniques for Echo State Networks, including Recycle Validation and modified K-Fold schemes that account for dynamical instability. These methods were shown to outperform standard grid search, especially when paired with Bayesian optimization, resulting in more robust generalization in chaotic systems.

Another original work [150] tackled performance optimization through novelty-driven evolutionary search. They defined a behavioral space for ESNs in terms of kernel rank, memory capacity, and generalization ability, and applied a novelty-search genetic algorithm to explore configurations that yield optimal performance on benchmarks like NARMA-10 and chaotic attractor datasets.

Another biologically inspired model implements neurological mechanisms in the study [151] proposed a method that enhances ESN performance by identifying dominant neurons — those whose output weights contribute most significantly to the final prediction — and retraining the reservoir to strengthen signal flow through these critical pathways. This biologically motivated refinement leads to improved error reduction and more efficient internal dynamics.

2.3.0.3 Efficiency-Oriented Innovations and Resource-Constrained Design

As the diversity of Reservoir Computing models grows, so does the need to improve their efficiency for practical applications — especially in scenarios constrained by memory, compute, or power. Recent research has introduced methods to reduce model complexity without sacrificing predictive performance, making ESNs more accessible for real-time and embedded systems.

In the rigorous study [152] authors proposed a reservoir minimization strategy based on the rank of the controllability matrix, which allows the determination of a minimal effective reservoir size needed to model complex dynamical systems. Their approach achieves performance comparable to larger ESNs while significantly reducing computational and memory overhead.

Similarly, authors in the study [153] developed a family of methods that expand the reservoir's effective dimensionality by concatenating internal states across time. This technique circumvents the need to increase neuron count, making it particularly suitable for memory-constrained environments where efficient temporal representation is critical.

From a deployment perspective, the *EchoBay* framework introduced as a well documented work [69] provides an automated hyperparameter tuning pipeline optimized for embedded and edge computing platforms. By minimizing reservoir memory and execution time, EchoBay makes ESNs more viable for low-latency applications beyond cloud infrastructure.

In one of the earliest efforts toward online adaptation, author of the work [154] introduced the use of Recursive Least Squares (RLS) for real-time output weight updates. This approach enabled efficient online learning in nonstationary or streaming data settings, laying the groundwork for adaptive ESNs with low computational overhead.

2.3.0.4 Augmentation-Based Enhancements and Higher-Order Reservoir Design

Recent work has shown that augmenting reservoir states with structured nonlinear transformations can significantly enhance the representational power and efficiency of Reservoir Computing models. In particular, higher-order operations and monomial-based augmentation techniques have emerged as powerful tools for embedding richer dynamics into otherwise fixed reservoir structures.

Early explorations of higher-order interactions appeared in imitation learning and sensorimotor representation tasks [75, 155], where polynomial expansions were used to capture complex mappings between sensory inputs and motor commands. Building on this intuition in our previous work [71], we introduced a family of resource-efficient reservoir architectures that augment the internal state space using nonlinear interactions. The Higher-order Augmented Reservoir Computer (haRC) enriches the state representation by introducing multiplicative monomials between randomly selected reservoir units, thereby increasing expressiveness without expanding the reservoir size. As a novelty we also propose a Selective haRC variant, which uses covariance-based [156] heuristics to choose the most variable and uncorrelated reservoir units to form the monomials. . Both variants preserve the training simplicity of ESNs while improving performance in memory-constrained scenarios.

Following our contribution, two notable studies have deepened the theoretical foundation for monomial-based reservoir augmentation. Authors of the study [73] extended standard ESNs by incorporating Taylor expansions in the readout layer, improving both short- and long-term predictions in chaotic systems. However, their method uses the full set of random monomials, resulting in quadratic or cubic growth in the number of parameters, unlike the selective approach.

In a parallel direction, the study [74] proposed the Generalized Reservoir Computing (GRC) framework, which relaxes the Echo State Property and introduces a time-invariant transformation in the readout. This allows the use of chaotic or nonstationary reservoirs by shifting complexity from the reservoir to the output layer, making the architecture more flexible. While this improves flexibility, it increases computational cost — diverging from efficient, dynamics-preserving approaches present in Reservoir Computing.

Complementing these works, authors of the study [72] proposed the Next Generation Reservoir Computing (NG-RC) paradigm, which eliminates the need for recurrent connections altogether. Instead, it constructs nonlinear feature vectors from delayed input embeddings, effectively mimicking a Volterra series expansion. This leads to state-of-the-art performance with fewer parameters and enhanced interpretability — reinforcing the broader value of nonlinear augmentation in reservoir design. While effective for specific tasks like Lorenz forecasting, it lacks internal state evolution, making it unsuitable for modeling true dynamical systems. Its generalization across data types and tasks remains to be explored.

3. HIGHER-ORDER AUGMENTED RESERVOIRS

3.1 Introduction

In this chapter, we introduce the core contributions presented in our conference paper, which laid the foundation for subsequent enhancements explored in later chapters. The central idea is the augmentation of classical reservoir computing architectures with higher-order interactions to improve memory efficiency and predictive performance. This architecture is termed the Higher-order Augmented Reservoir Computer (haRC).

We describe the baseline Standard Reservoir Computer (Standard Reservoir Computer (sRC)) and then detail the haRC formulation, which enriches the state representation without expanding the reservoir size. Both models are evaluated under two primary task categories: sequence memorization and dynamic system learning.

3.1.1 Standard Reservoir Computer (sRC)

The standard Reservoir Computer (sRC) is a foundational model composed of four main weight matrices: the input matrix $\boldsymbol{W^{in}} \in \mathbb{R}^{n \times r}$, the recurrent reservoir matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$, the output matrix $\boldsymbol{W^{out}} \in \mathbb{R}^{m \times n}$, and the feedback matrix $\boldsymbol{W^{fb}} \in \mathbb{R}^{n \times m}$. The model can be represented abstractly as:

$$\mathcal{E}(\boldsymbol{W^{in}}, \boldsymbol{W}, \boldsymbol{W^{out}}, \boldsymbol{W^{fb}}) \tag{3.1}$$

At each time step t, the reservoir state $\boldsymbol{x_t} \in \mathbb{R}^n$ is computed through a nonlinear update rule:

$$\boldsymbol{h_t} = \boldsymbol{W^{in}}\boldsymbol{u_t} + \boldsymbol{W}\boldsymbol{x_{t-1}} + \boldsymbol{W^{fb}}\hat{\boldsymbol{y}}_{\boldsymbol{t-1}}, \quad \boldsymbol{x_t} = \tanh(\boldsymbol{h_t}) \tag{3.2}$$

where $\boldsymbol{u_t} \in \mathbb{R}^r$ denotes the input at time t. During training, $\boldsymbol{u_t}$ corresponds to the ground truth signal. In the prediction phase (particularly in the dynamics learning task), this input is replaced with the model's previous output, i.e., $\boldsymbol{u_t} = \hat{\boldsymbol{y}}_{\boldsymbol{t-1}}$, making the system fully autoregressive.

Once all T reservoir states are collected into a matrix $\boldsymbol{X} \in \mathbb{R}^{T \times n}$, the output weights $\boldsymbol{W^{out}}$ are learned through ridge regression:

$$\boldsymbol{W^{out}} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{Y} \tag{3.3}$$

where $\boldsymbol{Y} \in \mathbb{R}^{T \times m}$ holds the target outputs, λ is a regularization constant, and $\boldsymbol{I}$ is the identity matrix. During inference, the model output is obtained via a simple linear projection:

$$\hat{\boldsymbol{y}}(\boldsymbol{t}) = \boldsymbol{W^{out}} \boldsymbol{x_t} \tag{3.4}$$

3.1.2 Higher-order Augmented Reservoir Computer (haRC)

To enrich the representational capacity of the reservoir, the haRC model introduces additional nonlinear features by augmenting the reservoir state with generated monomial combinations of recurrent unit activations. These monomials are computed from the outputs of the reservoir units at each time step and effectively serve as higher-order interaction terms. The core idea is to treat the reservoir state as a vector of variables and construct polynomial-like combinations of its components to form additional features. The motivation behind introducing higher-order multiplicative interactions into the reservoir stems from the nonlinear nature of many real-world dynamical systems. In such systems, especially those modeled by higher-order differential equations or exhibiting chaotic attractors, second- or higher-order terms naturally arise [25]. Different variables affecting each other in multiplicative nature can be captured strongly with the monomial augmentation method and used in many studies such as [72, 75, 155]. Another motivation is biological realism. Multiplication is often used to represent dendritic computation in biological circuits as investigated in the study [157] with experiments made on fruit flies. In a theoretical study [158], it was proven that any dichotomy in $\{-1, 1\}^n$ can be represented by a polynomial with a bounded number of monomials, providing mathematical motivation for monomial augmentation and linking it to neural computation in the cerebral cortex.

Formally, let the reservoir state at time t be denoted as $\boldsymbol{h}_t \in \mathbb{R}^N$. We define a monomial augmentation function $\phi_{\mathcal{M}} : \mathbb{R}^N \to \mathbb{R}^{|\mathcal{M}|}$, parameterized by an index set $\mathcal{M} \subset \bigcup_{d=1}^{D_{\max}} \{1, \ldots, N\}^d$, which maps the original state to a set of monomial terms:

$$\boldsymbol{S}_t = \phi_{\mathcal{M}}(\boldsymbol{h}_t) \tag{3.5}$$

Each multi-index $\boldsymbol{i} = (i_1, \ldots, i_d) \in \mathcal{M}$ corresponds to a specific monomial term, such that the component $[\boldsymbol{S}_t]_{\boldsymbol{i}}$ is computed as:

$$[\boldsymbol{S}_t]_{\boldsymbol{i}} = \prod_{k=1}^{d} h_t^{(i_k)} \tag{3.6}$$

The set $\mathcal{M}$ governs which higher-order interactions are included, and can be either randomly sampled or determined via heuristics.

Once the monomial augmentation $\boldsymbol{S}_t$ is computed for each time step, the full design matrix $\boldsymbol{Z} \in \mathbb{R}^{T \times (N+K)}$ is constructed by concatenating the original reservoir states and their corresponding monomial features:

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{h}_0 & \boldsymbol{S}_0 \\ \boldsymbol{h}_1 & \boldsymbol{S}_1 \\ \vdots & \vdots \\ \boldsymbol{h}_T & \boldsymbol{S}_T \end{bmatrix} \tag{3.7}$$

where $K = |\mathcal{M}|$ is the number of augmented features. Ridge Regression is then applied to $\boldsymbol{Z}$ to obtain the readout weight matrix:

$$\boldsymbol{W}^{\text{out}} = \left(\boldsymbol{Z}^\top \boldsymbol{Z} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{Z}^\top \boldsymbol{Y} \tag{3.8}$$

3.1.2.1 Implementation Details

In the implementation used for this thesis, we restrict the augmentation to second-order (quadratic) monomials. Specifically, given a pairwise index set $\mathcal{M} = \{(i, j) \mid 1 \leq i < j \leq N\}$, the augmentation features are computed as:

$$[\boldsymbol{S}_t]_{(i,j)} = \text{sign}(h_t^{(i)} h_t^{(j)}) \cdot \left| h_t^{(i)} h_t^{(j)} \right|^{\frac{1}{2}} \tag{3.9}$$

This transformation smooths the dynamic range of the features while preserving polarity. The resulting feature set is then used to build the regression matrix $\boldsymbol{Z}$ as defined in Equation 3.7, and readout weights are computed using Equation 3.8. The choice of monomial index set $\mathcal{M}$ can follow either a random selection strategy or the heuristic approach detailed in the next section.

3.1.3 Simulation Settings

We evaluate the proposed sparse haRC model against the baseline sparse sRC across two key tasks: (1) *sequential learning* (memorization) and (2) *dynamical system forecasting* (generalization). Each task highlights different aspects of temporal modeling ability.

In the sequential learning task, the model is trained to reproduce a given sequence without receiving external inputs during the prediction phase. The reservoir evolves solely based on its internal dynamics:

$$\boldsymbol{x}_t = \tanh(\boldsymbol{W}\boldsymbol{x}_{t-1}) \tag{3.10}$$

where $\boldsymbol{x}_t$ is the reservoir state at time t and $\boldsymbol{W}$ is the recurrent reservoir weight matrix. The network output is generated by:

$$\hat{\boldsymbol{y}}_t = \boldsymbol{W}^{\text{out}}\boldsymbol{x}_t \tag{3.11}$$

In the dynamical system forecasting task, the network is trained to generate long-term predictions by recursively feeding its own output back as input. During training, we use a teacher forcing mechanism [56] to guide learning:

$$\boldsymbol{x}_t = \tanh(\boldsymbol{W}^{\text{in}}\boldsymbol{y}_{t-1} + \boldsymbol{W}\boldsymbol{x}_{t-1}) \tag{3.12}$$

where $\boldsymbol{y}_{t-1}$ is the ground-truth signal at time t and $\boldsymbol{W}^{\text{in}}$ is the input projection matrix which is fixed as well as the reservoir matrix $\boldsymbol{W}$. The readout weights $\boldsymbol{W}^{\text{out}}$ are then optimized via ridge regression over the collected reservoir states.

During the testing phase, the reservoir is initialized using the final state and output from the training trajectory: $\boldsymbol{x}_T$ and $\hat{\boldsymbol{y}}_T$. The network then recursively generates future predictions by reusing its own outputs as inputs:

$$\boldsymbol{x}_t = \tanh(\boldsymbol{W}^{\text{fb}}\hat{\boldsymbol{y}}_{t-1} + \boldsymbol{W}\boldsymbol{x}_{t-1}) \tag{3.13}$$

where $\boldsymbol{W}^{\text{fb}} = \boldsymbol{W}^{\text{in}}$ as inherited from the training configuration. The prediction at each time step is computed as:

$$\hat{\boldsymbol{y}}_t = \boldsymbol{W}^{\text{out}}\boldsymbol{x}_t \tag{3.14}$$

This recursive setup enables the model to extrapolate future system behavior beyond the training horizon, using only its internal representations and generated outputs.

The number of neurons selected for higher-order augmentation was treated as a design parameter rather than empirically optimized. Our choice was guided by the need to simultaneously (i) demonstrate the expressive advantage of higher-order interactions and (ii) ensure a fair basis for comparison with the standard reservoir computer (s-RC). In particular, if the augmentation ratio were set too low, the resulting architecture would diverge from the s-RC baseline to the point of losing meaningful comparability. We therefore adopted a balanced proportion that highlights the benefits of higher-order augmentation while preserving experimental validity.

3.2 Datasets

We test sRC and haRC with the following datasets on the tasks at 3.1.3.

3.2.1 Butterfly Dataset

This dataset [159] is composed of samples from a 2D curve depicting a butterfly pattern and is generated by taking a range of sampling time points $t = 0, dt, 2dt, \ldots, 1000$ ms and computing $[x(t), y(t)] = C[r(t)\cos(qt), r(t)\sin(qt)]$, where

$r(t)=9-\sin(qt)+2\sin(3qt)+2\sin(5qt)-\sin(7qt)+3\cos(2qt)-2\cos(4qt)$ with $q = \frac{4\pi}{T}$, $T = 1000\,\text{ms}$, $dt = \frac{T}{N}$ (where N is the number of samples) and $C = \max(r(t))^{-1}$.

3.2.1.1 Uniformly Random Time Series Dataset

This dataset consists of values between -1 and 1 sampled from a uniform random distribution to serve as a baseline to investigate the ability of the model in representing unstructured data.

3.2.2 Mackey-Glass System

The Mackey-Glass system [25] is a classic example of a delayed feedback system that exhibits chaotic behavior for specific parameter regimes. It models physiological processes such as blood production and is widely used to benchmark time-series prediction algorithms due to its rich temporal structure and nonlinearity. The system is governed by the following delay differential equation:

$$\frac{dx(t)}{dt} = \beta \frac{x(t-\tau)}{1 + x(t-\tau)^n} - \gamma x(t) \tag{3.15}$$

For our experiments, we use $\beta = 0.2$, $\tau = 17$, $n = 10$, $\gamma = 0.1$, and $x(0) = 0$. The equation is integrated using Euler's method [160] with a time step of $dt = 1$ from $t = 0$ to $t = 516$. After discarding the initial delay period, we obtain 500 time steps of clean chaotic behavior.

3.2.3 Lorenz System

The Lorenz attractor [78], originally developed as a simplified model for atmospheric convection, is one of the most iconic examples of deterministic chaos. It is characterized by a butterfly-shaped trajectory and extreme sensitivity to initial conditions, making it ideal for testing generalization in long-term forecasting tasks. The governing

equations are:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z \end{aligned} \tag{3.16}$$

We set the standard parameters $\sigma = 10$, $\rho = 28$, $\beta = 2.667$, and initial state $(0, 1, 1.5)$. Integration is carried out using the Runge-Kutta [161] method with $dt = 0.01$ up to $t = 5$, resulting in 500 trajectory points.

3.2.4 Resource Calculation

We define *resource* as the total number of floating-point parameters and operations used by the network during training and inference. These include weights in the input, recurrent reservoir, and output layers, as well as the number of arithmetic operations required per time step.

Although this formulation generalizes to both dense and sparse reservoir configurations, the experiments discussed here assume **non-sparse** reservoirs (i.e., no zero-masking is applied; $p = 0$). Sparsity-based extensions are introduced and analyzed in Section 4.1.1.

3.2.4.1 Memory Resource

The total number of parameters in the standard reservoir computer (sRC) is given by:

$$R_{\text{sRC}} = hn_1 + n_1^2(1 - p) + mn_1 \tag{3.17}$$

where h is the input dimension, m is the output dimension, n_1 is the number of reservoir units, and $p \in [0, 1]$ is the sparsity ratio.

For the higher-order augmented reservoir (haRC), an additional term accounts for

the size of the monomial-augmented output layer:

$$R_{\text{haRC}} = hn_2 + n_2^2(1-p) + m(n_2 + \beta n_2^2) \tag{3.18}$$

where n_2 is the number of reservoir units in haRC, and $\beta \in [0, 1]$ represents the proportion of second-order monomials selected for augmentation.

3.2.4.2 Arithmetic Operations per Time Step

The total number of floating-point multiplications for sRC is:

$$M_{\text{sRC}} = hn_1 + n_1^2(1-p) + mn_1 \tag{3.19}$$

The corresponding number of summations is:

$$S_{\text{sRC}} = h(n_1 - 1) + n_1^2(1-p) - n_1 + m(n_1 - 1) \tag{3.20}$$

For haRC, additional multiplications are required for evaluating second-order monomials, which include square root operations that we conservatively count as two multiplications:

$$M_{\text{haRC}} = hn_2 + n_2^2(1-p) + m\left(n_2 + 3\beta n_2^2\right) \tag{3.21}$$

The corresponding number of summation operations is:

$$S_{\text{haRC}} = h(n_2 - 1) + n_2^2(1-p) - n_2 + m(n_2 + \beta n_2^2 - 1) \tag{3.22}$$

These formulas capture both the base matrix operations and the additional computational overhead introduced by higher-order augmentation. Notably, by tuning the parameter—which controls the augmentation ratio—the reservoir size in haRC can be reduced, effectively offsetting the cost of added operations. For a compact comparison of all expressions, see Table 3.1.

Table 3.1: *Memory, multiplication, and summation cost formulas for sRC and haRC. Here, p is the sparsity ratio, β is the augmentation ratio, h is the input dimensions, m is the output dimensions, and n_1, n_2 are the reservoir sizes of sRC and haRC respectively.*

Quantity	**sRC**	**haRC**
Memory Resource (R)	$hn_1 + n_1^2(1-p) + mn_1$	$hn_2 + n_2^2(1-p) + m(n_2 + \beta n_2^2)$
Multiplications (M)	$hn_1 + n_1^2(1-p) + mn_1$	$hn_2 + n_2^2(1-p) + m(n_2 + 3\beta n_2^2)$
Summations (S)	$h(n_1 - 1) + n_1^2(1-p) - n_1 + m(n_1 - 1)$	$h(n_2 - 1) + n_2^2(1-p) - n_2 + m(n_2 + \beta n_2^2 - 1)$

3.2.5 Spectral Radius Optimization

The spectral radius (SR) of the reservoir weight matrix critically influences the dynamical behavior and memory retention capacity of reservoir computing systems. It directly affects the richness of internal state transitions and the ability to represent long temporal dependencies. In particular, SR tuning has a pronounced impact on performance in memorization tasks.

To ensure consistency and fair comparison across models, an automated optimization procedure was employed to identify the optimal SR for each configuration and task. This procedure was applied under fixed resource budgets, meaning the number of trainable weights and reservoir size were held constant, while only the SR value was varied.

For each dataset and task, we scanned a predefined set of SR values and selected the one that maximized a composite fitness objective. This objective function was designed to balance predictive performance, stability, and dynamical scaling, and is defined

as follows:

$$\text{Fitness} = -\theta \cdot \text{Error} - \phi \cdot \text{Variance} - \zeta \cdot \text{SR} \tag{3.23}$$

where:

- Error refers to the average prediction error (e.g., RMSE),
- Variance captures the variability of the error across trials,
- SR is the spectral radius value under evaluation,

and the coefficients $\theta = 0.04$, $\phi = 0.3$, and $\zeta = 0.001$ are empirically chosen scaling factors to weight the terms appropriately. The inclusion of the SR term in the penalty discourages overly large spectral values that might violate the echo state property or induce chaotic internal dynamics, consistent with theoretical considerations in [56].

Algorithm 1 Spectral Radius Selection for Each Task and Model

Require: Spectral radius candidates `SR_Range` = [0.3, 5.0], step = 0.2

Initialize `OptimalSR` $\leftarrow 0$, `MaxFitness` $\leftarrow -\infty$
for each spectral radius value `sr` in `SR_Range` **do**
  for $i = 1$ to 30 **do**
    Run network with spectral radius `sr`
    Store RMSE in `Errors`
  end for
  Compute average and variance of RMSE
  Compute fitness score using the defined function
  if fitness $>$ `MaxFitness` **then**
    Update `OptimalSR` and `MaxFitness`
  end if
end for
return `OptimalSR`, `SR_Results`

This optimization was conducted using reservoir dimensions of 71×71 for haRC

and 350×350 for sRC. These sizes were previously determined via empirical testing as the minimal configurations that avoid underfitting while preserving representative behavior.

The optimal SR values identified through this process were as follows:

- **Memorization Tasks:** SR = 2 for both haRC and sRC
- **Lorenz Forecasting:** SR = 1 for both models
- **Mackey-Glass Forecasting:** SR = 0.88 for haRC, SR = 1 for sRC

These values were consistently observed across multiple runs and were used as fixed SR values in all experiments reported in this chapter.

3.3 Experimental Results

This section presents a comparative evaluation of haRC and sRC models under equivalent memory resource constraints. Performance is measured using Root Mean Square Error (RMSE) across memorization and generalization tasks.

3.3.1 Performance Comparison at Equal Resource Levels

Table 3.2 summarizes the performance of haRC versus sRC models on both memorization and generalization tasks using approximately 10^4 parameters. As shown, haRC consistently achieves significantly lower RMSE across all datasets, highlighting its superior memory efficiency.

Table 3.2: *Comparison of Forecasting Errors.*

Dataset	Parameter Count	haRC Error $\pm$ STD	sRC Error $\pm$ STD
Butterfly Memorization	1×10^4	$10^{-9} \pm 10^{-10}$	$5.8 \times 10^{-1} \pm 10^{-2}$
Random Data Memorization	1×10^4	$7 \times 10^{-9} \pm 10^{-8}$	$5.1 \times 10^{-1} \pm 10^{-2}$
Mackey-Glass Memorization	1×10^4	$7.34 \times 10^{-11} \pm 10^{-8}$	$8.01 \times 10^{-1} \pm 5 \times 10^{-1}$
Lorenz Memorization	1.02×10^4	$1.30 \times 10^{-7} \pm 10^{-8}$	2.4 ± 0.5
Lorenz Generalization	3.1×10^3	$4.5 \times 10^{-4} \pm 2 \times 10^{-4}$	$1.3 \times 10^{-1} \pm 1.1 \times 10^{-1}$
Mackey-Glass Generalization	1×10^4	$1.6 \times 10^{-3} \pm 3 \times 10^{-3}$	$7.6 \times 10^{-2} \pm 10^{-1}$

3.3.2 Sequence Learning Tasks

The memorization experiments are designed to evaluate each model's capacity to store and regenerate temporal patterns using only its internal dynamics. Each model is trained on a sequence of 500 time steps sampled from a given dynamical system. During training, the full sequence is provided as input. For evaluation, the model is reinitialized with the same starting reservoir state and allowed to evolve autonomously—without receiving any external input or feedback. The goal is to reproduce the training sequence as accurately as possible, thereby isolating the model's internal representational capabilities.

Butterfly Dataset. The models were tested on a 2D butterfly trajectory with 500 points. As shown in Figure 3.1a, haRC achieved an RMSE of 10^{-9} using 9×10^3 parameters (reservoir size: 86), whereas sRC required 3×10^5 parameters (reservoir size: 547) to reach the same performance. haRC is thus 30 times more memory efficient.

Random Data On a sequence of 500 uniform random values in (-1, 1), haRC again outperformed sRC. haRC required 9×10^3 parameters (reservoir size: 80) to reach an RMSE of 10^{-9}, compared to sRC needing 3×10^5 parameters (reservoir size: 547), achieving a 33× resource advantage.

