©Copyright 2026
Jan P. Williams

Data-driven recurrent models to forecast and control high-dimensional dynamical systems

Jan P. Williams

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2026

Reading Committee:

Krithika Manohar, Chair

J. Nathan Kutz, Chair

Stefania Fresca

Program Authorized to Offer Degree:
Department of Mechanical Engineering

University of Washington

Abstract

Data-driven recurrent models to forecast and control high-dimensional dynamical systems

Jan P. Williams

Co-Chairs of the Supervisory Committee:
Assistant Professor Krithika Manohar
Department of Mechanical Engineering

Professor J. Nathan Kutz
Departments of Applied Mathematics and Electrical and Computer Engineering

Many dynamical systems in science and engineering are high-dimensional, nonlinear, and only partially observable. Moreover, governing equations may be unknown and only limited observational data may be available. This dissertation develops recurrent neural network (RNN) methods that address three ubiquitous challenges in this setting: (i) state estimation from sparse sensors, (ii) data-driven surrogate modeling for control, and (iii) autonomous forecasting of high-dimensional dynamics. The key enabler throughout this work is the construction of recurrent latent structure. The recurrent architecture of RNNs provides an inductive bias well-matched to temporal dynamics, encoding high-dimensional nonlinear physics more expressively than linear projection-based techniques while learning far more data-efficiently than modern attention-based architectures such as transformers and foundation models.

This work develops temporally embedded shallow decoder and echo state network (ESN) architectures that enable high-dimensional estimation, control, and forecasting from partial, sparse sensor observations, with applications to turbulent fluid flows and chaotic dynamical systems. The SHallow REcurrent Decoder (SHRED) reconstructs high-dimensional states from time histories of sparse sensor measurements. By utilizing sequential embeddings of sensor measurements, SHRED leverages Takens' theorem and incorporates the temporal context that is absent from methods that reconstruct from static measurements alone, such

as gappy POD. Its nonlinear decoder also enables enhanced representation and compression capability. ESNs, a form of reservoir computer, are designed as surrogate models within model predictive control loops, demonstrated on lift control of flow past a cylinder and forecasting of chaotic dynamical systems. ESNs are orders of magnitude cheaper to train than standard RNN models, while remaining sufficiently lightweight to be embedded within optimization loops that require thousands of model evaluations per control step. To support reproducibility and community adoption, we develop OpenReservoirComputing, an open source, GPU accelerated reservoir computing library implemented in JAX. As one of the only autodiff-native software packages for ESNs, OpenReservoirComputing directly enables the use of ESNs within larger neural network models, as well as optimization and control loops.

Finally, the two methods are coupled to construct a reduced order model capable of forecasting directly from sparse sensor streams, following an offline training stage with full state information. SHRED maps sensor measurements to a learned latent representation, and the ESN propagates this latent state forward in time while allowing for the online assimilation of sensor measurements. This closes the loop between sensing and modeling, with preliminary results suggesting that the learned latent representation captures key geometric and statistical structure of the underlying attractor.

TABLE OF CONTENTS

Page

List of Figures iii

List of Tables ix

Chapter 1: Introduction 1

Chapter 2: Background 8
2.1 Dimensionality reduction and reduced order models 8
2.2 Recurrent Neural Networks 11
2.3 Reservoir Computing 16
2.4 Model Predictive Control 20

Chapter 3: Sensing with shallow recurrent decoder networks 23
3.1 Introduction 23
3.2 Background 27
3.3 Shallow recurrent decoder networks 29
3.4 Results 34
3.5 Discussion 47

Chapter 4: Reservoir computing for system identification and model predictive control 50
4.1 Introduction 50
4.2 Background 53
4.3 Methods 57
4.4 Forecasting Results 63
4.5 ESN Control Results 75
4.6 Discussion and Conclusion 80

Chapter 5: ORC: GPU-accelerated reservoir computing in JAX 83
5.1 Introduction 83

5.2 Statement of Need 84
5.3 State of the Field 85
5.4 Software Design 87
5.5 Research Impact Statement 89
5.6 RC Background 90
5.7 ORC Control 98
5.8 ORC Classification 105

Chapter 6: SHRED-ESN 112
6.1 Introduction 112
6.2 Methods 113
6.3 Experimental Systems 115
6.4 Results 116
6.5 Discussion 122

Chapter 7: Conclusion 126

Bibliography 129

LIST OF FIGURES

Figure Number Page

3.1 Summary diagram of SHallow REcurrent Decoder networks (SHRED) for flow reconstruction from sensor measurements. *A* A graphical representation of the SHRED method. The mulitvariate time-series of sensor measurements, $\{\mathbf{y}_i\}_{t-k}^{t}$, is fed into a stacked long short-term memory layer. The final output of the recurrent layer, $\mathbf{h}_t$, serves as an input to a fully-connected shallow decoder mapping from the hidden state to the high-dimensional field. *B* The time series of sensor measurements fed into the SHRED model. *C* The evolution of the output hidden state generated by the input sequence of sensor measurements. *D* Reconstruction errors of traditional methods (QR/POD), shallow decoders (SDN), and SHRED on a turbulent flow when three sensors are available. 24

3.2 Turbulent flow summary performance. *A* Example reconstructions obtained via SHRED and QR/POD of a turbulent flow when three sensors are available. Ground truth included for comparison. *B* Reconstruction errors of static sensing methods (QR/POD) and SHRED with a varying number of available sensors. The solid lines denote the median error from 32 trained estimators and the shaded region denotes the interquartile range. *C* Performance of reconstruction methods in the presence of varying levels of added Gaussian white noise. The added noise has mean zero and standard deviation equal to α times the mean absolute value of the field. The number in parentheses denotes the number of available sensors. 35

3.3 SST summary performance. *A* Example reconstructions obtained via SHRED and QR/POD of sea-surface temperature when three sensors are available. Ground truth included for comparison. *B* Reconstruction errors of static sensing methods (QR/POD) and SHRED with a varying number of available sensors. The solid lines denote the median error from 32 trained estimators and the shaded region denotes the interquartile range. *C* Performance of reconstruction methods in the presence of varying levels of added Gaussian white noise. The added noise has mean zero and standard deviation equal to α times the mean absolute value of the field. The number in parentheses denotes the number of available sensors. 39

3.4 Ozone summary performance. *A* Example reconstructions obtained via SHRED and QR/POD of atmospheric ozone concentration when three sensors are available. Only one of thirty elevations is shown. Ground truth included for comparison. *B* Reconstruction errors of static sensing methods (QR/POD) and SHRED with a varying number of available sensors. The solid lines denote the median error from 32 trained estimators and the shaded region denotes the interquartile range. *C* Performance of reconstruction methods in the presence of varying levels of added Gaussian white noise. The added noise has mean zero and standard deviation equal to α times the mean absolute value of the field. The number in parentheses denotes the number of available sensors. 41

3.5 Forecasting error for sea-surface temperature with Δt as one week. A LSTM is trained to forecast QR placed, panel *A*, or randomly placed, panel *B*, sensor measurements which are subsequently used to perform reconstructions using SHRED and gappy POD. 16 forecasts are performed and the median error is denoted by the solid lines, with the 25th and 75th percentiles of reconstruction error defining the shaded region. The dashed line represents an ensembled forecast. 42

3.6 Forecasting error for turbulent flow with $\Delta t = 0.006$s. A LSTM is trained to forecast QR placed, panel *A*, or randomly placed, panel *B*, sensor measurements which are subsequently used to perform reconstructions using SHRED and gappy POD. In this experiment, the validation set is selected from earlier in the data so that the forecast occurs immediately after following training data. 16 forecasts are performed and the median error is denoted by the solid lines, with the 25th and 75th percentiles of reconstruction error defining the shaded region. The dashed line represents an ensembled forecast. 43

3.7 Error results for SHRED models in the three considered datasets when the temporal spacing of the trajectories used for reconstruction is varied. k denotes the maximal time-delay of the trajectories used by a SHRED model. $n_{\Delta t}$ denotes the temporal spacing between utilized time-delays in terms of of the discretization of the data, Δt. The top row considers the case where k is approximately fixed to the values specified in Section 3.4.1 and the bottom row considers the case where k varies in addition to $n_{\Delta t}$ such that $k/n_{\Delta t}$ is fixed to the values of k specified in 3.4.1. Each plot shows the median and interquartile range of reconstruction error from 32 trained SHRED models. . 46

4.1 Summary diagram of echo state network (ESN) based model predictive control (MPC). An ESN trained to model plant dynamics acts as a surrogate model capable of rapidly forecasting the system under given control inputs. Online optimizations compute a sequence of control actions that minimize the deviation of the plant from a reference trajectory, yielding a controlled trajectory that tracks the reference. 51

4.2 Forecasting results for the linear spring mass system. Panel A shows the time-series of the validation data (top) and the corresponding control inputs (middle). The time-series of $\mathbf{x}$ has been scaled to improve readability across examples. The bottom of Panel A shows the ℓ_2 deviation of each surrogate model's forecast from the ground truth as a function of time. All forecasts are reinitialized every 50 time-steps, denoted by the dashed black line in the top plot. Panel B reports the mean ℓ_2 forecast error obtained across 32 trained surrogate models of each type for varying levels of additive Gaussian white noise. Panel C reports analogous results for the case of zero noise, but a varying number of training samples. We find the highest accuracy with linear system identification (DMDc) under no noise (bottom, A; C), as expected for an inherently linear system. With noisy data, ESNs achieve comparable performance to linear system identification (B). 64

4.3 For a description of each panel, refer to Fig. 4.2. Forecasting results for the stirred tank system show that ESNs achieve the best performance, both without noise (A,C) and with noise (B). ESNs outperform competing architectures by an order of magnitude when the number of training samples is held constant (C). 66

4.4 For a description of each panel, refer to Fig. 4.2. Forecasting results for the two-tank reservoir system show that that all architectures benefit greatly from more training data (C), but ESNs slightly outperform competing architectures both with noise (B) and without (A,C). 67

4.5 For a description of each panel, refer to Fig. 4.2. Forecasting results for the Lorenz system with control show that with zero or minimal noise, ESNs greatly outperform the competing architectures (A,B,C). As noise is increased, all forecasting methods fail (B), as to be expected in a chaotic system. 69

4.6 For a description of each panel, refer to Fig. 4.2. Forecasting results for the flow past a cylinder show that ESNs consistently achieve better performance across a range of noise levels (C). ESNs also require fewer training samples than competing architectures (B). 70

4.7 The left panel illustrates a 25 second sample of the generated training data for the considered flow past a cylinder, and the right panel shows the vorticity of the flow field for a single training snapshot. 71

4.8 The control objective for the spring mass system is to steer the displacement of both masses to -1.5, 0, and 1.5. Both ESN and LSTM based MPC achieve excellent reference tracking in the spring mass system and compute similar control actions, as to be expected in a simple linear system. 75

4.9 For the stirred tank system, the control task is to steer the system to $(x_1, x_2) = (x_1^*, x_2^*), (0.99x_1^*, 0.994x_2^*)$ and $(1.01x_1^*, 1.0066x_2^*)$, where $(x_1^*, x_2^*) = (0.877, 324.48)$ is the equilibrium concentration. The control performance of ESN and LSTM based MPC shows that both methods compute similar control actions and yield similar set points, even though the reference trajectory has been chosen to be infeasible. 77

4.10 For the two-tank reservoir system, the control task is to steer the system to $(x_1, x_2) = (0.25, 0.25), (0.5, 0.5)$ and $(0.1, 0.1)$. ESN based MPC computes slightly more aggressive control actions and achieves slightly better set-point tracking than LSTM based MPC. Transients between set points are similar for both methods. 78

4.11 The control task in the Lorenz system is to successively steer between the three fixed points of the system, at $(0, 0, 0)$, $(\sqrt{72}, \sqrt{72}, 27)$, and $(-\sqrt{72}, -\sqrt{72}, 27)$. Both the best and worst performing instances of 32 trials of LSTM based MPC are shown, as well as the worst performing instance of ESN based MPC. Even in the worst performing instance, ESN based MPC is able to steer the system to the fixed point at the origin and maintain the setpoint. Meanwhile the best performing instance of LSTM based MPC achieves this setpoint and maintains it through aggressive control actions. All shown controllers successfully steer the system to the fixed points at the center of the two butterfly lobes of the Lorenz system. 79

4.12 The control objective for the flow past a cylinder is to control the coefficient of lift C_ℓ to 0, 0.33, and -0.33. ESN based MPC with limited training data successfully tracks the desired coefficient of lift for the flow past a cylinder, while LSTM based MPC fails entirely. 80

5.1 ORC three-layer pipeline architecture. Each reservoir computer (RC) is decomposed into (i) an "embedding" function that lifts a low-dimensional signal u_t to the reservoir dimension, (ii) a "driver" function that propagates the reservoir state, and (iii) a "readout" that maps back to a target y. For control and forecasting RCs, the target y is typically u_{t+1} (either in the presence of a forcing term or not) and for classification the target y is a label. 87

5.2 ORC with GPU acceleration enables significantly faster performance than ReservoirPy, even when using ReservoirPy's JAX backend. Panel (a) shows the time per forecast step of RC models with varying reservoir dimension trained to forecast the Lorenz system, while panel (b) shows training tine for RC models with fixed reservoir dimension of 2000 but varying number of training samples. Performance of the two libraries with and without GPU acceleration are shown. GPU results were obtained running on an NVIDIA A40 GPU and CPU results were obtained with an Apple M2 chip. 89

6.1 Reconstruction error of the convolutional autoencoder (CAE) and shallow recurrent decoder (SHRED) on the KS system with $L = 22$ as a function of latent dimension d. N_{in} denotes the input dimension for each model. Left: aggregate relative ℓ_2 error. Right: maximum pointwise error. 117

6.2 Reconstruction error of CAE and SHRED on the KS system with $L = 44$ as a function of latent dimension d. N_{in} denotes the input dimension for each model. Left: aggregate relative ℓ_2 error. Right: maximum pointwise error. . . 117

6.3 Reconstruction error of the CAE and SHRED on the Kolmogorov flow at $Re = 30$ as a function of latent dimension d. N_{in} denotes the input dimension for each model. Left: aggregate relative ℓ_2 error. Right: maximum pointwise error. 118

6.4 Example reconstructions from CAE and SHRED on the KS system with $L = 22$ over 10 Lyapunov times. N_{in} denotes the input dimension for each model. Top: ground truth. Middle: reconstructions. Bottom: pointwise errors. Both models use latent dimension $d = 8$. 119

6.5 Example reconstructions from CAE and SHRED on the KS system with $L = 44$ over 10 Lyapunov times. N_{in} denotes the input dimension for each model. Top: ground truth. Middle: reconstructions. Bottom: pointwise errors. Both models use latent dimension $d = 16$. 120

6.6 Example reconstructions from CAE and SHRED on the Kolmogorov flow at $Re = 30$ over 50 dimensionless time units. N_{in} denotes the input dimension for each model. Top: ground truth. Middle rows: CAE and SHRED reconstructions. Bottom rows: pointwise errors. Both models use latent dimension $d = 8$. 121

6.7 Forecasting error for the KS system with $L = 22$ with a CAE + ESN and SHRED + ESN architecture, evaluated over $N = 50$ initial conditions. N_{in} denotes the input dimension for each model. Left: cumulative relative ℓ_2 error. Right: per-snapshot relative ℓ_2 error. Shaded regions denote interquartile ranges. 122

6.8 Forecasting error for the KS system with $L = 44$ with a CAE + ESN and SHRED + ESN architecture, evaluated over $N = 50$ initial conditions. N_{in} denotes the input dimension for each model. Left: cumulative relative ℓ_2 error. Right: per-snapshot relative ℓ_2 error. Shaded regions denote interquartile ranges. 123

6.9 Forecasting error for the Kolmogorov flow with a CAE + ESN and SHRED + ESN architecture, evaluated over $N = 30$ initial conditions. N_{in} denotes the input dimension for each model. Left: cumulative relative ℓ_2 error. Right: per-snapshot relative ℓ_2 error. Shaded regions denote interquartile ranges. . . 123

6.10 Lyapunov spectra of the true KS dynamics (black) and the learned SHRED + ESN latent dynamics (red, dashed) for $L = 22$ (left, top 10 exponents) and $L = 44$ (right, top 12 exponents). Spectra are augmented with two additional zero exponents following [138]. 124

LIST OF TABLES

Table Number Page

4.1 Forecasting performance across hyperparameters for the Lorenz system. Each block corresponds to varying one hyperparameter while holding others fixed. Results show 25th/75th percentiles of mean ℓ_2 error across 16 trained ESNs, all of which performed 10 forecasts of 50 timesteps. The bolded value in each row denotes the nominal hyperparameter value identified in the hyperparameter sweep and used when performing ablations of different variables. 73

4.2 Forecasting performance across hyperparameters for the flow past a cylinder. Each block corresponds to varying one hyperparameter while holding others fixed. Results show 25th/75th percentiles of mean ℓ_2 error across 16 trained ESNs, all of which performed 10 forecasts of 50 timesteps. The bolded value in each row denotes the nominal hyperparameter value identified in the hyperparameter sweep and used when performing ablations of different variables. 74

5.1 Comparison of reservoir computing libraries across key features. ✓ indicates full support; × indicates no support. *ReservoirPy's GPU support is available via its JAX backend (v0.4.0+) but does not fully exploit JAX's functional programming model. *Parallelizable* denotes native support for parallel RC architectures as in [136] and *vectorizable* denotes native support for vectorization (e.g. `vmap`). 85

ACKNOWLEDGMENTS

I am immensely appreciative to my advisors, Professor Krithika Manohar and Professor J. Nathan Kutz, for their support, encouragement, and trust throughout the duration of my graduate studies. I am fortunate to say that I learned about more than just research while working with them. I am also grateful to my committee members Professors Stefania Fresca and Marine Denolle for their feedback and support of this dissertation. In many ways, the Ph.D. is merely the tip of the student iceberg, and I would be remiss to ignore the formative role previous mentors have played in my academic career. Dennis Johnson, Michael McNally, and Jay Tasson have all taught me lessons I continue to carry with me years later.

The work contained in this dissertation would not have been possible without the contributions from my collaborators. Thank you all. Similarly, I am indebted to my labmates (MEB 123 and beyond) for the insights and levity they brought to my studies.

My family, assigned and chosen, has supported this work in ways they can't even realize. Dale, you have always inspired me with your work ethic. Mom and Dad, thank you for trusting me that good enough would be good enough, even when much evidence pointed to the contrary. Katie, no matter where we've been, it's better with you there.

DEDICATION

to K.T., A.P., and D.Ws.

Chapter 1

INTRODUCTION

The study of dynamical systems is the study of how quantities evolve in time. The word "quantity" here is deliberately vague and could refer to the position of a planet, the velocity field of air over a wing, or stock prices, among countless others. The connection between these seemingly disparate phenomena is that we can use shared mathematical approaches to understand the system or predict future behavior. At its basic level, this means that we are interested in relationships of the form $\frac{d}{dt}x = f(x)$, where the rate of change of some state x is expressed as a function of x itself. Given appropriately chosen x, an enormous range of phenomena in science and engineering can be approximated in this way. This is an old idea, dating back at least to the time of Newton and the invention of calculus. Consequently, centuries of work have produced powerful analytical tools for the study of dynamical systems [150]. For instance, there are numerous notions of stability, each formalizing a different aspect of predictable behavior. There are theories of bifurcation, linearization, perturbation, fixed points, limit cycles, and attractors. Theories of deterministic chaos have successfully described the behavior of many real world experiments [1,150]. Such ideas give us a common language for reasoning about dynamical systems across disciplines.

The application of such formal, mathematical approaches comes with its costs. Classical approaches typically require simplifying assumptions, such as linearity or exact knowledge of governing equations, that are violated by a real system. The approximation of a spherical cow, for example, has its limits. The recent explosion of machine learning [100] has demonstrated that data-driven methods can serve as an alternative modeling paradigm [22, 95]. Rather than derive a model from a physical law, one records how a system behaves and trains a model to approximate the observed dynamics. Data-driven approaches can enable accurate approximations when first principles models either do not exist or ignore relevant physics. They are also compelling because of their ability to find and exploit structure in

high-dimensional data that is not immediately obvious from a first principles view. Fluid dynamics is a particularly representative example. Direct numerical simulation of turbulence resolves every scale of motion, requiring massive, if not prohibitive, computational expense. However, the resulting flow might contain coherent structures that evolve on a low dimensional manifold. Data-driven approaches can learn to represent and propagate the lower dimensional dynamics directly. For tasks like prediction, monitoring, and control, the resulting computational savings can enable new optimization based approaches [22, 24].

The recent successes of large-scale neural networks have been driven in large part by the availability of massive datasets. The dominant architecture behind these advances has been the transformer [158]. While powerful, transformer based architectures have been shown to be data-hungry [177]. There is a fundamental mismatch between the realities of science and engineering and the availability of trillions of training samples used in large-scale commercial training. Typically, data is scarce and expensive to collect in scientific applications, and there are tighter tolerances on "accuracy" of the output. Recent works aimed at developing general purpose foundation models for time-series prediction [5, 105, 172] still struggle with scientific tasks that require accurate out-of-distribution predictions [173].

A more natural architecture for data-driven modeling of dynamical systems is the recurrent neural network (RNN). RNNs have a long history, dating to the work of Rumelhart, Hinton and Williams in the 1980s [143]. Subsequent developments like the long short-term memory (LSTM) network [68] extended their capacity to model longer range dependencies and improved their training behaviors. RNNs are particularly compelling in this context because their recurrent structure provides an inductive bias that is well-matched to temporal dynamics. A trained RNN can be viewed as a dynamical system itself. Most often, the RNN defines a discrete-time map, propagating a hidden state as a function of the previous hidden state and the current input, but continuous-time RNNs have also been developed [32]. This is exactly the structure of a dynamical system, and it means that the tools developed over centuries for analyzing dynamical systems can be applied to the learned model itself [156]. This structural alignment has practical consequences as well. RNNs perform well in low-data regimes that are typical of scientific applications [30, 161, 162]. They do not require the massive datasets transformer architectures typically rely on, and their sequential

nature reflects the causal, time-ordered structure of the systems they model. In this dissertation, we argue that RNN approaches, including traditional gradient trained networks as well as reservoir computing approaches, provide an efficient and expressive framework for addressing three central challenges in the modeling of real-world dynamical systems.

The sensing problem. The dynamical systems that arise in science and engineering are often too large to observe fully. A fluid flow may be defined on a grid of millions of spatial points, but we can only place sensors at a handful of locations [115]. Sea surface temperature varies continuously across the ocean, but we measure it at sparse, irregularly distributed stations and satellite passes [141]. In a nuclear reactor, the neutron flux is a spatially distributed quantity that we sample at a finite number of detector positions [83]. In each case, we have a small number of noisy, sparse sensor measurements, from which we need to infer the full global field.

The classical data-driven approach to this problem relies on dimensionality reduction. Methods such as proper orthogonal decomposition (POD) [50, 69], approximate the high-dimensional state as a linear combination of a small number of empirically derived modes. Given a new set of sparse measurements, one solves an inverse problem for the coefficients of these modes, producing an estimate of the full state [25, 48, 115]. This approach has been widely and successfully applied, but it has well-known limitations. The linear mode decomposition may be a poor fit for systems with strongly nonlinear dynamics, where the relevant structure cannot be captured by a fixed linear basis. Additionally, because the reconstruction is based on a single snapshot of sensor measurements, it has no mechanism for incorporating temporal information. When an underlying dynamical model is available, state estimation methods such as the Kalman filter [80] naturally incorporate the time history of observations. This temporal filtering is critical for ensuring estimates are robust to sensor noise. In the purely data-driven setting, where no explicit dynamical model is assumed, it is not obvious how to achieve the same effect.

Chapter 3 of this dissertation introduces the SHallow REcurrent Decoder (SHRED), an architecture built to incorporate trajectories of sensor measurements into data-driven, nonlinear reconstructions. SHRED uses a recurrent neural network to process trajectories

of sparse sensor measurements, encoding them into a latent representation from which a shallow decoder reconstructs the full high-dimensional state [48]. By operating on sensor trajectories rather than snapshots, SHRED naturally incorporates the temporal filtering that classical POD-based methods lack, while the learned nonlinear decoder avoids the limitations of a fixed linear basis. This time-delay approach is closely connected to Takens' embedding theorem [151], which guarantees that the dynamics of time-lagged observations of a dynamical system are diffeomorphic to the original attractor.