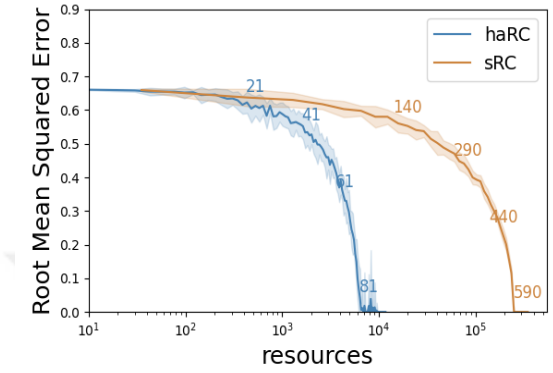
Lorenz System. For Lorenz system on sequential learning task, haRC achieved RMSE of 3×10^{-9} with 8×10^3 parameters (size: 78), while sRC required 3×10^5 parameters (size: 546), demonstrating a 37× efficiency gain. See Figure 3.1b.

Mackey-Glass System. In the Mackey-Glass memorization task, haRC achieved RMSE 10^{-9} with 6×10^3 parameters (size: 74), while sRC required 3×10^5 parameters (size: 547), marking a 30× efficiency gain (Figure 3.1c).

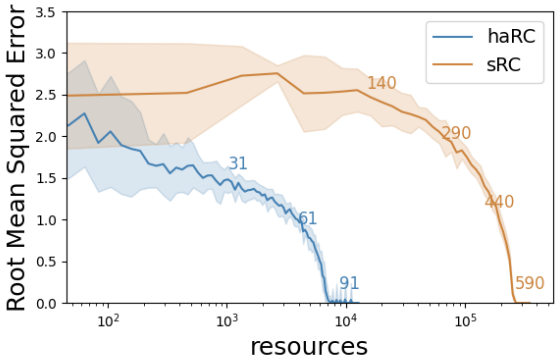
3.3.3 Dynamic System Forecasting Tasks

The forecasting experiments assess the models' ability to generalize and predict future states of a dynamical system from observed data. Each model is trained on a 500-timestep input sequence, where the first 100 time steps are used solely for washing out [56] the influence of the initial reservoir state and are excluded from training. After the readout layer is trained on the remaining 400 time steps, the model is evaluated by recursively predicting the next 50 time steps. During this prediction phase, the model's own outputs are fed back as inputs, creating a fully autoregressive closed-loop system. This setup highlights the stability and extrapolative power of the reservoir under sustained forecasting conditions.

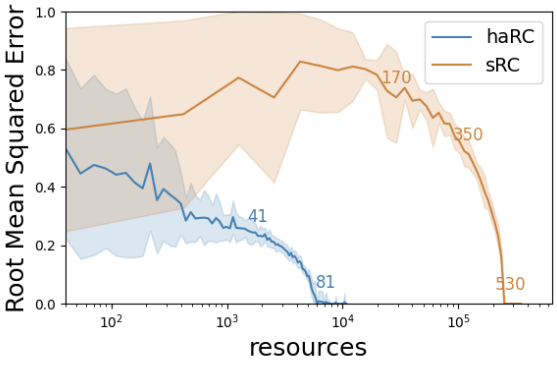
Figure 3.1: *Performance of haRC and sRC on memorization tasks. Each plot shows RMSE vs. parameter count across Butterfly, Lorenz, and Mackey-Glass datasets. haRC consistently achieves low error using significantly fewer parameters. (d) illustrates a direct trajectory forecast.*



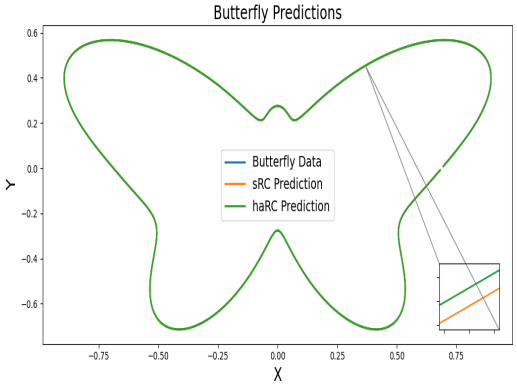
(a) *Butterfly memorization.*



(b) *Lorenz memorization.*



(c) *Mackey-Glass memorization.*



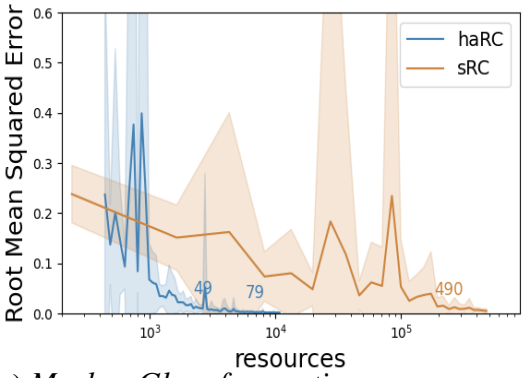
(d) *Butterfly forecast visualization.*

Lorenz System. Models were trained on 500 time steps of Lorenz data and tasked with predicting 50 steps ahead. haRC achieved near-zero RMSE with a 100-unit reservoir (Figure 3.2d), while sRC required 1000 units. The performance-resource curve (Figure 3.2b) shows haRC reaching RMSE 10^{-4} with 3×10^3 parameters versus sRC needing 3×10^4.

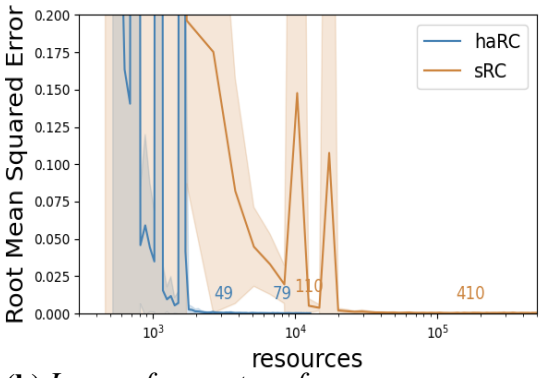
Mackey-Glass System. On the Mackey-Glass dataset, haRC achieved an RMSE of 5.1×10^{-3} with 3.5×10^3 parameters, while sRC needed 4.5×10^5 parameters to match

this accuracy — a 120× efficiency gain (Figure 3.2a and 3.2c).

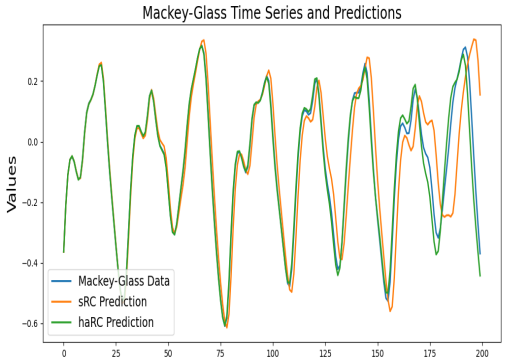
Figure 3.2: *Performance of haRC and sRC on dynamic systems forecasting tasks. Subfigures (a) and (b) show RMSE versus parameter count, highlighting haRC's efficiency on Mackey-Glass and Lorenz datasets. Subfigures (c) and (d) visualize long-term prediction trajectories on the same systems.*



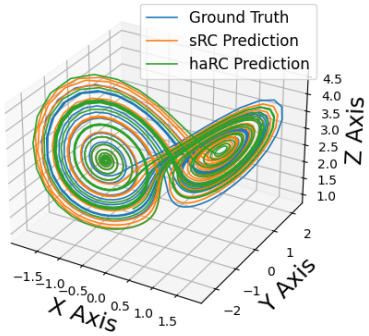
(a) *Mackey-Glass forecasting.*



(b) *Lorenz forecast performance.*



(c) *Mackey-Glass forecast visualization.*



(d) *Lorenz Forecast Visualization.*

3.4 Discussion for Chapter 3

Reservoir Computing (RC) has emerged as a powerful, resource-efficient alternative to gradient-based learning methods, offering strong performance in modeling nonlinear dynamical systems. In Chapter 3, we introduced a Higher-Order Augmented Reservoir Computing (haRC) model that enhances the expressivity of standard reservoirs by

randomly incorporating a subset of second-order multiplicative interactions among reservoir states. This approach demonstrates superior predictive accuracy compared to standard RC, especially under matched memory constraints, highlighting its potential for on-chip, embedded, and resource-limited deployments.

However, the design adopted in Chapter 3 introduces several limitations that warrant further investigation. Notably, the performance drops observed in certain trials—despite variations in random seeds—suggest underlying numerical or architectural sensitivities that remain unexplored. On top of this following points are potential improvements. First, the higher-order monomials are selected entirely at random, without consideration of their contribution to learning, which may lead to suboptimal performance. Second, sparse connectivity—an established approach for improving efficiency and generalization in RC architectures—was not applied to the reservoir itself, limiting insight into how sparsity interacts with higher-order augmentation. Third, the use of multiplicative states may inherently amplify the effects of input or state noise, raising questions about the noise robustness of the haRC model, which remains to be systematically evaluated.

Chapter 4 addresses these open questions through a more rigorous and structured analysis. First, we introduce a method for selectively constructing higher-order reservoir states based on internal signal statistics, moving beyond the random augmentation strategy of the previous chapter. We then evaluate both haRC and sRC under controlled noise injection, testing the robustness of their internal dynamics. To further explore architectural efficiency, we incorporate sparse reservoir connectivity, investigating its interaction with high-order augmentation. Finally, we perform a comparative analysis of haRC and sRC under equal memory budgets and equal readout sizes by varying their reservoir sizes and tuning their sparsity levels, ensuring fair conditions in terms of tunable parameters. Together, these experiments aim to deepen our understanding of haRC's capabilities and practical value across both constrained and high-dimensional learning tasks.

4. COMPREHENSIVE RC IMPROVEMENTS THROUGH UNIT SELECTION AND RESOURCE-EFFICIENT DESIGN

Building upon the findings in Chapter 3— which demonstrated that randomly augmenting reservoir states with a sparse subset of pairwise multiplicative interactions of reservoir units can enhance performance under low memory budgets — this chapter presents the extended contributions of the thesis beyond the initial conference work, forming the foundation of the journal-level study. While the preliminary haRC model showed promise, several key limitations remain unresolved, including noise sensitivity, the lack of principled augmentation strategies, and the absence of sparsity in reservoir connectivity.

To address these gaps, Chapter 4 introduces three core advancements aimed at improving efficiency, robustness, and structural adaptability. First, we incorporate reservoir sparsity to reduce computational and memory costs, leveraging known benefits of sparse connectivity in echo state networks. Second, we systematically evaluate the noise robustness of both standard and higher-order models by injecting controlled additive noise into the reservoir input, enabling a detailed comparison of their generalization stability. Finally, we propose the Selective haRC, a refined model that replaces random augmentation with a heuristic selection mechanism based on covariance analysis of internal dynamics. This selective approach identifies and augments only the most informative reservoir unit interactions, further reducing overhead while preserving expressive power.

Together, these extensions provide a comprehensive analysis of higher-order augmentation strategies under varied resource constraints, offering practical insights into the design of robust and efficient reservoir computing systems for real-world applications.

4.1 Methods

We will start by defining new processes of creating sparse RC's then we will include noisy reservoirs to understand their robustness to noise and finalize our introduction with the Selective haRC model with statistical heuristics.

4.1.1 Sparse Reservoir Implementation

Reservoir computing models can benefit substantially from sparse connectivity within the reservoir matrix, as sparse topologies are known to reduce memory consumption and computational overhead while sometimes improving performance [53]. In this study, we systematically compare both the standard (sRC) and higher-order augmented (haRC) reservoir computers under sparse configurations.

To introduce sparsity, we apply a controlled masking procedure to the recurrent weight matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ of the reservoir. Given a sparsity target $s \in [0, 1]$, indicating the fraction of weights to be set to zero, we uniformly sample sn^2 indices without replacement from the flattened matrix and zero out the corresponding entries. The modified matrix is then reshaped back to its original dimensions.

This process yields a sparsified matrix $\mathbf{W}_{\text{sparse}}$, defined as:

$$\mathbf{W}_{\text{sparse}} = \text{SparseMask}(\mathbf{W}, s) \tag{4.1}$$

where SparseMask$(\cdot)$ represents the masking operation described above. Since zeroing out elements alters the spectral properties of the matrix, particularly its spectral radius, we perform a post-sparsification normalization step to restore the intended spectral radius ρ. Specifically, we rescale $\mathbf{W}_{\text{sparse}}$ by:

$$\mathbf{W}_{\text{sparse}} \leftarrow \frac{\rho}{\lambda_{\max}} \cdot \mathbf{W}_{\text{sparse}} \tag{4.2}$$

where $\lambda_{\max}$ is the largest absolute eigenvalue of the sparsified matrix. This normalization ensures the dynamical regime of the reservoir remains controlled and consistent across

configurations. The same sparsification and rescaling procedure is applied to both sRC and haRC architectures to maintain a fair basis for comparison under equivalent resource constraints.

4.1.2 Reservoir Noise Implementation

For investigating noise robustness of haRC models we inject Gaussian noise during the reservoir operation as real-life systems usually contain Gaussian noise [162] in their operations or measurements. A signal from a zero mean Gaussian noise is represented by $\eta_t \sim \mathcal{N}(0, \sigma^2)$ where η_t is the noise added at time t and σ^2 indicates the variance of the distribution. The reservoir state dynamics with added noise can then be expressed as:

$$\mathbf{x}_t = \tanh(\mathbf{W}_{\text{in}}\mathbf{u}_t + \mathbf{W}(\mathbf{x}_{t-1} + \eta_t)) \tag{4.3}$$

where $\mathbf{x}_t$ represents the reservoir state at time t, $\mathbf{W}_{\text{in}}$ is the input weight matrix, $\mathbf{u}_t$ denotes the input at time t, and $\mathbf{W}$ corresponds to the reservoir weight matrix. To mitigate the effects of noise, Linear Regression with Ridge Regularization is applied during the training of the output layer: $\mathbf{W}_{\text{out}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{Y}$ where $\mathbf{X}$ is the matrix of reservoir states, $\mathbf{Y}$ denotes the target output matrix, λ is the Ridge regularization parameter, and $\mathbf{I}$ represents the identity matrix. The performance of RCs is evaluated by varying the noise variance σ^2 and the Ridge parameter λ.

4.1.3 Selective Higher-order Augmented Reservoir Computer (Selective haRC)

The selective haRC architecture introduces a systematic and task-agnostic approach to reservoir unit selection for monomial generation in order to enhance noise robustness and learning efficiency. While the standard haRC uses randomly sampled monomial terms, selective haRC ranks and selects features based on statistical properties of the reservoir dynamics during training. Although the following formulation focuses

on quadratic augmentation, the same selection strategy can be extended to higher-order monomials.

Let $\boldsymbol{h}_t \in \mathbb{R}^N$ denote the reservoir state at time t. Collecting these states across T time steps yields the state matrix $\boldsymbol{H} \in \mathbb{R}^{T \times N}$. The selective augmentation mechanism proceeds in the following steps:

4.1.3.1 Step 1: High-Variance Unit Selection

For each reservoir unit $i \in \{1, \ldots, N\}$, compute the sample variance over the training trajectory:

$$\text{Var}(h^{(i)}) = \frac{1}{T} \sum_{t=1}^{T} \left(h_t^{(i)} - \bar{h}^{(i)} \right)^2, \quad \bar{h}^{(i)} = \frac{1}{T} \sum_{t=1}^{T} h_t^{(i)} \tag{4.4}$$

Then, select the indices of the top-k most variant units to form the set $\mathcal{V} \subset \{1, \ldots, N\}$.

4.1.3.2 Step 2: Correlation Scoring and Pair Ranking

For each $i \in \mathcal{V}$, compute the Pearson correlation between $h^{(i)}$ and every other unit $h^{(j)}$:

$$\text{Corr}(h^{(i)}, h^{(j)}) = \frac{\text{Cov}(h^{(i)}, h^{(j)})}{\sigma(h^{(i)})\sigma(h^{(j)})} \tag{4.5}$$

A pairwise score is assigned to each (i, j) based on a linear combination of dissimilarity and variance:

$$\text{Score}(i, j) = \alpha \left(1 - |\text{Corr}(h^{(i)}, h^{(j)})|\right) + (1 - \alpha)\, \text{Var}(h^{(j)}) \tag{4.6}$$

where $\alpha \in [0, 1]$ balances preference for low-correlation (diversity) and high individual variance. The top-p scoring pairs are selected and compiled into the monomial index set $\mathcal{M} \subset \{(i, j)\}$.

Without relying on target task information, this procedure deterministically generates the augmentation set based solely on reservoir activations. Two hyperparameters govern the selection process: k, which defines the number of top-variance units selected

from a reservoir of size N (i.e., kN units), and h, which specifies the number of top pairwise interactions retained per selected unit (i.e., hN pairs in total). As a result, the overall monomial coverage ratio is approximately $hk \approx \beta$, where β controls the density of augmentation. This design makes the algorithm lightweight, repeatable, and inherently task-agnostic.

4.1.3.3 Step 3: Reservoir State Augmentation

At each time step t, the monomial augmentation vector $\boldsymbol{S}_t = \phi_{\mathcal{M}}(\boldsymbol{h}_t)$ is computed using the selected index set $\mathcal{M}$. These features are appended to the original reservoir states to form the design matrix $\boldsymbol{Z} \in \mathbb{R}^{T\times(N+|\mathcal{M}|)}$ according to the formula in 3.7.

4.1.3.4 Step 4: Readout Layer Training

The readout weights are computed using Ridge Regression, consistent with the standard haRC approach:

$$\boldsymbol{W}^{\text{out}} = \left(\boldsymbol{Z}^\top \boldsymbol{Z} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{Z}^\top \boldsymbol{Y} \tag{4.7}$$

The overall model structure remains identical to the random haRC, with the key distinction being the construction of the augmentation set $\mathcal{M}$.

4.2 Datasets

This chapter extends the experimental evaluation by reusing the Lorenz and Mackey-Glass systems (previously detailed in Chapter 3.2) and incorporating additional dynamical systems to assess robustness and sparsity effects.

4.2.1 Revisited Datasets

- **Lorenz System**: A chaotic 3D system used for generalization and memorization tasks. See section 3.2 for full description.

- **Mackey-Glass Time Series**: A 1D delay-differential system exhibiting chaotic behavior. See section 3.2 for full description.

4.2.2 Newly Introduced Datasets

- **Rössler Attractor** Introduced by Otto Rössler, this system [163] is a simpler continuous-time chaotic system compared to the Lorenz attractor but still produces a complex spiral attractor in three dimensions. It is especially useful for studying continuous, oscillatory chaos in a low-dimensional system. The system equations are:

$$\begin{aligned} \frac{dx}{dt} &= -y - z, \\ \frac{dy}{dt} &= x + ay, \\ \frac{dz}{dt} &= b + z(x - c) \end{aligned} \tag{4.8}$$

  With parameters $a = 0.2$, $b = 0.2$, $c = 5.7$, and initial condition $(1, 1, 1)$, the system is integrated using $dt = 0.05$ to yield a dataset with intricate periodic and chaotic transitions.

- **Henon Map**: The Hénon map [164] is a two-dimensional discrete-time system that demonstrates strange attractor behavior. It was initially proposed to model the motion of stars in galaxies and has become a cornerstone example in chaos theory. The map is defined as:

$$\begin{aligned} x_{n+1} &= 1 - ax_n^2 + y_n, \\ y_{n+1} &= bx_n \end{aligned} \tag{4.9}$$

  We use $a = 1.4$, $b = 0.3$, and initial condition $(0.1, 0.3)$, generating a chaotic trajectory that explores a folded and fractal-like attractor. Its simplicity and chaotic richness make it valuable for discrete-time modeling experiments.

- **Chua Circuit**: Chua's circuit [165] is one of the simplest electronic circuits capable of producing chaos, combining a nonlinear resistor (the Chua diode) with linear

capacitors and inductors. It provides a concrete example of chaos arising from deterministic nonlinear circuits. The system is modeled by:

$$\begin{aligned} \frac{dx}{dt} &= \alpha(y - x - f(x)), \\ \frac{dy}{dt} &= x - y + z, \\ \frac{dz}{dt} &= -\beta y \end{aligned} \tag{4.10}$$

The nonlinear function $f(x)$ is defined as:

$$f(x) = m_1 x + \frac{1}{2}(m_0 - m_1)(|x + 1| - |x - 1|) \tag{4.11}$$

We use parameters $\alpha = 15.6$, $\beta = 28$, $m_0 = -1.143$, $m_1 = -0.714$, and an initial condition $(0.1, 0, 0)$. The circuit is simulated with $dt = 0.05$ to produce a chaotic yet structured dataset.

- **Ikeda Map**: The Ikeda map [166] simulates the behavior of light within an optical cavity, where nonlinear and delay effects introduce chaotic behavior. It has been used in both physics and nonlinear optics to model multi-valued dynamical states. It is governed by:

$$\begin{aligned} x_{n+1} &= 1 + u(x_n \cos(t_n) - y_n \sin(t_n)), \\ y_{n+1} &= u(x_n \sin(t_n) + y_n \cos(t_n)), \\ t_n &= \kappa - \frac{\lambda}{1 + x_n^2 + y_n^2} \end{aligned} \tag{4.12}$$

We use $u = 1.00001$, $\kappa = 0.4$, $\lambda = 6.0$, and initial point $(0.1, 0.1)$ to generate 1000 steps of chaotic oscillations. Its structure exhibits sharp nonlinearity and complex folding of trajectories, offering a distinct challenge for prediction models.

- **ETHUSDT High-Frequency Trading Data**: To evaluate performance on real-world financial data, we introduce a dataset derived from Ethereum-to-Tether (ETH/USDT)

trading activity on the Binance exchange, collected between 2019 and 2024 [167]. This dataset contains high-frequency Open-High-Low-Close-Volume (OHLCV) entries recorded at 3-minute intervals, reflecting market microstructure dynamics. For use in our experiments, we focus on the core numerical features: `open`, `high`, `low`, `close`, and `volume`. The raw data is preprocessed using a Savitzky–Golay [168] filter with a window length of 20 and polynomial order of 2 to smooth out microstructural noise while preserving dynamic trends. To enhance temporal modeling performance and reduce computational load, we downsample the sequence by a factor of 20, effectively yielding a coarser but more stable signal. The resulting data is normalized using MinMax scaling and formatted to match the input structure used across our synthetic chaotic systems. This setup allows us to directly test our reservoir computing methods on volatile, nonlinear real-world sequences.

4.3 Experiments and Results

This chapter presents the empirical evaluation of the sRC and the haRC. The models are tested across several axes of performance, including:

- Robustness to internal Gaussian noise,
- The effect of varying reservoir sparsity levels,
- Comparison under equal learnable parameter counts,
- Evaluation of proposed selective augmentation technique.

All experiments are conducted on the Mackey-Glass dataset using both dense and highly sparse reservoir configurations. When applicable, haRC results include both randomly selected and statistically selected state pair augmentations.

4.3.1 Noise Robustness Evaluation

To evaluate the robustness of reservoir computing models under noisy conditions, we introduced Gaussian noise into the reservoir states during operation, as described in Equation 4.3. Following standard practice [162], we varied the standard deviation σ to simulate increasing noise levels and assess system degradation.

A critical component of this evaluation is the selection of the regularization parameter λ used in Ridge Regression. To ensure consistent and fair comparisons, we performed a grid search over $\lambda \in \{10^{-8}, 10^{-7}, \ldots, 1\}$, training each model configuration over 30 independent trials for each λ value and each noise level. The objective was to identify a value that provided stable performance across model types and noise regimes.

The grid search revealed that the relative performance ranking of the models remained stable across the entire range of λ. As expected, larger values of λ reduced output variance by regularizing the weights, but also biased predictions toward the mean, leading to underfitting. Conversely, very small λ values occasionally led to overfitting, which proved particularly detrimental under recursive forecasting due to the potential for error accumulation and signal divergence.

We selected $\lambda = 10^{-4}$ as the default regularization value for all noise robustness experiments, as it consistently yielded a favorable trade-off between forecast accuracy and output variance. The procedure used for selecting λ is summarized in Algorithm 2.

Algorithm 2 Ridge Parameter Selection Procedure

Require: Reservoir Model, Data $\mathcal{D}$, Noise levels σ_i, Grid $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$

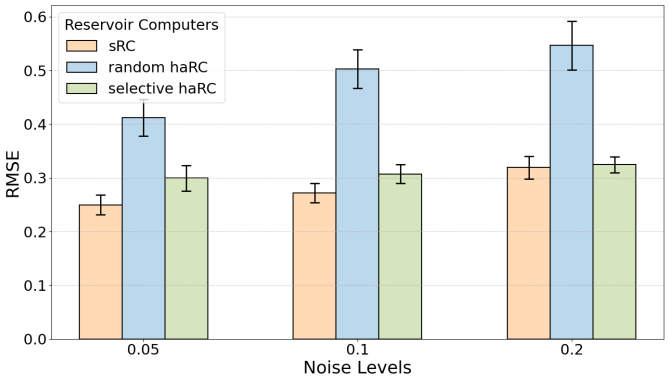
for each $\lambda \in \Lambda$ **do**
  for each noise level σ_i **do**
    for $k = 1$ to 30 **do**
      Add Gaussian noise $\eta_t \sim \mathcal{N}(0, \sigma_i^2)$ to reservoir state
      Train model using Ridge Regression with λ
      Evaluate RMSE and output variance
    end for
  end for
end for
Select λ^* that minimizes a composite trade-off between RMSE and prediction variance

Results Figure 4.1 shows RMSE under increasing Gaussian noise levels ($\sigma = 0.05$, 0.1, 0.2) for three models: standard sRC, random haRC, and selective haRC. All models are tested with matched output dimensions. Specifically, sRC uses 600 reservoir units, while haRC uses 72 reservoir units augmented with 600 output features. The total parameter count is $\approx 360,000$ for sRC and $\approx 6,000$ for haRC in the dense setting, and $\approx 36,000$ vs. ≈ 1000 in the 90% sparse setting.

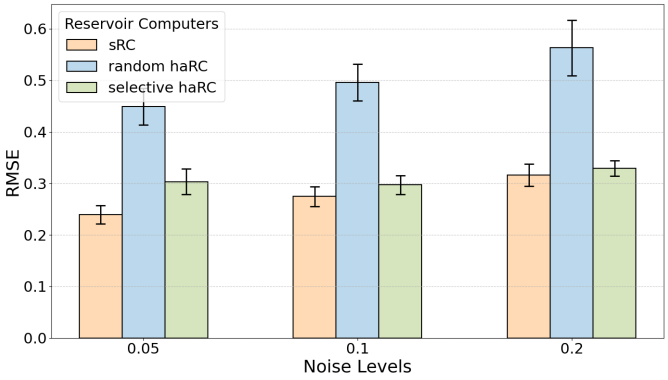
4.1a presents results in the dense configuration, while 4.1b displays results under 90% sparsity. In both graphs bars show the RMSE and error lines show the Standard Error of the Mean. In both settings, sRC (orange) shows gradual performance degradation with noise, maintaining its role as a robust baseline. Random haRC (blue) is more sensitive to perturbations, especially under sparsity, where its RMSE and variance increase more sharply. In contrast, selective haRC (green) demonstrates remarkable robustness: its performance closely matches or exceeds that of sRC, particularly in sparse conditions. It consistently outperforms the random haRC variant in both RMSE and variance. One plausible explanation is that selective haRC prioritizes high-variance reservoir units when

constructing augmented features. These high-variance states exhibit stronger internal dynamics that may suppress the relative impact of additive noise. In essence, perturbations are less influential when compared to the natural scale of the reservoir fluctuations.

Figure 4.1: *Noise robustness comparison on the Mackey-Glass dataset. haRC uses 72 reservoir units and 600 output features, while sRC uses 600 reservoir and output units.*



(a) *Dense reservoir (0% sparsity).*



(b) *Sparse reservoir (90% sparsity).*

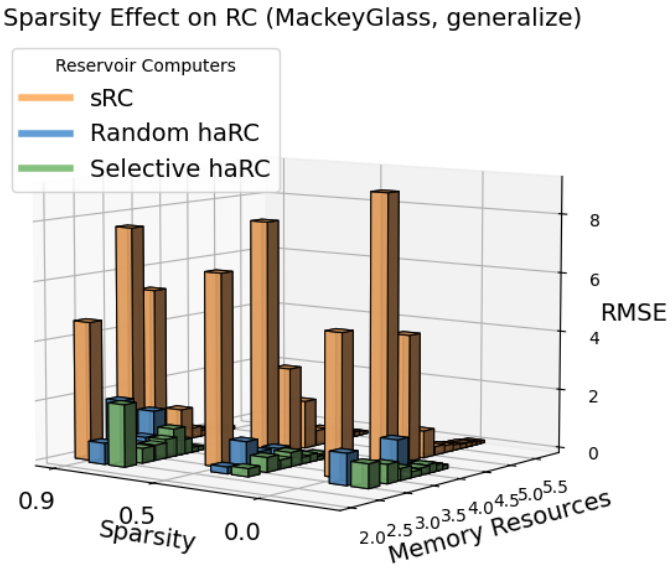
In summary, while higher-order augmentation tends to increase the system's sensitivity to noise relative to standard reservoir architectures, this drawback can be effectively mitigated through a principled, task-independent selection of monomials—such as the approach employed in selective haRC. This strategy enables the benefits of augmentation while maintaining robustness under noise.

4.3.2 Sparsity Analysis

To evaluate the impact of sparsity on model performance, we conducted systematic experiments comparing sparse sRC and haRC architectures on a dynamic system forecasting task using the Mackey-Glass dataset. The reservoir size for sRC was varied between 30 and 600 in increments of 50 units, while haRC reservoir sizes ranged from 10 to 100 with 10-unit steps. Each configuration was tested under three sparsity levels: 0% (fully dense), 50%, and 90% (highly sparse), representing the proportion of zero-valued connections in the reservoir weight matrix.

For every combination of model type, sparsity level, and reservoir size, we conducted 30 independent trials and measured performance using Root Mean Squared Error (RMSE). In Figure 4.2, each color-coded bar represents the average RMSE over 30 runs, plotted against the corresponding memory resource consumption. Memory resources were computed using the analytical expressions presented in Table 3.1, which take into account both the reservoir size and the sparsity ratio. Results indicate that sparse sRC architectures (in orange) can match the performance of their dense counterparts, suggesting that high sparsity does not inherently degrade predictive quality. In particular, the 90% sparse sRC variant offers good accuracy while requiring less memory resources making it a good baseline. Comparatively, both random (blue) and selective (green) haRC models outperform sRC across all sparsity levels, especially in high-sparsity regimes. Notably, at the 90% sparsity level, haRC consistently achieves lower RMSE than sRC for similar resource budgets, demonstrating its advantage in low-memory environments. Random and selective variants of haRC show comparable performance, indicating that the benefit derives primarily from the higher-order augmentation itself. A detailed breakdown of individual experiments—including reservoir sizes, RMSE values, sparsity levels, and corresponding memory resources—can be found in Table 4.1.

Figure 4.2: *Graph demonstrates how RC's are affected by sparsity across different memory resources. Each number in Memory Resources axis is in* 10^x *scale so 2 on the axis means,* 10^2*. Sparsity levels describe how sparse the reservoir is. 0.9 means reservoir is 90% sparse. haRC type Reservoir Computers perform more efficiently, making them better candidate for tasks that has a memory constraint.*



4.3.3 Equal Readout and Equal Memory Comparison

The fixed parameters of a reservoir system define the dimensionality of the feature space into which inputs are non-linearly projected, thereby influencing the system's representational capacity. In parallel, the learnable parameters—primarily the weights in the readout layer—govern the flexibility of the model during the regression phase by determining the available degrees of freedom. Previous experiments varied only one of these factors at a time. However, a more comprehensive analysis requires simultaneously controlling and balancing both the fixed and learnable parameter counts to enable

Table 4.1: *Sparsity experiment description table where memory resource, n is the reservoir size and rmse is the root mean square error. Lev.x corresponds to the level of memory resource used during the experiment data point corresponds to and the table refers to the Figure 4.2.*

	Resource Levels							
Model	**Lev. 1**	**Lev. 2**	**Lev. 3**	**Lev. 4**	**Lev. 5**	**Lev. 6**	**Lev. 7**	**Lev. 8**
Sparsity = 0.0								
sRC	960 (n=30) rmse=4.77	6560 (n=80) rmse=9.08	17160 (n=130) rmse=4.24	32760 (n=180) rmse=0.904	53360 (n=230) rmse=0.277	78960 (n=280) rmse=0.229	109560 (n=330) rmse=0.172	145160 (n=380) rmse=0.139
random haRC	130 (n=10) rmse=0.195	480 (n=20) rmse=1.05	1050 (n=30) rmse=0.197	1840 (n=40) rmse=0.184	2850 (n=50) rmse=0.204	4080 (n=60) rmse=1.08	5530 (n=70) rmse=0.239	7200 (n=80) rmse=0.107
selective haRC	130 (n=10) rmse=0.233	480 (n=20) rmse=0.802	1050 (n=30) rmse=0.632	1840 (n=40) rmse=0.39	2850 (n=50) rmse=0.187	4080 (n=60) rmse=0.235	5530 (n=70) rmse=0.281	7200 (n=80) rmse=0.209
Sparsity = 0.5								
sRC	510 (n=30) rmse=6.45	3360 (n=80) rmse=7.92	8710 (n=130) rmse=2.82	16560 (n=180) rmse=1.6	26910 (n=230) rmse=0.5	39760 (n=280) rmse=0.476	55110 (n=330) rmse=0.237	72960 (n=380) rmse=0.159
random haRC	80 (n=10) rmse=0.222	280 (n=20) rmse=0.222	600 (n=30) rmse=0.918	1040 (n=40) rmse=0.484	1600 (n=50) rmse=0.169	2280 (n=60) rmse=0.438	3080 (n=70) rmse=0.226	4000 (n=80) rmse=0.174
selective haRC	80 (n=10) rmse=0.209	280 (n=20) rmse=0.268	600 (n=30) rmse=0.489	1040 (n=40) rmse=0.302	1600 (n=50) rmse=0.466	2280 (n=60) rmse=0.212	3080 (n=70) rmse=0.242	4000 (n=80) rmse=0.112
Sparsity = 0.9								
sRC	150 (n=30) rmse=4.64	799 (n=80) rmse=7.6	1949 (n=130) rmse=5.36	3599 (n=180) rmse=0.293	5749 (n=230) rmse=1.06	8399 (n=280) rmse=0.339	11549 (n=330) rmse=0.186	15199 (n=380) rmse=0.164
random haRC	40 (n=10) rmse=0.258	120 (n=20) rmse=0.704	240 (n=30) rmse=2	400 (n=40) rmse=0.445	600 (n=50) rmse=0.603	840 (n=60) rmse=1.44	1120 (n=70) rmse=0.32	1440 (n=80) rmse=0.194
selective haRC	40 (n=10) rmse=0.344	120 (n=20) rmse=2.08	240 (n=30) rmse=0.493	400 (n=40) rmse=0.511	600 (n=50) rmse=0.621	840 (n=60) rmse=0.916	1120 (n=70) rmse=0.48	1440 (n=80) rmse=0.154

fair performance comparisons. This can be achieved by dynamically adjusting sparsity. Specifically, we begin with a predefined reservoir size and sparsity level for the haRC architecture, and then compute the appropriate reservoir size and sparsity level for the sRC model to ensure parity in both constant and trainable resource usage across architectures.

To avoid biasing our findings toward a specific reservoir size, we conduct experiments using a range of haRC reservoir sizes from 20 to 100, increasing in steps of 10. These experiments encompass both sequence learning and dynamical system forecasting

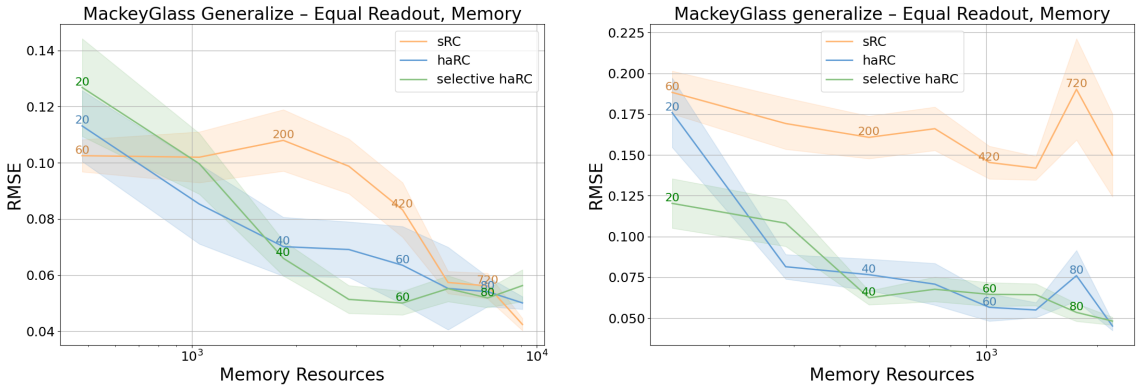
tasks, as described in Section 3.1.3, and are applied to the datasets introduced in Section 4.2. While comprehensive results are available in tabulated form in Appendix B.1, this section focuses on the Mackey-Glass and Chua Circuit attractors for illustrative purposes. We begin by comparing the forecasting performance of the different models.

4.3.3.1 Forecasting Tasks

The dynamical system forecasting experiments are configured with a 500-timestep training sequence, where the initial 100 time steps are reserved for warming up the reservoir [56]. These early states are excluded from the readout training phase to allow the internal dynamics to stabilize and to eliminate the effects of arbitrary initializations. After training, each model is evaluated by recursively predicting the subsequent 50 time steps, using the final state of the reservoir from the training phase as the initial state for forecasting.

To ensure fair comparisons, all models are configured to use the same number of fixed and learnable parameters. This is achieved by adjusting the reservoir size and sparsity level of the sRC model to match the total parameter budget (including readout weights) of the haRC configuration. Root Mean Squared Error (RMSE) is used as the primary evaluation metric, and performance variability is captured through shading that denotes the standard error across 30 independent trials.

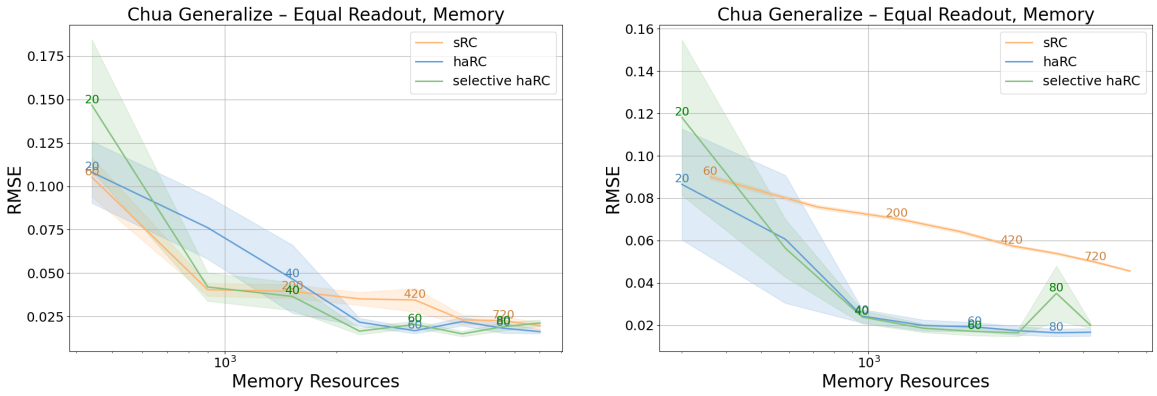
Figure 4.3: ***Dynamics Learning task.*** *These plots demonstrate the performance of RC's on forecasting task using the Mackey-Glass system. Both experiments have equal memory and equal readout parameter where reservoir sizes are annotated on the graph. haRC maintains competitive generalization accuracy requiring lower memory resources compared to sRC in both sparsity levels.*



(a) *non-sparse haRC.* **(b)** *sparse haRC (85%).*

Figure 4.3 illustrates model performance on the Mackey-Glass forecasting task. The horizontal axis represents total memory resources—i.e., the combined parameter count associated with each reservoir size shown. While haRC is evaluated with a fixed sparsity level, the sRC configurations vary in sparsity to satisfy the parameter-matching condition. The specific sparsity levels used are listed in Appendix Tables B.5 and B.1. Results show that haRC consistently outperforms sRC under resource-constrained conditions. In order to match haRC in memory efficiency, sRC must adopt extremely sparse connectivity, which in turn degrades its reservoir dynamics and forecasting accuracy.

Figure 4.4: ***Dynamics Learning task.****These plots demonstrate the performance of RC's on forecasting task using the Chua Attractor system. Both experiments have equal memory and equal readout parameter where reservoir sizes are annotated on the graph. haRC maintains competitive generalization accuracy requiring lower memory resources compared to sRC in both sparsity levels.*



(a) *sparse haRC (50%).* **(b)** *sparse haRC (85%).*

A similar trend is observed in experiments with the Chua Circuit. Figure 4.4 demonstrates that haRC, even with a 50% sparsity level, performs comparably to sRC. However, when the haRC sparsity is increased to 85%, the equivalent sRC configuration becomes a reservoir with zeros. This occurs because the 3-dimensional Chua system requires more input and output weights, leaving no parameter budget for internal recurrent connections in sRC. As a result, sRC behaves more like just an autoregressive model without a reservoir.

Despite the low number of parameters enforced, haRC continues to demonstrate effective forecasting performance under matched resource budgets with sRC. This can be visually verified in Figure A.6, which shows the generated trajectories of the Chua attractor. In this figure, sRC (orange curve) uses a 900-unit reservoir, while both random haRC (blue) and selective haRC (green) use only 90 units. Yet, all models are matched in terms of total constant and trainable parameters. In Figure A.6a, both haRC and sRC

successfully replicate the dynamics of the Chua attractor under moderate sparsity (50% in haRC, 99% in sRC), with trajectories closely following the ground truth (black curve). However, in Figure A.6b, the increased sparsity in haRC (85%) forces the sRC reservoir to degenerate into a reservoir with zeros. In this case, haRC still manages to learn the system dynamics, while sRC fails to move beyond its initial trajectory segment. For additional details and significance [1] of the results on forecasting experiments, see Appendix Tables B.5, B.3, B.1 for Mackey-Glass, and Tables B.11, B.13 for the Chua Circuit.

To extend the relevance of our findings to real-world scenarios, we incorporated a high-frequency financial time series (HFT) dataset as an additional benchmark. This dataset reflects the challenges of modeling noisy, high-dimensional, and highly dynamic systems found in financial applications. The performance of haRC on this dataset was consistent with the trends observed on chaotic systems, further supporting its practical applicability in real-time, resource-constrained environments. We can observe the visual prediction results on Figure A.9b and A.9a. On top of these we have the empirical equal parameter equal readout experiment results in the following Figures A.18a, A.18b. Outcome of the experiments are similar to that of chaotic attractor experiments and haRC methods perform better under parameter constraints.

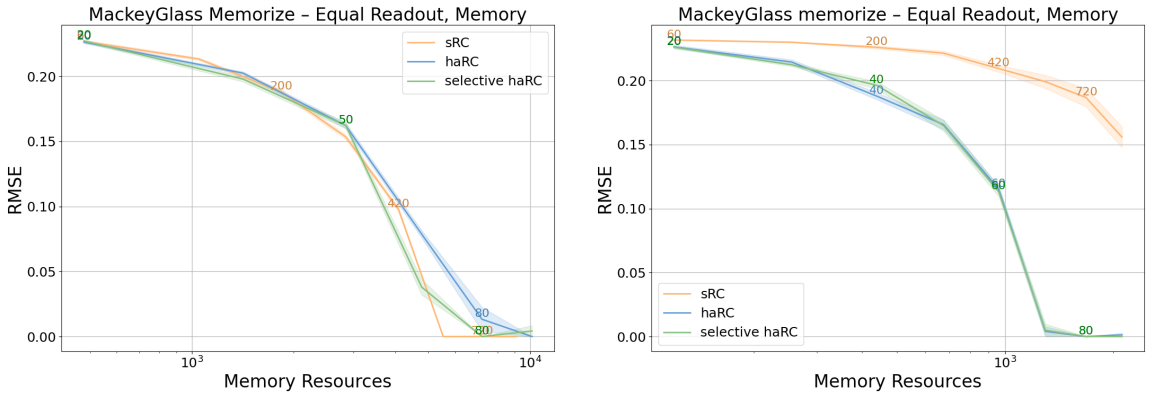
4.3.3.2 Memorization Tasks

In the sequential learning—or memorization—experiments, models are trained on 500-point sequences (either scalar or vector-valued) sampled from various dynamical systems described in Section 4.2. Unlike forecasting tasks, the goal here is to assess the model's ability to internally reconstruct the exact training sequence. Therefore, the test data is identical to the training set. During testing, the reservoir is initialized with its initial state from training and then allowed to evolve autonomously—without receiving any input or feedback—effectively relying on its internal dynamics to reproduce the sequence.

This setup allows us to directly compare the representational capacity of sRC and haRC models in memorizing complex temporal patterns.

As with the forecasting tasks, we present performance results in terms of RMSE and resource efficiency (see Figure 4.5). When haRC operates in a dense configuration, the corresponding sRC—adjusted to match the total number of fixed and learnable parameters—is typically capable of memorizing and regenerating the sequence with high accuracy.

Figure 4.5: ***Sequential Memorization task*** *These plots demonstrate the performance of RC's on sequence learning task using the Mackey-Glass system. Both experiments have equal memory and equal readout parameter where reservoir sizes are annotated on the graph. haRC clearly outperforms sRC in the scenario where haRC has an 85% sparse reservoir.*



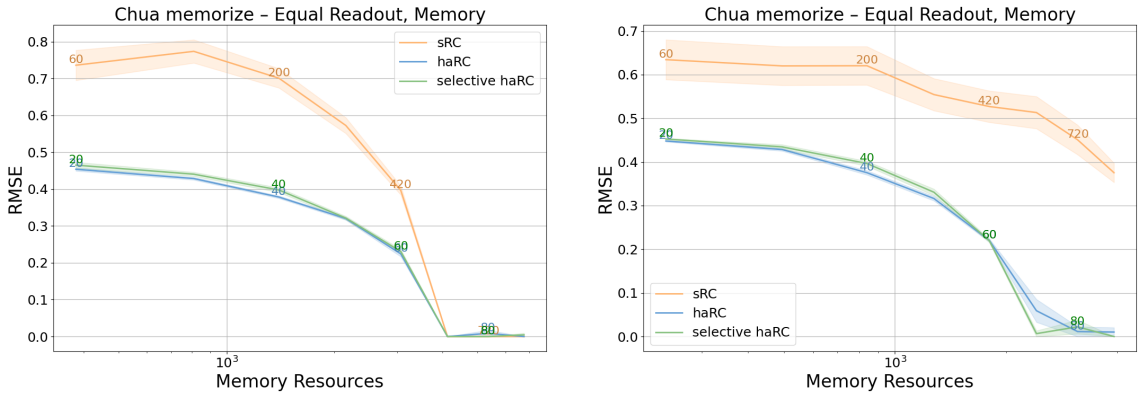
(a) *non-sparse haRC.* **(b)** *sparse haRC (85%).*

Similar behavior is observed in memorization experiments with the Chua Circuit dataset, as shown in Figure 4.6. Here, both 50% and 85% sparsity configurations are tested for haRC, with corresponding sRC models tuned to match their parameter counts.

Results indicate that haRC consistently outperforms sRC in both scenarios. Although the sRC reservoirs are non-zero in both cases, their limited parameter capacity renders them unable to effectively memorize the Chua attractor trajectory. In contrast,

haRC models demonstrate robust performance even under severe memory constraints. Further quantitative results and their significance [1] for the memorization experiments can be found in the Appendix. For the Mackey-Glass dataset, see Tables B.25 and B.23. For the Chua Circuit dataset, refer to Tables B.34 and B.32.

Figure 4.6: ***Sequential Memorization task*** *These plots demonstrate the performance of RC's on sequence learning task using the Chua Circuit Attractor. Both experiments have equal memory and equal readout parameter where reservoir sizes are annotated on the graph. haRC clearly outperforms sRC in both the scenarios where haRC has 50% and 85% sparse reservoirs.*



(a) *sparse haRC (50%).* **(b)** *sparse haRC (85%).*

A summarized reference list of all relevant tables for both generalization and memorization experiments are provided below.

- **Mackey-Glass Generalization:** Tables B.5, B.3, B.1
- **Chua Circuit Generalization:** Tables B.11, B.13
- **Mackey-Glass Memorization:** Tables B.25, B.23
- **Chua Circuit Memorization:** Tables B.34, B.32

4.3.4 Reservoir Unit Selection Algorithm

The reservoir unit selection algorithm introduced in Section 4.1.3 serves as a task-independent, data-driven heuristic designed to identify informative reservoir units for constructing higher-order augmentations in haRC. The primary goal of this selection mechanism is to improve the computational efficiency and robustness of haRC models by prioritizing units that contribute more diverse and informative features. While experimental results indicate that selective haRC does not exhibit a clear performance advantage over random haRC in ideal, noise-free environments, it consistently demonstrates superior robustness in the presence of internal Gaussian noise. In particular, under high noise levels, the performance of selective haRC approaches that of standard sRC, highlighting the importance of targeted unit selection for maintaining stability in noisy operational regimes. For completeness, selective haRC results are reported alongside random haRC and sRC in all relevant experiments. This enables a thorough evaluation of the proposed selection approach under both clean and noisy conditions.

However, once the haRC sparsity is increased to 85%, a notable gap emerges: the haRC model still successfully learns the sequence, while the sRC counterpart fails. Importantly, in this case, the sRC reservoir is not entirely null (as in some forecasting setups), but simply lacks sufficient parameter capacity to store the Mackey-Glass sequence under the imposed memory constraints.

5. CONCLUSIONS

Reservoir Computing (RC) continues to evolve as a powerful framework for learning temporal dependencies and modeling complex dynamical systems, especially under resource constraints. Its lightweight architecture and compatibility with edge computing have made it an attractive alternative to traditional deep learning methods for time series forecasting and dynamical system learning.