Control. In a discrete-time setting, autonomous dynamics take the form $x_{k+1} = f(x_k)$, and a trained RNN can learn this map directly, producing a dynamical system that generates forecasts by iterating its own predictions. The use of RNNs for time-series forecasting is well established and has a long history [30,64,162]. A less explored but equally important application is control, where the evolution of the state x depends additionally upon an exogenous control term u. That is, $x_{k+1} = f(x_k, u_k)$. If a learned RNN provides a sufficiently accurate forward model of such a system's dynamics, then that model can be embedded in a model predictive control (MPC) framework [14,146]. In MPC, control is posed as an online optimization problem, where at each time step, one optimizes over a sequence of candidate control inputs to steer the system toward a desired trajectory or target state. The forward model is used to evaluate the consequences of each candidate input sequence, and only the first input is applied before the optimization is re-solved with updated state information. This repeated re-optimization is what distinguishes MPC from open-loop control and makes it robust to model error and disturbances.

The quality of the forward model is critical in enabling MPC. If the surrogate model is grossly inaccurate or expensive to evaluate, MPC becomes unreliable or impractical [78]. Chapter 3 of this dissertation demonstrates that a particular class of RNN called an echo state network (ESN) [73] is well suited to serving as a surrogate model within an MPC loop, offering both accuracy and data-efficiency.

ESNs belong to a broader paradigm known as reservoir computing [73,114]. The central idea is to fix the majority of the weights of an RNN and train only a linear output layer. The recurrent layer known as the "reservoir" acts as a high-dimensional nonlinear feature

map that projects the input into a high-dimensional, nonlinearly evolving dynamical system. This approach has several benefits. Most importantly, because the reservoir weights are never updated, training reduces to a linear regression on the reservoir states [111]. Thus, training is orders of magnitude cheaper than for a fully trained RNN and there are no vanishing or exploding gradient problems. The resulting models are also lightweight enough to be evaluated thousands of times within an optimization loop, as is required by MPC. The trade-off is that the reservoir is not optimized for any particular task and its effectiveness depends on the reservoir being sufficiently rich and well-conditioned to represent the dynamics of interest. This, in turn depends on hyperparameter choices such as the spectral radius, reservoir size, and input scaling [111, 139].

ESNs have been shown to be effective for forecasting chaotic dynamics, reproducing both short-term trajectories and long-term statistical properties including Lyapunov spectra [137, 138, 161]. Their use as surrogate models for MPC has been explored in several contexts [8, 76, 132], as has the broader use of RNNs in learning-based control [14, 16, 153].

Chapter 4 of this dissertation uses echo state networks for both system identification and model predictive control, performing a comprehensive, standardized comparison between competing model architectures. Chapter 5 introduces OpenReservoirComputing (ORC). ORC is an open-source library implemented in JAX that provides GPU-accelerated reservoir computing with a modular architecture for experimenting with reservoir designs, training procedures, and downstream tasks.

Unknown or intractable dynamics. As systems grow in complexity, the governing equations may only be partially known or become too expensive to solve. The architectures introduced in the preceding sections can be combined into a single pipeline for forecasting directly from sparse sensor measurements. This approach to reduced order modeling is explored in Chapter 6. A trained SHRED model maps trajectories of sparse sensor data to a latent representation from which the full state can be decoded. However, this latent representation is itself a time-evolving quantity. If the latent dynamics are sufficiently structured, then an echo state network can be trained to propagate the latent state forward in time autonomously, producing forecasts that remain in the compressed latent space. The

full state can be recovered at any point by applying the shallow decoder.

This pipeline has practical advantages over alternative approaches to reduced-order forecasting. Methods based on autoencoders, for instance, learn a nonlinear dimensionality reduction and train a time stepper in the resulting latent space [29, 60, 118]. However, such approaches require full-state snapshots as input. In a real deployment where only sparse sensor measurements are available, one must first solve the nontrivial problem of estimating the full state from partial observations before the autoencoder can be applied. The SHRED-based pipeline avoids this issue entirely. The architecture is designed to operate on sparse sensor streams, and the latent state it produces is already conditioned on the available measurements.

Moreover, despite operating directly on sparse measurements, for several of the systems studied in Chapter 6, the autonomous latent-space time stepper replicates the Lyapunov spectrum of the true high-dimensional dynamics. The Lyapunov spectrum characterizes the rates of expansion and contraction along different directions in state space and is one of the most fundamental descriptors of a dynamical system's attractor [1]. Its replication in the learned latent dynamics provides strong empirical evidence that the latent representation faithfully captures the geometric and statistical structure of the underlying system. This finding is related to prior work demonstrating that reservoir computers can replicate Lyapunov exponents from data [138].

The remainder of this dissertation is organized as follows. Chapter 2 presents the mathematical foundations underlying the methods developed in this work, including dimensionality reduction, recurrent neural network architectures, sparse sensing, and model predictive control. Chapter 3 introduces the SHallow REcurrent Decoder (SHRED), a recurrent architecture for reconstructing high-dimensional states from trajectories of sparse sensor measurements. The chapter demonstrates SHRED's effectiveness across a range of dynamical systems and sensor configurations, and establishes it as a useful tool for data-driven state estimation. Chapter 4 develops reservoir computing as a framework for system identification and model predictive control. Echo state networks are shown to provide accurate surrogate models of nonlinear dynamics at a fraction of the computational cost of fully trained

RNNs, and these models are embedded in an MPC loop to achieve closed-loop control of dynamical systems from data. Chapter 5 presents OpenReservoirComputing, an open-source, GPU-accelerated reservoir computing library implemented in JAX. The library provides a modular platform for constructing, training, and deploying reservoir computing models, and is designed to support the computational demands of the methods developed in this dissertation. Chapter 6 unifies the sensing and modeling threads of the preceding chapters by using echo state networks to propagate the learned latent state of a SHRED model. This integration enables autonomous forecasting directly from sparse sensor streams. Chapter 7 offers concluding remarks, situating the contributions of this dissertation within the broader landscape of data-driven methods for dynamical systems and identifying directions for future work. Much of the content in Chapters 3-5 are reproduced with permission from the following works, respectively:

- Jan P. Williams, Olivia Zahn, and J. Nathan Kutz. Sensing with shallow recurrent decoder networks. *Proceedings of the Royal Society A, September*, 2024.

- Jan P. Williams, J. Nathan Kutz, and Krithika Manohar. Reservoir computing for system identification and model predictive control. *Neural Networks*, 2026.

- Jan P. Williams, Dima Tretiak, Steven L. Brunton, J. Nathan Kutz, and Krithika Manohar. OpenReservoirComputing: GPU-accelerated reservoir computing in JAX, *To appear in Journal of Open Source Software*, 2026.

Chapter 2

BACKGROUND

2.1 Dimensionality reduction and reduced order models

2.1.1 Linear methods

The dynamical systems that arise in science and engineering are often very high-dimensional. A fluid flow simulated on a grid of n spatial points, for instance, produces a state vector $x \in \mathbb{R}^n$ where n may be on the order of billions. In such cases, the state dimension can render tasks like prediction and control computationally infeasible. However, many such systems exhibit low-dimensional structure despite their high ambient dimensionality. The trajectories of the system may exist on a low-dimensional manifold in state space, or the dynamics may be well-approximated by a few dominant modes. This observation motivates the development of reduced-order models, which are compact representations that capture the essential behavior of the system in a lower dimensional space.

The most widely used framework for linear dimensionality reduction is proper orthogonal decomposition (POD), which is equivalent to principal component analysis (PCA) in the statistics literature [22, 69]. Suppose we collect m snapshots of a system state into a data matrix $X = [x_1 \; x_2 \; \cdots \; x_m] \in \mathbb{R}^{n \times m}$, where each column x_k is the state at time t_k. For notational simplicity, we assume that the temporal mean $\bar{x} = \frac{1}{m} \sum_{k=1}^{m} x_k$ has already been subtracted from the data. The data matrix admits a singular value decomposition (SVD),

$$X = U \Sigma V^T, \tag{2.1}$$

where $U \in \mathbb{R}^{n \times n}$ contains the left singular vectors (the POD modes), $\Sigma \in \mathbb{R}^{n \times m}$ is a diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$, and $V \in \mathbb{R}^{m \times m}$ contains the right singular vectors. The POD modes form an orthonormal basis, and the Eckart–Young theorem guarantees that the rank-r truncation $U_r \Sigma_r V_r^T$, obtained by retaining only the first r columns of U and V and the leading $r \times r$ block of Σ, is the best rank-r approximation to the data

in the sense of the Frobenius norm. A snapshot drawn from the same distribution then be approximated as

$$x_k \approx U_r a_k, \quad a_k = U_r^T x_k \in \mathbb{R}^r, \tag{2.2}$$

where a_k is the vector of POD coefficients. This decomposition replaces the n-dimensional state with an r-dimensional representation, where typically $r \ll n$.

The choice of truncation rank r is typically determined through analysis of the singular value spectrum. A common criterion is to select the smallest r such that the retained modes capture a prescribed fraction of the total variance, for example

$$\frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2} \geq 0.99. \tag{2.3}$$

Another common approach is the optimal hard threshold presented by Gavish and Donoho [58]. When the singular values decay rapidly, a small number of modes suffice. While many physical systems exhibit this rapid decay, others such as advection of sharp fronts or traveling waves, are known to produce slow singular value decay [125]. This means that a large number of POD modes are required for an accurate approximation and the savings from a reduced order model are lessened.

Such reduced order models typically retain the POD basis for spatial representation and fit a separate model to the temporal evolution of the coefficients a_k. A classical approach is Galerkin projection, which substitutes the modal expansion into the governing equations and projects onto the POD modes to obtain a reduced system of ordinary differential equations for the coefficients [69]. This produces interpretable, physics-informed reduced models, but requires knowledge of the governing equations and can suffer from stability issues. Alternatively, when the governing equations are unavailable or too complex, one can treat the coefficient trajectories as time-series data and fit a model directly. Neural networks, Gaussian process regression, and sparse regression have all been used in this role [22]. These data-driven approaches to learning the evolution of POD coefficients represent an intermediate modeling strategy where the interpretability and structure of a linear modal decomposition is retained, but nonlinear dynamics are allowed in the reduced space.

Such methods also have advantages beyond interpretability. Training is computationally inexpensive, typically amounting to an SVD and a regression. These methods are also

backed by decades of theoretical development and successful application across fluid dynamics, structural mechanics, and other fields [22, 69]. However, the fundamental assumption underlying all of these approaches is that the relevant dynamics can be well-represented in a fixed linear subspace. This assumption fails when the dynamics are strongly nonlinear or when the relevant structures change over time. Transport dominated systems are a canonical example where linear bases perform poorly. These limitations motivate the development of nonlinear dimensionality reduction methods, which learn flexible, data-dependent representations that can capture structure that linear bases cannot.

2.1.2 Nonlinear dimensionality reduction

The limitations of linear subspace methods motivate representations that can capture nonlinear structure in the data. The autoencoder [67] provides a natural framework for this. An autoencoder consists of an encoder $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}^r$ that maps the full state x to a low-dimensional latent vector $z = \mathcal{E}(x)$, and a decoder $\mathcal{D} : \mathbb{R}^r \to \mathbb{R}^n$ that maps back to the original space, producing a reconstruction $\hat{x} = \mathcal{D}(z)$. The encoder and decoder are typically parameterized as neural networks and trained jointly to minimize a reconstruction loss over a dataset of snapshots. POD can be viewed as the special case in which both the encoder and decoder are constrained to be linear maps. By allowing nonlinear encodings and decodings, autoencoders can learn latent representations that lie on curved manifolds in state space, capturing structure that linear bases cannot represent. This additional flexibility has made autoencoders an increasingly popular tool for dimensionality reduction in the modeling of complex dynamical systems [29, 102, 112].

Once a nonlinear latent representation has been learned, a reduced-order model is obtained by training a time stepper to propagate the learned latent state forward. That is, given the latent state $z_k = \mathcal{E}(x_k)$ at time t_k, a learned map $F : \mathbb{R}^r \to \mathbb{R}^r$ approximates the latent dynamics as $z_{k+1} = F(z_k)$. The full state can be recovered at any time via the decoder, $\hat{x}_{k+1} = \mathcal{D}(z_{k+1})$. The map F may itself be a neural network, or it may be a simpler model such as a linear operator or a sparse regression. This is the nonlinear analogue of learning the evolution of POD coefficients discussed in the preceding section: the spatial

representation is now nonlinear, but the overall strategy of separating dimensionality reduction from temporal modeling is the same. A number of works have pursued this approach. Gonzalez and Balajewicz [60] combined convolutional autoencoders with LSTM networks to forecast fluid flows in reduced coordinates. Lusch et al. [112] trained autoencoders alongside linear Koopman-based time steppers to learn latent dynamics that are approximately linear. Champion et al. [29] extended this to nonlinear ODE models. The paradigm of learning a compressed representation and then modeling its evolution has become a standard template in the data-driven reduced-order modeling literature [102, 118, 130].

Both POD and autoencoder based reduced order models share the practical limitation of requiring a full-state snapshot $x \in \mathbb{R}^n$ as input. In real systems, full-state measurements are rarely available. Instead, one typically has access to a small number of noisy, sparse sensor measurements. Before an autoencoder can be applied, one must first solve the nontrivial problem of reconstructing the full state from these partial observations. Autoencoder-based reduced-order models thus implicitly assume that the state estimation problem has already been solved, an assumption that limits their applicability in sensor-constrained settings. This limitation is one of the central motivations for the SHRED architecture introduced in Chapter 3, which sidesteps the full-state requirement entirely by operating directly on trajectories of sparse sensor measurements.

2.2 Recurrent Neural Networks

2.2.1 Foundations

A standard feedforward neural network consists of layers of affine transformations interspersed by pointwise nonlinearities to approximate a function from inputs to outputs. As an example, a single hidden layer network can be written as

$$\hat{y} = W_2\phi(W_1 x + b_1) + b_2, \tag{2.4}$$

where ϕ is a nonlinear activation. The universal approximation theorem [38, 70] guarantees that such networks can approximate any continuous function on a compact domain given sufficient width. While feedforward networks are powerful function approximators, they treat each input independently. That is, there is no mechanism for incorporating the order

or history of a sequence of inputs. For tasks where the current output depends not only on the current input but on prior states as well, this is a fundamental limitation.

Recurrent neural networks [143] were introduced to address this limitation by incorporating a hidden state that is propagated across time steps. At each time t, the hidden state $h_t \in \mathbb{R}^{d_h}$ is updated as a function of the previous hidden state and the current input $x_t \in \mathbb{R}^{d_x}$. That is,

$$h_t = \phi(W_h h_{t-1} + W_x x_t + b), \tag{2.5}$$

where $W_h \in \mathbb{R}^{d_h \times d_h}$ and $W_x \in \mathbb{R}^{d_h \times d_x}$ are weight matrices, $b \in \mathbb{R}^{d_h}$ is a bias vector, and ϕ is a nonlinear activation function such as tanh. An output $\hat{y}_t \in \mathbb{R}^{d_y}$ can be produced at any time step via a separate readout layer, $\hat{y}_t = W_y h_t + b_y$. The same weight matrices W_h, W_x, and W_y are shared across all time steps, so the number of parameters is independent of the sequence length. The hidden state can thus be interpreted as a learned state vector that evolves according to a parameterized map that depends on the current state h_{t-1} and an external driving input x_t. W_y can be interpreted as a readout or measurement function. A trained RNN is therefore itself a dynamical system, and the concepts developed over centuries for analyzing dynamical systems can in principle be applied to the learned model [8, 150, 153].

To train an RNN, one unrolls the recurrence over the length of the input sequence and applies the chain rule to compute gradients of the loss with respect to the shared weights. This procedure is known as backpropagation through time (BPTT) [143, 164]. Let

$$\mathcal{L} = \sum_t \ell_t(\hat{y}_t, y_t) \tag{2.6}$$

denote the total loss as a sum over time steps. The gradient of $\mathcal{L}$ with respect to the recurrent weights W_h involves contributions from every time step, and the contribution from time t depends on how the hidden state at time t was influenced by the weights at all previous times. By the chain rule, this requires computing products of Jacobians of the form

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}, \tag{2.7}$$

for all $k < t$. Each factor $\frac{\partial h_i}{\partial h_{i-1}}$ is a $d_h \times d_h$ matrix that depends on the activation function and the weights.

2.2.2 Vanishing gradients, exploding gradients, and gating mechanisms

The product of Jacobians $\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$ that appears in the BPTT gradient computation involves the repeated multiplication of matrices across time steps. If the largest singular value of the Jacobian $\frac{\partial h_i}{\partial h_{i-1}}$ is consistently less than one, the product contracts exponentially and the gradient signal from future time steps is reduced to near zero by the time it reaches much earlier timesteps. This is the vanishing gradient problem [12, 68]. Conversely, if the largest singular value is consistently greater than one, the product can grow exponentially and the gradients become numerically unstable. This is the exploding gradient problem. In either case, the network cannot effectively train.

The long short-term memory (LSTM) network, introduced by Hochreiter and Schmidhuber [68], addresses the vanishing gradient problem by providing a "gradient highhway" for gradient information to flow across many time steps. This is accomplished by introducing the cell state $c_t \in \mathbb{R}^{d_h}$. The cell state is another memory vector, but it is updated through additive interactions rather than the repeated matrix multiplications that cause gradients to vanish or explode. Access to the cell state is regulated by three learned gates, each of which is a sigmoid-activated function of the current input and previous hidden state. The recurrence of an LSTM can be summarized as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad \text{(forget gate)} \tag{2.8a}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad \text{(input gate)} \tag{2.8b}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o). \quad \text{(output gate).} \tag{2.8c}$$

The forget gate f_t determines what fraction of the previous cell state to retain. The input gate i_t controls how much of a candidate update $\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$ is written to the cell state. The cell state is then updated as

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{2.9}$$

where $\odot$ denotes elementwise multiplication. Finally, the output gate o_t determines what portion of the cell state is exposed as the hidden state,

$$h_t = o_t \odot \tanh(c_t). \tag{2.10}$$

Because c_t depends on c_{t-1} through an elementwise multiplication by f_t rather than a full matrix multiplication, the gradient of c_t with respect to c_{t-1} is a diagonal matrix with entries in $[0, 1]$. When the forget gate is "open", gradient information passes through nearly unchanged. The network learns when to remember and when to forget, and this learned control over the gradient pathway is what enables LSTMs to capture dependencies across long time intervals.

The gated recurrent unit (GRU), introduced by Cho et al. [33], simplifies the LSTM by combining the forget and input gates into a single update gate and merging the cell state and hidden state into a single vector. The GRU update equations are

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \quad \text{(update gate)} \tag{2.11a}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r), \quad \text{(reset gate)} \tag{2.11b}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h), \tag{2.11c}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t. \tag{2.11d}$$

The update gate z_t plays the role of both the forget and input gates in the LSTM, interpolating between retaining the previous hidden state and replacing it with the candidate $\tilde{h}_t$. The reset gate r_t controls how much of the previous hidden state influences the candidate. GRUs have fewer parameters than LSTMs and often achieve comparable performance in practice [34]. More broadly, the introduction of gating mechanisms was the critical development that transformed recurrent neural networks from theoretically appealing but practically limited architectures into effective tools for sequence modeling. For roughly two decades following the introduction of the LSTM, gated RNNs were the dominant architecture for tasks involving sequential data, from speech recognition and machine translation to time-series prediction [64, 107].

2.2.3 RNNs as dynamical systems models

With the gating machinery in place, we return to the dynamical systems perspective introduced in Section 2.2.1. A trained LSTM or GRU defines a discrete-time dynamical system whose state evolution is governed by learned, nonlinear maps. The gates act as learned switches that control the flow of information through the system, determining what is "remembered" and "discarded" at each time step. The hidden state accumulates information from past observations according to learned rules, and the recurrent map propagates it forward in a manner determined by the data. Unlike feedforward architectures that process each input in isolation, or attention-based architectures that access the entire sequence simultaneously, RNNs process data sequentially and causally. Each hidden state depends only on previous inputs, reflecting the time-ordered, causal structure of the physical systems they are designed to model.

This structural alignment has practical consequences for data efficiency. The transformer architecture [158], which has become dominant in natural language processing and increasingly in scientific applications, relies on self-attention mechanisms that can learn arbitrary dependencies across a sequence. This flexibility, however, typically comes with inctreased data requirements. Transformers are data-hungry, typically requiring enormous training sets to learn effectively [177]. In scientific and engineering applications, data is scarce and expensive to collect. A single high-fidelity simulation of a turbulent flow may require days or weeks of compute time, and experimental measurements are limited by sensor budgets and facility access. RNNs perform well in these low-data regimes because their architectural assumptions provide an inductive bias that reduces the amount of data needed to learn an accurate model of the dynamics [30, 162]. Recent work on general-purpose foundation models for time-series prediction has shown that transformer-based approaches still struggle with scientific tasks [173, 174], underscoring the value of architectures whose structure is matched to the problem.

Gradient-trained RNNs, particularly gated architectures like the LSTM and GRU, are powerful and expressive models of temporal dynamics. However, training via BPTT remains computationally expensive, especially for long sequences or large hidden states. The

optimization landscape can be difficult to navigate, and the computational cost of unrolling and differentiating through many time steps limits the scalability of these models. An alternative paradigm avoids these difficulties entirely by fixing the recurrent weights and training only a linear output layer. This approach is known as reservoir computing, and is the subject of the next section.

2.3 Reservoir Computing

2.3.1 Computational approach

Reservoir computing emerged independently from two lines of work in the early 2000s. From the ML side, Jaeger [73] introduced echo state networks, demonstrating that a randomly initialized recurrent network with fixed weights could be trained to perform complex time-series tasks by fitting only a linear output layer. At the same time, Maass, Natschlager, and Markram [114] introduced the liquid state machine. The liquid state machine arose from a computational neuroscience perspective, showing that a recurrent network of "spiking neurons" could act as a computational medium when paired with a trained readout layer. Both works rest on the same idea that a a linear map can be learned from a random, high-dimensional dynamical system to a lower dimensional signal of interest. The term "reservoir computing" was later introduced [160] as an umbrella for both the liquid state machine and echo state network approaches.

This approach can loosely be understood through analogy to kernel methods in machine learning. A kernel maps data into a high-dimensional feature space in which a linear classifier or regressor can solve problems that are nonlinear in the original input space. The reservoir plays an analogous role, but with the addition of a temporal structure. The reservoir is a dynamical system with memory, meaning its state at any time t is a nonlinear function of the history of inputs to that point. The reservoir can therefore be viewed as a temporal nonlinear feature map. It projects the input sequence into a high-dimensional trajectory in the reservoir state space. There, the temporal structure of the input is encoded in that trajectory. If the reservoir is sufficiently high-dimensional and its dynamics are sufficiently expressive, a linear readout from the reservoir state can solve tasks that would require a more

complex nonlinear model operating in lower dimensions. In some sense, the computational burden is shifted from training to architecture. The reservoir needs to be designed so that its dynamics are rich enough to approximate the data, but once this is achieved, learning reduces to a linear regression.

2.3.2 Echo State Networks

Throughout this dissertation, we will rely on the echo state network (ESN) architecture. An input $x_t \in \mathbb{R}^{d_x}$ is mapped into the reservoir through a fixed matrix $W_{in} \in \mathbb{R}^{N \times d_x}$, where N is the reservoir dimension. The reservoir state $r_t \in \mathbb{R}^N$ evolves according to

$$r_t = (1 - \alpha)\, r_{t-1} + \alpha \, \tanh(W_{in}\, x_t + W\, r_{t-1}), \tag{2.12}$$

where $W \in \mathbb{R}^{N \times N}$ is the fixed reservoir matrix and $\alpha \in (0, 1]$ is a leak rate hyperparameter. When $\alpha = 1$, the update reduces to $r_t = \tanh(W_{in}\, x_t + W\, r_{t-1})$, and the reservoir has no explicit smoothing of its state across time steps. For $\alpha < 1$, the reservoir state is a convex combination of the previous state and the nonlinear activation, which has the effect of slowing the reservoir dynamics and can improve performance on tasks with slower timescales [111]. The output at each time step is a linear function of the reservoir state

$$\hat{y}_t = W_{out}\, r_t, \tag{2.13}$$

where $W_{out} \in \mathbb{R}^{d_y \times N}$ is the output weight matrix. W_{out} is the only matrix of trained weights. The input weights W_{in} and reservoir weights W are initialized randomly and held fixed.