This thesis introduced and evaluated a novel augmentation strategy based on higher-order monomials applied to the reservoir state matrix. The proposed **Higher-Order Augmented Reservoir Computer (haRC)** and its selective variant demonstrated significant gains in predictive accuracy, memory efficiency, and representational richness without increasing the reservoir size or introducing additional learnable parameters. These findings suggest that such augmentations can enable smaller reservoirs to match or outperform larger ones, supporting both efficient design and practical deployment in constrained environments. The inclusion of a high-frequency trading dataset demonstrated that our proposed architectures generalize beyond synthetic systems, showing promise for deployment in more complex, real-world prediction tasks.

In addition, we explored reservoir unit selection strategies based on correlation analysis, which prioritize the most informative and uncorrelated reservoir activations. This approach ensures that only high-utility features contribute to the readout, leading to improved generalization and reduced overfitting.

5.1 Contributions to the Literature

Introduced a novel Higher-Order Augmented Reservoir Computing (haRC) framework with random and selective augmentation schemes.

Demonstrated that higher-order monomial augmentation improves both memory capacity and prediction accuracy, especially in constrained reservoirs.

Proposed a covariance-based reservoir unit selection method to enhance the efficiency of regression by filtering redundant or correlated reservoir states.

Systematically benchmarked standard RC, haRC, and selective haRC across multiple tasks including memorization and chaotic forecasting.

Provided a comprehensive survey and structured classification of the broader Reservoir Computing literature based on recent developments

5.2 Outlook and Future Work

The Higher-order Augmented Reservoir Computer (haRC) introduced in this thesis presents a lightweight, expressive, and resource-efficient alternative to conventional reservoir computing architectures. By augmenting a subset of reservoir state interactions through multiplicative monomials, the haRC framework enhances representational power without increasing the number of tunable parameters. The proposed Selective haRC model further advances this paradigm by using internal signal statistics to heuristically choose the most informative reservoir interactions, leading to improved performance under memory constraints, noise injection, and sparse connectivity.

These properties position haRC as a strong candidate for several real-world applications. Its low memory footprint and computational efficiency make it well-suited for *on-chip learning*, *neuromorphic computing*, and *real-time embedded AI*. For instance, it can be deployed in *robotics* for adaptive control or trajectory forecasting, or in *financial engineering* for forecasting and anomaly detection over high-dimensional time series using compact architectures for adapting to higher order moments of market movements. The Selective haRC's ability to capture essential dynamic interactions using fewer resources offers particular promise for control systems, and spatio-temporal sensor platforms operating under strict energy and latency constraints.

Building on these promising results, several research directions emerge. First, the

performance of randomly augmented haRC and its selective counterpart is often comparable, particularly in equal-readout scenarios. While selective augmentation reduces internal noise and provides better control over interaction terms, it does not always yield substantial improvements. More data-oriented methods for reservoir unit selection—such as mutual information, entropy measures, or sparse coding—could refine the Selective haRC strategy by replacing data-agnostic principles with data-driven criteria. In addition, the β parameter, which determines the number of monomials used in augmentation, remains a crucial factor for optimization. Exploring the joint effect of advanced selection techniques and systematic β optimization holds significant potential for improving the efficiency and adaptability of haRC. Second direction, is to investigate the numerical and architectural sensitivities that led to unexplained performance drops in certain forecasting experiments. These sensitivities could be better understood and potentially mitigated through improved initialization schemes, or additional stability checks.

Third, task-aware monomial construction and adaptive augmentation policies may offer improved expressiveness while minimizing overhead. A particularly novel direction involves integrating foundational dynamical systems—such as Lorenz or Rössler attractors—into the reservoir via state doping, thereby embedding known sub-dynamics into the reservoir space as dynamic priors to improve generalization. Finally, hardware-level integration with mixed-signal processors or spiking neuromorphic architectures could enable ultra-efficient implementations of haRC in edge computing scenarios.

In summary, the haRC and its selective variant not only expand the theoretical design space of reservoir computers, but also demonstrate practical potential across a range of real-world tasks where efficiency, robustness, and compactness are paramount.

APPENDICES

APPENDIX A. FORECAST GRAPHS OF RESERVOIR COMPUTERS

A.1 Prediction Visualizations of the Reservoir Computers

In this section we can observe the datasets and how the Reservoir Computers are reproduction of the trajectory according to the underlying dynamics of the datasets.

Figure A.1: *Reservoirs perform sequential memorization task on the butterfly dataset. haRC has fitted the equation with way less error that ground truth is not visible under the green trace of haRC.*

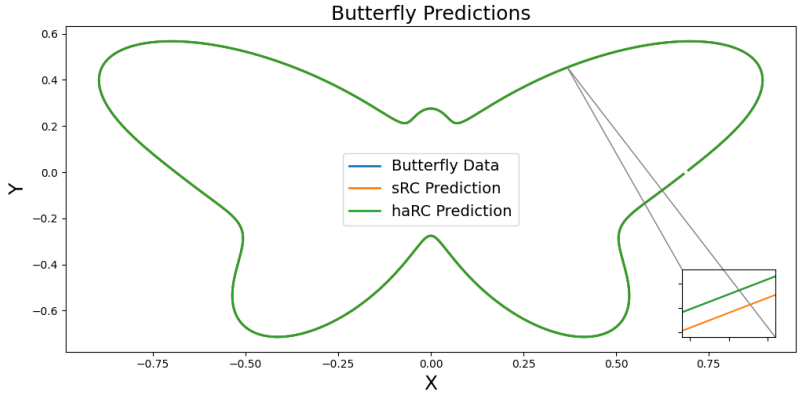


Figure A.2: *Reservoirs perform dynamics learning task on the Lorenz dataset. Both reservoir computers succesfully captured the dynamics of the Lorenz system.*

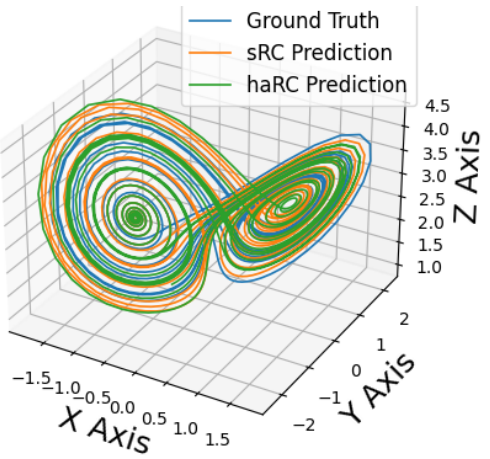


Figure A.3: *Reservoirs perform dynamics learning task on the Mackey-Glass dataset. Both reservoir computers successfully captured the dynamics of the Mackey-Glass system yet sRC lost composure after 150 time steps.*

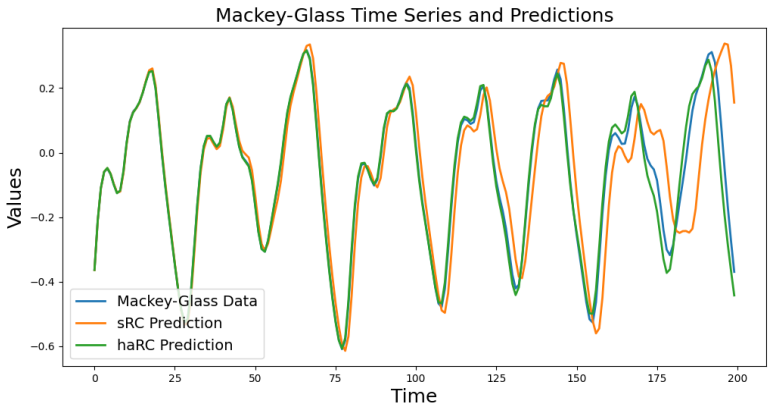
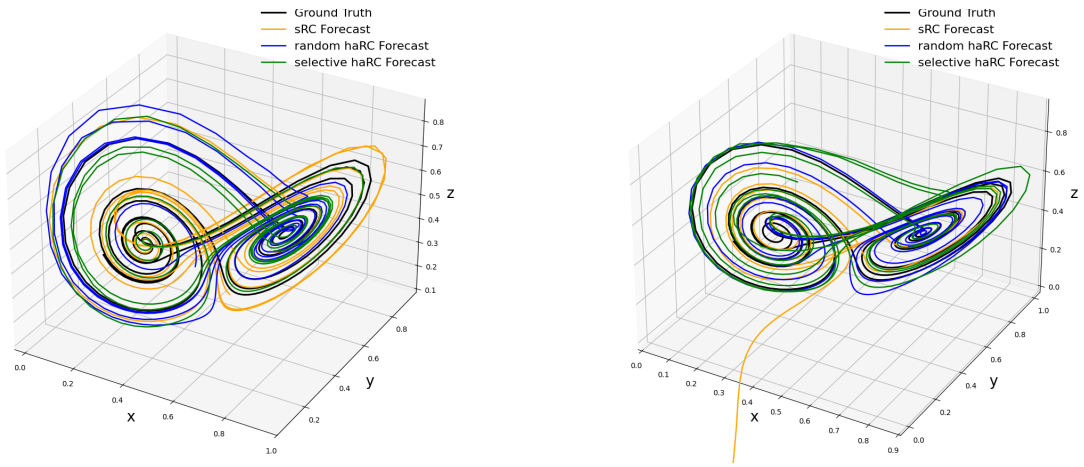


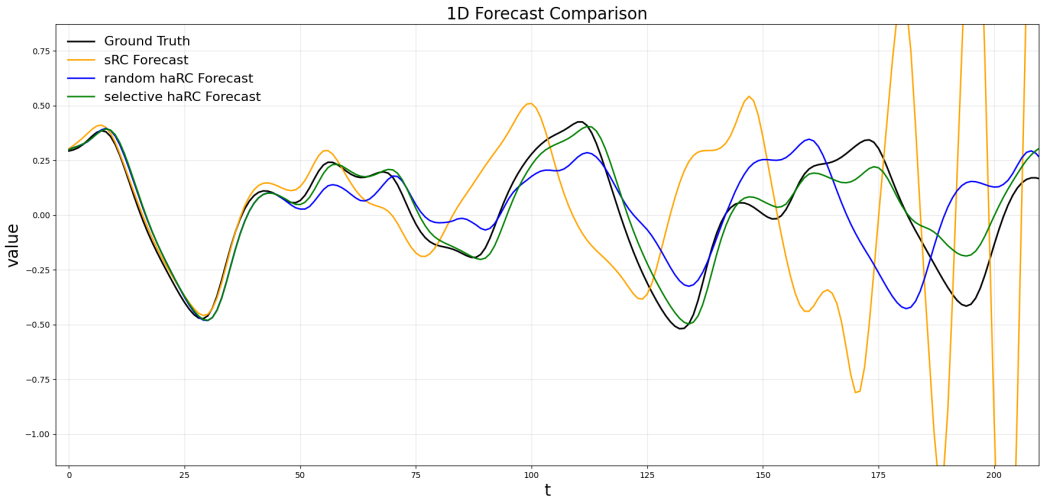
Figure A.4: ***Lorenz Dynamics Learning task equal parameter.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Root Mean squared error. (a)When haRC reservoir is 50%sparse both sRC and haRC captures the dynamics of the system to forecast.(b) When the haRC reservoirs are made to be 85% sparse we can see that sRC diverges rapidly after a few steps of forecasting.*



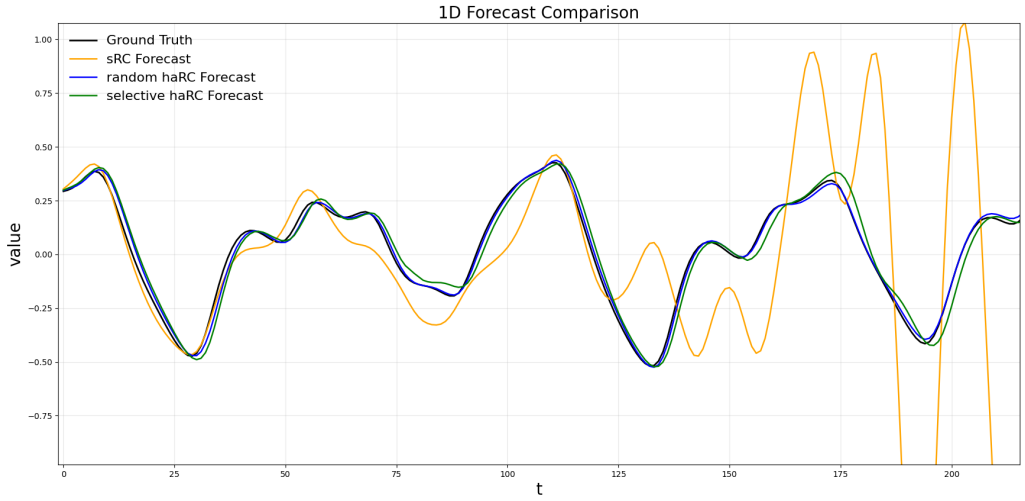
(a) *sparse haRC (50%).*

(b) *sparse haRC (85%).*

Figure A.5: ***Mackey-Glass Dynamics Learning task equal parameter.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Root Mean squared error. sRC has 900 unit reservoirs size and haRC's have 90 unit reservoirs (a) Higher Order Augmented computers prevail in long term accuracy with 50%sparseness.(b) When haRC reservoirs to be 85% sparse we can see that sRC algorithm starts deteriorating quickly while both random and selective haRC proceeds to forecast data accurately.*

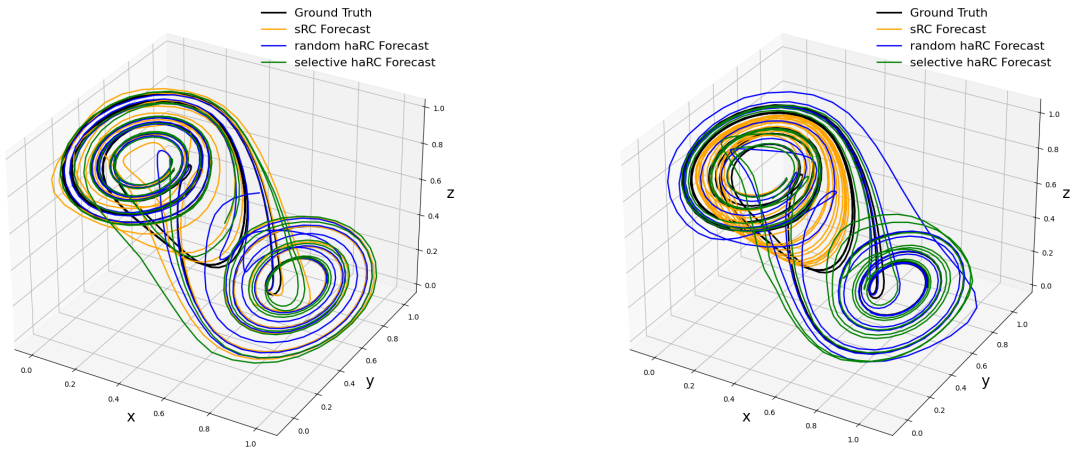


(a) *sparse haRC (50%).*



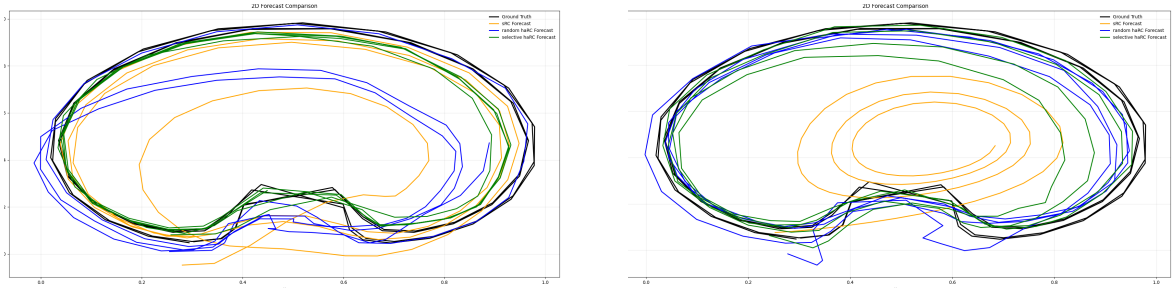
(b) *sparse haRC (85%).*

Figure A.6: ***Chua Circuit Dynamics Learning task equal parameter, equal readout.*** *sRC has 900 unit reservoirs size and haRC's have 90 unit reservoirs (a) Both haRC and sRC prevail in long term accuracy with 50%sparseness level for haRC. (b) When we make the haRC reservoirs to be 85% sparse we can see that haRC algorithms beat the sRC algorithms by keeping accurate predictions in the long term.*



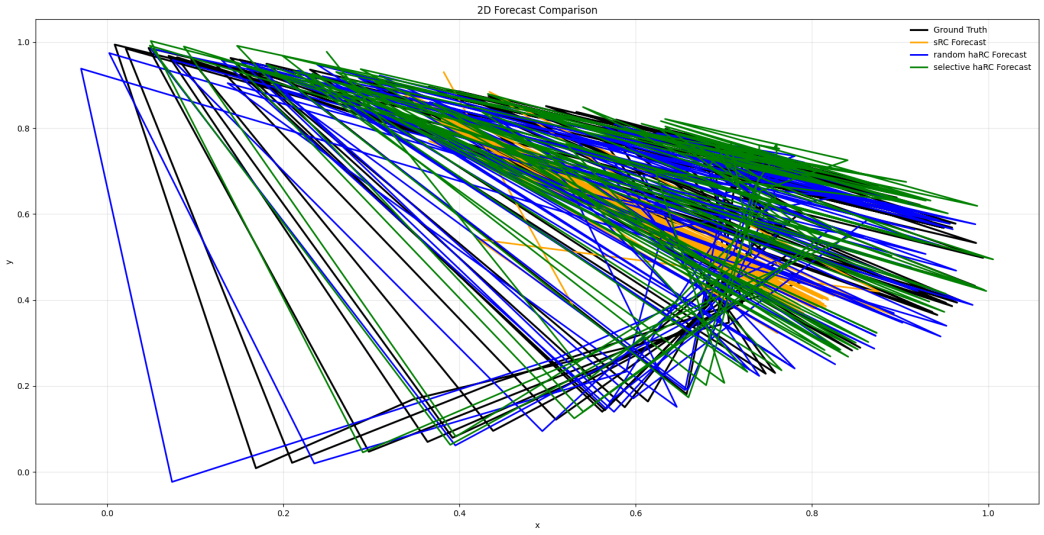
(a) *sparse haRC (50%).* **(b)** *sparse haRC (85%).*

Figure A.7: ***Ikeda map Dynamics Learning task equal parameter, equal readout.*** *sRC has 900 unit reservoirs size and haRC's have 90 unit reservoirs (a) Both sRC and haRC can capture the dynamics with haRC has 50% sparsity level.(b) When we make the haRC reservoirs to be 85% sparse we can see that haRC algorithms beat the sRC algorithms by keeping accurate predictions in the long term. While sRC diverges after a few timesteps.*

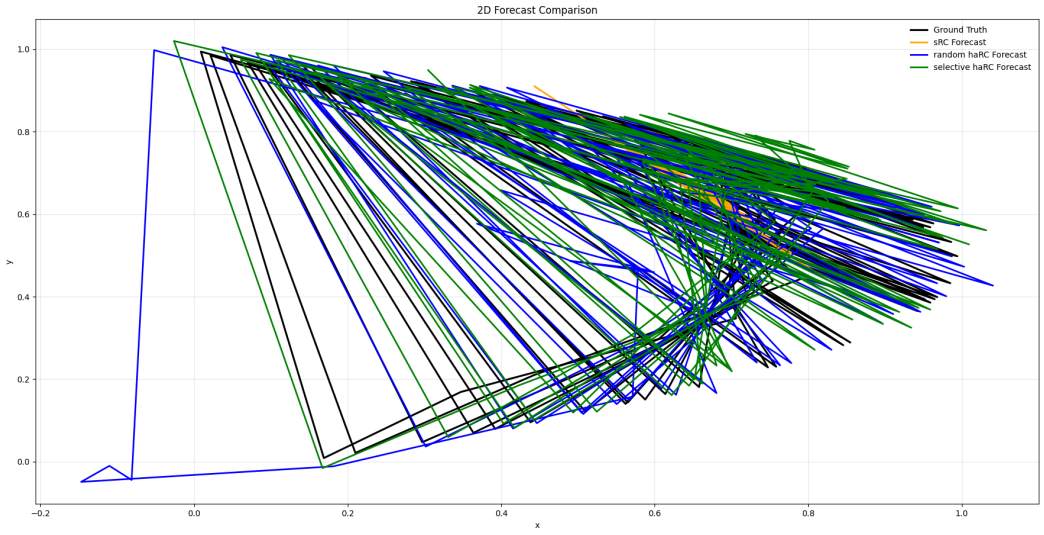


(a) *sparse haRC (50%).* **(b)** *sparse haRC (85%).*

Figure A.8: ***Henon map Dynamics Learning task equal parameter.*** *sRC has 900 unit reservoirs size and haRC's have 90 unit reservoirs (a) As it can be seen Higher Order Augmented computers prevail in long term accuracy with 50% sparseness.(b) When we make the haRC reservoirs to be 85% sparse we can see that haRC algorithms beat the sRC algorithms by keeping accurate predictions in the long term.*

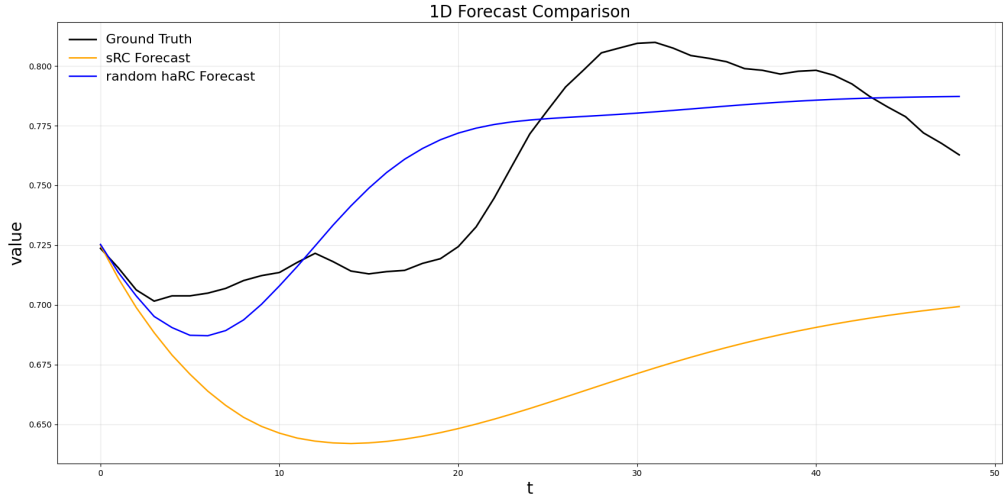


(a) *sparse haRC (50%).*

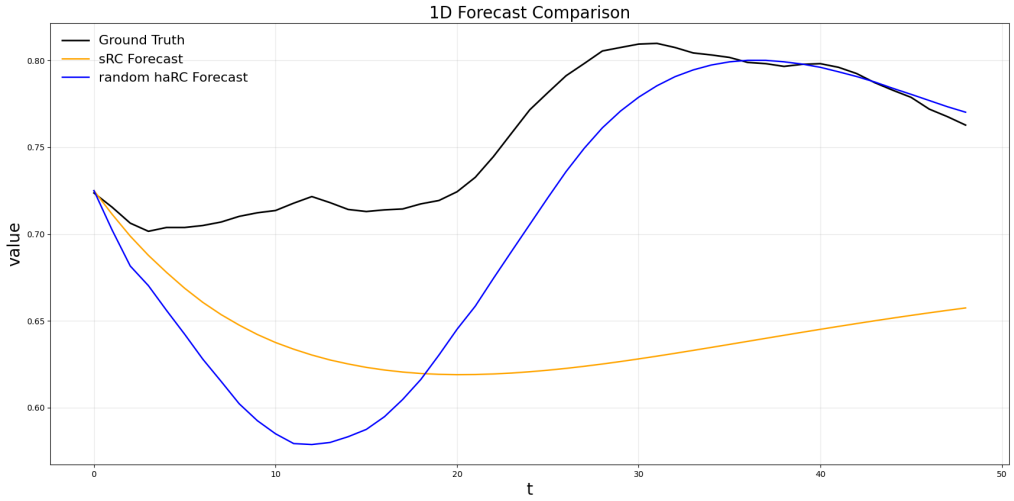


(b) *sparse haRC (85%).*

Figure A.9: ***HFT Forecasting task equal parameter.*** *sRC has 900 unit reservoirs size and haRC's have 90 unit reservoirs. As it can be seen Higher Order Augmented computers prevail in long term accuracy with 85% sparseness.*



(a) *sparse haRC (85%).*



(b) *sparse haRC (85%).*

A.2 Graphs of the Resource Analysis Experiments

Figure A.10: *Reservoirs are compared for the sequence memorization task on the Butterfly dataset. haRC requires way less resources to perform with non- sparse reservoirs.*