The echo state property is a condition that ensures the reservoir's behavior is a well-defined function of an input sequence [111]. It requires that the effect of the reservoir's initial conditions on its state decays asymptotically, so that after a sufficient number of time steps, the reservoir state depends only on the history of the driving input and not on how the reservoir was initialized. This fading memory condition ensures that the reservoir does not depend on an arbitrarily chosen initial condition. A sufficient condition for the echo state property is that the spectral radius $\rho(W)$ of the reservoir weight matrix satisfies $\rho(W) < 1$ [73,111]. Intuitively, this ensures that perturbations to the reservoir state contract

over time, preventing the reservoir from amplifying its own internal dynamics at the expense of the input signal. In practice, reservoirs with spectral radius near or slightly above one often perform well, particularly for tasks that require sensitivity to longer-range temporal dependencies [139]. The spectral radius is therefore taken as a tunable parameter that trades.

Since W_{in} and W are fixed, the reservoir states $r_1, r_2, \ldots, r_m$ are determined entirely by the input sequence. Training the output weights W_{out} therefore reduces to solving a linear regression problem. Given the matrix of reservoir states $R = [r_1\ r_2\ \cdots\ r_m] \in \mathbb{R}^{N\times m}$ and the corresponding target outputs $Y = [y_1\ y_2\ \cdots\ y_m] \in \mathbb{R}^{d_y\times m}$, the output weights are obtained in closed form as

$$W_{out} = YR^T(RR^T + \beta I)^{-1}, \tag{2.14}$$

where $\beta > 0$ is a Tikhonov regularization parameter that prevents overfitting and ensures numerical stability when the reservoir states are correlated. Adding low levels of noise to the input stream is also often used as a regularization strategy [165]. Because this training relies only on a linear regression, training an echo state network is orders of magnitude faster than training a comparably sized RNN via backpropagation through time. The resulting model is also computationally inexpensive to evaluate. This combination of cheap training and cheap evaluation makes echo state networks particularly attractive for applications that require the model to be trained on limited data and evaluated many times in rapid succession. As we will see in Chapter 4, this property is critical for embedding ESNs as surrogate models within model predictive control loops, where the forward model may be called thousands of times per control step.

2.3.3 Hyperparameters

The performance of an echo state network depends heavily on the selection of hyperparameters. The spectral radius $\rho(W)$, as discussed above, controls the effective memory of the reservoir. Larger values allow the reservoir to retain information about past inputs over longer intervals but can be unstable, while smaller values produce faster forgetting but greater stability. The reservoir size N determines the dimension of the feature space in which

the linear readout operates. Larger reservoirs can represent more complex input-output relationships, but they increase computational cost and, because the number of parameters in W_{out} scales linearly with N, they also increase the risk of overfitting when training data is limited. How the random entries W_{in} (e.g. the distribution from which values are drawn) are selected controls the amplitude of the driving signal relative to the reservoir's internal dynamics. Large input scaling drives the tanh nonlinearity into saturation, producing a reservoir that responds nonlinearly to the input. Small input scaling keeps the reservoir in a nearly linear regime, which may be appropriate for systems with approximately linear dynamics. The sparsity of the reservoir weight matrix W, typically parameterized as the fraction of nonzero entries, affects both the computational cost of the reservoir update and the diversity of dynamical timescales within the reservoir. Sparse reservoirs tend to produce more useful dynamics, as different subsets of reservoir nodes are coupled to different degrees [111, 139]. Sparse reservoirs also allow for computational speedups.

Since the reservoir weights are not directly learned for the task at hand, the effectiveness of an echo state network depends largely on whether the fixed reservoir happens to be sufficiently rich and well-conditioned to represent the dynamics of interest. This is the central trade-off of reservoir computing approaches and means that hyperparameter selection is a critical part of training. A reservoir whose spectral radius, size, or input scaling is mismatched to the target dynamics will produce poor results regardless of how the output layer is trained. Consequently, hyperparameter selection is typically performed by grid search or random search over a validation set [139]. For complex systems where the interaction between hyperparameters is nontrivial, this search can itself become computationally demanding, particularly when the reservoir is large or the training data requires long time series. This dependence on hyperparameter selection, and the computational cost of exploring the design space, largely motivates the work of Chapter 5, which develops a software infrastructure to make it easy and fast to construct, train, and evaluate reservoir computing models at scale. The software contributions in that chapter can also be helpful in enabling optimization based control techniques.

2.4 Model Predictive Control

2.4.1 Problem Formulation

The prior sections have focused on modeling and observing dynamical systems whose evolution is either autonomous or driven by fixed external conditions. In many applications, however, the goal is not to observe or predict the state of a system but rather to control it. This introduces a control input $u_k \in \mathbb{R}^{d_u}$ that influences the state evolution, giving dynamics of the form

$$x_{k+1} = f(x_k, u_k), \tag{2.15}$$

where $x_k \in \mathbb{R}^n$ is the state and f is the dynamics map. A control objective is to determine a sequence of inputs $u_0, u_1, \ldots$ that drives the system towards a target behavior [92]. This can be viewed as the minimization of a cost function such as

$$J = \sum_{k=0}^{H-1} \ell(x_k, u_k) + \ell_f(x_H), \tag{2.16}$$

where $\ell(x_k, u_k)$ penalizes deviation from a reference trajectory and the magnitude of the control effort, $\ell_f(x_H)$ is a terminal cost, and H is the planning horizon. The inputs and states are often also subject to constraints that restrict the set of feasible control actions.

Open loop control functions by solving for an entire control sequence in advance. While straightforward, this approach is rarely effective. Any mismatch between the model f and the true system, or any unmodeled disturbance, causes the actual trajectory to diverge from the planned one with no mechanism for correction. Model predictive control (MPC) [92, 146] achieves closed-loop robustness through repeated optimization. At each time step k, the controller measures or estimates the current state x_k and solves the finite-horizon optimization problem

$$\min_{u_k, \ldots, u_{k+H-1}} \sum_{j=0}^{H-1} \ell(x_{k+j}, u_{k+j}) + \ell_f(x_{k+H}) \quad \text{subject to} \quad x_{k+j+1} = f(x_{k+j}, u_{k+j}), \tag{2.17}$$

along with any input or state constraints. Only the first control input u_k is applied to the system. At the next time step, the state is updated, the horizon shifts forward, and the optimization is solved again. The repeated re-optimization at each time step is the source of

MPC's robustness. The controller continuously corrects for model errors, unmodeled disturbances, and accumulated drift. Even when the forward model f is only an approximation of the true dynamics, the closed-loop structure limits the horizon over which prediction errors can compound. This makes MPC practical in settings where the model is approximate, which is exactly the situation that arises in data-driven applications.

The forward model f sits at the center of the MPC loop. At every iteration of the optimization solver and at every step within the prediction horizon, the model must be evaluated to propagate candidate state trajectories forward. The quality of the resulting controller therefore depends on the forward model in two distinct senses. It must be accurate enough that the trajectories it predicts are well approximate the true system, and it must be computationally cheap enough to ensure that the optimization problem can be solved within the time budget of a single control step. A high-fidelity simulation may be accurate but too expensive to call thousands of times within an optimizer. A simple linear model may be fast but too inaccurate to capture the relevant nonlinear dynamics.

2.4.2 Data-Driven Surrogate Models for MPC

A data-driven surrogate model for MPC must therefore satisfy three requirements: (i) it must be accurate enough to produce useful predictions over the control horizon, (ii) it must be fast enough to evaluate within an optimization loop that may call it thousands of times per control step, and (iii) it must be data-efficient enough to train from the limited datasets typical of scientific and engineering applications. The echo state networks introduced previously are well matched to all three. Echo state networks achieve excellent performance generating faithful approximations of dynamical systems, require only two matrix-vector multiplication to propagate forward in time, and are more data-efficient than many competing methods as a result of their training via linear regression. This combination of accuracy, speed, and data efficiency makes echo state networks a natural choice for the surrogate model within an MPC framework.

Echo state networks are not the only data-driven surrogate that has been considered for predictive control. Fully trained neural networks [14, 71, 153], Gaussian process mod-

els [66], and linear models identified via system identification or DMD [78,140] have all been embedded in MPC loops, each occupying a different point in the accuracy-speed-data efficiency trade-off. Gaussian processes, for instance, provide principled uncertainty estimates but scale poorly with dataset size. Fully trained RNNs can be highly expressive but are expensive to train and may require careful regularization to avoid overfitting in low-data regimes [16]. The contribution of Chapter 4 is to demonstrate that echo state networks occupy a particularly favorable point in this trade-off for the class of nonlinear dynamical systems considered in this dissertation.

Chapter 3

SENSING WITH SHALLOW RECURRENT DECODER NETWORKS

3.1 Introduction

Emerging sensor technologies are transforming every science and engineering domain, with the quantity and quality of data collected also driving fundamental advances through data-science and machine learning methods. In many areas of interest, measurements of the full-state are at best impractical and often impossible. Thus sensors are commonly used to infer the current and future behavior of high-dimensional systems with a limited number of sensor locations. With severely limited and noisy sensor measurements, this task is exceptionally difficult and frequently requires principled sensor placement schemes to yield faithful reconstructions [115]. Accurate and robust reconstruction techniques are vital in enabling downstream tasks such as system identification, forecasting, and control. Existing data-driven techniques to learn mappings from sensor measurements to the full-state fall into two categories: those that rely only on static sensor measurements [15, 25, 48, 176] and those that involve delay embedding techniques [59, 151, 156, 175]. The former have been used extensively in the reconstruction of large spatio-temporal fields, while the latter have primarily been applied with the aim of reconstructing chaotic attractors. In this work, we apply such delay embedding techniques to high-dimensional spatio-temporal flows. The aim is not to replicate an attractor, but rather infer the full-state from sparse sensor measurements. As will be shown, not only is there a significant performance increase in comparison to static sensing, but a minimal number of sensors, randomly placed, can be used.

The reconstruction of spatio-temporal dynamics from limited sensors relies on low-rank features of the data. Computing low-rank embeddings of such high-dimensional data is often achieved through the *singular value decomposition* (SVD), also known as *proper orthogonal decomposition* (POD) [22, 95]. The coefficients of the dominant correlated modes are deter-

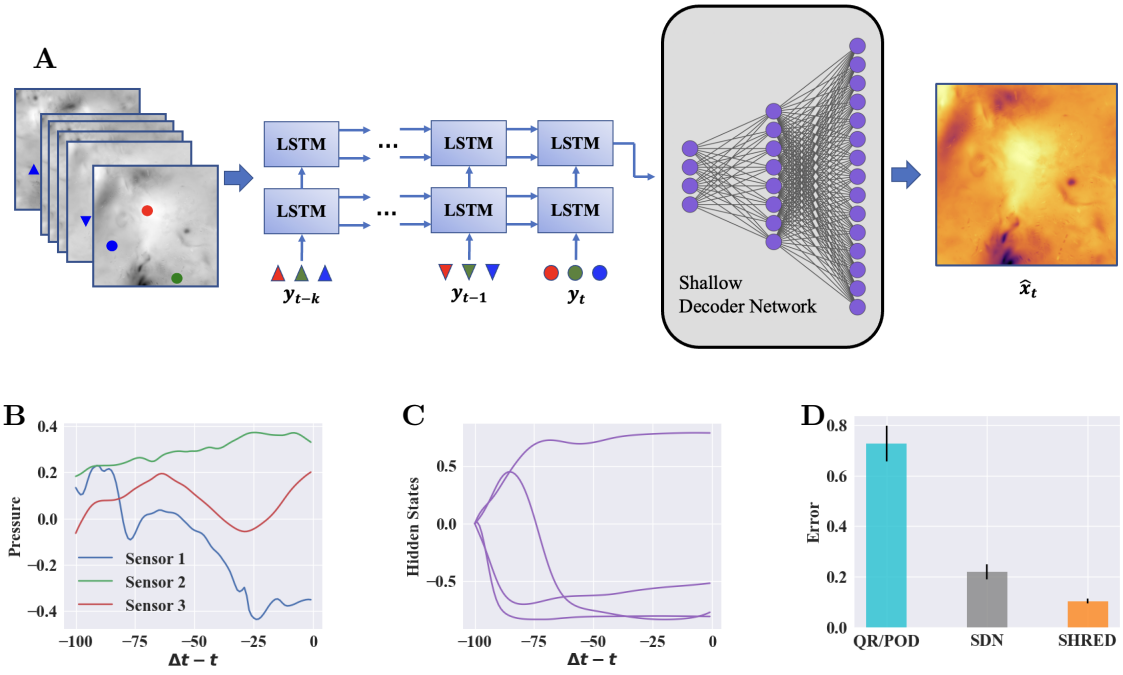


Figure 3.1: Summary diagram of SHallow REcurrent Decoder networks (SHRED) for flow reconstruction from sensor measurements. *A* A graphical representation of the SHRED method. The mulitvariate time-series of sensor measurements, $\{\mathbf{y}_i\}_{t-k}^{t}$, is fed into a stacked long short-term memory layer. The final output of the recurrent layer, $\mathbf{h}_t$, serves as an input to a fully-connected shallow decoder mapping from the hidden state to the high-dimensional field. *B* The time series of sensor measurements fed into the SHRED model. *C* The evolution of the output hidden state generated by the input sequence of sensor measurements. *D* Reconstruction errors of traditional methods (QR/POD), shallow decoders (SDN), and SHRED on a turbulent flow when three sensors are available.

mined by solving a linear inverse problem and the modes themselves serve as a linear map between measurements and the spatio-temporal state space. Many of these linear methods are built upon the mathematical framework of gappy POD and have been successful across many disciplines [11, 31, 45, 50, 115]. More recently, shallow decoder networks (SDN) have leveraged advances in machine learning and AI to learn end-to-end, nonlinear maps between measurements and high-dimensional state spaces. SDNs have been demonstrated to outperform their linear counterparts, particularly when the number of available sensors is exceedingly low [27, 48, 144, 166]. Both derivatives of gappy POD and SDNs rely on measurements at a single snapshot in time to reconstruct the corresponding high-dimensional state at that time.

As a result of this static sensing, linear reconstruction methods are heavily reliant on optimally placed sensors for the inverse problem to be well-conditioned [45,115]. In general, determining optimal sensor locations is a combinatorially hard problem and is infeasible for large search spaces. For smaller spaces, there exist a number of well-known techniques for sensor placement [18,28,77,93,108,133,147], but they become computationally intractable in high-dimensional systems. Instead, convex relaxations of the problem become necessary for obtaining approximate solutions. A greedy search using the QR decomposition with column pivoting offers one approximate solution and such methods have been critical in enabling accurate reconstructions of high-dimensional systems [45,115]. More recent work has sought to relax the reliance on optimal sensor placement through regularization and probabilistic reconstruction. Bayesian and regularized formulations of gappy POD enable stable reconstruction under sensor undersampling and provide uncertainty quantification absent from purely deterministic schemes [79,90], and the PySensors library provides open implementations of many of these sparse sensing and placement algorithms [39, 82]. Sparse variants of the discrete empirical interpolation method have similarly improved reconstruction from few sensors [51], including recent approaches that, like the present work, combine sparse sensing with recurrent neural networks [52]. However, greedy algorithms suffer from the fact that there is no guarantee that the sensor locations found are physically implementable; for instance, measuring sea-surface temperature in the middle of the Pacific Ocean is significantly more challenging than taking measurements along the coast. Variations of greedy search al-

gorithms can incorporate additional constraints, such as cost and sensor fidelity [35–37,116], but often at the expense of reconstruction accuracy. The use of SDNs helps mitigate the dependence on sensor location, but still is greatly aided by principled sensor placement schemes [166].

The data-driven method presented here incorporates temporal trajectories of sensor measurements to improve reconstruction accuracy, robustness to noise, and eliminate the need for optimally placed sensors. We use a type of recurrent network layer, *long short-term memory networks* (LSTM) [68], to process a time-series of sensor measurements. The latent representation of the LSTM is the input into a fully-connected, shallow decoder for reconstruction. We demonstrate that these *SHallow REcurrent Decoders* (SHRED) outperform existing linear and nonlinear techniques on three example datasets: a forced isotropic turbulent flow [104], weekly sea-surface temperature [141], and atmospheric ozone concentration [13]. In all cases, SHRED with as few as three randomly placed sensors achieves superior reconstruction accuracy than existing techniques using a far greater number of optimally placed sensors. Moreover, because the inputs to a trained SHRED model consist of just a few sensor measurements, SHRED offers an *on-the-fly* compression for modeling physical and engineering systems. Fig. 3.1 gives a graphical summary of SHRED. The use of both constituents of SHRED in this paper, LSTMs and feed-forward shallow decoders, follows a long line of work demonstrating the efficacy of each component and their use together replicating chaotic attractors from partial observations [48, 84, 156]. That being said, there exist a litany of different architectures that could be used to encode sensor trajectories (basic RNNs, GRUs [34], ESNs [111], transformers [157], etc.) or decode from a learned latent state (CNNs [100], transformers [157], residual layers [62], etc.); the fundamental objective of SHRED is to combine implicit time-delay embeddings of sensor measurements with an expressive decoder for high-dimensional state reconstruction. The investigation of the many possible architectures to perform this task is an important area of future exploration. In this work, we have shown explicitly that the combination of the two key components, *recurrence* and *decoding*, allow for a successful mathematical scheme for sensing.

3.2 Background

3.2.1 Proper orthogonal decomposition and QR

Traditional techniques for high-dimensional state reconstruction from limited sensor measurements typically rely on the SVD to compute a low-rank embedding of the spatio-temporal dynamics. Given N snapshots of an m dimensional state, we construct a data matrix $\mathbf{X} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \mathbf{x}_N]$ where $\mathbf{x}_i \in \mathbb{R}^m$ is the i-th temporal snapshot of the state. The SVD factors the data matrix into the product of an orthogonal matrix, $\mathbf{U} \in \mathbb{R}^{m \times m}$, a diagonal matrix with decreasing entries, $\mathbf{\Sigma} \in \mathbb{R}^{m \times N}$, and another orthogonal matrix $\mathbf{V} \in \mathbb{R}^{N \times N}$. An optimal rank r approximation of $\mathbf{X}$ can be directly obtained through the SVD by retaining only the first r columns of $\mathbf{U}$,

$$\mathbf{X} \approx \hat{\mathbf{X}} = \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T. \tag{3.1}$$

Thus, each temporal snapshot $\mathbf{x}_i$ can be approximated by a linear combination of the first r columns of $\mathbf{U}$, otherwise known as the dominant POD modes.

POD based methods estimate a high-dimensional state, $\mathbf{x}$, drawn from the same distribution as that of the training data by solving for these r coefficients using a subsampling of the state,

$$\mathbf{y} = \mathbf{C}\mathbf{x} \approx \mathbf{C}\mathbf{U}_r \mathbf{b}, \tag{3.2}$$

where $\mathbf{C}$ is a "sensing" matrix consisting of rows of the $m \times m$ identity matrix and $\mathbf{b} \in \mathbb{R}^r$. A full-state estimate from a set of sensor measurements, $\mathbf{y}$, can then be obtained by

$$\mathbf{x} \approx \hat{\mathbf{x}} = \mathbf{U}_r (\mathbf{C}\mathbf{U}_r)^{-1} \mathbf{y}. \tag{3.3}$$

In practice, it is critical to design $\mathbf{C}$ such that the inversion in (3.3) is well-conditioned. An indirect bound on the condition number of $\mathbf{C}\mathbf{U}_r$ can be found by determining $\mathbf{C}$ where $|\det(\mathbf{C}\mathbf{U}_r)|$ is maximized. That is, we seek

$$\mathbf{C}. = \underset{\mathbf{C}}{\operatorname{argmax}} |\det \mathbf{C}\mathbf{U}_r| = \underset{\mathbf{C}}{\operatorname{argmax}} \prod_i |\lambda_i(\mathbf{C}U_r)|$$
$$= \underset{\mathbf{C}}{\operatorname{argmax}} \prod_i \sigma_i(\mathbf{C}U_r). \tag{3.4}$$

Unfortunately, for large systems this NP-hard, combinatorial search becomes computationally intractable. The QR decomposition with column pivoting of $\mathbf{U}_r^T$ offers an approximate greedy solution. The decomposition utilizes Householder transformations to find a column permutation matrix $\mathbf{P}^T$ such that

$$\mathbf{U}_r^T \mathbf{P}^T = \mathbf{Q}\mathbf{R} \tag{3.5}$$

where $\mathbf{Q}$ is orthogonal and $\mathbf{R}$ is upper triangular with decreasing, positive entries on the diagonal. Because at each iteration of the algorithm the diagonal entry of $\mathbf{R}$ is selected to be as large as possible and $\mathbf{Q}$ is orthogonal, we have a a greedy maximization of $\det(\mathbf{U}_r^T \mathbf{P}^T) = \det(\mathbf{P}\mathbf{U}_r)$. Thus, setting $\mathbf{C} = \mathbf{P}$ can improve the reconstructions obtained by POD based methods. We refer to reconstructions obtained in this manner as QR/POD.

3.2.2 Shallow decoder networks

Although QR/POD methods have demonstrated success in a variety of fields, the method can, at most, estimate the coefficients of the first n POD modes if there are n sensors available. The included POD modes are insufficiently expressive to obtain accurate reconstructions when exceedingly few sensors are available. This limitation motivates the development of more expressive, nonlinear methods for reconstruction. Among these are shallow decoder networks (SDN) [48].

As before, let

$$\mathbf{y} = \mathbf{C}\mathbf{x} \tag{3.6}$$

be a point sampling of a high-dimensional state $\mathbf{x}$. SDNs are shallow, fully-connected neural networks that learn a mapping from the space of sensor measurements, $\mathbf{y}$, back to $\mathbf{x}$. SDNs can be denoted as

$$\mathcal{F}(\mathbf{y}; \mathbf{W}_{SD}) := R(\mathbf{W}^b R(\mathbf{W}^{b-1} \cdots R(\mathbf{W^1 s}))) \tag{3.7}$$

where $\mathbf{y}$ is the input sensor data, R is a scalar, nonlinear activation function, $\mathbf{W}_{SD} = \{\mathbf{W^i}\}_{i=1}^{b}$ are trainable weights, and k denotes the number of layers in the SDN. Formally, we seek

$$\mathcal{F} \in \underset{\widetilde{\mathcal{F}} \in \mathscr{F}}{\operatorname{argmin}} \sum_{i=1}^{N} ||\mathbf{x}_i - \widetilde{\mathcal{F}}(\mathbf{y}_i)||_2, \tag{3.8}$$

so that reconstruction error is minimized over a set of N training states $\{\mathbf{x}_i\}_{i=1}^{N}$. The parameters of the network are randomly initialized and then trained by a gradient descent method. Once the network is trained, reconstructions given subsequent sensor measurements are found by

$$\hat{\mathbf{x}} = \mathcal{F}(\mathbf{y}). \tag{3.9}$$

SDNs can be trained for any set of sensor measurements, and unlike QR/POD are not entirely reliant on principled sensor placement schemes. Still, previous empirical work suggests that even neural-network based reconstructions can be improved through the use of QR placement [166]. Both SDNs and QR/POD have been used successfully in the reconstruction of high-dimensional dynamical systems, but also for broader classes of images [115]. This extension is possible because both methods rely only on static, point measurements and each reconstruction is performed individually. Dynamical systems, on the other hand, inherently depend on the temporal evolution of the state, and incorporating this dependence into a method for reconstruction offers a natural improvement upon existing techniques. Our work includes these dynamics through the use of trajectories of sensor measurements.

3.3 Shallow recurrent decoder networks

3.3.1 SHRED and separation of variables

The SHRED architecture is based upon the separation of variables technique for solving linear partial differential equations (PDEs) [54]. Separation of variables assumes that a solution can be separated into a product of spatial and temporal functions $u(x,t) = T(t)X(x)$.

This solution form is then used to reduce the PDE into ordinary differential equations: one for time $T(t)$ and one for space $X(x)$. Such a decomposition also constitutes the underpinnings of spectral methods for the numerical solution of the PDE, linear or nonlinear [95]. Consider the constant coefficient linear PDE

$$\dot{u} = \mathcal{L}(\partial_x, \partial_x^2, \cdots)u \tag{3.10}$$

where $u(x,t)$ specifies the spatio-temporal field of interest. Typically the initial condition (IC) and boundary conditions (BCs) are given by

$$\text{IC:} \qquad u(x,0) = u_0(x) \tag{3.11a}$$

$$\text{BCs:} \qquad \alpha_1 u(0,t) + \beta_1 u_x(0,t) = g_1(t) \text{ and}$$

$$\alpha_2 u(L,t) + \beta_2 u_x(L,t) = g_2(t). \tag{3.11b}$$

This may be generalized to systems of several spatial variables, or a system with no time dependence. The linear operator $\mathcal{L}$ specifies the spatial derivatives, which in turn model the underlying physics of the system. Simple examples of $\mathcal{L}$ include $\mathcal{L} = c\partial_x$ (the one-way wave equation) and $\mathcal{L} = \kappa\partial_x^2$ (the heat equation) [95].

The earliest solutions of linear PDEs assumed separation of variables whereby $u(x,t) = \exp(\lambda t)X(x)$ was a product of a temporal (exponential) function multiplied by a spatial function. The parameter λ is in general complex. This gives the eigenfunction solution of (3.10) to be

$$u(x,t) = \sum_{n=1}^{N} a_n \exp(\lambda_n t)\phi_n(x) \tag{3.12}$$

where $\phi_n(x)$ are the eigenfunctions of the linear operator and λ_n are its eigenvalues ($\mathcal{L}\phi_n(x) = \lambda_n\phi_n(x)$). Here a finite dimensional approximation N is assumed, which is standard in practice for numerical evaluation.

Typically, initial conditions $u(x,0) = u_0(x)$ are imposed in order to uniquely determine the coefficients b_n. Specifically, at time $t = 0$ (3.12) becomes

$$u_0(x) = \sum_{n=1}^{N} a_n\phi_n(x). \tag{3.13}$$

Taking the inner product of both sides with respect to $\phi_m(x)$ and making use of orthogonality gives

$$a_n = \langle u_0(x), \phi_n(x) \rangle \tag{3.14}$$

SHRED instead has measurements at a single spatial (sensor) location x_s, but with a temporal history. Thus if SHRED has, for example, N temporal trajectory points, this gives at each time point of the measurement:

$$u(x_s, t_j) = \sum_{n=1}^{N} a_n \exp(\lambda_n t_j)\phi_n(x_s) \quad \text{for } j = 1, 2, \cdots N. \tag{3.15}$$

This results in N equations for the N unknowns a_n. Specifically, the $N \times N$ system of equations $\mathbf{Ax} = \mathbf{b}$ is prescribed by the vector components $x_k = a_k$ and $b_k = u(x_s, t_k)$ and matrix components $(a_{kj}) = \exp(\lambda_k t_j)\phi_k(x_s)$. As with the initial condition (3.11a), the time trajectory of measurements at a single location uniquely prescribes the solution. This analysis can be generalized to allow for spatially mobile sensors or include multiple sensor measurements at a single time point. The former case is addressed in [46] and generally requires that the sensor's movement be periodic. In the latter case, if there are two measurements at a given time t_j, then only $N/2$ trajectory points are needed to uniquely determine the solution. Likewise, three sensor measurements at a given time requires only $N/3$ trajectory points. As a caveat, it should be noted that sensors must actually measure the physics in order to perform a reconstruction, i.e. if a shock wave has not hit the sensor yet, one cannot construct the spatio-temporal dynamics. Moreover, SHRED cannot reconstruct regions of space where the dynamics are statistically independent.

Thus, temporal trajectory information at a single spatial location, or with a moving sensor, can be equivalent to knowing the full spatial field at a given point in time. The motivation of SHRED is a nonlinear generalization of this concept. The proposed architecture has the form $u = X(T(\{y_i\}_{i=t-k}^{i=t}))$. In separation of variables, one assumes the form $u = X(x)T(t)$ which allows for the construction of differential equations for time and space separately: $dT/dt = cT$ and $\mathcal{L}X = cX$. However, they are not independent, but are related through the constant c. Specifically, the solution $T(t)$ is constrained to constants that are the eigenvalues of $X(x)$. In a SHRED model, an analogous result is obtained through the

joint training of X and T. Of course, as a generalization to nonlinear dynamics, rigorous theoretical bounds of SHRED are difficult to achieve, much like analytic and numerical solutions are difficult to rigorously bound in computational PDE settings.

That being said, standard numerical techniques for non-linear PDEs can also be obtained with time-series of sparse sensor measurements. Consider a spectral solution technique [95] whereby numerical solutions are approximated by a spectral basis

$$u(x,t) = \sum_{n=1}^{N} a_n(t)\phi_n(x). \tag{3.16}$$

This spectral decomposition turns the PDE into a system of N coupled ordinary differential equations with constants of integration that can be uniquely determined by requiring the solution satisfy N temporal trajectory points.

Coupled PDEs

We can also consider coupled, constant coefficient linear PDEs. For example, the coupled system

$$\dot{u} = \mathcal{L}_1 u + \mathcal{L}_2 v \tag{3.17a}$$

$$\dot{v} = \mathcal{L}_3 u + \mathcal{L}_4 v \tag{3.17b}$$

where $u(x,t)$ and $v(x,t)$ specifies the spatio-temporal fields of interest. The PDEs can be instead be written in the form

$$\ddot{u} = \mathcal{L}_1 \dot{u} + \mathcal{L}_2 \mathcal{L}_3 u + \mathcal{L}_2 \mathcal{L}_4 \left(\mathcal{L}_2^{-1} (\dot{u} - \mathcal{L}_1 u) \right) \tag{3.18}$$

where (3.17a) is differentiated with respect to time and (3.17b) is used in order to write the PDEs as a function of $u(x,t)$ alone. Thus knowledge of the field $u(x,t)$ alone is capable of constructing the solution fields $u(x,t)$ and $v(x,t)$. For this second-order (in time) PDE, both an initial condition $u(x,0)$ and an initial *velocity* require specification $\dot{u}(x,0)$ in order to uniquely determine the solution. As with the previous arguments, a time trajectory embedding of $2N$ measurements can be used to uniquely determine the solution.

3.3.2 SHRED architecture

To model temporal dynamics, we rely on recurrent neural networks (RNN) [143] and, more specifically, long short-term memory networks (LSTM). The most basic RNN accepts as an input a sequence of vectors $\{\mathbf{v}_i\}_{i=1}^t$ and outputs a single hidden state, $\mathbf{h}_t$. This can be expressed through the recursion relation

$$\mathbf{h}_t = R(\mathbf{W}_c\mathbf{v}_t + \mathbf{W}_r\mathbf{h}_{t-1} + \mathbf{b}_r) \tag{3.19}$$

where R is a scalar, nonlinear activation function, $\mathbf{W}_c$, $\mathbf{W}_r$ and $\mathbf{b}_r$ are trainable weights, and $\mathbf{h_0} = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}^T$. Generic RNNs are notorious for suffering from the vanishing gradient problem, causing them to have difficulty in identifying long term dependencies. LSTMs address this issue through the introduction of a so-called "gradient super-highway." Instead of outputting a single hidden state, LSTMs incorporate an additional "cell state" which only undergoes minor, pointwise operations, allowing gradients to flow easily from many time steps in the past. LSTMs are perhaps the most commonly used class of RNN at the time of writing, due to their ability to learn long and short term dependencies [107]. They are frequently used in time-series forecasting, natural language processing, and video analysis, among many other domains [107]. For a more complete discussion of LSTMs, we direct the reader to [68]. The recursion relation of an LSTM can be written as

$$\mathbf{h}_t = \sigma\left(\mathbf{W}_o\left[\mathbf{h}_{t-1}, \mathbf{v}_t\right] + \mathbf{b}_o\right) \odot \tanh(\mathbf{c}_t) \tag{3.20}$$

$$\begin{aligned}\mathbf{c}_t = \sigma\left(\mathbf{W}_f\left[\mathbf{h}_{t-1}, \mathbf{v}_t\right] + \mathbf{b}_f\right) \odot \mathbf{c}_{t-1} \\ +\sigma\left(\mathbf{W}_i\left[\mathbf{h}_{t-1}, \mathbf{v}_t\right] + \mathbf{b}_f\right) \odot \tanh\left(\mathbf{W}_g\left[\mathbf{h}_{t-1}, \mathbf{v}_t\right] + \mathbf{b}_g\right)\end{aligned} \tag{3.21}$$

for trainable weights and biases $\mathbf{W}_{RN} = \{\mathbf{W}_o, \mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_g, \mathbf{b}_o, \mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_g\}$. Both σ, the sigmoid function, and tanh operate pointwise. Note that $\mathbf{h}_t$ can be written as function in terms of only $\{\mathbf{v}_i\}_{i=1}^t$ and parameterized by the trainable weights and biases,

$$\mathbf{h}_t = \mathcal{G}(\{\mathbf{v}_i\}_{i=1}^t; \mathbf{W}_{RN}). \tag{3.22}$$

We now introduce the *SHallow REcurrent Decoder* (SHRED), which merges an LSTM with the SDN architecture. Suppose we have a multivariate time-series of a high-dimensional state $\{\mathbf{x}_i\}_{i=1}^T$ and corresponding measurements of the system $\{\mathbf{y}_i\}_{i=1}^T$. Rather than reconstruct $\mathbf{x}_i$ using only $\mathbf{y}_i$, as is done by QR/POD and SDN, we let an LSTM learn a latent representation using the previous k sets of sensor measurements. This latent representation is then used by a shallow decoder to reconstruct the high-dimensional state. The SHRED architecture can be written as

$$\mathcal{H}\left(\{\mathbf{y}_i\}_{i=t-k}^t\right) = \mathcal{F}\left(\mathcal{G}(\{\mathbf{y}_i\}_{i=t-k}^t; \mathbf{W}_{RN}); \mathbf{W}_{SD}\right) \tag{3.23}$$

using eqs. 3.7 and 3.22. As in the case of SDNs, we use a gradient descent method to train the network weights to minimize reconstruction loss,

$$\mathcal{H} \in \underset{\widetilde{\mathcal{H}} \in \mathscr{H}}{\operatorname{argmin}} \sum_{i=1}^{N} ||\mathbf{x}_i - \widetilde{\mathcal{H}}\left(\{\mathbf{y}_i\}_{i=t-k}^t\right)||_2. \tag{3.24}$$

$\mathcal{H}$, so defined, represents a map from sensor trajectories to the full-state.

3.4 Results

3.4.1 Reconstruction of high-dimensional spatio-temporal fields

We demonstrate the performance of SHRED trained with the Adam optimizer [89] on three example datasets. In each case, we compare the reconstruction error obtained by SHRED with randomly placed sensors to SHRED with QR placed sensors (QR SHRED), shallow decoder networks with randomly placed sensors (R-SDN), shallow decoder networks with QR placed sensors (Q-SDN), and linear reconstructions with QR placed sensors (QR/POD). The considered reconstruction error is defined to be the normalized mean square error over each state in a test set [48],

$$\text{Error} = \frac{||\mathcal{H}\left(\{\mathbf{y}_j\}_{i-k}^i\right) - \mathbf{x}_i||_2}{||\mathbf{x}_i||_2}. \tag{3.25}$$

Because SHRED models rely on trajectories of sensor measurements to perform state estimation, we truncate each dataset to reconstruct only the final $N-k$ temporal snapshots,

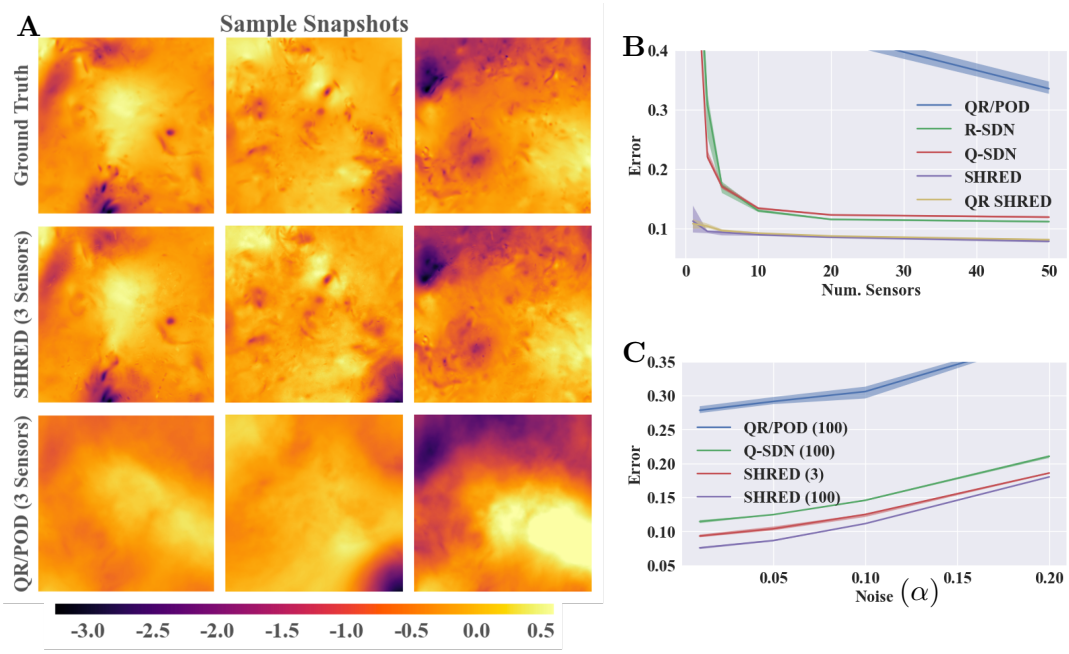


Figure 3.2: Turbulent flow summary performance. *A* Example reconstructions obtained via SHRED and QR/POD of a turbulent flow when three sensors are available. Ground truth included for comparison. *B* Reconstruction errors of static sensing methods (QR/POD) and SHRED with a varying number of available sensors. The solid lines denote the median error from 32 trained estimators and the shaded region denotes the interquartile range. *C* Performance of reconstruction methods in the presence of varying levels of added Gaussian white noise. The added noise has mean zero and standard deviation equal to α times the mean absolute value of the field. The number in parentheses denotes the number of available sensors.

where N is the initial number of samples and k is the length of the utilized trajectories. This length can be viewed as a hyper-parameter that can be tuned according to the data. Detailed network parameters and training protocols can be found in the SI. The used parameters of each model were selected via hyperparameter search. Finally, we note that in this section, training, validation, and test samples are temporally interspersed. In later sections, we demonstrate results in the case that training and test sets are temporally distinct, although in all cases we assume no errors in sensor drift or synchronisation. The study of our method under these more difficult conditions is an important area of future work.

The first application we consider is the pressure field of a forced isotropic turbulent flow from the Johns Hopkins Turbulence Database [104]. The flow was generated by direct numerical simulation using 1024^3 nodes and the pseudo-spectral method. We select a 350 by 350 two dimensional cutout of computed pressure over 1667 evenly spaced temporal snapshots from the three dimensional simulation. The temporal spacing of the snapshots is approximately 0.006 seconds, while the DNS used to generate the data utilized time steps of 0.0002 seconds. We seek to reconstruct these high-dimensional states from trajectories of point measurements of the states. There are many options for determining the optimal parameter k, where $k\Delta t$ is the maximal time-delay utilized, for reconstructions generated by SHRED. In this case, a simple hyperparameter search over $k \in \{10, 25, 50, 75, 100\}$ was used with the number of sensors held constant at 3. From these possible k, the best performing value of $k = 100$ was selected. Correspondingly, the final 1567 temporal snapshots of the pressure field are randomly split into training, validation, and test sets consisting of 1100, 234, and 233 snapshots, respectively. For each considered number of sensors, we generate 32 reconstructions with all considered methods. We plot the median performance and denote the interquartile range by the shaded region. These results are shown in panel B of Fig. 3.2. Even with only a single, randomly placed sensor, the reconstructions obtained by SHRED yield significantly lower error than competing methods with as many as 50 sensors. Moreover, placement via the greedy QR algorithm appears to have a negligible impact on reconstructive performance. Panel A of Fig. 3.2 shows sample reconstructions obtained by SHRED and QR/POD with three sensors. While QR/POD is only able to identify large scale features, SHRED accurately reconstructs fine grain features as well. Numerically, with

1 sensor the median computed error of QR/POD is 0.89 while with SHRED the median computed error is 0.11, an approximately 8 times improvement in performance. The poor performance of QR/POD in this example highlights a well-known limitation of gappy-POD based techniques. The turbulent flow is both tri-periodic and nonstationary, presenting challenges that POD is fundamentally unable to handle. Such techniques remain widespread in part because of their relative interpretability and in part because there remains a lack of methods for inferring high-dimensional states from sparse sensor measurements alone. The comparison to POD illustrates the need for nonlinear methods in such cases.

In the vast majority of real world applications, and in contrast to numerical simulations, data is often corrupted by noise. As a result, methods for state estimation must exhibit resilience to noise. To measure this resilience, we corrupt the data with Gaussian noise of mean zero and standard deviation of $\alpha \times |\bar{x}|$, where $|\bar{x}|$ is the average absolute value over all points in all snapshots of the training set. We then generate 32 state estimates exactly as before using all considered methods. The resulting median reconstruction error and interquartile range for varying α is shown in panel C of Fig. 3.2. Again, SHRED outperforms competing techniques using a far greater number of sensors.

The second application we examine is that of sea-surface temperature (SST). We consider weekly mean sea-surface temperature from the years 1992 to 2019 as reported by NOAA [141]. Unlike the previous example of a simulated turbulent flow, SST is a sensor generated dataset for which governing equations are not known and thus represents a more practical application of SHRED. The data consists of 1400 snapshots of a 180 by 360 grid, of which 44,219 spatial locations correspond to the sea-surface. Each snapshot corresponds to the weekly mean sea-surface temperature for the respective week. In this case, it is easy to make physical arguments for determining k, the number of lags used by SHRED. Since each data snapshot represents weekly mean sea-surface temperature, we expect a strong signal with period 52 corresponding to annual fluctuations in temperature. As a result, we select $k = 52$ to capture this annual variation. We allow the input trajectories to SHRED to have a length of 52, corresponding to one year of measurements. The final 1348 samples are divided into training, test, and validation sets consisting of 1000, 174, and 174 snapshots, respectively. Analogous to Fig. 3.2, Fig. 3.3 shows example reconstructions and

reconstruction error distributions of SHRED and existing techniques both in the presence and absence of added Gaussian noise. The contour plots shown in panel A of Figure 3 show the ground truth and reconstructions with the mean field subtracted. We choose to present this visualization because in comparison to other example datasets, the global mean-field dominates and the mean subtracted contour plots more clearly demonstrate the difference between QR/POD and SHRED. Again, the reconstructions obtained by SHRED are both visually and empirically superior when few sensors are available. When one sensor is available, the median error from QR/POD (0.12) is approximately 6 times greater than that of SHRED (0.02). Again, the reconstructions obtained by SHRED are both visually and empirically superior, while requiring far fewer sensors, which can be randomly placed without a loss of accuracy.

The last system for which we consider the performance of SHRED is a simulation of atmospheric chemistry. Chemical transport models (CTM) simulate the evolution of an ensemble of interacting chemical species through a transport operator [13, 159]

$$\frac{\partial n_i}{\partial t} = -\nabla \cdot (n_i \mathbf{U}) \tag{3.26}$$

and a chemical operator

$$\frac{dn_i}{dt} = (P_i - L_i)(\mathbf{n}) + E_i - D_i, \tag{3.27}$$

where each entry $\mathbf{n} = \begin{bmatrix} n_1 & n_2 & \cdots n_K \end{bmatrix}^T$ represents the number density of a specific chemical species, $\mathbf{U}$ is the wind vector, $(P_i - L_i)(\mathbf{n})$ is the local chemical production and loss term, E_i the emission rate of a species, and D_i the deposition rate. The output of a CTM, then, consists of concentrations for K chemical species for a grid of latitudes, longitudes, and elevations. The data we consider is drawn from the work of Velegar et al. [159] and contains simulated atmospheric ozone concentration generated by the CTM software GEOS-Chem [13] over the course of a year with dynamical time steps of 20 minutes. The accessed data from [159] is a compressed SVD representation using the first 50 POD modes. The decompressed data matrix consists of 26,208 snapshots of a 46 by 72 by 30 (latitude, longitude, elevation) grid with temporal spacing of 20 minutes, from which we

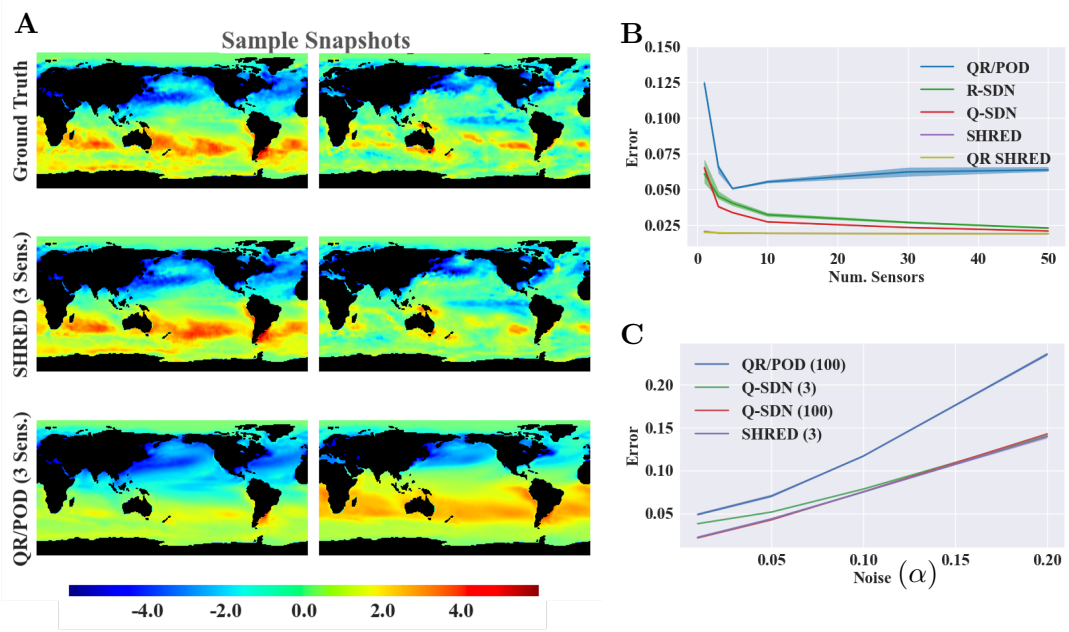


Figure 3.3: SST summary performance. *A* Example reconstructions obtained via SHRED and QR/POD of sea-surface temperature when three sensors are available. Ground truth included for comparison. *B* Reconstruction errors of static sensing methods (QR/POD) and SHRED with a varying number of available sensors. The solid lines denote the median error from 32 trained estimators and the shaded region denotes the interquartile range. *C* Performance of reconstruction methods in the presence of varying levels of added Gaussian white noise. The added noise has mean zero and standard deviation equal to α times the mean absolute value of the field. The number in parentheses denotes the number of available sensors.

further downsample to obtain 2,600 global ozone concentration fields with temporal spacing of roughly 200 minutes. To account for the fact that the data matrix is inherently rank 50, we add Gaussian noise with standard deviation $\alpha = 0.05$ to the data before performing any analyses. Identical to the turbulent flow example, $k = 100$ for the reconstructions generated by SHRED was chosen by a hyperparameter search. The resulting snapshots are divided into training, validation, and test sets consisting of 1900, 300, and 300 entries, respectively.