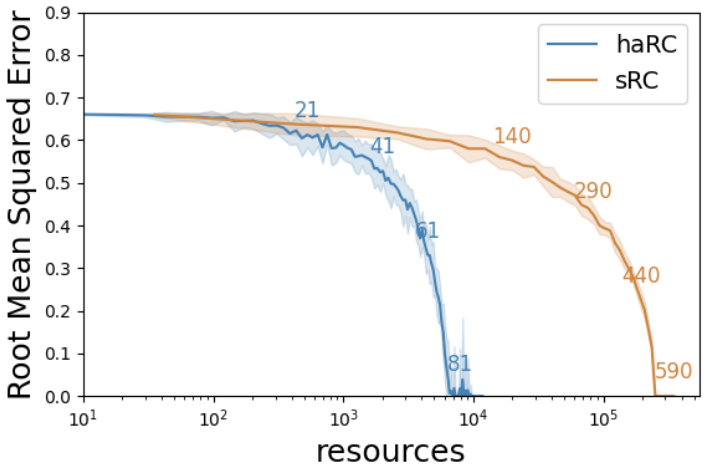


Figure A.11: *Reservoirs are compared for the sequence memorization task on the Random Time Series dataset. haRC requires way less resources to perform with non- sparse reservoirs.*

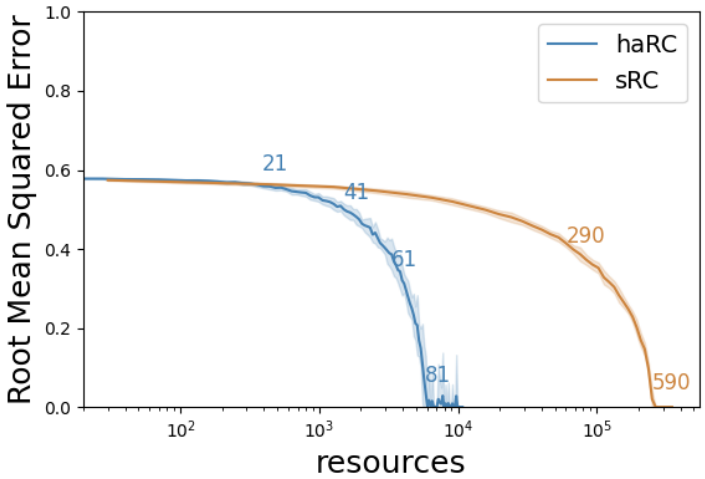


Figure A.12: *Reservoirs are compared for the dynamics learning task on the Lorenz dataset. haRC requires way less resources to perform with non- sparse reservoirs.*

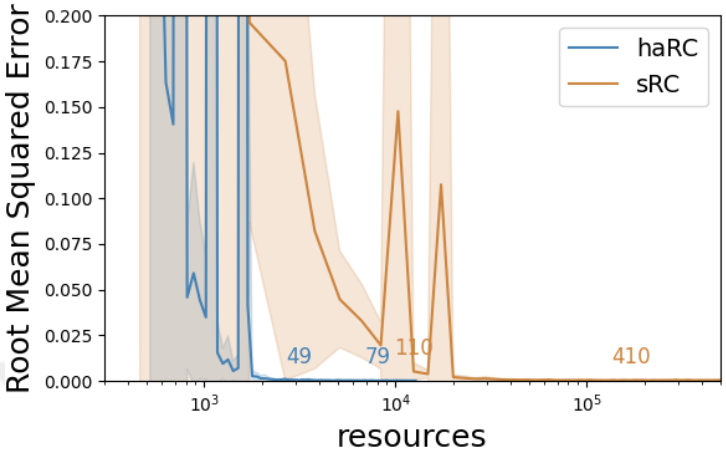


Figure A.13: *Reservoirs are compared for the dynamics learning task on the Mackey-Glass dataset. haRC requires way less resources to perform with non- sparse reservoirs.*

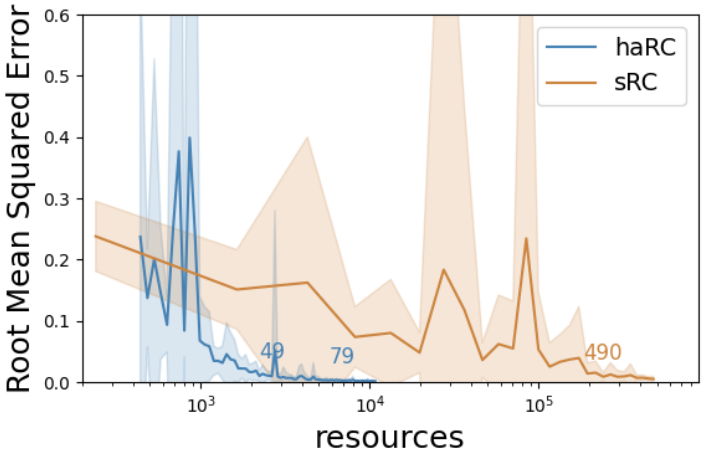
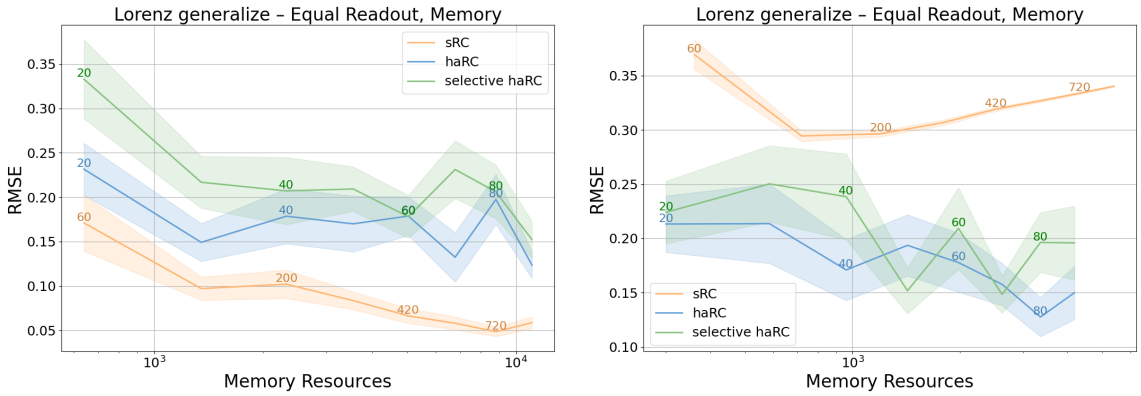
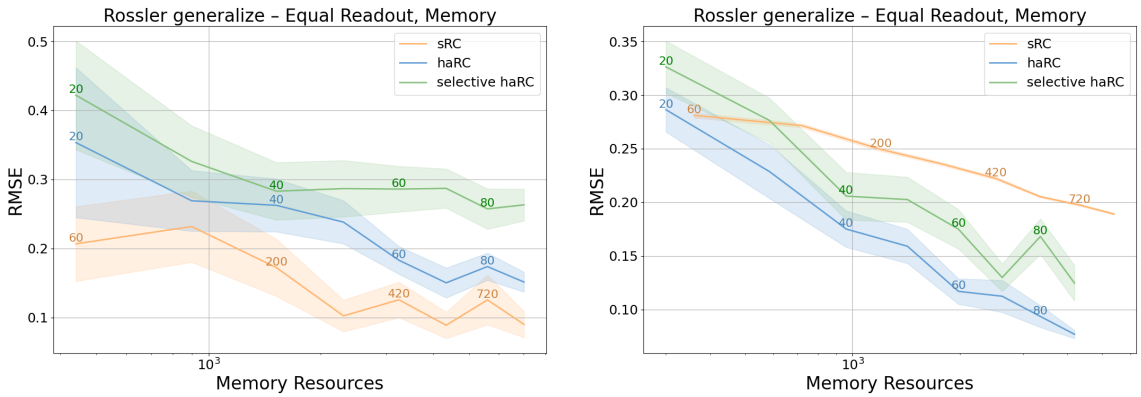


Figure A.14: ***Lorenz System Dynamics Learning task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Mean. (a) sRC has a 90% sparse reservoir it performs better compared to non-sparse haRC. (b) When we make the haRC reservoir to be 85% sparse we can see that sRC can not compete with it and haRC still performs reliably on the prediction of the Lorenz dynamics.*



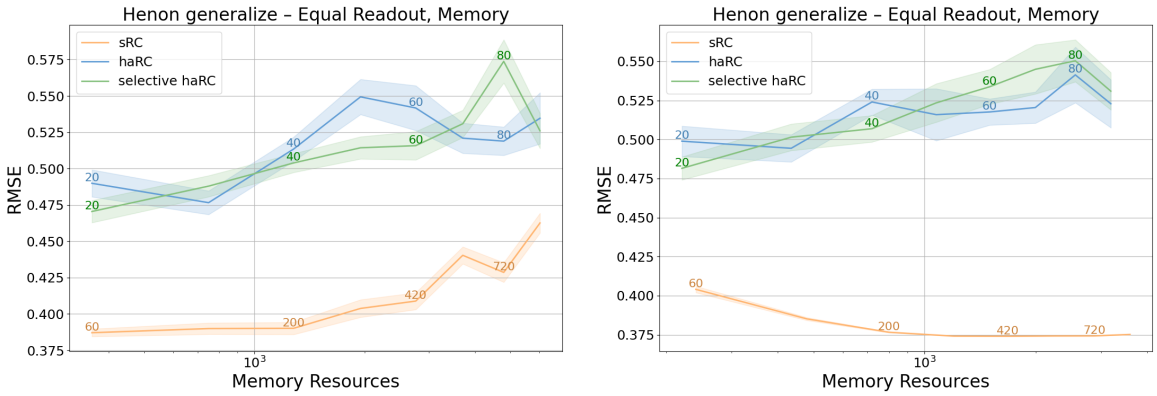
(a) *Non-sparse haRC.* **(b)** *Sparse haRC (85%).*

Figure A.15: ***Rössler Attractor Dynamics Learning task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Mean. (a) sRC has a 90% sparse reservoir it performs better compared to non-sparse haRC. (b) When we make the haRC reservoir to be 85% sparse we can see that sRC can not compete with it and haRC still performs reliably on the prediction of the Lorenz dynamics.*



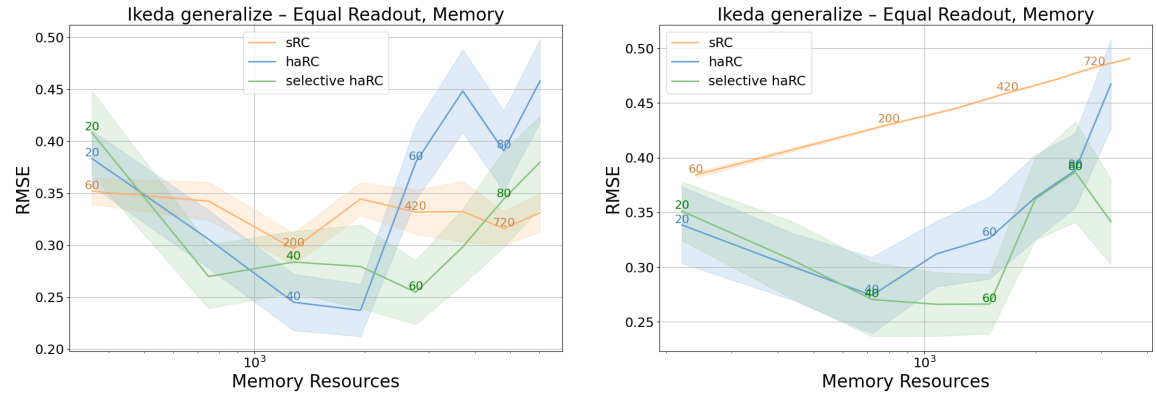
(a) *Sparse haRC (50%).* **(b)** *Sparse haRC (85%).*

Figure A.16: ***Henon Map Dynamics Learning task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Root Mean squared error. (a)Predictions graphs at Figure A.6 shows that especially random haRC is showing closer dynamics learning. (b) When we make the haRC reservoirs to be 85% sparse we can see that sRC gives out constant output while variance of haRC match dataset*



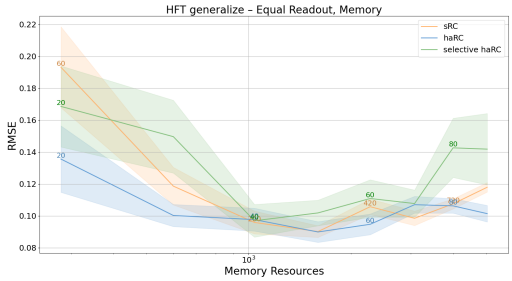
(a) *Sparse haRC (50%).* **(b)** *Sparse haRC (85%).*

Figure A.17: ***Ikeda Dynamics Learning task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Mean. (a) Predictions graphs at Figure A.7.(b) When we make the haRC reservoirs to be 85% sparse we can see that sRC follows a sub dynamic output while variance of haRC matches the dataset. Figure A.7*

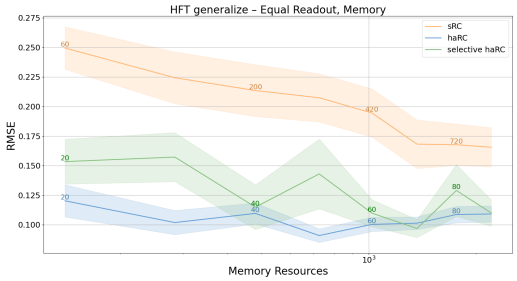


(a) *Sparse haRC (50%).* **(b)** *Sparse haRC (85%).*

Figure A.18: ***HFT forecasting task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Mean. When Reservoirs are 85% sparse we can see that sRC generates the wrong output while haRC matches the dataset.*

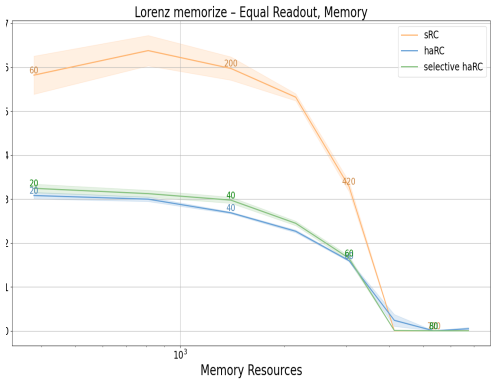


(a) *Sparse haRC (50%).*

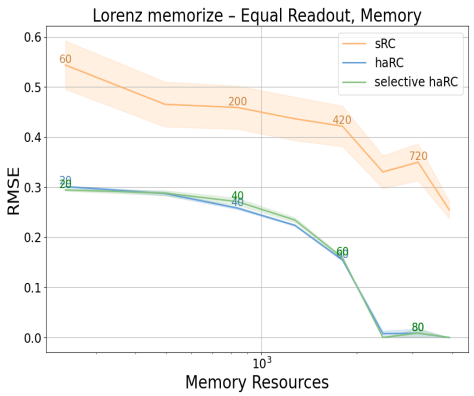


(b) *Sparse haRC (85%).*

Figure A.19: ***Lorenz Sequential Memorization task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Mean. (a)As it can be seen haRC' prevails in 50% sparseness.(b) When haRC reservoirs have 85% sparse sRC does not have enough parameters to memorize the whole sequence of the Lorenz system while haRC efficiently memorizes the whole sequence.*

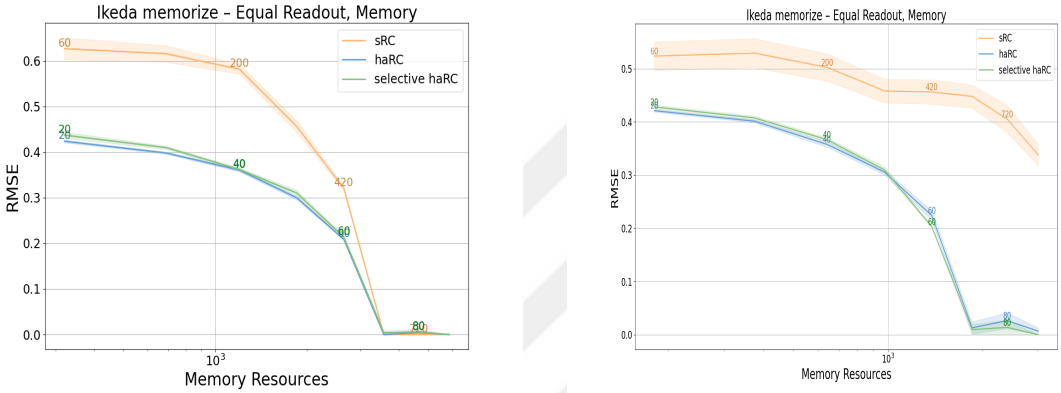


(a) *Sparse haRC (50%).*



(b) *Sparse haRC (85%).*

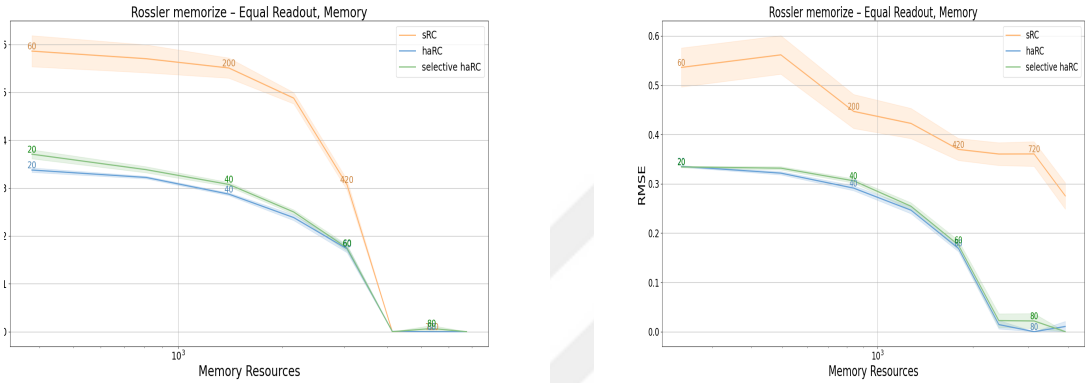
Figure A.20: ***Ikeda Sequential Memorization task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Root Mean squared error. (a)As it can be seen haRC' prevails in 50% sparseness where they score less error compared to sRC.(b) When we make the haRC reservoirs to be 85% sparse we can see that sRC does not have enough parameters to memorize the whole sequence of the Ikeda system while haRC efficiently memorizes the whole sequence.*



(a) *Sparse haRC (50%).*

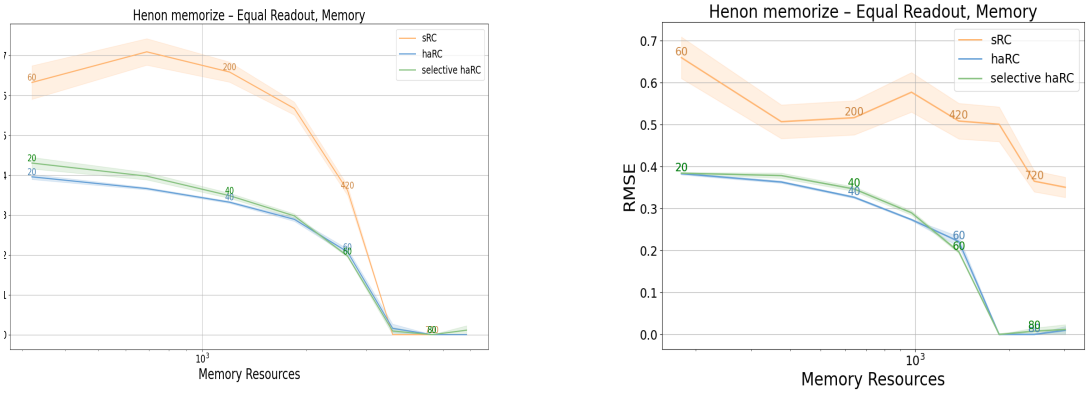
(b) *Sparse haRC (85%).*

Figure A.21: ***Rossler Sequential Memorization task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Root Mean squared error. (a)As it can be seen haRC' prevails in 50% sparseness.(b) When we make the haRC reservoirs to be 85% sparse we can see that sRC does not have enough parameters to memorize the whole sequence while haRC efficiently memorizes it.*



(a) *Sparse haRC (50%).* **(b)** *Sparse haRC (85%).*

Figure A.22: ***Henon Sequential Memorization task.*** *All data points are 30 experiments per configurations and shading is the Standard Error of the Root Mean squared error. (a) As it can be seen haRC' prevails in 50% sparseness (b) When we make the haRC reservoirs to be 85% sparse we can see that sRC does not have enough parameters to memorize the whole sequence while haRC efficiently memorizes it all.*



(a) *Sparse haRC (50%).* **(b)** *Sparse haRC (85%).*

APPENDIX B. EQUAL RESOURCE EVALUATION OF RESERVOIR COMPUTERS

B.1 Equal Resource Experiments

Tables in this section compares the performance of Reservoir Computers with corresponding resources used. In these experiments number of constant parameters used by the RC's and the number of tunable parameters used are equalized so that we can have a fair comparison in both sequence learning and dynamic systems forecasting tasks. To summarize, sRC requires higher number of parameters to complete these tasks meanwhile haRC algorithms can achieve success at extreme minimum parameter values which standard Reservoir Computers can not even operate or be created due to memory constraints.

Table B.1: *Mackey-Glass Dataset forecasting results are obtained with equalized resource and readout layer size at 85% sparse haRC. Refer to Fig 4.3b.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
140	20	60	60	0.850	0.994	141	220	139	138	0.176 ± 0.116	**0.120 ± 0.083**	0.188 ± 0.072	1.57
285	30	120	120	0.850	0.997	283	465	281	283	**0.081 ± 0.041**	0.108 ± 0.078	0.169 ± 0.086	2.08
480	40	200	200	0.850	0.998	480	800	478	478	0.077 ± 0.052	**0.062 ± 0.023**	0.161 ± 0.071	2.57
725	50	300	300	0.850	0.999	690	1225	688	723	0.071 ± 0.070	**0.068 ± 0.041**	0.166 ± 0.073	2.45
1020	60	420	420	0.850	0.999	1016	1740	1014	1018	**0.057 ± 0.046**	0.065 ± 0.040	0.145 ± 0.055	2.57
1365	70	560	560	0.850	0.999	1433	2345	1431	1363	**0.055 ± 0.025**	0.064 ± 0.037	0.142 ± 0.040	2.58
1760	80	720	720	0.850	0.999	1958	3040	1956	1758	0.076 ± 0.085	**0.053 ± 0.030**	0.190 ± 0.170	3.56
2205	90	900	900	0.850	1.000	1800	3825	1798	2203	**0.045 ± 0.015**	0.048 ± 0.013	0.150 ± 0.139	3.32

Table B.2: *Equal readout equal resource experiment with MackeyGlass dataset used in Generalize task performance at 85.0% sparsity level, with significance checks of (Welch's t-test [1], $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
140	20	60	0.1759 ± 0.1164	0.1883 ± 0.07187	0.6215	0	No
285	30	120	$\mathbf{0.08147 \pm 0.0412}$	0.1692 ± 0.0862	9.837e-06	1	Yes
480	40	200	$\mathbf{0.07659 \pm 0.05247}$	0.1607 ± 0.07134	3.184e-06	1	Yes
725	50	300	$\mathbf{0.07077 \pm 0.07001}$	0.1661 ± 0.07297	3.133e-06	1	Yes
1020	60	420	$\mathbf{0.05656 \pm 0.04612}$	0.1453 ± 0.05491	7.754e-09	1	Yes
1365	70	560	$\mathbf{0.05495 \pm 0.02519}$	0.1419 ± 0.03965	1.244e-13	1	Yes
1760	80	720	$\mathbf{0.07601 \pm 0.08493}$	0.1902 ± 0.17	0.00201	1	Yes
2205	90	900	$\mathbf{0.04512 \pm 0.01512}$	0.1498 ± 0.139	0.0002926	1	Yes

Table B.3: *Mackey-Glass Dataset forecasting results are obtained with equalized resource and readout layer size at 49% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
284	20	60	60	0.490	0.954	285	364	283	282	**0.092 ± 0.047**	0.146 ± 0.133	0.105 ± 0.035	1.14
609	30	120	120	0.490	0.974	614	789	612	607	**0.074 ± 0.053**	0.083 ± 0.044	0.096 ± 0.026	1.30
1056	40	200	200	0.490	0.984	1040	1376	1038	1054	0.073 ± 0.090	**0.070 ± 0.035**	0.099 ± 0.049	1.41
1625	50	300	300	0.490	0.989	1590	2125	1588	1623	**0.062 ± 0.025**	0.065 ± 0.031	0.091 ± 0.048	1.47
2316	60	420	420	0.490	0.992	2251	3036	2249	2314	0.065 ± 0.044	**0.052 ± 0.028**	0.090 ± 0.045	1.73
3129	70	560	560	0.490	0.994	3001	4109	2999	3127	**0.052 ± 0.031**	0.053 ± 0.019	0.083 ± 0.059	1.60
4064	80	720	720	0.490	0.995	4032	5344	4030	4062	0.063 ± 0.057	0.074 ± 0.116	**0.057 ± 0.042**	0.914
5121	90	900	900	0.490	0.996	5040	6741	5038	5119	**0.047 ± 0.014**	0.048 ± 0.011	0.051 ± 0.024	1.07

Table B.4: *Equal readout equal resource experiment with MackeyGlass dataset used in Generalize task performance at 49.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
284	20	60	0.09234 ± 0.047	0.1055 ± 0.03548	0.2268	0	No
609	30	120	$\mathbf{0.07422 \pm 0.05263}$	0.09646 ± 0.0256	0.04357	1	Yes
1056	40	200	0.07256 ± 0.09003	0.09863 ± 0.04854	0.1697	0	No
1625	50	300	$\mathbf{0.06192 \pm 0.02466}$	0.09085 ± 0.04808	0.005361	1	Yes
2316	60	420	$\mathbf{0.06506 \pm 0.04406}$	0.09011 ± 0.04514	0.03374	1	Yes
3129	70	560	$\mathbf{0.05209 \pm 0.03102}$	0.08338 ± 0.05926	0.01392	1	Yes
4064	80	720	0.06266 ± 0.05743	0.05728 ± 0.04213	0.681	0	No
5121	90	900	0.04738 ± 0.01367	0.05051 ± 0.02419	0.5411	0	No

Table B.5: *Mackey-Glass Dataset forecasting results are obtained with equalized resource and readout layer size at 0% sparse haRC. Please refer to Fig 4.3a.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
480	20	60	60	0.000	0.900	479	560	477	478	0.113 ± 0.069	0.127 ± 0.095	**0.103 ± 0.031**	0.906
1050	30	120	120	0.000	0.944	1046	1230	1044	1048	**0.085 ± 0.078**	0.100 ± 0.059	0.102 ± 0.050	1.20
1840	40	200	200	0.000	0.964	1840	2160	1838	1838	0.070 ± 0.057	**0.066 ± 0.034**	0.108 ± 0.060	1.64
2850	50	300	300	0.000	0.975	2850	3350	2848	2848	0.069 ± 0.054	**0.051 ± 0.027**	0.099 ± 0.053	1.92
4080	60	420	420	0.000	0.982	4015	4800	4013	4078	0.063 ± 0.076	**0.050 ± 0.023**	0.083 ± 0.053	1.66
5530	70	560	560	0.000	0.986	5510	6510	5508	5528	0.055 ± 0.081	**0.055 ± 0.025**	0.057 ± 0.022	1.04
7200	80	720	720	0.000	0.989	7142	8480	7140	7198	0.054 ± 0.029	**0.052 ± 0.019**	0.056 ± 0.025	1.08
9090	90	900	900	0.000	0.991	9090	10710	9088	9088	0.050 ± 0.012	0.056 ± 0.031	**0.042 ± 0.012**	0.847

Table B.6: *Lorenz Dataset forecasting results are obtained with equalized resource and readout layer size at 0% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
640	20	60	60	0.000	0.922	640	880	634	634	0.232 ± 0.160	0.333 ± 0.243	**0.171 ± 0.174**	0.738
1350	30	120	120	0.000	0.956	1353	1890	1347	1344	0.149 ± 0.117	0.217 ± 0.161	**0.097 ± 0.072**	0.651
2320	40	200	200	0.000	0.972	2320	3280	2314	2314	0.179 ± 0.170	0.207 ± 0.207	**0.102 ± 0.088**	0.571
3550	50	300	300	0.000	0.981	3510	5050	3504	3544	0.170 ± 0.172	0.209 ± 0.137	**0.083 ± 0.055**	0.491
5040	60	420	420	0.000	0.986	4989	7200	4983	5034	0.179 ± 0.120	0.178 ± 0.131	**0.066 ± 0.047**	0.372
6790	70	560	560	0.000	0.989	6809	9730	6803	6784	0.132 ± 0.152	0.231 ± 0.178	**0.058 ± 0.040**	0.437
8800	80	720	720	0.000	0.991	8985	12640	8979	8794	0.197 ± 0.154	0.206 ± 0.165	**0.048 ± 0.030**	0.245
11070	90	900	900	0.000	0.993	11070	15930	11064	11064	0.123 ± 0.083	0.152 ± 0.113	**0.059 ± 0.036**	0.474

Table B.7: *Lorenz Dataset forecasting results are obtained with equalized resource and readout layer size at 49% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
444	20	60	60	0.490	0.977	442	684	436	438	0.253 ± 0.181	**0.229 ± 0.173**	0.239 ± 0.198	1.04
909	30	120	120	0.490	0.987	907	1449	901	903	0.240 ± 0.186	0.233 ± 0.151	**0.142 ± 0.139**	0.612
1536	40	200	200	0.490	0.992	1520	2496	1514	1530	0.250 ± 0.264	0.181 ± 0.189	**0.127 ± 0.094**	0.698
2325	50	300	300	0.490	0.994	2340	3825	2334	2319	0.217 ± 0.201	0.206 ± 0.175	**0.068 ± 0.047**	0.327
3276	60	420	420	0.490	0.996	3225	5436	3219	3270	0.201 ± 0.161	0.191 ± 0.138	**0.093 ± 0.072**	0.489
4389	70	560	560	0.490	0.997	4300	7329	4294	4383	0.161 ± 0.139	0.168 ± 0.167	**0.073 ± 0.053**	0.455
5664	80	720	720	0.490	0.997	5875	9504	5869	5658	0.113 ± 0.092	0.121 ± 0.096	**0.062 ± 0.043**	0.544
7101	90	900	900	0.490	0.998	7020	11961	7014	7095	0.133 ± 0.110	0.195 ± 0.184	**0.063 ± 0.038**	0.476

Table B.8: *Lorenz Dataset forecasting results are obtained with equalized resource and readout layer size at 85% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
300	20	60	60	0.850	1.020	287	540	281	294	**0.213 ± 0.143**	0.224 ± 0.158	0.370 ± 0.074	1.73
585	30	120	120	0.850	1.010	575	1125	569	579	**0.214 ± 0.200**	0.250 ± 0.192	0.294 ± 0.027	1.38
960	40	200	200	0.850	1.010	799	1920	793	954	**0.171 ± 0.154**	0.238 ± 0.217	0.296 ± 0.015	1.73
1425	50	300	300	0.850	1.000	1800	2925	1794	1419	0.194 ± 0.155	**0.152 ± 0.115**	0.307 ± 0.014	2.02
1980	60	420	420	0.850	1.000	2520	4140	2514	1974	**0.178 ± 0.149**	0.209 ± 0.206	0.319 ± 0.010	1.80
2625	70	560	560	0.850	1.000	3360	5565	3354	2619	0.158 ± 0.107	**0.148 ± 0.094**	0.327 ± 0.007	2.20
3360	80	720	720	0.850	1.000	4320	7200	4314	3354	**0.128 ± 0.099**	0.196 ± 0.151	0.334 ± 0.006	2.62
4185	90	900	900	0.850	1.000	5400	9045	5394	4179	**0.150 ± 0.135**	0.196 ± 0.186	0.340 ± 0.005	2.27

Table B.9: *Equal readout equal resource experiment with Lorenz dataset used in Generalize task performance at 85.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
300	20	60	**0.2133 ± 0.1432**	0.3696 ± 0.07442	3.588e-06	1	Yes
585	30	120	**0.2136 ± 0.2003**	0.2945 ± 0.02732	0.03639	1	Yes
960	40	200	**0.171 ± 0.1541**	0.2965 ± 0.015	0.0001159	1	Yes
1425	50	300	**0.1937 ± 0.155**	0.3069 ± 0.01445	0.000407	1	Yes
1980	60	420	**0.1778 ± 0.1493**	0.3192 ± 0.01023	1.517e-05	1	Yes
2625	70	560	**0.1576 ± 0.1068**	0.3269 ± 0.007495	1.437e-09	1	Yes
3360	80	720	**0.1276 ± 0.09871**	0.3338 ± 0.006279	2.697e-12	1	Yes
4185	90	900	**0.15 ± 0.1347**	0.3403 ± 0.005124	1.551e-08	1	Yes

Table B.10: *Chua Circuit Dataset forecasting results are obtained with equalized resource and readout layer size at 0% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
640	20	60	60	0.000	0.922	640	880	634	634	**0.419 ± 0.265**	0.435 ± 0.243	0.495 ± 0.223	1.18
1350	30	120	120	0.000	0.956	1353	1890	1347	1344	0.441 ± 0.263	0.431 ± 0.262	**0.393 ± 0.196**	0.912
2320	40	200	200	0.000	0.972	2320	3280	2314	2314	0.428 ± 0.284	0.346 ± 0.268	**0.343 ± 0.160**	0.992
3550	50	300	300	0.000	0.981	3510	5050	3504	3544	0.330 ± 0.284	0.340 ± 0.268	**0.264 ± 0.096**	0.799
5040	60	420	420	0.000	0.986	4989	7200	4983	5034	0.381 ± 0.243	0.379 ± 0.235	**0.279 ± 0.108**	0.737
6790	70	560	560	0.000	0.989	6809	9730	6803	6784	0.376 ± 0.266	0.380 ± 0.266	**0.260 ± 0.105**	0.692
8800	80	720	720	0.000	0.991	8985	12640	8979	8794	0.397 ± 0.252	0.357 ± 0.250	**0.241 ± 0.112**	0.675
11070	90	900	900	0.000	0.993	11070	15930	11064	11064	0.407 ± 0.270	0.311 ± 0.232	**0.200 ± 0.080**	0.643

Table B.11: *Chua Circuit Dataset forecasting results are obtained with equalized resource and readout layer size at 50% sparse haRC. Refer to figure 4.4a.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
440	20	60	60	0.500	0.978	439	680	433	434	0.108 ± 0.098	0.147 ± 0.207	**0.105 ± 0.062**	0.973
900	30	120	120	0.500	0.988	892	1440	886	894	0.076 ± 0.099	0.042 ± 0.045	**0.040 ± 0.020**	0.963
1520	40	200	200	0.500	0.992	1520	2480	1514	1514	0.047 ± 0.108	**0.037 ± 0.044**	0.039 ± 0.022	1.08
2300	50	300	300	0.500	0.994	2340	3800	2334	2294	0.022 ± 0.012	**0.017 ± 0.011**	0.035 ± 0.020	2.13
3240	60	420	420	0.500	0.996	3225	5400	3219	3234	**0.017 ± 0.009**	0.020 ± 0.011	0.034 ± 0.037	2.05
4340	70	560	560	0.500	0.997	4300	7280	4294	4334	0.022 ± 0.013	**0.015 ± 0.008**	0.023 ± 0.016	1.54
5600	80	720	720	0.500	0.998	5356	9440	5350	5594	**0.018 ± 0.016**	0.019 ± 0.013	0.023 ± 0.012	1.24
7020	90	900	900	0.500	0.998	7020	11880	7014	7014	**0.016 ± 0.006**	0.021 ± 0.010	0.020 ± 0.011	1.21

Table B.12: *Equal readout equal resource experiment with Chua dataset used in Generalize task performance at 50.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
440	20	60	0.1081 ± 0.09758	0.1052 ± 0.06226	0.8913	0	No
900	30	120	0.07615 ± 0.09894	0.04041 ± 0.02026	0.06162	0	No
1520	40	200	0.04655 ± 0.1076	0.03947 ± 0.02165	0.7261	0	No
2300	50	300	**0.02172 ± 0.01239**	0.03515 ± 0.02039	0.003379	1	Yes
3240	60	420	**0.0168 ± 0.009088**	0.03449 ± 0.03715	0.01635	1	Yes
4340	70	560	0.02204 ± 0.01339	0.02307 ± 0.01554	0.7856	0	No
5600	80	720	0.01821 ± 0.01573	0.02256 ± 0.01164	0.2286	0	No
7020	90	900	0.01624 ± 0.006104	0.01968 ± 0.01111	0.1434	0	No

Table B.13: *Chua Circuit Dataset forecasting results are obtained with equalized resource and readout layer size at 85% sparse haRC. Please refer to figure 4.4b.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
300	20	60	60	0.850	1.020	287	540	281	294	**0.087 ± 0.143**	0.118 ± 0.201	0.090 ± 0.006	1.04
585	30	120	120	0.850	1.010	575	1125	569	579	0.061 ± 0.165	**0.056 ± 0.075**	0.076 ± 0.005	1.35
960	40	200	200	0.850	1.010	799	1920	793	954	0.024 ± 0.017	**0.024 ± 0.017**	0.070 ± 0.004	2.95
1425	50	300	300	0.850	1.000	1800	2925	1794	1419	0.020 ± 0.014	**0.019 ± 0.011**	0.064 ± 0.004	3.46
1980	60	420	420	0.850	1.000	2520	4140	2514	1974	0.019 ± 0.012	**0.017 ± 0.011**	0.057 ± 0.004	3.36
2625	70	560	560	0.850	1.000	3360	5565	3354	2619	0.017 ± 0.011	**0.016 ± 0.010**	0.054 ± 0.003	3.28
3360	80	720	720	0.850	1.000	4320	7200	4314	3354	**0.016 ± 0.010**	0.035 ± 0.071	0.050 ± 0.003	3.03
4185	90	900	900	0.850	1.000	5400	9045	5394	4179	**0.017 ± 0.010**	0.020 ± 0.007	0.046 ± 0.002	2.73

Table B.14: *Equal readout equal resource experiment with Chua dataset used in Generalize task performance at 85.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
300	20	60	0.08651 ± 0.1433	0.09008 ± 0.006132	0.8928	0	No
585	30	120	0.0606 ± 0.1653	0.07582 ± 0.004991	0.618	0	No
960	40	200	$\mathbf{0.02432 \pm 0.01743}$	0.07033 ± 0.00423	2.464e-15	1	Yes
1425	50	300	$\mathbf{0.0199 \pm 0.01433}$	0.06425 ± 0.004403	1.104e-17	1	Yes
1980	60	420	$\mathbf{0.01921 \pm 0.01178}$	0.0575 ± 0.004009	1.47e-18	1	Yes
2625	70	560	$\mathbf{0.0174 \pm 0.01114}$	0.05388 ± 0.003309	2.316e-18	1	Yes
3360	80	720	$\mathbf{0.01647 \pm 0.009705}$	0.04982 ± 0.00295	4.609e-19	1	Yes
4185	90	900	$\mathbf{0.01671 \pm 0.009751}$	0.0456 ± 0.001877	1.599e-16	1	Yes

Table B.15: *Rössler Dataset forecasting results are obtained with equalized resource and readout layer size at 50% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
440	20	60	60	0.500	0.978	439	680	433	434	0.353 ± 0.595	0.422 ± 0.431	**0.207 ± 0.296**	0.585
900	30	120	120	0.500	0.988	892	1440	886	894	0.269 ± 0.241	0.326 ± 0.284	**0.232 ± 0.283**	0.861
1520	40	200	200	0.500	0.992	1520	2480	1514	1514	0.262 ± 0.210	0.283 ± 0.228	**0.172 ± 0.227**	0.656
2300	50	300	300	0.500	0.994	2340	3800	2334	2294	0.238 ± 0.172	0.287 ± 0.224	**0.102 ± 0.126**	0.430
3240	60	420	420	0.500	0.996	3225	5400	3219	3234	0.183 ± 0.110	0.286 ± 0.182	**0.126 ± 0.141**	0.687
4340	70	560	560	0.500	0.997	4300	7280	4294	4334	0.150 ± 0.119	0.287 ± 0.155	**0.089 ± 0.104**	0.591
5600	80	720	720	0.500	0.998	5356	9440	5350	5594	0.174 ± 0.108	0.257 ± 0.160	**0.125 ± 0.197**	0.722
7020	90	900	900	0.500	0.998	7020	11880	7014	7014	0.151 ± 0.079	0.263 ± 0.127	**0.090 ± 0.104**	0.595

Table B.16: *Rössler Dataset forecasting results are obtained with equalized resource and readout layer size at 85% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
300	20	60	60	0.850	1.020	287	540	281	294	0.286 ± 0.113	0.326 ± 0.133	**0.281 ± 0.013**	0.982
585	30	120	120	0.850	1.010	575	1125	569	579	**0.229 ± 0.137**	0.276 ± 0.114	0.272 ± 0.010	1.19
960	40	200	200	0.850	1.010	799	1920	793	954	**0.175 ± 0.093**	0.206 ± 0.122	0.249 ± 0.010	1.42
1425	50	300	300	0.850	1.000	1800	2925	1794	1419	**0.159 ± 0.087**	0.203 ± 0.115	0.235 ± 0.008	1.48
1980	60	420	420	0.850	1.000	2520	4140	2514	1974	**0.117 ± 0.065**	0.175 ± 0.101	0.222 ± 0.008	1.90
2625	70	560	560	0.850	1.000	3360	5565	3354	2619	**0.112 ± 0.082**	0.130 ± 0.070	0.205 ± 0.005	1.82
3360	80	720	720	0.850	1.000	4320	7200	4314	3354	**0.094 ± 0.056**	0.168 ± 0.091	0.198 ± 0.006	2.11
4185	90	900	900	0.850	1.000	5400	9045	5394	4179	**0.077 ± 0.021**	0.125 ± 0.091	0.189 ± 0.005	2.46

Table B.17: *Equal readout equal resource experiment with Rossler dataset used in Generalize task performance at 85.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
300	20	60	0.2864 ± 0.1132	0.2811 ± 0.01307	0.8008	0	No
585	30	120	0.2287 ± 0.137	0.2716 ± 0.01023	0.09812	0	No
960	40	200	$\mathbf{0.1751 \pm 0.09311}$	0.2494 ± 0.00963	0.0001498	1	Yes
1425	50	300	$\mathbf{0.1591 \pm 0.08743}$	0.2354 ± 0.008095	4.746e-05	1	Yes
1980	60	420	$\mathbf{0.117 \pm 0.06485}$	0.2223 ± 0.007778	8.146e-10	1	Yes
2625	70	560	$\mathbf{0.1125 \pm 0.08176}$	0.2052 ± 0.005302	8.921e-07	1	Yes
3360	80	720	$\mathbf{0.09365 \pm 0.05591}$	0.1976 ± 0.005779	3.941e-11	1	Yes
4185	90	900	$\mathbf{0.07702 \pm 0.02086}$	0.1892 ± 0.004514	2.694e-24	1	Yes

Table B.18: *Ikeda Dataset forecasting results are obtained with equalized resource and readout layer size at 0% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
560	20	60	60	0.000	0.911	560	720	556	556	**0.303 ± 0.159**	0.334 ± 0.151	0.335 ± 0.116	1.11
1200	30	120	120	0.000	0.950	1200	1560	1196	1196	**0.236 ± 0.112**	0.274 ± 0.173	0.337 ± 0.092	1.43
2080	40	200	200	0.000	0.968	2080	2720	2076	2076	0.295 ± 0.189	**0.220 ± 0.120**	0.326 ± 0.099	1.48
3200	50	300	300	0.000	0.978	3180	4200	3176	3196	0.308 ± 0.185	**0.285 ± 0.176**	0.308 ± 0.094	1.08
4560	60	420	420	0.000	0.984	4502	6000	4498	4556	0.321 ± 0.172	**0.279 ± 0.206**	0.321 ± 0.112	1.15
6160	70	560	560	0.000	0.988	6003	8120	5999	6156	0.338 ± 0.198	0.359 ± 0.203	**0.297 ± 0.072**	0.878
8000	80	720	720	0.000	0.990	8064	10560	8060	7996	0.412 ± 0.192	**0.323 ± 0.195**	0.329 ± 0.100	1.02
10080	90	900	900	0.000	0.992	10080	13320	10076	10076	0.487 ± 0.212	0.403 ± 0.267	**0.314 ± 0.110**	0.778

Table B.19: *Ikeda Dataset forecasting results are obtained with equalized resource and readout layer size at 50% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
360	20	60	60	0.500	0.967	358	520	354	356	0.383 ± 0.146	0.408 ± 0.220	**0.352 ± 0.069**	0.918
750	30	120	120	0.500	0.981	753	1110	749	746	0.306 ± 0.152	**0.270 ± 0.167**	0.342 ± 0.100	1.27
1280	40	200	200	0.500	0.988	1280	1920	1276	1276	**0.245 ± 0.148**	0.284 ± 0.162	0.297 ± 0.082	1.21
1950	50	300	300	0.500	0.992	1920	2950	1916	1946	**0.237 ± 0.138**	0.280 ± 0.220	0.344 ± 0.087	1.45
2760	60	420	420	0.500	0.994	2738	4200	2734	2756	0.380 ± 0.199	**0.255 ± 0.169**	0.332 ± 0.118	1.30
3710	70	560	560	0.500	0.995	3808	5670	3804	3706	0.448 ± 0.218	**0.298 ± 0.199**	0.332 ± 0.159	1.11
4800	80	720	720	0.500	0.996	4953	7360	4949	4796	0.391 ± 0.211	0.344 ± 0.253	**0.316 ± 0.090**	0.918
6030	90	900	900	0.500	0.997	6030	9270	6026	6026	0.458 ± 0.217	0.380 ± 0.242	**0.331 ± 0.100**	0.872

Table B.20: *Ikeda Dataset forecasting results are obtained with equalized resource and readout layer size at 85% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
220	20	60	60	0.850	1.010	203	380	199	216	**0.339 ± 0.192**	0.351 ± 0.147	0.384 ± 0.016	1.14
435	30	120	120	0.850	1.000	480	795	476	431	**0.301 ± 0.167**	0.307 ± 0.193	0.411 ± 0.009	1.36
720	40	200	200	0.850	1.000	800	1360	796	716	0.274 ± 0.191	**0.271 ± 0.185**	0.430 ± 0.004	1.59
1075	50	300	300	0.850	1.000	1200	2075	1196	1071	0.312 ± 0.164	**0.266 ± 0.160**	0.445 ± 0.004	1.67
1500	60	420	420	0.850	1.000	1680	2940	1676	1496	0.327 ± 0.205	**0.266 ± 0.148**	0.459 ± 0.004	1.73
1995	70	560	560	0.850	1.000	2240	3955	2236	1991	0.364 ± 0.212	**0.362 ± 0.207**	0.471 ± 0.003	1.30
2560	80	720	720	0.850	1.000	2880	5120	2876	2556	0.389 ± 0.186	**0.387 ± 0.250**	0.483 ± 0.004	1.25
3195	90	900	900	0.850	1.000	3600	6435	3596	3191	0.467 ± 0.222	**0.342 ± 0.215**	0.491 ± 0.002	1.44

Table B.21: *Henon Dataset forecasting results are obtained with equalized resource and readout layer size at 50% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
360	20	60	60	0.500	0.967	358	520	354	356	0.490 ± 0.051	0.471 ± 0.042	**0.387 ± 0.014**	0.823
750	30	120	120	0.500	0.981	753	1110	749	746	0.477 ± 0.045	0.488 ± 0.040	**0.390 ± 0.021**	0.818
1280	40	200	200	0.500	0.988	1280	1920	1276	1276	0.513 ± 0.045	0.504 ± 0.036	**0.390 ± 0.023**	0.774
1950	50	300	300	0.500	0.992	1920	2950	1916	1946	0.549 ± 0.066	0.514 ± 0.042	**0.404 ± 0.033**	0.785
2760	60	420	420	0.500	0.994	2738	4200	2734	2756	0.542 ± 0.084	0.516 ± 0.054	**0.409 ± 0.032**	0.793
3710	70	560	560	0.500	0.995	3808	5670	3804	3706	0.521 ± 0.057	0.531 ± 0.051	**0.440 ± 0.032**	0.845
4800	80	720	720	0.500	0.996	4953	7360	4949	4796	0.519 ± 0.053	0.574 ± 0.082	**0.429 ± 0.037**	0.826
6030	90	900	900	0.500	0.997	6030	9270	6026	6026	0.535 ± 0.096	0.526 ± 0.066	**0.463 ± 0.037**	0.879

Table B.22: *Henon Dataset forecasting results are obtained with equalized resource and readout layer size at 85% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
220	20	60	60	0.850	1.010	203	380	199	216	0.499 ± 0.053	0.482 ± 0.040	**0.404 ± 0.011**	0.839
435	30	120	120	0.850	1.000	480	795	476	431	0.494 ± 0.048	0.501 ± 0.047	**0.385 ± 0.005**	0.779
720	40	200	200	0.850	1.000	800	1360	796	716	0.524 ± 0.045	0.507 ± 0.047	**0.377 ± 0.002**	0.743
1075	50	300	300	0.850	1.000	1200	2075	1196	1071	0.516 ± 0.091	0.523 ± 0.067	**0.374 ± 0.001**	0.725
1500	60	420	420	0.850	1.000	1680	2940	1676	1496	0.518 ± 0.046	0.534 ± 0.061	**0.374 ± 0.001**	0.723
1995	70	560	560	0.850	1.000	2240	3955	2236	1991	0.520 ± 0.055	0.545 ± 0.086	**0.374 ± 0.001**	0.719
2560	80	720	720	0.850	1.000	2880	5120	2876	2556	0.541 ± 0.098	0.550 ± 0.074	**0.374 ± 0.001**	0.691
3195	90	900	900	0.850	1.000	3600	6435	3596	3191	0.523 ± 0.084	0.531 ± 0.065	**0.375 ± 0.001**	0.718