Fig. 3.4 shows the performance of SHRED and existing techniques on this atmospheric ozone experiment. With 1 sensor, the median computed error of QR/POD (0.44) is roughly 7 times that of SHRED (0.06). However, in this case when many sensors are available (> 50) QR/POD performs comparably to nonlinear reconstructions. This performance is an artifact of the use of a compressed, rank 50 representation of the data.

We note that for both the atmospheric ozone concentration and turbulent flow results, we have trained on synthetic data from validated high-fidelity models. As full-state measurements of physical high-dimensional systems are frequently impossible, such training procedures are common in data-driven reconstruction methods. However, the performance of any machine learning model trained using synthetic data is critically reliant upon the fidelity of the model used to generate the synthetic data. In real-world applications devoid of a well-validated numerical model or a training set of high-dimensional state measurements, it is unlikely that any ML model mapping from sparse sensor measurements to the full-state will perform well. One manner in which this simulation-to-reality gap can be addressed is by performing an expensive data collection period using a large number of deployed sensors. A shallow recurrent decoder could then be used to map from a small number of sensors to the larger set of sensor measurements as opposed to the full state. After the training period, costs can be reduced by removing many of the sensors that were initially deployed and interpolating between the sensor locations to which the map has been learned.

3.4.2 Forecasting of high-dimensional spatio-temporal fields

In this section, we explore how the expressiveness of SHRED models with few sensors can be leveraged to perform forecasts of high-dimensional spatio-temporal states using only

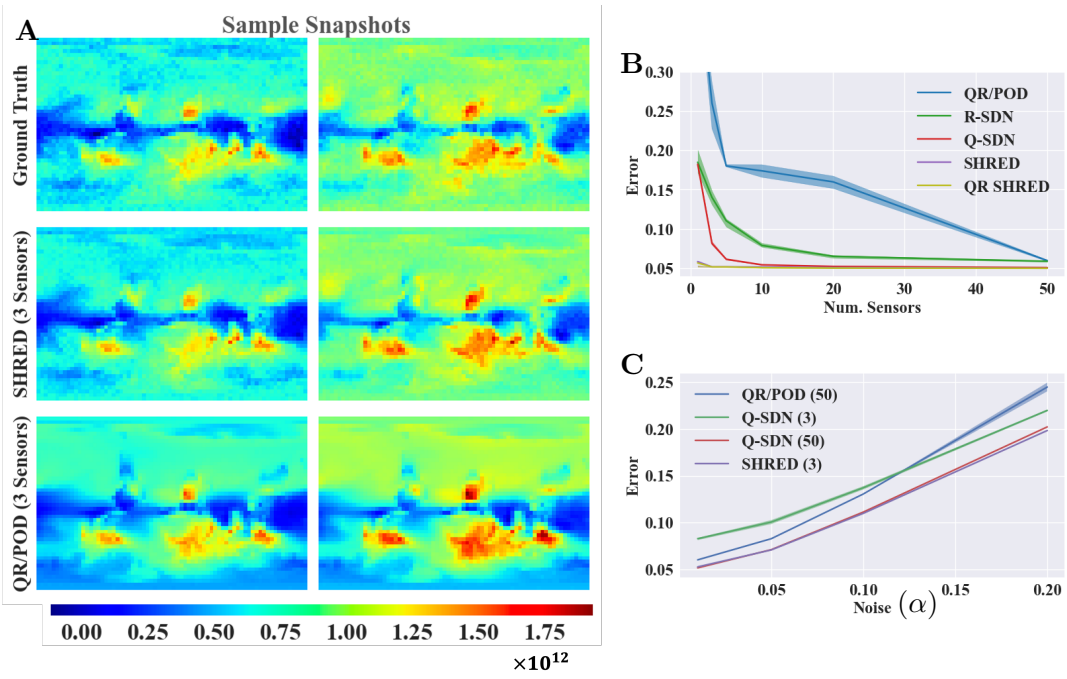


Figure 3.4: Ozone summary performance. *A* Example reconstructions obtained via SHRED and QR/POD of atmospheric ozone concentration when three sensors are available. Only one of thirty elevations is shown. Ground truth included for comparison. *B* Reconstruction errors of static sensing methods (QR/POD) and SHRED with a varying number of available sensors. The solid lines denote the median error from 32 trained estimators and the shaded region denotes the interquartile range. *C* Performance of reconstruction methods in the presence of varying levels of added Gaussian white noise. The added noise has mean zero and standard deviation equal to α times the mean absolute value of the field. The number in parentheses denotes the number of available sensors.

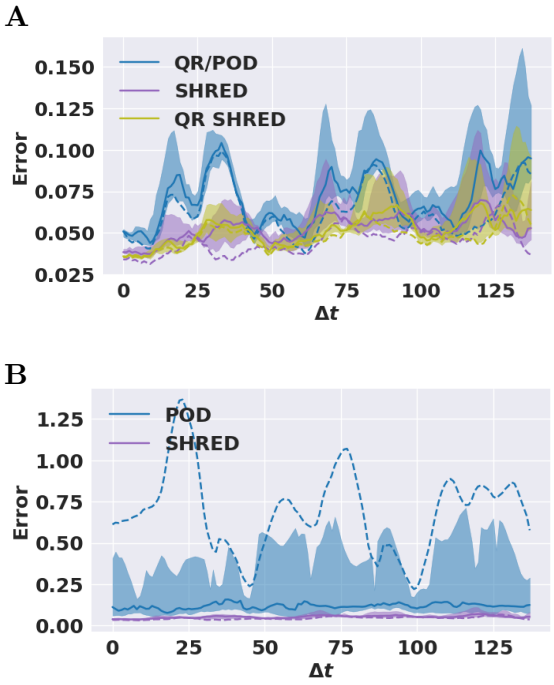


Figure 3.5: Forecasting error for sea-surface temperature with Δt as one week. A LSTM is trained to forecast QR placed, panel A, or randomly placed, panel B, sensor measurements which are subsequently used to perform reconstructions using SHRED and gappy POD. 16 forecasts are performed and the median error is denoted by the solid lines, with the 25th and 75th percentiles of reconstruction error defining the shaded region. The dashed line represents an ensembled forecast.

a limited subsampling of the state. Forecasting the evolution of high-dimensional states from sensor measurements is an exceedingly challenging task, owing to the fact that doing so combines the problem of system identification with the difficulties of forecasting. We propose a two step approach that leverages the success of LSTMs for low-dimensional time-series forecasting [107] and SHRED for state estimation.

Let $\{\mathbf{x}_i\}_1^{T_t}$ represent a training dataset of temporally ordered states with corresponding sensor measurements $\{\mathbf{y}_i\}_1^{T_t}$. We train an LSTM network, $\mathcal{G}\left(\{\mathbf{y}_i\}_{t-k}^{t}\right)$, to map between a trajectory of sensor measurements and the subsequent measurement, $\mathbf{y}_{t+1}$, by finding

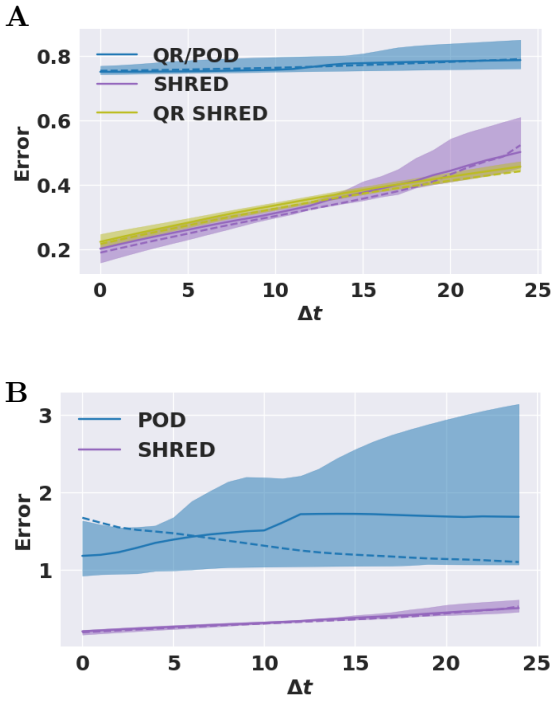


Figure 3.6: Forecasting error for turbulent flow with $\Delta t = 0.006$s. A LSTM is trained to forecast QR placed, panel A, or randomly placed, panel B, sensor measurements which are subsequently used to perform reconstructions using SHRED and gappy POD. In this experiment, the validation set is selected from earlier in the data so that the forecast occurs immediately after following training data. 16 forecasts are performed and the median error is denoted by the solid lines, with the 25th and 75th percentiles of reconstruction error defining the shaded region. The dashed line represents an ensembled forecast.

$$\mathcal{G} \in \underset{\widetilde{\mathcal{G}} \in \mathscr{G}}{\operatorname{argmin}} \sum_{i=1}^{N} ||\mathbf{y}_{t+1} - \widetilde{\mathcal{G}}\left(\{\mathbf{y}_j\}_{i-k}^{i}\right)||_2 \tag{3.28}$$

using the ADAM optimizer on the training data. Doing so, we then forecast beyond the time interval of the training data to obtain forecasted sensor measurements $\{\hat{\mathbf{y}}_i\}_{T_t+1}^{T_t+p}$ for $p > 1$. The forecasted measurements are used by a SHRED model, trained on $\{\mathbf{x}_i\}_1^{T_t}$, to obtain forecasts of the high-dimensional state $\mathbf{x}$. Detailed training protocols are included in SI.

The forecasted states $\{\hat{\mathbf{x}}_i\}_{T_t+1}^{T_t+p}$ are evaluated against the ground truth for each Δt forecast rather than the mean error across all forecasts to demonstrate performance over forecasts of varying length. The error at time $p\Delta t$ is thus given by

$$\text{Error} = \frac{\left|\left|\mathcal{H}\left(\{\hat{\mathbf{y}}_j\}_{p-k}^{p}\right) - \mathbf{x}_p\right|\right|_2}{||\mathbf{x}_p||_2}. \tag{3.29}$$

We also include reconstructions from forecasted sensor measurements obtained by gappy POD for comparison, akin to the work of [134]. Finally, we only show results for SST and turbulent flow data, as the compressed form of the atmospheric ozone data is biased towards both QR placement and POD reconstructions.

For the SST example, we select the first 85% of the dataset to act as training data, the subsequent 20 snapshots as validation data, and the remainder as the test set. A LSTM for forecasting is trained on the training data and sensor measurements are forecast beyond the validation set. These forecasted sensor measurements are then used to construct a forecast of the high-dimensional spatio-temporal field using a trained SHRED model and gappy POD. We consider the cases that the sensors are placed via QR or randomly. We perform 16 runs for each experiment, and consider an ensembled forecast found by averaging the forecast of each run. The results for QR placed and randomly placed sensors are shown in Fig. 3.5 with Δt representing one week. With QR placement, the forecasts obtained by SHRED outperform that of POD in the short-term and are similar for longer forecast horizons. However, with randomly placed sensors SHRED is still able to obtain comparably accurate forecasts while POD fails to consistently yield faithful reconstructions.

Unlike sea-surface temperature, even medium-range forecasts of a turbulent flow are impossible due to the fact that the flow is not quasi-periodic or stationary in nature. For this reason, we focus on achieving accurate short-term forecasts in this application. To do so, we use the same scheme as before, with the exception that the validation set is selected to occur earlier in the training data. Of the 1567 snapshots of the turbulent flow with sufficiently long preceding measurement histories, the first 1000 are selected as training, followed by 50 samples selected for validation. The next 100 samples are also used for training and the remainder constitute the test set. This scheme allows for more accurate short-term forecasts because the forecast occurs directly after the training data, as opposed

to directly after the validation set. Fig. 3.6 shows the results for the cases that sensors are placed via QR and randomly with Δt representing 0.006 seconds. In both cases, the forecast obtained by SHRED deteriorates quickly but greatly outperforms that obtained by POD reconstructions.

3.4.3 Temporal spacing of time-delays

To this point, we have only considered SHRED models where the temporal spacing between utilized sensor measurements is equal to the discretization, Δt, of the data snapshots. However, time-delay embedding techniques can often be sensitive to the selection of both the maximal time-delay incorporated, $k\Delta t$, and the temporal spacing of those time-delays, $n_{\Delta t}\Delta t$ [1]. In this notation, Sections 3.4.1 and 3.4.2 show results for $n_{\Delta t} = 1$ and in this section we consider the performance of SHRED for varying $n_{\Delta t}$.

We perform two sets of experiments for each considered dataset. The first varies $n_{\Delta t}$ but keeps k approximately constant. This corresponds to a decrease in the total number of sensor measurements in the input to a SHRED model. The second set of experiments varies $n_{\Delta t}$ and k such that the total number of utilized sensor measurements is held constant. As a result, the first $N - kn_{\Delta t}$ snapshots in each dataset must be discarded. We therefore modify the number of samples in the training, validation, and test sets such that there are 900 training samples for the turbulent flow, 1000 for the SST, and 1000 for the atmospheric ozone data. The results for both experiments are reported in Fig. 3.7. In both cases, all other network hyperparameters and the manner in which the train/validation/test split is performed remain unchanged from Section 3.4.1.

We find that the performance of SHRED in the cases of SST and atmospheric ozone concentration remains largely unchanged when $n\Delta t$ is varied, provided k is also scaled to use the same number of sensor measurements. If $n_{\Delta t}$ is increased and k is held fixed, performance deteriorates for larger values of $n_{\Delta t}$.

On the other hand, our results are less clear when SHRED is applied to the turbulent flow data. It appears that commensurately increasing $n_{\Delta t}$ and k moderately improves performance, while increasing only $n_{\Delta t}$ does not have a strong effect on reconstruction

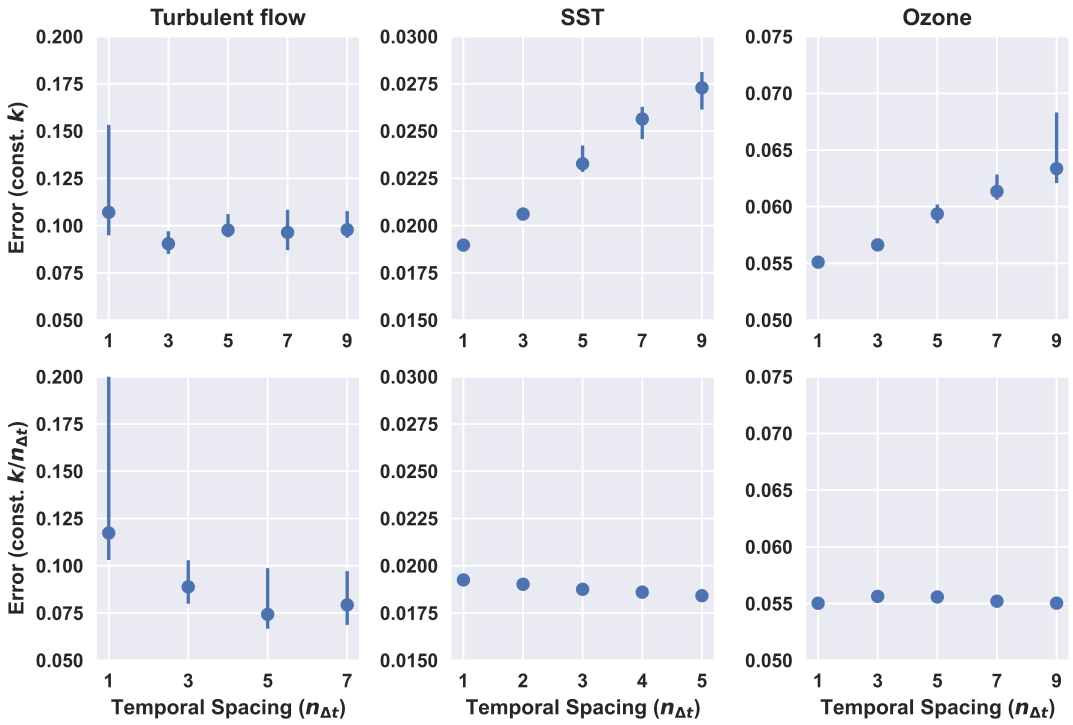


Figure 3.7: Error results for SHRED models in the three considered datasets when the temporal spacing of the trajectories used for reconstruction is varied. k denotes the maximal time-delay of the trajectories used by a SHRED model. $n_{\Delta t}$ denotes the temporal spacing between utilized time-delays in terms of of the discretization of the data, Δt. The top row considers the case where k is approximately fixed to the values specified in Section 3.4.1 and the bottom row considers the case where k varies in addition to $n_{\Delta t}$ such that $k/n_{\Delta t}$ is fixed to the values of k specified in 3.4.1. Each plot shows the median and interquartile range of reconstruction error from 32 trained SHRED models.

accuracy. However, it is difficult to draw strong conclusions from these results as there are strong indications that 900 training samples are insufficient to train the model. This is illustrated by moderately worse median performance and much larger variance for the cases $n_{\Delta t} = 1$ in Fig. 3.7. The only difference between these models and those from Section 3.4.1 is a reduction in access to training data. The effect of this reduction is likely more visible in the turbulent flow data than SST or atmospheric ozone due to more complex intrinsic dynamics.

3.5 Discussion

We have demonstrated SHRED as a method for system identification/state estimation and the forecasting of high-dimensional time-series data. By including sensor trajectories in the model for reconstruction, SHRED outperformed competing techniques based on static reconstructions from gappy POD or shallow decoder networks. We considered three high-dimensional, spatio-temporal datasets, a synthetically generated forced turbulent flow, a simulation of atmospheric ozone concentration from a chemical transport model, and weekly mean sea-surface temperature.

The superior performance of SHRED held across both interpolatory (reconstruction) and extrapolatory (forecasting) regimes. In Section 3.4.1, the task was purely an interpolatory reconstruction. Training and test data, upon which we evaluate reconstruction accuracy, were drawn from the same distribution. As a result, overfitting to the training data was not an issue, and SHRED was able to capture fine grain details of complex flows with as few as one or three sensors. Applied in this manner, SHRED offers an attractive method for compression of large datasets as well as a framework for developing reduced order models of dynamical systems. The performance of SHRED in this section is also indicative of the promise of SHRED for high-dimensional data that is stationary or periodic. Notably, SHRED demonstrated remarkable indifference towards sensor placement. In this interpolatory regime, issues arising from the necessity of optimal sensor placement can be mitigated by increasing the information content of inputs to a neural network through the use of sensor trajectories. These results show that the sensor trajectories encode a significant amount of information, just as is expected of time-delayed embeddings of dy-

namics [6, 10, 21, 81, 103, 131].

In Section 3.4.2, the task at hand was forecasting the evolution of high-dimensional spatio-temporal data. In contrast to Section 3.4.1, training and test sets are temporally separated and thus the results are extrapolatory in nature. High-dimensional forecasts were accomplished by forecasting low-dimensional sensor measurements and constructing states from the predicted measurements. Forecasting is an inherently difficult task for complex systems, especially in the high-dimensional fields often encountered in the engineering and the physical sciences. Still, SHRED's use of sensor trajectories allows for improved forecasting performance as compared to existing methods. Furthermore, SHRED for forecasting exhibits robustness to sensor placement not found in the other methods considered. We thus advocate for the use of SHRED for both the tasks of state estimation and state prediction from forecasted sensor measurements. Further work must be done to improve the accuracy of low-dimensional sensor forecasts to be used in conjunction with SHRED for state prediction. Improving the forecasts of high-dimensional data could enable on-the-fly compression and novel sensing mechanisms.

The comparisons in this chapter are made against gappy POD with QR-based sensor placement, which has historically been among the most widely used approaches for reconstruction from sparse measurements. Since the development of these methods, a number of regularized, Bayesian, and tensor-based formulations have been proposed that relax the dependence on optimal sensor placement, accommodate undersampling, and quantify reconstruction uncertainty [51, 53, 79, 90]. A deeper comparison of SHRED against these more recent reconstruction techniques is an important subject of future work.

Finally, future work will explore the extension of SHRED to multi-modal data, as briefly discussed in the section on coupled PDEs. This extension can be viewed as an augmentation of the full-state representation to include sensor measurements from various modalities. The inputs to the network then have the interpretation of encoding both the spatial location and sensing modality. As in the cases considered here, the application of SHRED in multi-modal settings will require a high-fidelity model with priors that well-represent the physical system of interest. Provided such a model and coupling between the field of observations and augmented full-state, in principle it is possible to apply our methodology to reconstruct

the full-state from measurements of one of its constituent fields.

Chapter 4

RESERVOIR COMPUTING FOR SYSTEM IDENTIFICATION AND MODEL PREDICTIVE CONTROL

4.1 Introduction

Recent advances in the availability of large-scale data and computational resources have driven transformative progress in neural network performance, with applications ranging from natural language processing [158, 177] to protein design [91] to advanced manufacturing [7]. These successes have relied heavily on immense datasets and computational budgets. In contrast, many problems in science and engineering are characterized by data scarcity and strict computational constraints [24]. This limitation is especially true for system identification in control settings, in which dynamics must be modeled with limited observational data obtained under diverse control inputs.

Neural networks have long been explored as tools for dynamical system modeling. Early work employed small feedforward or wavelet networks to approximate system trajectories [98, 106]. Such architectures continue to be used in control contexts, such as photovoltaic systems and electrical drive systems [9,61,126]. However, as architectures and computational resources have matured, increasingly sophisticated models have been developed, culminating in modern time series foundation models [5, 105, 172]. Somewhere between these extremes lie recurrent neural networks (RNNs). RNNs are a particularly natural fit for modeling dynamical systems, as they were explicitly designed to process sequential data such as time series [68, 143].

RNN-based models of dynamical systems typically follow one of two paradigms: RNNs trained by backpropagation through time (BPTT) or reservoir computing (RC) approaches, often called echo state networks (ESNs). Their variants have been successfully applied to high-dimensional and chaotic systems, with studies comparing a wide variety of architectures [161]. However, most of this research has focused on autonomous dynamics, leaving

important questions about forced or controlled systems.

Learning the dynamics of forced systems presents unique challenges. Training data are typically generated under one set of input signals, but effective control requires accurate prediction under novel inputs and unexplored regions of phase space. Surrogate models must therefore generalize to new control regimes while remaining lightweight enough for online use. Given such a surrogate, model predictive control (MPC) provides an attractive control framework. MPC is a widely used control strategy that computes optimal actions over a receding horizon to steer the system toward a target trajectory or state [92, 146]. Its flexibility and proven success in managing nonlinear, multi-input, multi-output (MIMO) systems have led to widespread adoption in engineering practice.

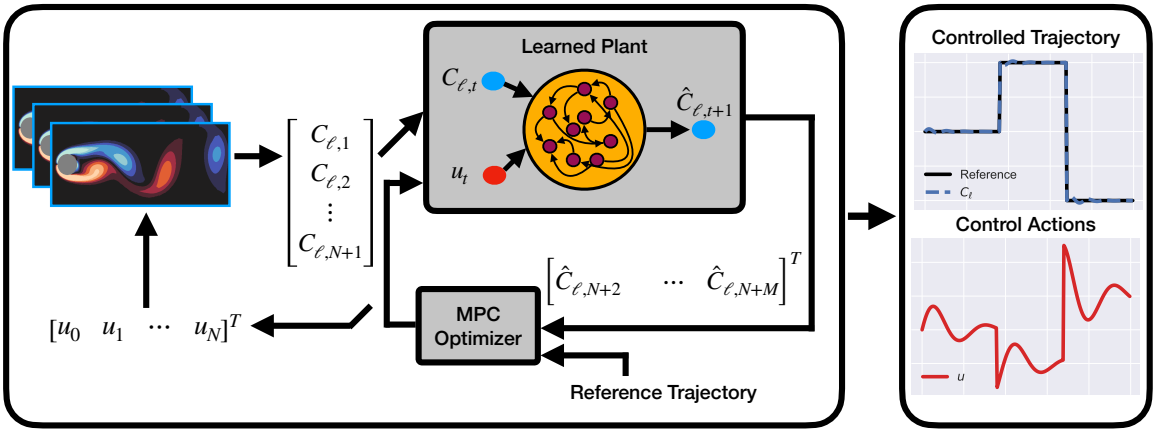


Figure 4.1: Summary diagram of echo state network (ESN) based model predictive control (MPC). An ESN trained to model plant dynamics acts as a surrogate model capable of rapidly forecasting the system under given control inputs. Online optimizations compute a sequence of control actions that minimize the deviation of the plant from a reference trajectory, yielding a controlled trajectory that tracks the reference.