Table B.23: *Mackey-Glass Dataset memorization results are obtained with equalized resource and readout layer size at 85% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
120	20	60	60	0.850	0.983	120	240	59	58	0.226 ± 0.007	**0.226 ± 0.005**	0.232 ± 0.001	1.02
255	30	120	120	0.850	0.991	255	525	134	133	0.215 ± 0.010	**0.212 ± 0.007**	0.230 ± 0.002	1.08
440	40	200	200	0.850	0.994	440	920	239	238	**0.188 ± 0.015**	0.196 ± 0.016	0.226 ± 0.006	1.20
675	50	300	300	0.850	0.996	675	1425	374	373	0.166 ± 0.020	**0.165 ± 0.024**	0.221 ± 0.009	1.34
960	60	420	420	0.850	0.997	960	2040	539	538	0.115 ± 0.015	**0.113 ± 0.021**	0.209 ± 0.015	1.85
1295	70	560	560	0.850	0.998	1295	2765	734	733	**0.004 ± 0.021**	0.005 ± 0.027	0.199 ± 0.028	49
1680	80	720	720	0.850	0.998	1680	3600	959	958	**0.000 ± 0.000**	0.000 ± 0.000	0.187 ± 0.040	100+
2115	90	900	900	0.850	0.999	2115	4545	1214	1213	0.001 ± 0.007	**0.000 ± 0.000**	0.156 ± 0.045	100+

Table B.24: *Equal readout equal resource experiment with MackeyGlass dataset used in Memorize task performance at 85.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
120	20	60	**0.2265 ± 0.0075**	0.2318 ± 0.001327	0.0006011	1	Yes
255	30	120	**0.2146 ± 0.01029**	0.2301 ± 0.00184	3.834e-09	1	Yes
440	40	200	**0.1879 ± 0.01468**	0.2261 ± 0.006471	6.184e-16	1	Yes
675	50	300	**0.1657 ± 0.02016**	0.2215 ± 0.009136	6.672e-17	1	Yes
960	60	420	**0.1153 ± 0.01507**	0.2095 ± 0.01517	6.245e-32	1	Yes
1295	70	560	**0.004037 ± 0.02056**	0.1992 ± 0.02834	2.853e-35	1	Yes
1680	80	720	**2.049e-10 ± 4.101e-10**	0.1868 ± 0.03964	1.475e-21	1	Yes
2115	90	900	**0.001306 ± 0.007155**	0.1559 ± 0.04479	3.128e-18	1	Yes

Table B.25: *Mackey-Glass Dataset memorization results are obtained with equalized resource and readout layer size at 0% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
460	20	60	60	0.000	0.889	460	580	399	398	**0.224 ± 0.008**	0.226 ± 0.005	0.226 ± 0.006	1.01
1020	30	120	120	0.000	0.938	1020	1290	899	898	0.211 ± 0.006	**0.210 ± 0.010**	0.214 ± 0.007	1.02
1800	40	200	200	0.000	0.960	1800	2280	1599	1598	0.190 ± 0.007	**0.188 ± 0.009**	0.191 ± 0.008	1.02
2800	50	300	300	0.000	0.972	2800	3550	2499	2498	0.161 ± 0.008	0.161 ± 0.012	**0.154 ± 0.010**	0.955
4020	60	420	420	0.000	0.980	4020	5100	3599	3598	0.115 ± 0.013	0.115 ± 0.026	**0.098 ± 0.012**	0.851
5460	70	560	560	0.000	0.984	5460	6930	4899	4898	0.005 ± 0.027	0.008 ± 0.035	**0.000 ± 0.000**	0.000
7120	80	720	720	0.000	0.988	7120	9040	6399	6398	0.006 ± 0.033	0.000 ± 0.000	**0.000 ± 0.000**	0.003
9000	90	900	900	0.000	0.990	9000	11430	8099	8098	0.000 ± 0.000	0.000 ± 0.000	**0.000 ± 0.000**	0.575

Table B.26: *Equal readout equal resource experiment with MackeyGlass dataset used in Memorize task performance at 0.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
460	20	60	0.2241 ± 0.008347	0.2256 ± 0.006104	0.4415	0	No
1020	30	120	0.2108 ± 0.006385	0.2135 ± 0.006937	0.1188	0	No
1800	40	200	0.1901 ± 0.00719	0.1913 ± 0.008082	0.5401	0	No
2800	50	300	0.1609 ± 0.007951	**0.1536 ± 0.009716**	0.002331	1	Yes
4020	60	420	0.1146 ± 0.01316	**0.09757 ± 0.01224**	2.717e-06	1	Yes
5460	70	560	0.005011 ± 0.02744	8.244e-10 ± 2.445e-10	0.3256	0	No
7120	80	720	0.006072 ± 0.03326	1.015e-10 ± 1.277e-11	0.3256	0	No
9000	90	900	7.551e-11 ± 1.953e-10	4.342e-11 ± 4.707e-12	0.3756	0	No

Table B.27: *Lorenz Dataset memorization results are obtained with equalized resource and readout layer size at 85% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
240	20	60	60	0.850	0.983	240	360	177	174	0.301 ± 0.011	**0.294 ± 0.010**	0.544 ± 0.265	1.85
495	30	120	120	0.850	0.991	495	765	372	369	**0.287 ± 0.021**	0.288 ± 0.027	0.465 ± 0.246	1.62
840	40	200	200	0.850	0.994	840	1320	637	634	**0.258 ± 0.013**	0.271 ± 0.038	0.459 ± 0.236	1.78
1275	50	300	300	0.850	0.996	1275	2025	972	969	**0.224 ± 0.011**	0.234 ± 0.025	0.436 ± 0.238	1.95
1800	60	420	420	0.850	0.997	1800	2880	1377	1374	**0.155 ± 0.012**	0.159 ± 0.015	0.422 ± 0.222	2.72
2415	70	560	560	0.850	0.998	2415	3885	1852	1849	0.008 ± 0.030	**0.000 ± 0.000**	0.330 ± 0.179	100+
3120	80	720	720	0.850	0.998	3120	5040	2397	2394	**0.009 ± 0.048**	0.009 ± 0.049	0.350 ± 0.202	40
3915	90	900	900	0.850	0.999	3915	6345	3012	3009	**0.000 ± 0.000**	0.000 ± 0.000	0.255 ± 0.096	100+

Table B.28: *Equal readout equal resource experiment with Lorenz dataset used in Memorize task performance at 85.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
240	20	60	**0.3011 ± 0.01122**	0.5438 ± 0.2653	2.467e-05	1	Yes
495	30	120	**0.2873 ± 0.02098**	0.4653 ± 0.2456	0.0004426	1	Yes
840	40	200	**0.2581 ± 0.01343**	0.4589 ± 0.2361	6.608e-05	1	Yes
1275	50	300	**0.2238 ± 0.01057**	0.4364 ± 0.2384	3.497e-05	1	Yes
1800	60	420	**0.1548 ± 0.01201**	0.4217 ± 0.2215	3.121e-07	1	Yes
2415	70	560	**0.007807 ± 0.02967**	0.3303 ± 0.1794	7.308e-11	1	Yes
3120	80	720	**0.008684 ± 0.04756**	0.3498 ± 0.2019	2.606e-10	1	Yes
3915	90	900	**5.11e-09 ± 1.611e-08**	0.2546 ± 0.09579	7.286e-15	1	Yes

Table B.29: *Lorenz Dataset memorization results are obtained with equalized resource and readout layer size at 50% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
380	20	60	60	0.500	0.944	380	500	317	314	**0.308 ± 0.041**	0.324 ± 0.054	0.582 ± 0.238	1.89
810	30	120	120	0.500	0.969	810	1080	687	684	**0.300 ± 0.028**	0.312 ± 0.043	0.638 ± 0.189	2.13
1400	40	200	200	0.500	0.980	1400	1880	1197	1194	**0.269 ± 0.011**	0.298 ± 0.038	0.597 ± 0.145	2.22
2150	50	300	300	0.500	0.986	2150	2900	1847	1844	**0.227 ± 0.017**	0.245 ± 0.028	0.532 ± 0.044	2.35
3060	60	420	420	0.500	0.990	3060	4140	2637	2634	**0.160 ± 0.011**	0.165 ± 0.031	0.329 ± 0.063	2.06
4130	70	560	560	0.500	0.992	4130	5600	3567	3564	0.024 ± 0.074	0.001 ± 0.003	**0.000 ± 0.000**	0.000
5360	80	720	720	0.500	0.994	5360	7280	4637	4634	**0.000 ± 0.000**	0.000 ± 0.000	0.000 ± 0.000	1.35
6750	90	900	900	0.500	0.995	6750	9180	5847	5844	0.005 ± 0.018	**0.000 ± 0.000**	0.000 ± 0.000	1.58

Table B.30: *Rössler Dataset memorization results are obtained with equalized resource and readout layer size at 85% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
240	20	60	60	0.850	0.983	240	360	177	174	0.335 ± 0.009	**0.334 ± 0.008**	0.536 ± 0.216	1.60
495	30	120	120	0.850	0.991	495	765	372	369	**0.322 ± 0.018**	0.332 ± 0.018	0.562 ± 0.214	1.75
840	40	200	200	0.850	0.994	840	1320	637	634	**0.291 ± 0.028**	0.306 ± 0.024	0.447 ± 0.189	1.53
1275	50	300	300	0.850	0.996	1275	2025	972	969	**0.246 ± 0.038**	0.254 ± 0.040	0.422 ± 0.168	1.71
1800	60	420	420	0.850	0.997	1800	2880	1377	1374	**0.169 ± 0.036**	0.176 ± 0.051	0.370 ± 0.122	2.19
2415	70	560	560	0.850	0.998	2415	3885	1852	1849	**0.014 ± 0.047**	0.022 ± 0.076	0.360 ± 0.126	25
3120	80	720	720	0.850	0.998	3120	5040	2397	2394	**0.000 ± 0.000**	0.022 ± 0.083	0.361 ± 0.137	100+
3915	90	900	900	0.850	0.999	3915	6345	3012	3009	0.010 ± 0.057	**0.000 ± 0.000**	0.275 ± 0.146	100+

Table B.31: *Rössler Dataset memorization results are obtained with equalized resource and readout layer size at 50% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
380	20	60	60	0.500	0.944	380	500	317	314	**0.338 ± 0.026**	0.371 ± 0.051	0.586 ± 0.178	1.74
810	30	120	120	0.500	0.969	810	1080	687	684	**0.322 ± 0.015**	0.338 ± 0.035	0.570 ± 0.159	1.77
1400	40	200	200	0.500	0.980	1400	1880	1197	1194	**0.287 ± 0.016**	0.308 ± 0.024	0.551 ± 0.115	1.92
2150	50	300	300	0.500	0.986	2150	2900	1847	1844	**0.238 ± 0.033**	0.250 ± 0.026	0.488 ± 0.064	2.05
3060	60	420	420	0.500	0.990	3060	4140	2637	2634	**0.173 ± 0.042**	0.175 ± 0.037	0.307 ± 0.060	1.77
4130	70	560	560	0.500	0.992	4130	5600	3567	3564	0.000 ± 0.003	0.000 ± 0.000	**0.000 ± 0.000**	0.004
5360	80	720	720	0.500	0.994	5360	7280	4637	4634	**0.000 ± 0.000**	0.007 ± 0.036	0.000 ± 0.000	3.49
6750	90	900	900	0.500	0.995	6750	9180	5847	5844	0.000 ± 0.000	0.000 ± 0.000	**0.000 ± 0.000**	0.532

Table B.32: *Chua Circuit Dataset memorization results are obtained with equalized resource and readout layer size at 85% sparse haRC. Please refer to figure 4.6b.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
240	20	60	60	0.850	0.983	240	360	177	174	**0.448 ± 0.010**	0.453 ± 0.008	0.634 ± 0.250	1.42
495	30	120	120	0.850	0.991	495	765	372	369	**0.429 ± 0.018**	0.435 ± 0.029	0.620 ± 0.243	1.45
840	40	200	200	0.850	0.994	840	1320	637	634	**0.375 ± 0.027**	0.396 ± 0.030	0.621 ± 0.242	1.65
1275	50	300	300	0.850	0.996	1275	2025	972	969	**0.316 ± 0.028**	0.331 ± 0.041	0.554 ± 0.202	1.75
1800	60	420	420	0.850	0.997	1800	2880	1377	1374	0.220 ± 0.019	**0.220 ± 0.023**	0.527 ± 0.197	2.40
2415	70	560	560	0.850	0.998	2415	3885	1852	1849	0.059 ± 0.143	**0.007 ± 0.037**	0.513 ± 0.202	77
3120	80	720	720	0.850	0.998	3120	5040	2397	2394	**0.011 ± 0.062**	0.022 ± 0.087	0.451 ± 0.186	40
3915	90	900	900	0.850	0.999	3915	6345	3012	3009	0.010 ± 0.056	**0.000 ± 0.000**	0.375 ± 0.121	100+

Table B.33: *Equal readout equal resource experiment with Chua dataset used in Memorize task performance at 85.0% sparsity level, with significance checks of (Welch's t-test, $\alpha = 0.05$).*

Memory Resource	haRC Size	sRC Size	Error haRC	Error sRC	p-value	h	Signif.
240	20	60	**0.448 ± 0.01019**	0.6345 ± 0.2502	0.0003216	1	Yes
495	30	120	**0.4285 ± 0.01826**	0.6201 ± 0.2431	0.0001706	1	Yes
840	40	200	**0.3754 ± 0.02702**	0.6205 ± 0.2415	5.462e-06	1	Yes
1275	50	300	**0.3162 ± 0.02797**	0.5545 ± 0.2017	4.411e-07	1	Yes
1800	60	420	**0.2204 ± 0.01922**	0.5271 ± 0.1975	2.149e-09	1	Yes
2415	70	560	**0.05909 ± 0.1432**	0.5133 ± 0.2018	8.117e-14	1	Yes
3120	80	720	**0.01134 ± 0.06212**	0.4509 ± 0.1859	2.554e-14	1	Yes
3915	90	900	**0.01024 ± 0.05608**	0.3754 ± 0.1209	3.135e-18	1	Yes

Table B.34: *Chua Equal Output results memorization sparsity 50% please refer to figure 4.6a.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
380	20	60	60	0.500	0.944	380	500	317	314	**0.454 ± 0.024**	0.465 ± 0.044	0.736 ± 0.224	1.62
810	30	120	120	0.500	0.969	810	1080	687	684	**0.429 ± 0.020**	0.441 ± 0.031	0.774 ± 0.172	1.81
1400	40	200	200	0.500	0.980	1400	1880	1197	1194	**0.378 ± 0.019**	0.398 ± 0.034	0.701 ± 0.144	1.85
2150	50	300	300	0.500	0.986	2150	2900	1847	1844	**0.320 ± 0.021**	0.322 ± 0.027	0.573 ± 0.118	1.79
3060	60	420	420	0.500	0.990	3060	4140	2637	2634	**0.224 ± 0.041**	0.231 ± 0.032	0.397 ± 0.067	1.77
4130	70	560	560	0.500	0.992	4130	5600	3567	3564	0.000 ± 0.000	0.000 ± 0.000	**0.000 ± 0.000**	0.124
5360	80	720	720	0.500	0.994	5360	7280	4637	4634	0.009 ± 0.047	**0.000 ± 0.000**	0.000 ± 0.000	1.55
6750	90	900	900	0.500	0.995	6750	9180	5847	5844	0.000 ± 0.000	0.005 ± 0.025	**0.000 ± 0.000**	0.571

Table B.35: *Ikeda Dataset memorization results are obtained with equalized resource and readout layer size at 85% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
180	20	60	60	0.850	0.983	180	300	118	116	**0.421 ± 0.013**	0.428 ± 0.020	0.524 ± 0.147	1.24
375	30	120	120	0.850	0.991	375	645	253	251	**0.401 ± 0.016**	0.408 ± 0.021	0.529 ± 0.150	1.32
640	40	200	200	0.850	0.994	640	1120	438	436	**0.358 ± 0.029**	0.367 ± 0.021	0.503 ± 0.142	1.40
975	50	300	300	0.850	0.996	975	1725	673	671	**0.306 ± 0.029**	0.310 ± 0.033	0.458 ± 0.122	1.50
1380	60	420	420	0.850	0.997	1380	2460	958	956	0.224 ± 0.048	**0.203 ± 0.010**	0.456 ± 0.125	2.25
1855	70	560	560	0.850	0.998	1855	3325	1293	1291	0.013 ± 0.063	**0.009 ± 0.052**	0.448 ± 0.118	47
2400	80	720	720	0.850	0.998	2400	4320	1678	1676	0.026 ± 0.081	**0.014 ± 0.075**	0.405 ± 0.143	29
3015	90	900	900	0.850	0.999	3015	5445	2113	2111	0.007 ± 0.036	**0.000 ± 0.000**	0.338 ± 0.111	100+

Table B.36: *Ikeda Dataset memorization results are obtained with equalized resource and readout layer size at 50% sparse haRC.*

Memory Resource	haRC Size	sRC Size	Readout Size	haRC Sparsity	sRC Sparsity	sRC Mult Resource	haRC Mult Resource	sRC Summ Resource	haRC Summ Resource	Random haRC Error ± STD	Selective haRC Error ± STD	sRC Error ± STD	Winning Ratio
320	20	60	60	0.500	0.944	320	440	258	256	**0.424 ± 0.017**	0.437 ± 0.037	0.627 ± 0.131	1.48
690	30	120	120	0.500	0.969	690	960	568	566	**0.398 ± 0.013**	0.410 ± 0.020	0.615 ± 0.099	1.55
1200	40	200	200	0.500	0.980	1200	1680	998	996	**0.360 ± 0.019**	0.362 ± 0.019	0.582 ± 0.058	1.61
1850	50	300	300	0.500	0.986	1850	2600	1548	1546	**0.300 ± 0.023**	0.310 ± 0.034	0.456 ± 0.074	1.52
2640	60	420	420	0.500	0.990	2640	3720	2218	2216	**0.209 ± 0.008**	0.213 ± 0.017	0.320 ± 0.026	1.54
3570	70	560	560	0.500	0.992	3570	5040	3008	3006	0.000 ± 0.000	0.004 ± 0.022	**0.000 ± 0.000**	0.174
4640	80	720	720	0.500	0.994	4640	6560	3918	3916	0.004 ± 0.023	0.006 ± 0.032	**0.000 ± 0.000**	0.000
5850	90	900	900	0.500	0.995	5850	8280	4948	4946	0.000 ± 0.000	**0.000 ± 0.000**	0.000 ± 0.000	1.15

APPENDIX C. NOISE ROBUSTNESS AND REGULARIZATION EXPERIMENTS

C.1 Noise Robustness Ridge Parameter Evaluation Experiments

These experiments were made to empirically evaluate the best regularization parameter to be used in the Noise Robustness and equal output experiments. They are derived for different datasets as well, Following visuals show the Mackey Glass dataset evaluations.

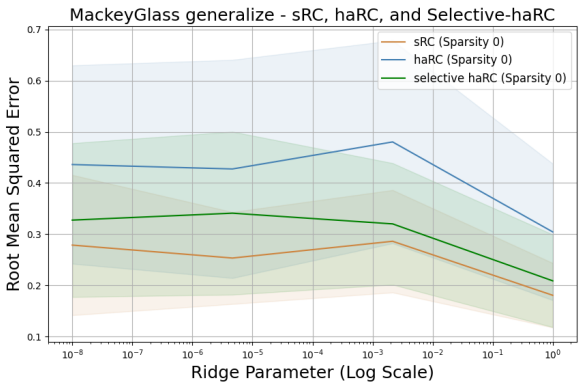


Figure C.1: *Reservoir Computers are tested on dynamic system learning task with added in-reservoir Gaussian noise of 0.05 on non-sparse reservoirs.*

Figure C.2: *Reservoir Computers are tested on dynamic system learning task with added in-reservoir Gaussian noise of 0.05 on 90% sparse reservoirs.*

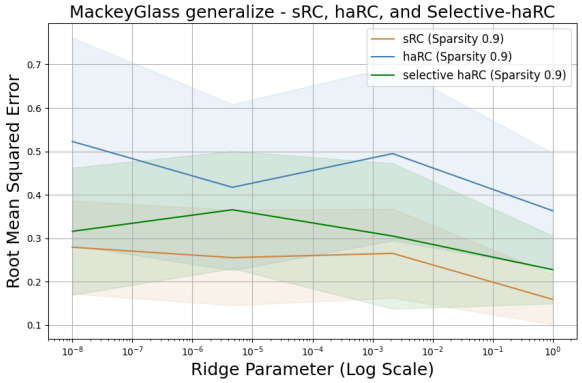


Figure C.3: *Reservoir Computers are tested on dynamic system learning task with added in-reservoir Gaussian noise of 0.1 on non-sparse reservoirs.*

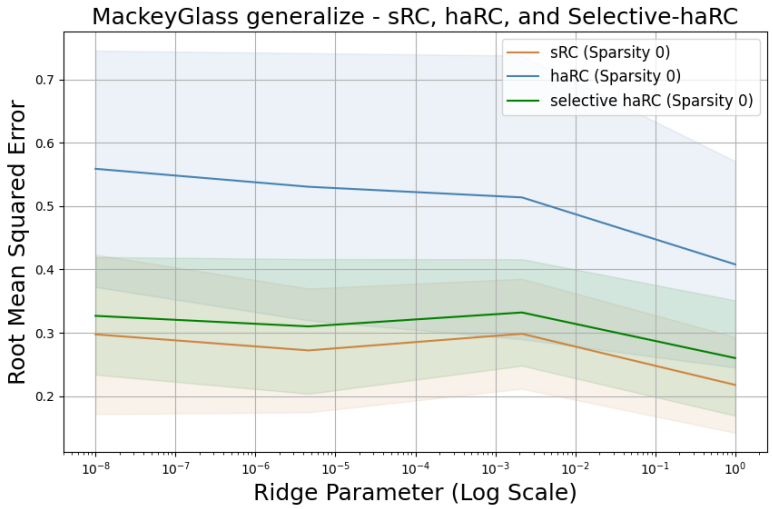
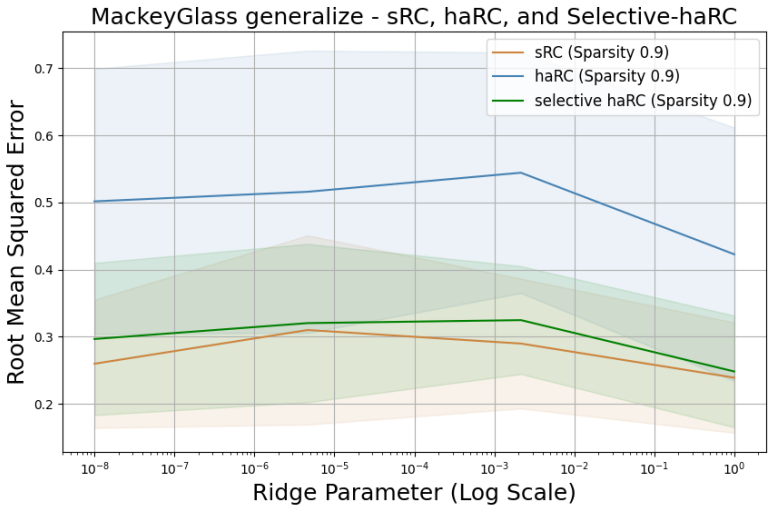


Figure C.4: *Reservoir Computers are tested on dynamic system learning task with added in-reservoir Gaussian noise of 0.1 on 90% sparse reservoirs.*



REFERENCES

[1] B. L. Welch, "The generalization of 'student's'problem when several different population varlances are involved," *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 1947.

[2] S. H. Strogatz, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. Chapman and Hall/CRC, 2024.

[3] D. ASTRONOMY, "Celestial mechanics and dynamical astronomy,"

[4] R. V. Rohli, C. Li, R. V. Rohli, and C. Li, "The seven basic equations in weather forecasting models," *Meteorology for Coastal Scientists*, pp. 171–185, 2021.

[5] K. Miyakoda and R. Moyer, "A method of initialization for dynamical weather forecasting," *Tellus*, vol. 20, no. 1, pp. 115–128, 1968.

[6] H. Lomax, T. H. Pulliam, D. W. Zingg, and T. Kowalewski, "Fundamentals of computational fluid dynamics," *Appl. Mech. Rev.*, vol. 55, no. 4, pp. B61–B61, 2002.

[7] K. Engelborghs, V. Lemaire, J. Belair, and D. Roose, "Numerical bifurcation analysis of delay differential equations arising from physiological modeling," *Journal of mathematical biology*, vol. 42, no. 4, pp. 361–385, 2001.

[8] E. Platen and N. Bruti-Liberati, *Numerical solution of stochastic differential equations with jumps in finance*, vol. 64. Springer Science & Business Media, 2010.

[9] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of political economy*, vol. 81, no. 3, pp. 637–654, 1973.

[10] N. McClamroch, "Singular systems of differential equations as dynamic models for constrained robot systems," in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 21–28, IEEE, 1986.

[11] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[12] D. Wilcox, "Turbulence modeling-an overview," in *39th aerospace sciences meeting and exhibit*, p. 724, 2001.

[13] I. Georgiev, V. Centeno, V. Mihova, and V. Pavlov, "A modified ordinary differential equation approach in price forecasting," in *Aip conference proceedings*, vol. 2459, AIP Publishing, 2022.

[14] C. Cobelli, G. Federspil, G. Pacini, A. Salvan, and C. Scandellari, "An integrated mathematical model of the dynamics of blood glucose and its hormonal control," *Mathematical Biosciences*, vol. 58, no. 1, pp. 27–60, 1982.

[15] B. Denis, "An overview of numerical and analytical methods for solving ordinary differential equations," *arXiv preprint arXiv:2012.07558*, 2020.