A growing body of work has explored the integration of MPC with neural network surrogates. Fully connected, convolutional, and recurrent network architectures have all been proposed [14, 43, 118, 124]. In particular, both BPTT- and RC-based surrogate models have demonstrated success in enabling MPC for nonlinear systems [8, 71, 75, 76, 163]. BPTT-

trained RNN-based MPC has even been applied to challenging chaotic fluid systems [14]. Despite progress with both approaches, the relative strengths and limitations of ESN-based versus BPTT-trained, RNN-based MPC have not been systematically characterized. Key open questions remain regarding predictive accuracy, control performance, and generalization. Direct comparisons between the two paradigms remain limited, and results across studies are difficult to reconcile due to differences in benchmarks, training protocols, and evaluation criteria [173]. Addressing these gaps is critical for guiding the design of data-driven control strategies. This motivates our work: providing a systematic comparison of RC- and BPTT-based surrogate models for model predictive control in the context of challenging and complex dynamics.

Here, we benchmark ESN-based MPC against BPTT-trained RNNs across a set of representative nonlinear systems, ranging from canonical, low-dimensional examples to fluid flows. Our results demonstrate that ESNs achieve excellent accuracy, generalization, and efficiency, positioning them as a compelling candidate for data-driven control. A summary diagram of this approach is shown in Fig. 4.1. Specifically, we find that:

- ESN surrogate models require significantly less training data than networks trained via backpropagation and gradient descent.

- ESNs are substantially faster to train than networks trained via backpropagation and gradient descent.

- In standardized comparisons with LSTM networks, gated recurrent unit (GRU) networks, fully connected networks (FCNs) [42], and dynamic mode decomposition with control (DMDc) [140], ESNs demonstrated superior control performance and more accurate dynamics approximation across all nonlinear benchmark systems.

- For the two most challenging benchmark tasks (controlling lift in flow past a cylinder and steering the chaotic Lorenz system to fixed points), ESNs were the only surrogate model capable of achieving the control objective with limited training data.

The remainder of the paper is organized as follows: Section 2 provides technical background for model predictive control, linear system identification, fully connected networks, and RNNs. In Section 3, we detail the instantiation and training of an ESN surrogate model and provide a pseudocode algorithm of the control scheme. Section 4 reports open-loop results for the different forms of surrogate model considered on a variety of benchmark systems, assessing how well each class of surrogate forecasts given a predetermined control input. In Section 5, we demonstrate the performance of each trained surrogate in closed-loop MPC. Finally, Section 6 summarizes our key findings and future directions for this work.

4.2 Background

Model predictive control (MPC) is the iterative optimization of a finite-horizon, open-loop optimal control problem, requiring re-optimization for new time horizons. Typically, MPC assumes a discrete-time model of system dynamics, $\hat{\mathbf{x}}_{j+1} = f(\mathbf{x}_j, \mathbf{u}_j)$ where $\mathbf{x}_j$ is the system state at time j and $\mathbf{u}_j$ is the exogenous control signal. Redefining $t = 0$ for each optimization, we pose a cost function of the form

$$J(U,t) = \sum_{j=1}^{T} \epsilon_j^T(f) Q_1 \epsilon_j(f) + \mathbf{u}_j^T Q_2 \mathbf{u}_j + \Delta\mathbf{u}_j^T Q_3 \Delta\mathbf{u}_j$$
$$\epsilon_j(f) = \mathbf{r}_j - f(\hat{\mathbf{x}}_{j-1}, \mathbf{u}_{j-1}) \tag{4.1}$$

where $U = \{\mathbf{u}_{t+1}, \ldots, \mathbf{u}_{t+M}\}$, $\mathbf{r}_j$ is the desired system state at time j, and $\Delta\mathbf{u}_j = \mathbf{u}_j - \mathbf{u}_{j-1}$. Q_1 is a positive semi-definite weight matrix penalizing deviation from the desired trajectory, and Q_2, Q_3 are positive definite matrices that penalize the magnitude of the control input and rapid control fluctuations, respectively. T is the finite forecast horizon of the model predictive controller, and M is the number of control actions over which to optimize and must satisfy $M \leq T$. After the M-th step of the forecast, control actions are frequently set to a fixed value [146]. At time t, the model predictive controller computes

$$U_* = \underset{U \in \mathcal{B}}{\operatorname{argmin}}\, J(U,t) \tag{4.2}$$

where $\mathcal{B}$ defines the set of feasible control actions.

The requirements of a surrogate model to be used with MPC are twofold. First, the model must be capable of generating accurate forecasts of the control system given a sequence of control inputs. Second, the model must provide computationally inexpensive predictions to enable online minimization of the MPC objective function. In this work, we consider a total of five candidate architectures for data-driven surrogate modeling. Each model is trained for one-step forecasts that can be recursively applied for multi-step prediction.

4.2.1 Linear System Identification

The control of linear time-invariant (LTI) systems has been studied for over a century and many control techniques exist when such a model is available, including linear quadratic regulators (LQR), linear quadratic Gaussian regulators (LQG), proportional-integral-derivative (PID) control, etc. For discrete-time systems with state dimension m, control dimension n, and measurement dimension p, such methods assume a system of the form

$$\begin{aligned} \mathbf{x}_{j+1} &= \mathbf{A}\mathbf{x}_j + \mathbf{B}\mathbf{u}_j \\ \mathbf{y}_j &= \mathbf{C}\mathbf{x}_j, \end{aligned} \tag{4.3}$$

where $\mathbf{x}_j \in \mathbb{R}^m$ represents the system state at time j, $\mathbf{y}_j \in \mathbb{R}^p$ denotes the corresponding measurement of the system, and $\mathbf{u}_j \in \mathbb{R}^n$ is the exogenous control input. $\mathbf{A} \in \mathbb{R}^{m\times m}$ is the system matrix, $\mathbf{B} \in \mathbb{R}^{m\times n}$ is an input matrix, and $\mathbf{C} \in \mathbb{R}^{p\times m}$ is a measurement or sensing matrix. Because the control of linear systems is well-understood and the model is interpretable, constructing a data-driven model of the form Eq. 4.3 has been an attractive paradigm for system identification. In the simplest case when full-state measurements are available, i.e. $\mathbf{C} = \mathbb{I}$, the identity matrix, one assumes access to a set of training data pairs $(\mathbf{x}_j, \mathbf{u}_j)$ for $0 \leq j \leq N$.

From this training data, we construct data matrices of the form

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} \mathbf{x}_0 & \mathbf{x}_1 & \cdots & \mathbf{x}_{N-1} \end{bmatrix}, \\ \mathbf{X}' &= \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{N} \end{bmatrix}, \\ \mathbf{U} &= \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \cdots & \mathbf{u}_{N-1} \end{bmatrix}. \end{aligned} \tag{4.4}$$

The matrices $\mathbf{A}$ and $\mathbf{B}$ can then be estimated using a linear regression problem of the form

$$\underset{\Theta}{\operatorname{argmin}} \| \begin{bmatrix} \mathbf{X}^T & \mathbf{U}^T \end{bmatrix} \Theta - (\mathbf{X}')^T \|_F^2, \tag{4.5}$$

for the parameters $\Theta = \begin{bmatrix} \mathbf{A} & \mathbf{B} \end{bmatrix}^T$. For an LTI system with nondegenerate eigenvalues and no noise, this procedure exactly recovers $\mathbf{A}$ and $\mathbf{B}$.

A similar result can be obtained when $\mathbf{C} \neq \mathbb{I}$ through the incorporation of time-delays in the model, provided the resulting system is observable. This follows from the fact that an n-th order ordinary differential equation (ODE) can be rewritten as a system of n first order ODEs and vice versa. Let k denote the number of included time-delays and construct the time-delayed matrices given by

$$\begin{aligned} \mathbf{Y} &= \begin{bmatrix} \mathbf{y}_k & \mathbf{y}_{k+1} & \cdots & \mathbf{y}_{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{y}_0 & \mathbf{y}_1 & \cdots & \mathbf{y}_{N-k} \end{bmatrix} \\ \mathbf{Y}' &= \begin{bmatrix} \mathbf{y}_{k+1} & \mathbf{y}_{k+2} & \cdots & \mathbf{y}_N \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_{N-k+1}. \end{bmatrix}. \end{aligned} \tag{4.6}$$

As before, we learn a dynamical operator, Θ, using linear regression of the form

$$\underset{\Theta}{\operatorname{argmin}} \| \begin{bmatrix} \mathbf{Y}^T & \mathbf{U}^T \end{bmatrix} \Theta - (\mathbf{Y}')^T \|_F^2. \tag{4.7}$$

This formulation is equivalent to dynamic mode decomposition with control (DMDc) [140] assuming no rank truncation. Such DMDc models can be directly used for MPC, but they are limited by the inherent assumption of linearity. Highly nonlinear systems often require more complex methods for satisfactory MPC performance.

4.2.2 Neural Network Models

Fully connected neural networks (FCN) offer greater expressiveness in comparison to linear models, at the cost of interpretability. The nonlinearity of FCNs allows them to approximate far more complex dynamics than can be modeled with linear methods. FCNs were among the first nonlinear, data-driven models used for MPC and can be an effective architecture for

many systems [42,124,126]. They also provide an important intermediate level of complexity between DMDc and the RNN architectures we will subsequently consider.

FCNs act as a series of matrix multiplications composed with nonlinear activation functions. A two-layer fully connected network f_Θ can be expressed as

$$f_\Theta(\mathbf{x}_j^k, \mathbf{u}_j^k) = W_1 \left(R \left(W_0 \begin{bmatrix} \mathbf{x}_j^k \\ \mathbf{u}_j^k \end{bmatrix} + b_0 \right) \right) + b_1, \tag{4.8}$$

where $\Theta = \{W_0, W_1, b_0, b_1\}$ are trainable parameters and $\mathbf{x}_j^k$, $\mathbf{u}_j^k$ are the state and control vectors at time j augmented by up to k time-delays. R is a nonlinear activation function that operates elementwise. We seek Θ such that the mean square error (MSE) of a one-step forecast, over a set of $N+1$ training states and control inputs is minimized. More concretely, we seek to minimize

$$J(\Theta) = \frac{1}{N-k} \sum_{i=k}^{N} \| f_\Theta \left(\mathbf{x}_j^k, \mathbf{u}_j^k \right) - \mathbf{x}_{j+1} \|_2^2. \tag{4.9}$$

This minimization is typically performed with a variant of stochastic gradient descent (SGD). As with the linear case, the above methodology can be applied when only measurements $\mathbf{y}_j$ are available. MPC with FCNs dates back to at least 1995 [42], but has largely fallen out of favor with the advent of more advanced neural network architectures. Although the previously introduced moels are capable of incorporating time-delays, they do so by augmenting the input dimension of the system model; an n dimensional system modeled with k time-delays will require an input dimension of nk. This results in a model that can be highly computationally inefficient.

A more natural architecture for modeling a dynamical system is that of a recurrent neural network (RNN), specifically designed to allow for a sequence of inputs without increasing the model input dimension. RNN architectures evolve a latent state, $\mathbf{h}_j$, parameterized by a set of weights and biases Θ_r according to the expression

$$\mathbf{h}_{j+1} = f_{\Theta_r}(\mathbf{x}_j, \mathbf{h}_j). \tag{4.10}$$

In the context of time-series forecasting, an additional output operator f_{Θ_o} is learned that maps $\mathbf{h}_j$ to $\hat{\mathbf{x}}_j$. The system can then be expressed as

$$\mathbf{h}_{j+1} = f_{\Theta_r} \left(f_{\Theta_o}(\mathbf{h}_j), \mathbf{h}_j \right) \tag{4.11}$$

and is an autonomous dynamical system that can be used to generate a forecast of arbitrary length. This interpretation has allowed variants of RNNs to achieve remarkable performance characterizing chaotic dynamical systems, capturing their true Lyapunov exponents and long-term power spectra [138, 161].

In the control setting, the system of interest is nonautonomous, but the learned RNN can be simplified so that the only explicit time dependence arises from the control action, $\mathbf{u}_j$. The dynamical system of the RNN can then be represented by

$$\mathbf{h}_{j+1} = f_{\Theta_r}(\mathbf{x}_j, \mathbf{u}_j, \mathbf{h}_j) \tag{4.12a}$$

$$\hat{\mathbf{x}}_j = f_{\Theta_o}(\mathbf{h}_j). \tag{4.12b}$$

An important consequence of such RNN representations of dynamical systems is that the implicit incorporation of time-delays in the model allows generalization to partial observations $\mathbf{y}_j$ without augmenting the size of the network.

The first form of RNN, known as the Elman RNN, assumed f_{Θ_r} and f_{Θ_o} such that

$$\mathbf{h}_{j+1} = \sigma_R(W_x\mathbf{x}_j + W_u\mathbf{u}_j + W_h\mathbf{h}_j + b_h) \tag{4.13a}$$

$$\hat{\mathbf{x}}_j = W_{\hat{x}}\sigma_o(W_o\mathbf{h}_j + b_o) + b_{\hat{x}}, \tag{4.13b}$$

where $\Theta_r = \{W_x, W_u, W_h, b_h\}$ are the trainable recurrent parameters, $\Theta_o = \{W_{\hat{x}}, W_o, b_o, b_{\hat{x}}\}$ are the trainable output parameters, and σ_R, σ_o are the recurrent and output nonlinearities, respectively. Since its introduction, the Elman RNN has largely fallen out of favor due to vanishg/exploding gradient issues [68]. In this work, we consider two variants of RNN inspired by the Elman RNN that help ameliorate these problems: long-short term memory networks (LSTMs) [68] and gated recurrent units (GRUs) [34]. Both have been used as MPC surrogates, but extensive benchmark comparisons are lacking.. Each of these RNNs has the form given in (4.12a) and (4.12b), but differ in their explicit forms of f_{Θ_r} and f_{Θ_o}.

4.3 Methods

Another class of RNN is the echo state network (ESN) [74, 113], a type of reservoir computer. While LSTM networks rely on gating mechanisms to control the flow of information and alleviate the problems of vanishing/exploding gradients, ESNs avoid backpropagation

altogether. Instead, a random, high-dimensional "reservoir" is constructed that is stimulated by a comparably low-dimensional input. The reservoir evolves nonlinearly and it is assumed that the random encodings of this high-dimensional dynamical system can be linearly mapped to the desired low-dimensional output. A trained ESN can then be used as a surrogate model for MPC [8].

4.3.1 ESN Surrogate Model

Concretely, the recurrent mapping of the ESN is given by

$$\mathbf{h}_{j+1} = (1-\alpha)\mathbf{h}_j + \alpha \tanh\left(A\mathbf{h}_j + W_{ih}\bar{\mathbf{x}}_j + \sigma_b \mathbf{1}\right), \tag{4.14}$$

where α is the leak rate, $\mathbf{1}$ denotes a vector of ones, and σ_b scales the additive bias in the reservoir. The vector concatenation of $\mathbf{x}$ and $\mathbf{u}$ is denoted as $\bar{\mathbf{x}}$. W_{ih} and A are randomly initialized and left fixed; there is no direct "learning" of the dynamics of the reservoir. Nevertheless, the initialization of W_{ih} and A, and the selection of α and σ_b play a critical role in the performance of the network. The role of the reservoir matrix A has been especially explored in the literature about ESNs. ESNs rely on a type of generalized synchronization commonly referred to as the *echo state property* which mandates that given a sufficiently long driving sequence $\mathbf{x}$, the state of the reservoir is not dependent on $\mathbf{h}_0$ [111]. This desirable property can be achieved through careful initialization of A. In this work, A is selected to be a random, sparse matrix with dimensions $N_r \times N_r$, density ρ, and spectral radius ρ_{sr}. In all considered ESNs, $N_r = 1000$ and $\rho = 0.02$. The spectral radius ρ_{sr} is selected by a hyperparameter sweep, although other methods relying on Bayesian optimization have been used to select ρ_{sr} [139]. The entries of W_{ih} are drawn from the uniform distribution $\mathcal{U}(-\sigma, \sigma)$, where σ is another hyperparameter chosen through a hyperparameter sweep. The leak rate parameter α and additive bias σ_b are similarly chosen.

Despite a larger number of hyperparameters that must be selected, ESNs are typically orders of magnitude faster to train than their LSTM counterparts. This is because only the output operator, f_{Θ_0}, is trained and f_{Θ_0} is frequently selected to be linear. Consequently, the training of an ESN amounts to a computationally efficient ridge regression. The reservoir is teacher forced with a sequence of state measurements and control inputs,

$\{(\mathbf{x}_0, \mathbf{u}_0), \ldots, (\mathbf{x}_N, \mathbf{u}_N)\}$ to generate a corresponding sequence $\{\mathbf{h}(1), \ldots, \mathbf{h}(N+1)\}$ where $\mathbf{h}_0 = \mathbf{0}$. Here, the echo state property ensures that our selection of $\mathbf{h}_0$ will be "forgotten" after a sufficiently long sequence of teacher forcing. In general, the minimum length of this sequence, which we call N_{spin}, depends on the problem at hand, but in all the examples we consider, $N_{spin} = 200$ time-steps appears sufficient. This is substantially fewer than suggested in some existing literature, perhaps as a consequence of an external driving force as opposed to autonomous dynamics. Larger N_{spin} effectively limits the amount of data on which the output operator can be trained.

The output operator f_{Θ_o} takes the form

$$f_{\Theta_o}(\mathbf{h}_j) = W_o \mathbf{h}_j. \tag{4.15}$$

Subsequently, the best fit output operator W_o is determined so that

$$W_o \begin{bmatrix} \mathbf{h}_{N_{spin}} & \cdots & \mathbf{h}_{N+1} \end{bmatrix} \approx \begin{bmatrix} \mathbf{x}_{N_{spin}} & \cdots & \mathbf{x}_{N+1} \end{bmatrix}. \tag{4.16}$$

This is achieved by solving a ridge regression of the form $W_o = (H^T H + \beta I)^{-1} H^T X$. The Tikhonov regularization parameter β is another hyperparameter that is tuned using grid search. Some previous works have relaxed the assumption that the output operator be linear and take f_{Θ_o} to include quadratic nonlinearities in some entries of $\mathbf{h}_j$. In this form of an ESN, a ridge regression is still used for training. However, recent work has shown that this inclusion is unnecessary when σ_b is allowed to be non-zero, as is the case here [139].

This training protocol performs a minimization of the error on a one-step forecast. A multi-step forecast is achieved by iterating Eq. 4.14 according to

$$\mathbf{h}_{j+1} = f_{\Theta_r}(\hat{\mathbf{x}}_j, \mathbf{u}_j, \mathbf{h}_j). \tag{4.17}$$

One advantage of this approach is that performing a T-step forecast requires only T forward passes of the network, without any need to reinitialize the reservoir state $\mathbf{h}$. Thus, an ESN maintains the benefits of a stateful RNN without the additional training difficulties stateful models introduce in backpropagation based RNN training procedures [161].

4.3.2 Training Protocols

The training of surrogate models for each example system we will consider was performed in the same manner. For any example system, let $\mathbf{X} = \begin{bmatrix} \mathbf{x}_0 & \cdots & \mathbf{x}_N \end{bmatrix}$ denote the training data matrix. The data matrix was then corrupted column-wise with zero-mean Gaussian white noise with covariance such that $\Sigma_{ii}^2 = 0.001\sigma_{tr,i}^2$ and $\Sigma_{i,j} = 0$ for $i \neq j.$, where σ_{tr} is a vector containing the componentwise empirical standard deviation of the training data. The corruption of training data in this manner acts as a regularization and improves the downstream forecasting performance of surrogate models [161]. The data is then min-max scaled row-wise.

For each system, a validation dataset was also generated. The validation forcing signal is unique to each system and is described in Section 4.4. An extensive hyperparameter sweep was then conducted for each surrogate model class. For a given set of hyperparameters, we trained the model on the training data and evaluated its 50-step forecast performance using root mean squared error (RMSE) on the validation data. A 50-step forecast was selected, as longer horizons would make online optimization in MPC computationally prohibitive. For the ESN and DMDc models, one of the hyperparameters included in the sweep is β, the Tikhonov regularization parameter used in the ridge regression for training. Because this parameter alone can reduce overfitting, 100% of the training data was used in training these models.

On the other hand, the models trained via backpropagation and an Adam optimizer [88] do not have a hyperparameter that directly controls overfitting to the data. As a result, during training, 20% of the training data is withheld to serve as a first validation set. An early stopping criteria is introduced based on performance in this validation set to prevent overfitting.

From the many models trained in the hyperparameter search, we select the best performing set of hyperparameters for subsequent forecast and control performance demonstrations. However, we do not use the specific model found during the hyperparameter sweep. This is because the random initializations of neural networks can have a significant impact on subsequent performance, potentially causing the continued use of the model from the hy-

perparameter sweep to yield better results than can be expected in most cases. We note that although we consider significantly more hyperparameter combinations for the ESN compared to the LSTM, the total time required to perform the sweep is still much larger for the LSTM. This is a consequence and benefit of the rapid training enabled by reservoir computing approaches.

4.3.3 MPC Formulation

The total MPC cost function is a linear combination of $J(U,t)$ (4.1) and additional terms that represent a type of soft control barrier function (CBF) [4]. The cost is therefore given by

$$J_{tot}(U,t) = J(U,t) + \alpha_1 \| \max(U, b_1) - b_1 \|_2^2 + \alpha_1 \| \min(U, b_2) - b_2 \|_2^2, \tag{4.18}$$

where the constant matrices b_1 and b_2 are chosen with entries to be 0.95 and 0.05, respectively, for all datasets and the max/min functions operate elementwise [4]. The same constants can be used for all benchmark systems because the models perform forecasts in min-max scaled data. Similarly, $\alpha_1 = 100$ is chosen for all systems. Because the introduced CBF does not impose hard constraints, any entries of U found by the numerical solver are clamped to the range [0,1]. The introduction of the soft constraints encourages the solver to identify control actions between 0.05 and 0.95, while the hard constraints ensure that the maximum/minimum control values do not exceed those found in the training data.

The forecast horizon, T, and control horizon, M, are selected to be 50, while the number of control steps used from a single optimization, M_c, is taken to be 20. These selections provide a more stringent test of the learned surrogate models, as even very poor models can be effective with MPC when T, M, and M_c are small. Finally, the weighting matrices for $J(U,t)$ from Eq. (4.1) are selected as $Q_1 = 100\mathbb{I}$, $Q_2 = 1\mathbb{I}$, and $Q_3 = 20\mathbb{I}$. Q_1 determines the cost of deviation from the desired reference trajectory, while Q_2 and Q_3 penalize the magnitude of control actions and their rate of change.. Following existing work using RNNs for MPC, we use the limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm to perform the required online minimizations of (4.18) [14]. A pseudocode algorithm detailing our implementation of ESN-based MPC is included in Algorithm 1.

Algorithm 1 Pseudocode algorithm for Echo State Network Based Model Predictive Control. The algorithm consists of an offline ESN training phase followed by online control.