[16] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, "Learning data-driven discretizations for partial differential equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 31, pp. 15344–15349, 2019.

[17] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science advances*, vol. 3, no. 4, p. e1602614, 2017.

[18] J. Kim, H. Kim, H. Kim, D. Lee, and S. Yoon, "A comprehensive survey of deep learning for time series forecasting: architectural diversity and open challenges," *Artificial Intelligence Review*, vol. 58, no. 7, pp. 1–95, 2025.

[19] P. R. Vlachas, G. Arampatzis, C. Uhler, and P. Koumoutsakos, "Multiscale simulations of complex systems by learning their effective dynamics," *Nature Machine Intelligence*, vol. 4, no. 4, pp. 359–366, 2022.

[20] R. Yu and R. Wang, "Learning dynamical systems from data: An introduction to physics-guided deep learning," *Proceedings of the National Academy of Sciences*, vol. 121, no. 27, p. e2311808121, 2024.

[21] V. G. Kulkarni, *Modeling and analysis of stochastic systems*. Chapman and Hall/CRC, 2016.

[22] H. Poincaré, *Les méthodes nouvelles de la mécanique céleste*, vol. 2. Gauthier-Villars et fils, imprimeurs-libraires, 1893.

[23] L. Boltzmann, *Lectures on gas theory*. Univ of California Press, 2022.

[24] L. Pujo-Menjouet, "Blood cell dynamics: half of a century of modelling," *Mathematical Modelling of Natural Phenomena*, vol. 11, no. 1, pp. 92–115, 2016.

[25] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, no. 4300, pp. 287–289, 1977.

[26] K. McGoff, S. Mukherjee, and N. Pillai, "Statistical inference for dynamical systems: A review," *Statistics Surveys*, vol. 9, no. none, pp. 209 – 252, 2015.

[27] R. P. Masini, M. C. Medeiros, and E. F. Mendes, "Machine learning advances for time series forecasting," *Journal of Economic Surveys*, vol. 37, pp. 76–111, Feb. 2023.

[28] N. Sapankevych and R. Sankar, "Time Series Prediction Using Support Vector Machines: A Survey," *IEEE Computational Intelligence Magazine*, vol. 4, pp. 24–38, May 2009.

[29] J. F. Torres, D. Hadjout, A. Sebaa, F. Martínez-Álvarez, and A. Troncoso, "Deep Learning for Time Series Forecasting: A Survey," *Big Data*, vol. 9, pp. 3–21, Feb. 2021.

[30] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[31] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006.

[32] M. Stitson, J. Weston, A. Gammerman, V. Vovk, and V. Vapnik, "Theory of support vector machines," *University of London*, vol. 117, no. 827, pp. 188–191, 1996.

[33] A. Gasparin, S. Lukovic, and C. Alippi, "Deep learning for time series forecasting: The electric load case," *CAAI Transactions on Intelligence Technology*, vol. 7, no. 1, pp. 1–25, 2022.

[34] J. Z. Kim, Z. Lu, E. Nozari, G. J. Pappas, and D. S. Bassett, "Teaching recurrent neural networks to infer global temporal structure from local examples," *Nature Machine Intelligence*, vol. 3, pp. 316–323, Apr. 2021.

[35] M. Kaur and A. Mohta, "A review of deep learning with recurrent neural network," in *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 460–465, IEEE, 2019.

[36] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," *arXiv preprint arXiv:1912.05911*, 2019.

[37] R. Laje and D. V. Buonomano, "Robust timing and motor patterns by taming chaos in recurrent neural networks," *Nature neuroscience*, vol. 16, no. 7, pp. 925–933, 2013.

[38] P. F. Dominey, "Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning," *Biological cybernetics*, vol. 73, no. 3, pp. 265–274, 1995.

[39] K.-i. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural networks*, vol. 6, no. 6, pp. 801–806, 1993.

[40] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[41] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[42] F. Wong, "Time series forecasting using backpropagation neural networks," *Neurocomputing*, vol. 2, pp. 147–159, July 1991.

[43] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, Oct. 1990.

[44] G. Lachtermacher and J. D. Fuller, "Back propagation in time-series forecasting," *Journal of forecasting*, vol. 14, no. 4, pp. 381–393, 1995.

[45] X. Wang and E. K. Blum, "Dynamics and bifurcation of neural networks," 1995.

[46] K. Doya, "Bifurcations of recurrent neural networks in gradient descent learning," *IEEE Transactions on neural networks*, vol. 1, no. 75, p. 218, 1993.

[47] U. D. Schiller and J. J. Steil, "Analyzing the weight dynamics of recurrent learning algorithms," *Neurocomputing*, vol. 63, pp. 5–23, 2005.

[48] A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves, "Memory-efficient backpropagation through time," *Advances in neural information processing systems*, vol. 29, 2016.

[49] L. Manneschi and E. Vasilaki, "An alternative to backpropagation through time," *Nature Machine Intelligence*, vol. 2, no. 3, pp. 155–156, 2020.

[50] R. Liao, Y. Xiong, E. Fetaya, L. Zhang, K. Yoon, X. Pitkow, R. Urtasun, and R. Zemel, "Reviving and improving recurrent back-propagation," in *International conference on machine learning*, pp. 3082–3091, PMLR, 2018.

[51] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[52] E. A. Wan, "Discrete time neural networks," *Applied Intelligence*, vol. 3, pp. 91–105, 1993.

[53] H. Jaeger and H. Haas, "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication," *Science*, vol. 304, pp. 78–80, Apr. 2004.

[54] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German national research center for information technology gmd technical report*, vol. 148, no. 34, p. 13, 2001.

[55] W. Maass, "Liquid State Machines: Motivation, Theory, and Applications," in *Computability in Context*, pp. 275–296, IMPERIAL COLLEGE PRESS, Feb. 2011.

[56] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade: Second Edition*, pp. 659–686, Springer, 2012.

[57] K. Rajan, L. Abbott, and H. Sompolinsky, "Stimulus-dependent suppression of chaos in recurrent neural networks," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 82, no. 1, p. 011903, 2010.

[58] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, "Reservoir observers: Model-free inference of unmeasured variables in chaotic systems," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 4, 2017.

[59] 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*, vol. 115, pp. 100–123, 2019.

[60] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee, and W. D. Lu, "Reservoir computing using dynamic memristors for temporal information processing," *Nature communications*, vol. 8, no. 1, p. 2204, 2017.

[61] A. Morán, V. Canals, F. Galan-Prado, C. F. Frasser, D. Radhakrishnan, S. Safavi, and J. L. Rosselló, "Hardware-optimized reservoir computing system for edge intelligence applications," *Cognitive Computation*, pp. 1–9, 2023.

[62] Z. Wang, S. Joshi, S. Savel'Ev, W. Song, R. Midya, Y. Li, M. Rao, P. Yan, S. Asapu, Y. Zhuo, *et al.*, "Fully memristive neural networks for pattern classification with unsupervised learning," *Nature Electronics*, vol. 1, no. 2, pp. 137–145, 2018.

[63] P. Mujal, R. Martínez-Peña, J. Nokkala, J. García-Beni, G. L. Giorgi, M. C. Soriano, and R. Zambrini, "Opportunities in Quantum Reservoir Computing and Extreme Learning Machines," *Advanced Quantum Technologies*, vol. 4, p. 2100027, Aug. 2021.

[64] S. Wang, Y. Li, D. Wang, W. Zhang, X. Chen, D. Dong, S. Wang, X. Zhang, P. Lin, C. Gallicchio, X. Xu, Q. Liu, K.-T. Cheng, Z. Wang, D. Shang, and M. Liu, "Echo state graph neural networks with analogue random resistive memory arrays," *Nature Machine Intelligence*, vol. 5, pp. 104–113, Feb. 2023.

[65] H. Wang, J. Hu, Y. Baek, K. Tsuchiyama, M. Joly, Q. Liu, and S. Gigan, "Optical next generation reservoir computing," *arXiv preprint arXiv:2404.07857*, 2024.

[66] T. Matsuo, D. Sato, S.-G. Koh, H. Shima, Y. Naitoh, H. Akinaga, T. Itoh, T. Nokami, M. Kobayashi, and K. Kinoshita, "Dynamic Nonlinear Behavior of Ionic Liquid-Based Reservoir Computing Devices," *ACS Applied Materials & Interfaces*, vol. 14, pp. 36890–36901, Aug. 2022.

[67] K. Nakajima, "Physical reservoir computing—an introductory perspective," *Japanese Journal of Applied Physics*, vol. 59, p. 060501, June 2020.

[68] M. Inubushi and K. Yoshimura, "Reservoir Computing Beyond Memory-Nonlinearity Trade-off," *Scientific Reports*, vol. 7, p. 10199, Aug. 2017.

[69] L. Cerina, M. D. Santambrogio, G. Franco, C. Gallicchio, and A. Micheli, "EchoBay: Design and Optimization of Echo State Networks under Memory and Time Constraints," *ACM Transactions on Architecture and Code Optimization*, vol. 17, pp. 1–24, Sept. 2020.

[70] H. Cui, X. Liu, and L. Li, "The architecture of dynamic reservoir in the echo state network," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, p. 033127, Sept. 2012.

[71] B. Çelebi, M. Asada, and E. Oztop, "Augmenting Reservoirs with Higher Order Terms for Resource Efficient Learning," in *2024 International Joint Conference on Neural Networks (IJCNN)*, (Yokohama, Japan), pp. 1–8, IEEE, June 2024.

[72] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa, "Next generation reservoir computing," *Nature communications*, vol. 12, no. 1, pp. 1–8, 2021.

[73] A. Ohkubo and M. Inubushi, "Reservoir computing with generalized readout based on generalized synchronization," *Scientific Reports*, vol. 14, p. 30918, Dec. 2024.

[74] T. Kubota, Y. Imai, S. Tsunegi, and K. Nakajima, "Reservoir Computing Generalized," 2024.

[75] E. Oztop, "Sign-representation of Boolean functions using a small number of monomials," *Neural Networks*, vol. 22, pp. 938–948, Sept. 2009.

[76] S. Lawrie, R. Moreno-Bote, and M. Gilson, "Covariance-based information processing in reservoir computing systems," *Biorxiv*, pp. 2021–04, 2021.

[77] J. Jaurigue, J. Robertson, A. Hurtado, L. Jaurigue, and K. Lüdge, "Post-processing methods for delay embedding and feature scaling of reservoir computers," *Communications Engineering*, vol. 4, no. 1, p. 10, 2025.

[78] E. N. Lorenz, "Deterministic Nonperiodic Flow," *Journal of the Atmospheric Sciences*, vol. 20, pp. 130–141, Mar. 1963.

[79] E. De Santis, M. D. Di Benedetto, *et al.*, "Observability of hybrid dynamical systems," *Foundations and Trends® in Systems and Control*, vol. 3, no. 4, pp. 363–540, 2016.

[80] L. A. Aguirre, L. L. Portes, and C. Letellier, "Structural, dynamical and symbolic observability: From dynamical systems to networks," *PLoS One*, vol. 13, no. 10, p. e0206180, 2018.

[81] C. W. Rowley and S. T. Dawson, "Model reduction for flow analysis and control," *Annual Review of Fluid Mechanics*, vol. 49, no. 1, pp. 387–417, 2017.

[82] B. M. Lucey and M. Dowling, "The role of feelings in investor decision-making," *Journal of economic surveys*, vol. 19, no. 2, pp. 211–237, 2005.

[83] K. S. Krane, *Introductory nuclear physics*. John Wiley & Sons, 1991.

[84] E. Rutherford and F. Soddy, "Xli. the cause and nature of radioactivity.—part i," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 4, no. 21, pp. 370–396, 1902.

[85] E. Ott, *Chaos in dynamical systems*. Cambridge university press, 2002.

[86] W. B. Arthur, "Complexity and the economy," in *Handbook of Research on Complexity*, Edward Elgar Publishing, 2009.

[87] B. B. Mandelbrot and B. B. Mandelbrot, *The variation of certain speculative prices*. Springer, 1997.

[88] J. C. Hull and S. Basu, *Options, futures, and other derivatives*. Pearson Education India, 2016.

[89] P. A. M. Dirac, *The principles of quantum mechanics*. No. 27, Oxford university press, 1981.

[90] W. Heisenberg, "Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik," *Zeitschrift für Physik*, vol. 43, no. 3, pp. 172–198, 1927.

[91] E. Schrödinger, "An undulatory theory of the mechanics of atoms and molecules," *Physical review*, vol. 28, no. 6, p. 1049, 1926.

[92] M. B. Elowitz and S. Leibler, "A synthetic oscillatory network of transcriptional regulators," *Nature*, vol. 403, no. 6767, pp. 335–338, 2000.

[93] A. Raj and A. Van Oudenaarden, "Nature, nurture, or chance: stochastic gene expression and its consequences," *Cell*, vol. 135, no. 2, pp. 216–226, 2008.

[94] S. A. Kauffman, "The origins of order: Self-organization and selection in evolution," in *Spin glasses and biology*, pp. 61–100, World Scientific, 1992.

[95] A. Vandesompele, G. Urbain, F. Wyffels, and J. Dambre, "Populations of spiking neurons for reservoir computing: Closed loop control of a compliant quadruped," *Cognitive Systems Research*, vol. 58, pp. 317–323, Dec. 2019.

[96] B. Lim and S. Zohren, "Time-series forecasting with deep learning: a survey," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, p. 20200209, 2021.

[97] T. Santos and R. Kern, "A literature survey of early time series classification and deep learning.," *Sami@ iknow*, 2016.

[98] J. Runge, S. Bathiany, E. Bollt, G. Camps-Valls, D. Coumou, E. Deyle, C. Glymour, M. Kretschmer, M. D. Mahecha, J. Muñoz-Marí, *et al.*, "Inferring causation from time series in earth system sciences," *Nature communications*, vol. 10, no. 1, p. 2553, 2019.

[99] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005-2019," 2019.

[100] B. Krollner, B. Vanstone, and G. Finnie, "Financial Time Series Forecasting with Machine Learning Techniques: A Survey," Jan. 2010.

[101] F. Martinez Alvarez, A. Troncoso, J. C. Riquelme, and J. S. Aguilar Ruiz, "Energy Time Series Forecasting Based on Pattern Sequence Similarity," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 1230–1243, Aug. 2011.

[102] P. Lara-Benítez, M. Carranza-García, J. M. Luna-Romera, and J. C. Riquelme, "Temporal Convolutional Networks Applied to Energy-Related Time Series Forecasting," *Applied Sciences*, vol. 10, p. 2322, Mar. 2020.

[103] N. Vakitbilir, L. Froese, A. Gomez, A. S. Sainbhi, K. Y. Stein, A. Islam, T. J. Bergmann, I. Marquez, F. Amenta, Y. Ibrahim, *et al.*, "Time-series modeling and forecasting of cerebral pressure–flow physiology: A scoping systematic review of the human and animal literature," *Sensors*, vol. 24, no. 5, p. 1453, 2024.

[104] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[105] Z. Zamanzadeh Darban, G. I. Webb, S. Pan, C. Aggarwal, and M. Salehi, "Deep learning for time series anomaly detection: A survey," *ACM Computing Surveys*, vol. 57, no. 1, pp. 1–42, 2024.

[106] J. Park, N. Yang, and N. Chandramoorthy, "When are dynamical systems learned from time series data statistically accurate?," *arXiv preprint arXiv:2411.06311*, 2024.

[107] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Information systems*, vol. 53, pp. 16–38, 2015.

[108] J. Renn, "Einstein's invention of brownian motion," *Annalen der Physik*, vol. 517, pp. 23–37, 2005.

[109] J. D. Hamilton, *Time series analysis*. Princeton university press, 2020.

[110] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*. Springer, 2002.

[111] X. Chen, H. Wang, Y. Wei, J. Li, and H. Gao, "Autoregressive-model-based methods for online time series prediction with missing values: an experimental evaluation," *arXiv preprint arXiv:1908.06729*, 2019.

[112] Z. Ivanovski, A. Milenkovski, and Z. Narasanov, "Time series forecasting using a moving average model for extrapolation of number of tourist," *UTMS Journal of Economics*, vol. 9, no. 2, pp. 121–132, 2018.

[113] R. Agrawal and R. Adhikari, "An introductory study on time series modeling and forecasting," *Nova York: CoRR*, pp. 200–212, 2013.

[114] P. Chen, A. Niu, D. Liu, W. Jiang, and B. Ma, "Time series forecasting of temperatures using sarima: An example from nanjing," in *IOP conference series: materials science and engineering*, vol. 394, p. 052024, IOP Publishing, 2018.

[115] E. Ostertagova and O. Ostertag, "Forecasting using simple exponential smoothing method," *Acta Electrotechnica et Informatica*, vol. 12, no. 3, p. 62, 2012.

[116] J.-X. Pan, K.-T. Fang, J.-X. Pan, and K.-T. Fang, "Maximum likelihood estimation," *Growth curve models and statistical diagnostics*, pp. 77–158, 2002.

[117] K. Pearson, "Method of moments and method of maximum likelihood," *Biometrika*, vol. 28, no. 1/2, pp. 34–59, 1936.

[118] V. N. Vapnik, "An overview of statistical learning theory," *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.

[119] G. Bontempi, S. Ben Taieb, and Y.-A. Le Borgne, "Machine learning strategies for time series forecasting," in *European Big Data Management and Analytics Summer School*, pp. 62–77, Springer, 2012.

[120] J. R. Quinlan, “Induction of decision trees,” *Machine learning*, vol. 1, pp. 81–106, 1986.

[121] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.

[122] T. G. Dietterich, “Ensemble methods in machine learning,” in *International workshop on multiple classifier systems*, pp. 1–15, Springer, 2000.

[123] Y. Grandvalet and Y. Bengio, “Semi-supervised learning by entropy minimization,” *Advances in neural information processing systems*, vol. 17, 2004.

[124] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, pp. 273–297, 1995.

[125] N. S. Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[126] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[127] S. Si, H. Zhang, S. S. Keerthi, D. Mahajan, I. S. Dhillon, and C.-J. Hsieh, “Gradient boosted decision trees for high dimensional sparse output,” in *International conference on machine learning*, pp. 3182–3190, PMLR, 2017.

[128] M. Weller, “Recurrent neural networks for time series forecasting,” *Novatec, Oct*, vol. 16, 2018.

[129] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[130] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[131] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[132] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need. dvances in neural information processing s stems," 2017.

[133] K. E. Hoque and H. Aljamaan, "Impact of hyperparameter tuning on machine learning models in stock price forecasting," *IEEE Access*, vol. 9, pp. 163815–163830, 2021.

[134] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence, Warwick 1980: proceedings of a symposium held at the University of Warwick 1979/80*, pp. 366–381, Springer, 2006.

[135] I. B. Yildiz, H. Jaeger, and S. J. Kiebel, "Re-visiting the echo state property," *Neural networks*, vol. 35, pp. 1–9, 2012.

[136] M. Buehner and P. Young, "A tighter bound for the echo state property," *IEEE transactions on neural networks*, vol. 17, no. 3, pp. 820–824, 2006.

[137] A. Ceni and C. Gallicchio, "Edge of Stability Echo State Network," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–10, 2024.

[138] Q. Ma, W. Chen, J. Wei, and Z. Yu, "Direct model of memory properties and the linear reservoir topologies in echo state networks," *Applied Soft Computing*, vol. 22, pp. 622–628, Sept. 2014.

[139] Y. Kawai, J. Park, and M. Asada, "A small-world topology enhances the echo state property and signal propagation in reservoir computing," *Neural Networks*, vol. 112, pp. 15–23, Apr. 2019.

[140] X. Sun, S. Ma, Y. Li, D. Wang, Z. Li, N. Wang, and G. Gui, "Enhanced echo-state restricted Boltzmann machines for network traffic prediction," *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 1287–1297, 2019.

[141] X. Na, W. Ren, and X. Xu, "Hierarchical delay-memory echo state network: A model designed for multi-step chaotic time series prediction," *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104229, June 2021.

[142] G. K. Venayagamoorthy and B. Shishir, "Effects of spectral radius and settling time in the performance of echo state networks," *Neural Networks*, vol. 22, pp. 861–863, Sept. 2009.

[143] M. Goldmann, F. Köster, K. Lüdge, and S. Yanchuk, "Deep time-delay reservoir computing: Dynamics and memory capacity," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 30, no. 9, 2020.

[144] X. Liu, M. Chen, C. Yin, and W. Saad, "Analysis of Memory Capacity for Deep Echo State Networks," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, (Orlando, FL), pp. 443–448, IEEE, Dec. 2018.

[145] K. Inoue, K. Nakajima, and Y. Kuniyoshi, "Designing spontaneous behavioral switching via chaotic itinerancy," *Science advances*, vol. 6, no. 46, p. eabb3989, 2020.

[146] S.-x. Lun, H.-f. Hu, and X.-s. Yao, "The modified sufficient conditions for echo state property and parameter optimization of leaky integrator echo state network," *Applied Soft Computing*, vol. 77, pp. 750–760, 2019.

[147] Y. Kawai, J. Park, I. Tsuda, and M. Asada, "Learning long-term motor timing/-patterns on an orthogonal basis in random neural networks," *Neural Networks*, vol. 163, pp. 298–311, June 2023.

[148] Y. Kawai, J. Park, and M. Asada, "Reservoir computing using self-sustained oscillations in a locally connected neural network," *Scientific Reports*, vol. 13, p. 15532, Sept. 2023.

[149] A. Racca and L. Magri, "Robust Optimization and Validation of Echo State Networks for learning chaotic dynamics," *Neural Networks*, vol. 142, pp. 252–268, Oct. 2021.

[150] Z. Zhang, Y. Zhu, X. Wang, and W. Yu, "Optimal echo state network parameters based on behavioural spaces," *Neurocomputing*, vol. 503, pp. 299–313, Sept. 2022.

[151] H.-T. Fan, W. Wang, and Z. Jin, "Performance optimization of echo state networks through principal neuron reinforcement," in *2017 International Joint Conference on Neural Networks (IJCNN)*, (Anchorage, AK, USA), pp. 1717–1723, IEEE, May 2017.

[152] B. Whiteaker and P. Gerstoft, "Reducing echo state network size with controllability matrices," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 32, p. 073116, July 2022.

[153] Y. Sakemi, K. Morino, T. Leleu, and K. Aihara, "Model-size reduction for reservoir computing by concatenating internal states through time," *Scientific Reports*, vol. 10, p. 21794, Dec. 2020.

[154] H. Jaeger, "Adaptive Nonlinear System Identification with Echo State Networks," in *Advances in Neural Information Processing Systems* (S. Becker, S. Thrun, and K. Obermayer, eds.), vol. 15, MIT Press, 2002.

[155] T. Chaminade, E. Oztop, G. Cheng, and M. Kawato, "From self-observation to imitation: Visuomotor association on a robotic hand," *Brain Research Bulletin*, vol. 75, pp. 775–784, Apr. 2008.

[156] B. Kathirgamanathan and P. Cunningham, "Correlation based feature subset selection for multivariate time-series data," *arXiv preprint arXiv:2112.03705*, 2021.

[157] L. N. Groschner, J. G. Malis, B. Zuidinga, and A. Borst, "A biophysical account of multiplication by a single neuron," *Nature*, vol. 603, no. 7899, pp. 119–123, 2022.

[158] E. Oztop, "An upper bound on the minimum number of monomials required to separate dichotomies of {- 1, 1} n," *Neural computation*, vol. 18, no. 12, pp. 3119–3138, 2006.

[159] R. Pyle and R. Rosenbaum, "A reservoir computing model of reward-modulated motor learning and automaticity," *Neural computation*, vol. 31, no. 7, pp. 1430–1461, 2019.

[160] B. N. Biswas, S. Chatterjee, S. Mukherjee, and S. Pal, "A discussion on euler method: A review," *Electronic Journal of Mathematical Analysis and Applications*, vol. 1, no. 2, pp. 2090–2792, 2013.

[161] J. H. Cartwright and O. Piro, "The dynamics of runge–kutta methods," *International Journal of Bifurcation and Chaos*, vol. 2, no. 03, pp. 427–449, 1992.

[162] G. Frigo, A. Derviškadić, A. Bach, and M. Paolone, "Statistical model of measurement noise in real-world pmu-based acquisitions," in *2019 International Conference on Smart Grid Synchronized Measurements and Analytics (SGSMA)*, pp. 1–8, IEEE, 2019.

[163] O. E. Rössler, “An equation for continuous chaos,” *Physics Letters A*, vol. 57, no. 5, pp. 397–398, 1976.

[164] M. Hénon, “A two-dimensional mapping with a strange attractor,” *The theory of chaotic attractors*, pp. 94–102, 2004.

[165] L. O. Chua, C. W. Wu, A. Huang, and G.-Q. Zhong, “A universal circuit for studying and generating chaos. i. routes to chaos,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 10, pp. 732–744, 1993.

[166] K. Ikeda, “Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system,” *Optics communications*, vol. 30, no. 2, pp. 257–261, 1979.

[167] S. Sahithi, “Multi timeframe ethusdt ohlcv data (2019–2024),” 2024. Accessed: 2025-08-13.

[168] A. Savitzky and M. J. E. Golay, “Smoothing and differentiation of data by simplified least squares procedures,” *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.