Offline Training:
Given:

- Randomly initialized ESN
- Pseudo-random control sequence $\{\mathbf{u}_j\}_{j=1}^{N_{\text{train}}}$
- Initial state $\mathbf{x}_0$
- Length of transient to discard N_{spin}

procedure ESN TRAINING($\mathbf{x}_0$, $\{\mathbf{u}_j\}_{j=1}^{N_{\text{train}}+1}$)
    Collect state trajectory $\{\mathbf{x}_j\}_{j=1}^{N_{\text{train}}}$
    Teacher-force the ESN to generate hidden states $\{\mathbf{h}_j\}_{j=1}^{N_{\text{train}}+1}$
    Solve for output operator $f_o(\mathbf{h}) = W_o\mathbf{h}$ by ridge regression:

$$W_o \begin{bmatrix} \mathbf{h}_{N_{\text{spin}}} & \cdots & \mathbf{h}_{N_{\text{train}}+1} \end{bmatrix} \approx \begin{bmatrix} \mathbf{x}_{N_{\text{spin}}} & \cdots & \mathbf{x}_{N_{\text{train}}+1} \end{bmatrix}$$

end procedure

Online Control:
Given:

- Trained ESN surrogate model
- Prediction horizon M, control update interval M_c
- Reference trajectory $\{\mathbf{x}_j^*\}_{j=1}^{N}$
- Initial state $\mathbf{x}_0$
- Cost function $J(\mathbf{U}, k)$

procedure ESN-MPC($\mathbf{x}_0$, $\{\mathbf{x}_j^*\}_{j=1}^{N}$)
    $k \leftarrow 0$ ▷ time index
    $m \leftarrow 0$ ▷ steps since last optimization
    while $k < N$ **do**
        if $m = M_c$ **then**
            Compute optimal control sequence using ESN surrogate:

$$\{\mathbf{u}_j^\star\}_{j=1}^{M_c} \approx \arg\min_{\mathbf{U}} J(\mathbf{U}, k)$$

            $m \leftarrow 0$
        end if
        Apply control action $\mathbf{u}_m^\star$
        Observe new state $\mathbf{x}_k$
        $k \leftarrow k + 1$
        $m \leftarrow m + 1$
    end while
end procedure

4.4 Forecasting Results

A ubiquitous issue in the training of surrogate models for control systems is that the structure of the control signal used for training does not mimic that of effective control. While one may find that a trained surrogate model performs well on data resembling that seen in training, the model may prove ineffective for different control signals. For this reason, we construct a validation dataset for each example system with control inputs that switch between constants and smoothly varying sinusoids. Such control signals are not seen in training, where the control resembles a series of rapidly varying step inputs. Although the validation control signal also does not mimic effective control, it provides a first test of a model's ability to function under previously unseen control inputs.

In a departure from existing work on surrogate models for autonomous systems, we emphasize the accuracy of short-term, 50-step forecasts, as this ability is most relevant to subsequent MPC. The selection of 50-step forecasts is somewhat arbitrary. In this work, it was selected because it represents an approximate upper bound on the forecast horizon for which MPC is computationally feasible with the discretizations of the majority of considered models.

For all of the following benchmark systems, the results figures follow the same format. Panel A shows the time-series of the validation data (top) and the corresponding control inputs (middle). The time-series of system states $\mathbf{x}$ has been scaled to improve readability across examples. The bottom of Panel A shows the ℓ_2 deviation of each surrogate model's forecast from the ground truth as a function of time. All forecasts are reinitialized every 50 time-steps, denoted by the dashed black line in the top plot. Panel B reports the mean ℓ_2 forecast error obtained across 32 trained surrogate models of each type for varying levels of additive Gaussian white noise. Panel C reports analogous results for the case of zero noise, but a varying number of training samples.

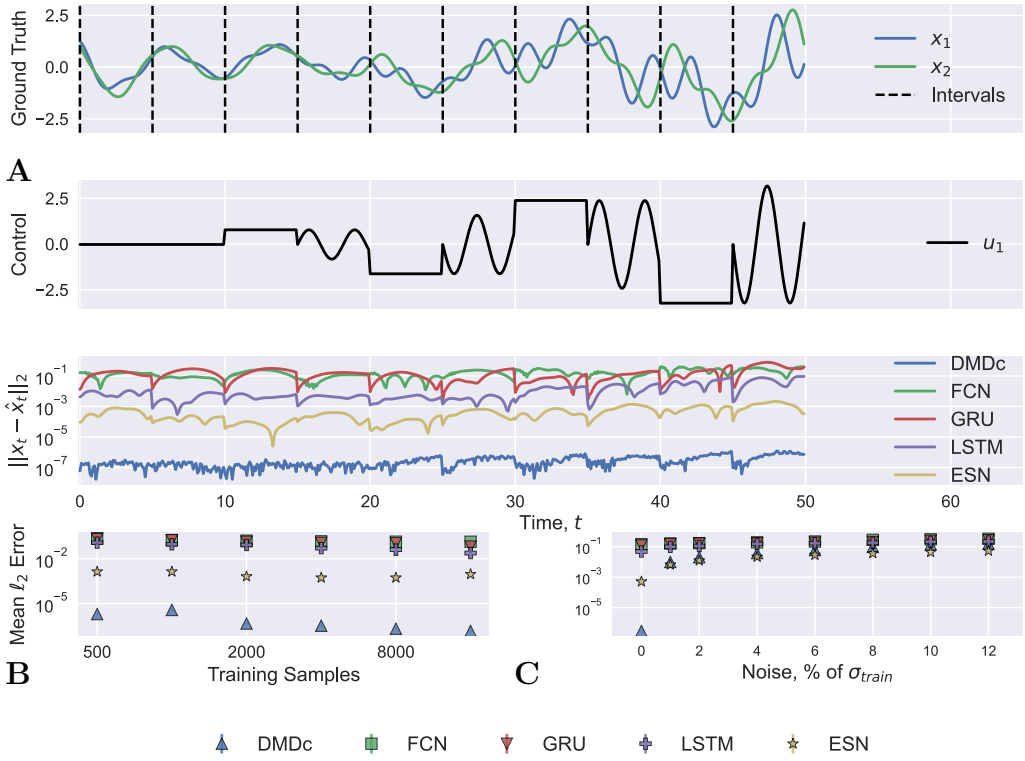


Figure 4.2: Forecasting results for the linear spring mass system. Panel A shows the time-series of the validation data (top) and the corresponding control inputs (middle). The time-series of $\mathbf{x}$ has been scaled to improve readability across examples. The bottom of Panel A shows the ℓ_2 deviation of each surrogate model's forecast from the ground truth as a function of time. All forecasts are reinitialized every 50 time-steps, denoted by the dashed black line in the top plot. Panel B reports the mean ℓ_2 forecast error obtained across 32 trained surrogate models of each type for varying levels of additive Gaussian white noise. Panel C reports analogous results for the case of zero noise, but a varying number of training samples. We find the highest accuracy with linear system identification (DMDc) under no noise (bottom, A; C), as expected for an inherently linear system. With noisy data, ESNs achieve comparable performance to linear system identification (B).

4.4.1 Linear spring mass system

The first example control system is inspired by a simple, coupled spring-mass system. The continuous single-input, multi-output (SIMO) system is represented by the following system

state-space representation:

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-2k}{m} & \frac{-b}{m} & \frac{k}{m} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k}{m} & 0 & \frac{-k}{m} & \frac{-b}{m} \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ \frac{1}{m} \\ 0 \\ 0 \end{bmatrix} \mathbf{u}, \tag{4.19a}$$

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \mathbf{x}. \tag{4.19b}$$

where k, m, and b are constants. The system variables x_1, x_3 denote the positions of two masses and x_2, x_4 their respective velocities. We assume access only to measurements of the masses' positions. The control discretization is taken to be $\Delta t = 0.1$s. Training data is generated by simulating the system under a pseudo-random control signal for 5000 seconds, corresponding to 50,000 temporal snapshots. The control signal was generated a priori by drawing samples from $\mathcal{U}(-3, 3)$ at intervals of 0.5 seconds and repeating each drawn value for 5 time-steps. To obtain a more physical control signal, the signal was then convolved with a uniform filter of width 2.

The validation control signal and the performance of a particular instance of each candidate model for the linear, spring mass system are shown in Panel A of Fig. 4.2. In the case of zero noise, as in Panel A and C, DMDc models achieve the best performance because the underlying system is linear and observable. Thus, the DMDc models inherently encode the best possible representation of the dynamics. ESNs provide the second best approximation of the dynamics with no noise, and outperform all competing methods in the presence of all considered non-zero noise levels (Fig. 4.2, Panel C). All candidate models achieve similar performance when the number of training samples is varied between 500 and 32,000 (Fig. 4.2, Panel B).

4.4.2 Continuous stirred tank reactor

The second example system we consider is also a SIMO system, but now includes nonlinearities. The control aim of the system is to maintain the temperature of a chemical reaction by modulating the jacket temperature of the reactor. We let x_1 denote the concentration of a chemical A_{chem} and x_2 be the reactor temperature. u denotes the jacket temperature.

The example system was proposed by AP Monitor [63] and we rely on the Python package neuromancer for simulation [44]. The control discretization is set as $\Delta t = 0.1$s and the system was simulated for 5,000s. The control signal used for the generation of training data was constructed by drawing samples from $\mathcal{U}(297, 303)$ to mimic the operating parameters of the cooling jacket. Samples were drawn every 0.5s and held constant until the next sample was drawn. Again, we apply a smoothing uniform filter of width 2.

Fig. 4.3 reports results for the continuous stirred tank reactor system. Because of the inherently nonlinear dynamics, DMDc models are no longer accurate and yield the worst performance. ESNs achieve the best forecasting performance across all considered noise

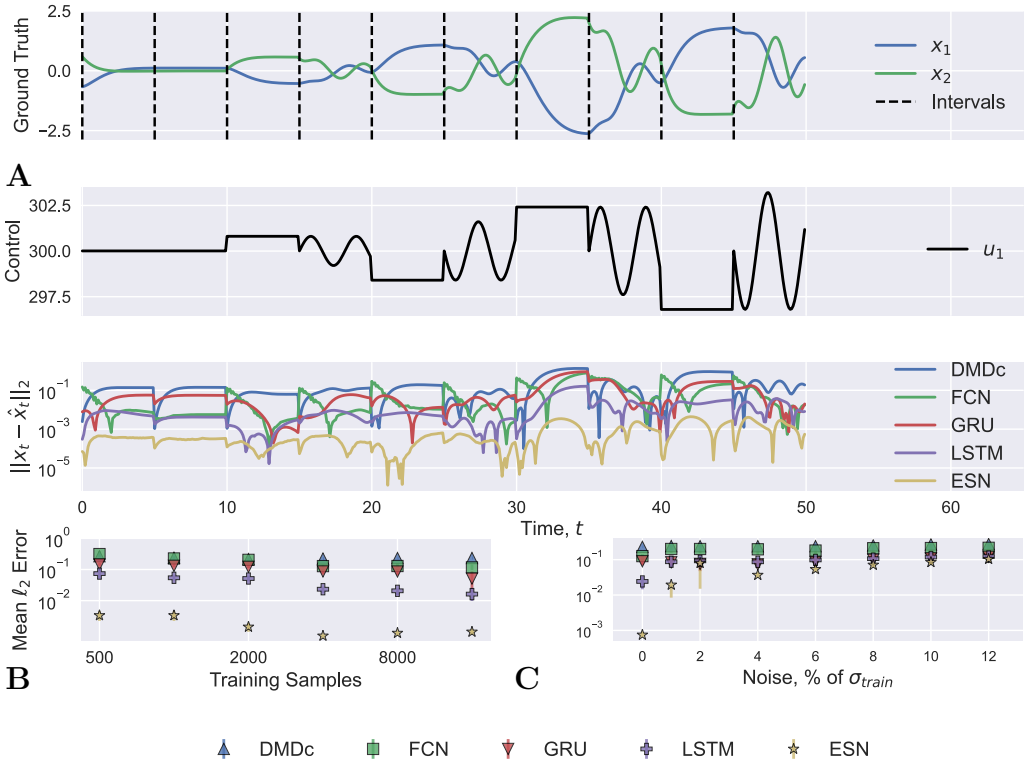


Figure 4.3: For a description of each panel, refer to Fig. 4.2. Forecasting results for the stirred tank system show that ESNs achieve the best performance, both without noise (A,C) and with noise (B). ESNs outperform competing architectures by an order of magnitude when the number of training samples is held constant (C).

levels (Fig. 4.3, Panel A, C) and number of training samples (Fig. 4.3, Panel B). As before, the gap in performance is largest in the case of zero noise and diminishes as noise is increased. LSTMs consistently provide the second best surrogate model. All models exhibit moderate reductions in mean ℓ_2 forecasting error as the number of training samples is increased.

4.4.3 Two-tank reservoir

We now turn to a nonlinear, multi-input, multi-output (MIMO) system consisting of coupled water tanks. Water can be pumped into, u_1, or from, u_2, the tanks. Tank 1 drains into

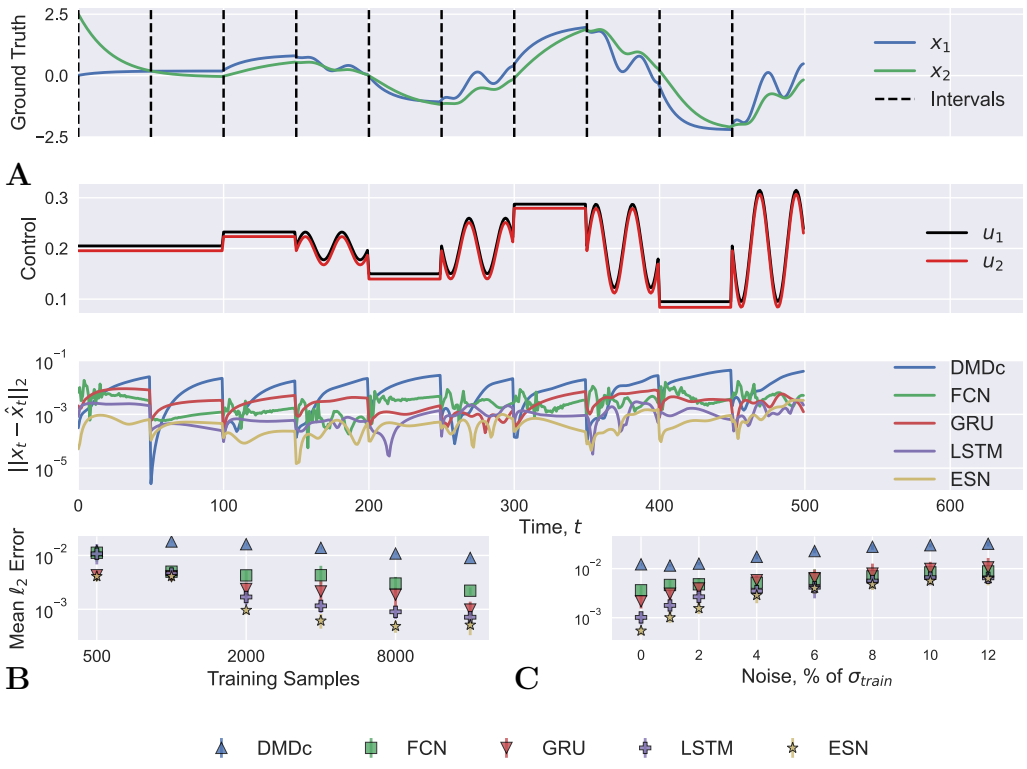


Figure 4.4: For a description of each panel, refer to Fig. 4.2. Forecasting results for the two-tank reservoir system show that that all architectures benefit greatly from more training data (C), but ESNs slightly outperform competing architectures both with noise (B) and without (A,C).

Tank 2, and Tank 2 drains into a large reservoir. The water levels in each tank are taken as the state variables x_1, x_2 and control aims to achieve set-points in each tank. As in the case of the stirred tank reactor, the system was proposed by AP Monitor [63] and we rely on the package neuromancer for simulation [44]. The dynamics occur on longer time-scales than in the previous examples, leading us to select $\Delta t = 1$s as the control discretization. In the generation of training data, both u_1 and u_2 were drawn independently from $\mathcal{U}(0, 0.4)$ at intervals of 50 seconds for 50,000 seconds. A smoothing uniform filter of width 50 was then applied entrywise.

Fig. 4.4 reports the forecasting performance for each considered model in the two-tank reservoir system. ESNs provide the best approximation of the dynamics across all noise-levels and number of training samples (Fig. 4.4, Panels A and B), although the disparity between ESNs and their LSTM counterparts is greatly reduced in this system. All models show larger error reductions with increasing training samples than observed in the previous two systems (Fig. 4.4, Panel C). This is likely a result of the MIMO nature of the two-tank system requiring more data to fully explore the state space of the system.

4.4.4 Lorenz 63 with control

Proposed in 1963 by Edward Lorenz, the Lorenz system is frequently used as an example of chaotic dynamics. Indeed, much existing work using RNNs for the forecasting of autonomous dynamical systems demonstrates algorithmic performance on the Lorenz system [138, 139, 161]. Here, we augment the classic Lorenz system with a control term $u(t)$. The system is given by

$$\dot{x}_1 = \sigma(x_2 - x_1) + u(t), \tag{4.20a}$$

$$\dot{x}_2 = x_1(\rho - x_3) - x_2, \tag{4.20b}$$

$$\dot{x}_3 = x_1 x_2 - \beta x_3, \tag{4.20c}$$

where $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. The control task will be to successively steer between the three fixed points of the autonomous Lorenz system, and the considered control discretization is $\Delta t = 0.01$s. Following the work of Kaiser et al. [78], we generate training

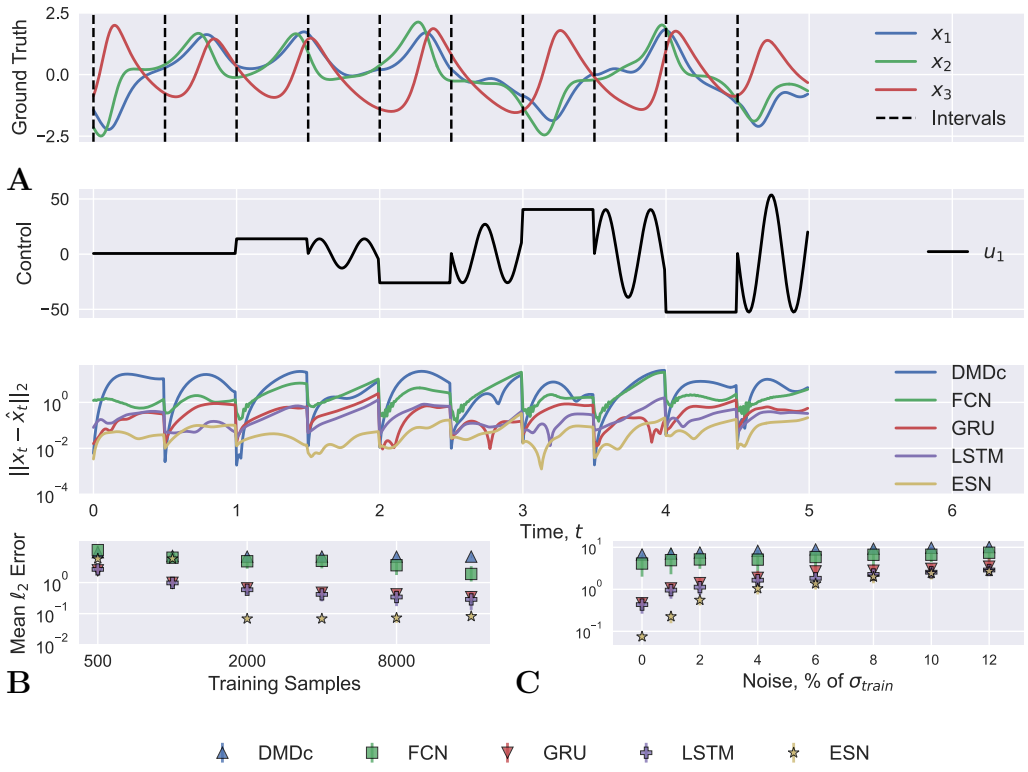


Figure 4.5: For a description of each panel, refer to Fig. 4.2. Forecasting results for the Lorenz system with control show that with zero or minimal noise, ESNs greatly outperform the competing architectures (A,B,C). As noise is increased, all forecasting methods fail (B), as to be expected in a chaotic system.

data by sampling u from $\mathcal{U}(-50, 50)$ every 0.05 seconds for 500 seconds and smoothing the control signal with a uniform filter of width 2.

Fig. 4.5 reports the forecasting performance and selected hyperparameters for each considered model in the Lorenz system with control. In contrast to the previous systems, the Lorenz system exhibits chaotic behavior, rendering forecasts significantly more challenging. This is exhibited by the higher mean ℓ_2 error for all models (Fig, 4.5, Panels A, B, and C). Consequently, all models fail to achieve accurate forecasts when only 500 training samples are available (Panel B). However, with 1,000 training samples, ESNs outperform all competing models with as many as 32,000 training samples. Beyond low-levels of noise, all

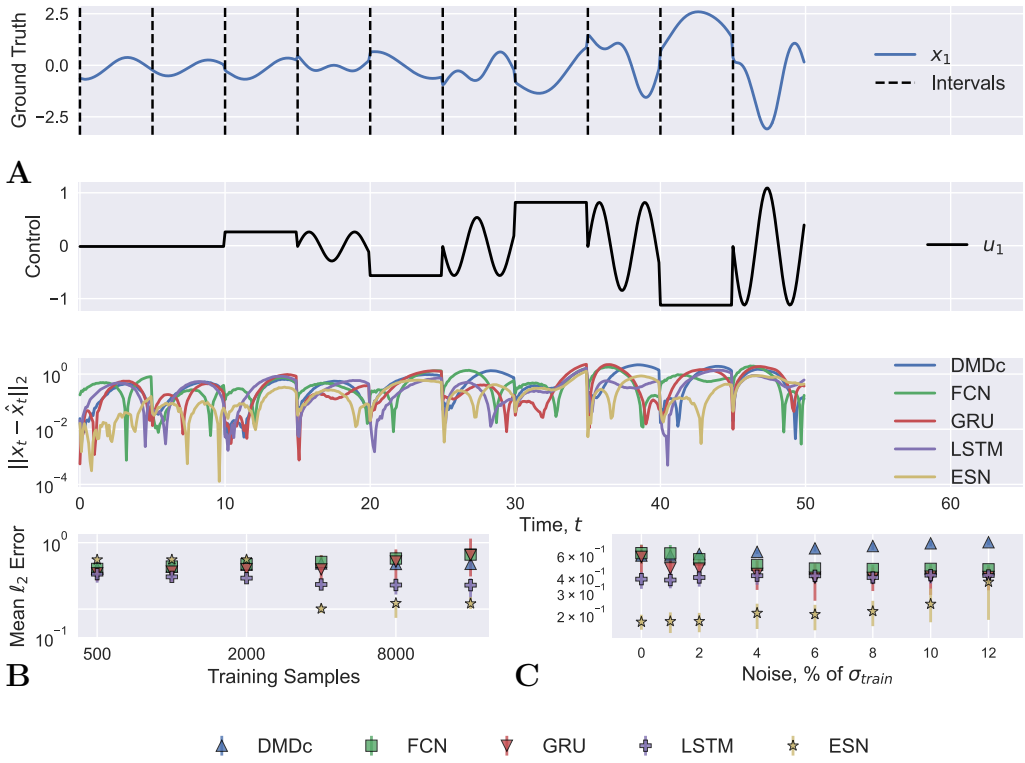


Figure 4.6: For a description of each panel, refer to Fig. 4.2. Forecasting results for the flow past a cylinder show that ESNs consistently achieve better performance across a range of noise levels (C). ESNs also require fewer training samples than competing architectures (B).

models fail to perform accurate forecasts, but ESNs still provide the best approximation (Panel C).

4.4.5 Flow past a cylinder

Our final benchmark task is controlling the lift generated by fluid flow around a cylinder. The Navier-Stokes equations governing this flow are given by

$$\frac{\partial u}{\partial t} + u \cdot (\nabla u) = -\nabla p + \frac{1}{\mathrm{Re}} \Delta u \tag{4.21a}$$

$$\nabla \cdot u = 0, \tag{4.21b}$$

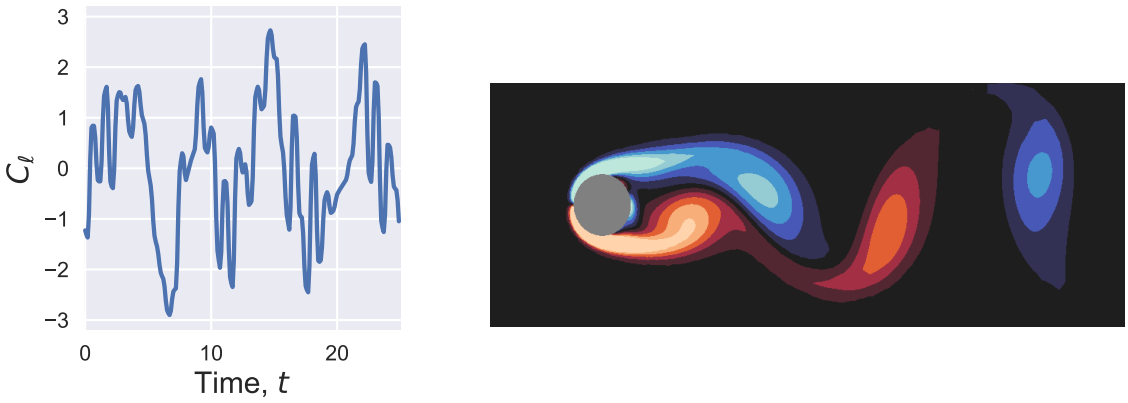


Figure 4.7: The left panel illustrates a 25 second sample of the generated training data for the considered flow past a cylinder, and the right panel shows the vorticity of the flow field for a single training snapshot.

where u, t, and p are nondimensional velocity, time, and pressure, respectively, and Re, the Reynolds number, is set to 100. A sample visualization of the flow is given in Fig. 4.7 (left).

Control is accomplished by rotating the cylinder, and we assume a no-slip condition at the interface of the cylinder and the fluid. The objective is to control the lift generated by the cylinder, and all surrogate models are constructed assuming only access to lift measurements. Training data is generated by simulating the system under a pseudo-random control sequence with values drawn from $\mathcal{U}(-\pi/2, \pi/2)$ for 5,000 seconds with a control discretization of $\Delta t = 0.1$s. Control values are selected every 0.5s and the resulting signal is smoothed with a uniform filter of width 2. Fig. 4.7 (right) shows the evolution of the coefficient of lift, C_ℓ, for a 25 second period of the training data.

Fig. 4.6 displays the forecasting results for each considered class of surrogate model. Once again, ESNs consistently outperform all competing methods, albeit by a smaller margin in this case, as all methods yield higher mean ℓ_2 errors than in previous examples. With limited training samples (Fig. 4.6, Panel B) all models fail to provide accurate forecasts of the dynamics. However, with 4,000 or more training samples, ESNs achieve notably supe-

rior performance to all competing methods. In particular, with low to moderate noise (Fig. 4.6, Panel C), ESNs achieve roughly half the mean ℓ_2 error of the next best architecture, LSTMs.

4.4.6 Ablations

The performance of ESNs can be critically dependent on the selected hyperparameters [111]. In Section 4.3.3, we introduced five hyperparameters: (1) the leak rate α, (2) the Tikhonov regularization β, (3) the input scaling σ, (4) the additive bias σ_b, and (5) ρ_{sr}, the spectral radius of the reservoir matrix A from Eq. 4.14. We subsequently investigated the sensitivity to each hyperparameter by ablating across a range of values, while keeping all other hyperparameters fixed. Tables 1 and 2 show the results of these ablations for the Lorenz system and flow past a cylinder, respectively. In each case, the tables report the 25th and 75th percentile of the mean ℓ_2 error across 10 forecasts of 50 timesteps. The forcing signals for these validation forecasts are selected to be identical to those in Section 4.4.4 and 4.4.5.

In general, we observe high sensitivity in performance to all parameters, but especially σ and ρ_{sr}. While the spectral radius ρ_{sr} has long been known to be an important quantity in determining ESN performance [111, 139], the high sensitivity to input scaling that we observe is less well studied. We conjecture that in the context of control systems, the input scaling takes on heightened importance because the requirement of a control input makes the system inherently non-autonomous. Finally, we remark that the choice of $\sigma_b = 1.33$ is exceptionally poor for the flow past a cylinder, as no trained model produced a stable forecast. This result highlights the importance of careful hyperparameter selection for ESNs.

Hyperparameter	**Values**				
α	**0.2**	0.4	0.6	0.8	—
Result	0.217/0.262	0.235/0.307	0.321/0.409	0.501/0.595	—
β	10^{-4}	10^{-5}	10^{-6}	$\mathbf{10^{-7}}$	—
Result	0.481/0.619	0.249/0.324	0.197/0.247	0.214/0.283	—
σ	0.005	0.01	**0.1**	0.25	0.5
Result	6.960/7.240	3.343/3.716	0.225/0.295	0.205/0.298	0.278/0.402
σ_b	0.33	0.66	1	**1.33**	1.66
Result	0.208/0.280	0.214/0.310	0.213/0.314	0.178/0.244	0.172/0.236
ρ_{sr}	0.2	**0.4**	0.6	0.8	—
Result	0.203/0.265	0.207/0.261	0.221/0.292	0.328/0.412	—

Table 4.1: Forecasting performance across hyperparameters for the Lorenz system. Each block corresponds to varying one hyperparameter while holding others fixed. Results show 25th/75th percentiles of mean ℓ_2 error across 16 trained ESNs, all of which performed 10 forecasts of 50 timesteps. The bolded value in each row denotes the nominal hyperparameter value identified in the hyperparameter sweep and used when performing ablations of different variables.

Hyperparameter	**Values**				
α	**0.2**	0.4	0.6	0.8	—
Result	0.161/0.200	0.229/0.258	0.269/0.332	0.260/0.612	—
β	10^{-4}	10^{-5}	10^{-6}	$\mathbf{10^{-7}}$	—
Result	0.209/0.226	0.237/0.265	0.246/0.264	0.159/0.182	—
σ	0.005	0.01	0.1	0.25	**0.5**
Result	0.439/0.444	0.380/0.385	0.320/0.324	0.257/0.289	0.154/0.191
σ_b	**0.33**	0.66	1	1.33	1.66
Result	0.162/0.197	0.325/0.368	0.308/0.420	6.862/143.022	0.396/0.909
ρ_{sr}	0.2	**0.4**	0.6	0.8	—
Result	0.316/0.337	0.160/0.191	0.274/0.546	0.302/0.436	—

Table 4.2: Forecasting performance across hyperparameters for the flow past a cylinder. Each block corresponds to varying one hyperparameter while holding others fixed. Results show 25th/75th percentiles of mean ℓ_2 error across 16 trained ESNs, all of which performed 10 forecasts of 50 timesteps. The bolded value in each row denotes the nominal hyperparameter value identified in the hyperparameter sweep and used when performing ablations of different variables.

4.5 ESN Control Results

To clarify the presentation and because the results of the previous section consistently show that ESNs and LSTMs yield the best forecasts, we present only the closed loop performance of LSTM and ESN based MPC in this section, and change the color scheme to more clearly distinguish ESN and LSTM performance.

4.5.1 Linear, spring-mass system

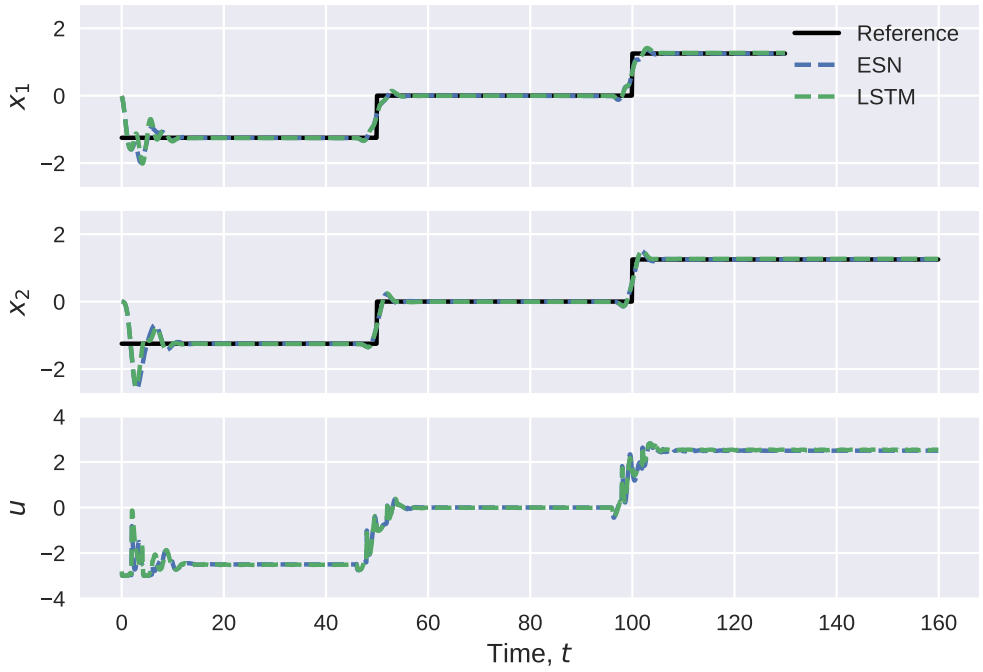


Figure 4.8: The control objective for the spring mass system is to steer the displacement of both masses to -1.5, 0, and 1.5. Both ESN and LSTM based MPC achieve excellent reference tracking in the spring mass system and compute similar control actions, as to be expected in a simple linear system.

For the spring mass system, each MPC is tasked with successively steering the system to $(x_1, x_2) = (-1.5, -1.5)$ for 50 seconds, $(x_1, x_2) = (0, 0)$ for 50 seconds, and $(x_1, x_2) =$

$(1.5, 1.5)$ for 50 seconds. Results for a particular instance of LSTM and ESN based MPC is shown in Fig. 4.11. As seen in the figure, both controllers result in nearly identical control actions that successfully steer the system to the desired reference trajectory. The controllers also appear to correctly identify the steady state inputs required to maintain the system at each set point. To provide a more sensitive metric for comparing ESN and LSTM based MPC, we compute the statistics of $J(U,t)$ over 32 trials of each class of controller. The mean and standard deviation for the LSTM and ESN approaches are nearly equivalent $\mu_{LSTM} = 159, \sigma_{LSTM} = 4$, and $\mu_{ESN} = 159, \sigma_{ESN} = 1$. The excellent performance of MPC with both surrogate models is to be expected, as the underlying dynamics of the system are linear.

4.5.2 Continuous stirred tank reactor

In contrast to the previous example where all set points were achievable, here the selected reference trajectory contains only one set point that is feasible, the equilibrium concentration $(x_1^*, x_2^*) = (0.877, 324.48)$. However, the specified control task is to steer the system to $(x_1, x_2) = (x_1^*, x_2^*), (0.99x_1^*, 0.994x_2^*)$ and $(1.01x_1^*, 1.0066x_2^*)$. Both LSTM and ESN based MPC rapidly steer the system to this first set point and correctly identify the required constant control input to maintain the equilibrium. As shown in Fig. 4.9, both MPC algorithms yield similar control signals that strike a balance between achieving the set point in x_1 and x_2. Both approaches achieve near identical performance as measured by $J(U,t)$, with the mean and standard deviation over 32 ESN models as $\mu_{ESN} = 326$ and $\sigma_{ESN} = 0.2$ and $\mu_{LSTM} = 326$ and $\sigma_{LSTM} = 0.3$ for the 32 LSTM models.

4.5.3 Two-tank reservoir

In the case of the two-tank reservoir system, MPC must solve for two actuations at each time-step as opposed to only one. Fig. 4.10 illustrates the control actions and state evolution of a particular instance of ESN and LSTM based MPC. The control task is to steer the system to $(x_1, x_2) = (0.25, 0.25), (0.5, 0.5)$ and $(0.1, 0.1)$. In both cases, MPC achieves excellent performance following the desired reference trajectory. However, ESN based MPC

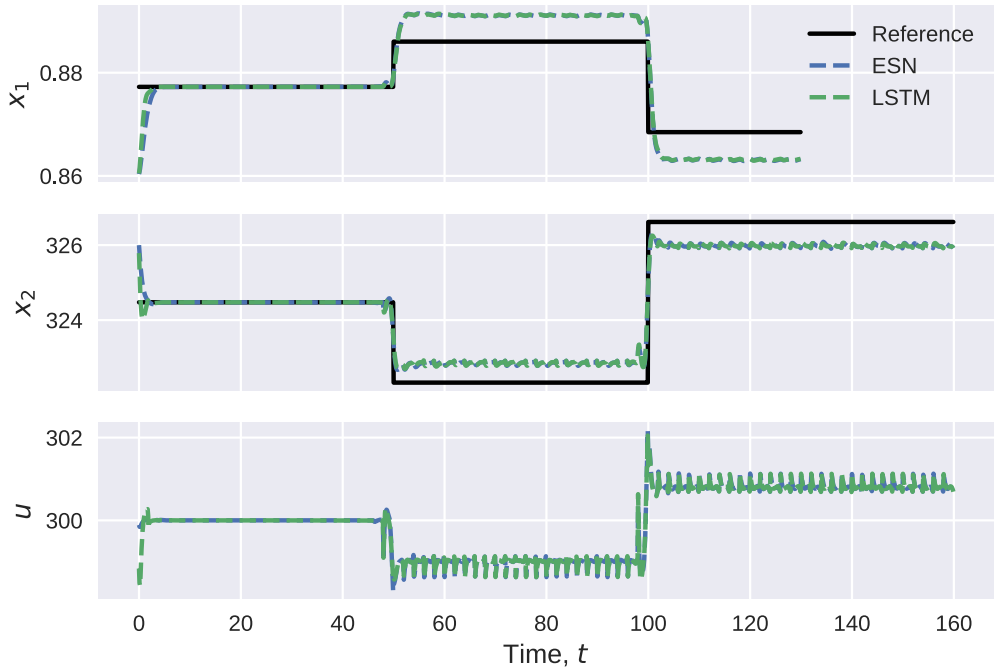


Figure 4.9: For the stirred tank system, the control task is to steer the system to $(x_1, x_2) = (x_1^*, x_2^*), (0.99x_1^*, 0.994x_2^*)$ and $(1.01x_1^*, 1.0066x_2^*)$, where $(x_1^*, x_2^*) = (0.877, 324.48)$ is the equilibrium concentration. The control performance of ESN and LSTM based MPC shows that both methods compute similar control actions and yield similar set points, even though the reference trajectory has been chosen to be infeasible.

appears to be more aggressive in the selected control actions (Fig. 4.10, bottom plot) when the set point switches between the three selected values. The mean and standard deviation of $J(U, t)$ over 32 instances of ESN based control are $\mu_{ESN} = 363$ and $\sigma_{ESN} = 5$, while for the LSTM approach $\mu_{LSTM} = 399$ and $\sigma_{LSTM} = 35$. ESN based MPC consistently outperforms LSTM based MPC and does so with significantly less variance in the cost, J.

4.5.4 Lorenz Control

The control task in the Lorenz system is to successively steer between the three fixed points of the system. These fixed points can be shown to exist at $(0, 0, 0)$, $(\sqrt{72}, \sqrt{72}, 27)$, and

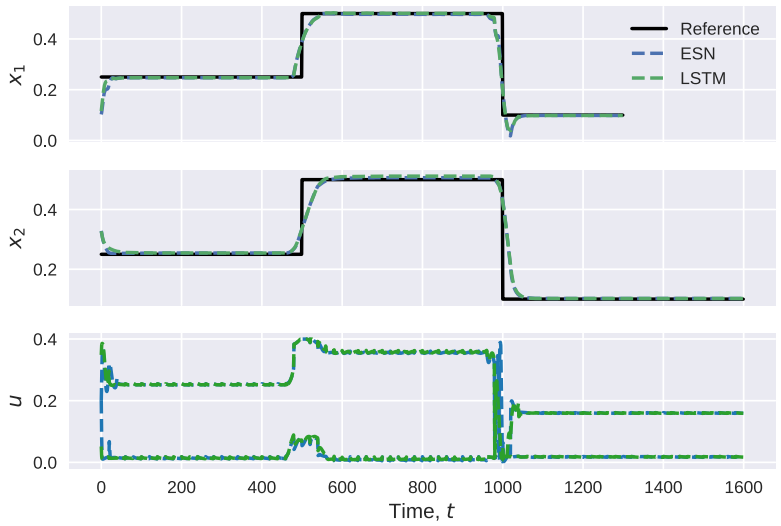


Figure 4.10: For the two-tank reservoir system, the control task is to steer the system to $(x_1, x_2) = (0.25, 0.25), (0.5, 0.5)$ and $(0.1, 0.1)$. ESN based MPC computes slightly more aggressive control actions and achieves slightly better set-point tracking than LSTM based MPC. Transients between set points are similar for both methods.

$(-\sqrt{72}, -\sqrt{72}, 27)$ for the selected values of σ, ρ, and β. Previous work has demonstrated that data-driven MPC methods can successfully steer the system to the latter two of these fixed points [78], however stabilizing the system at the origin is a significantly more challenging task. This difficulty arises for at least two reasons. First, the maximal real part of an eigenvalue associated with the Jacobian evaluated at the origin is larger than the maximal real part of eigenvalues of the Jacobian evaluated at the other two fixed points. Second, setting aside any initial transient, the training data never explores the phase space of the system near the origin. Consequently, we find that while all considered LSTM and ESN based controllers stabilize the system at $(\pm\sqrt{72}, \pm\sqrt{72}, 27)$, not all LSTM based MPC schemes successfully control the system to the origin.

The mean and standard deviation of the MPC cost $J(U, t)$ across 32 trials of ESN and LSTM based MPC were $\mu_{ESN} = 1042$, $\sigma_{ESN} = 11$ and $\mu_{LSTM} = 1050$, $\sigma_{LSTM} = 149$. Fig. 4.11 illustrates the best and worst performing instances of LSTM-MPC, as well as

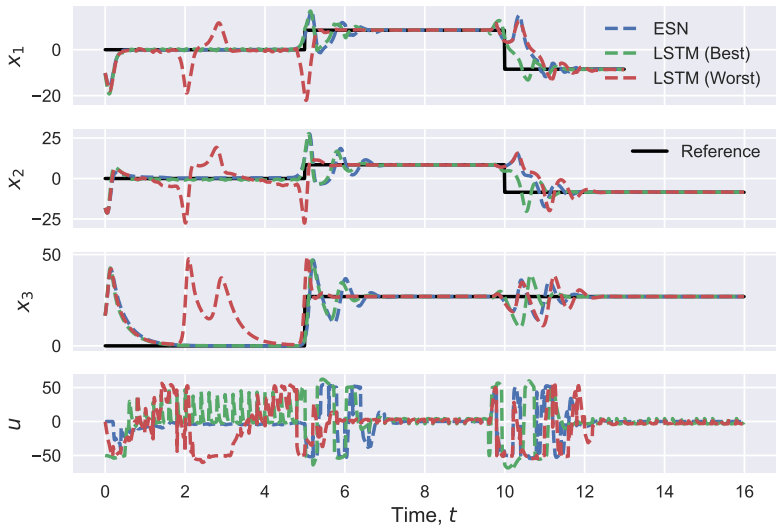


Figure 4.11: The control task in the Lorenz system is to successively steer between the three fixed points of the system, at $(0, 0, 0)$, $(\sqrt{72}, \sqrt{72}, 27)$, and $(-\sqrt{72}, -\sqrt{72}, 27)$. Both the best and worst performing instances of 32 trials of LSTM based MPC are shown, as well as the worst performing instance of ESN based MPC. Even in the worst performing instance, ESN based MPC is able to steer the system to the fixed point at the origin and maintain the set-point. Meanwhile the best performing instance of LSTM based MPC achieves this setpoint and maintains it through aggressive control actions. All shown controllers successfully steer the system to the fixed points at the center of the two butterfly lobes of the Lorenz system.

the worst performing instance of ESN-MPC. Not only does the worst case of LSTM based MPC fail to stabilize at the origin, even the best case of LSTM-MPC requires aggressive control actions to maintain the state at the origin (Fig. 4.11, bottom plot). The worst case of ESN based MPC, on the other hand, still successfully steers the system to the origin and maintains the unstable set point with minimal control input. All models are able to achieve and maintain the other two set points.

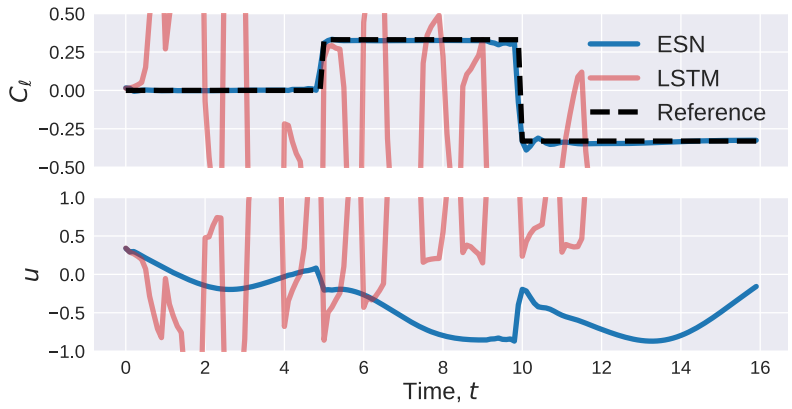


Figure 4.12: The control objective for the flow past a cylinder is to control the coefficient of lift C_ℓ to 0, 0.33, and -0.33. ESN based MPC with limited training data successfully tracks the desired coefficient of lift for the flow past a cylinder, while LSTM based MPC fails entirely.

4.5.5 Flow past a cylinder

The control objective for the flow past a cylinder is to control the coefficient of lift C_ℓ to 0, 0.33, and -0.33. Results for an ESN and an LSTM based MPC is shown in Fig. 4.12. As demonstrated in the figure, the LSTM based MPC entirely fails to control the system, while ESN based MPC provides excellent reference tracking. The work of Bieker et al. [14] demonstrated that RNNs trained with gradient descent can be used to control the same system, but the amount of training data used in that work was much greater. This finding is consistent with our observations in Section IV, where we found that the ESNs required less training data to achieve satisfactory forecasting performance.

4.6 Discussion and Conclusion

In this work, we empirically demonstrated that ESNs outperform other considered surrogate models (DMDc, FCNs, LSTMs, and GRUs) across a diverse set of example systems, including a linear spring–mass system, stirred tank reaction, coupled reservoir/tank system, Lorenz system with control, and flow past a cylinder. Across both forecasting and

control tasks, ESNs consistently achieved superior performance. We attribute this to two key advantages. First, ESNs avoid the difficulties of backpropagation through time. Their recurrent weights are fixed after random initialization, and only the readout layer is trained via linear regression. This enables rapid, numerically stable training and efficient hyperparameter exploration. Second, linear regression often requires fewer training samples than gradient-based methods, leading to greater data efficiency. For instance, in the Lorenz control problem, LSTMs required an order of magnitude more training data to achieve comparable forecasting accuracy. Finally, ESN-based MPC was computationally lightweight, with average control input computation times of under 0.1s on a standard laptop. This makes the method feasible for real-time control with temporal discretizations around 0.1 s, as in all benchmark systems except the Lorenz system.

These advantages were especially pronounced in two of our most challenging benchmarks. ESN-based MPC successfully controlled the lift generated by flow past a cylinder, assuming access to only a fraction of the training data used in prior work [14]. No other tested architecture achieved stable control. Similarly, ESN-based MPC reliably stabilized the chaotic Lorenz system at a fixed point in a region of phase space not well-explored during training, providing strong evidence that ESNs generalize better to unseen control and measurement inputs. Alongside their superior accuracy and data efficiency, these results demonstrate that ESNs can meet the computational demands of real-time implementation, at least for the moderate time resolutions considered here. Together, these findings suggest that ESNs provide a uniquely effective tradeoff between computational efficiency, data efficiency, extrapolation ability, and real-time applicability in model predictive control.

That being said, our study has several limitations. First, while ESNs outperform the selected baseline architectures, the comparison is not exhaustive. Many recently proposed architectures, such as structured state space sequence (S4) models and other state-space-inspired deep networks, may offer competitive performance. Second, although our implementation achieved sub-0.1 s computation times, this restricts applicability to problems with control discretizations no finer than 0.1 s. Real-time performance for systems with faster dynamics, or for high-dimensional systems with large control inputs, remains to be demonstrated. Finally, our training procedure relied on offline data and did not incorporate

noise or model mismatch beyond what was present in the simulation environments. Extending these methods to real-world experimental data will likely require additional robustness considerations.

These limitations suggest several promising directions for future research. One avenue is to incorporate online adaptation into ESNs using recursive least squares, which could enable real-time learning and improve robustness to nonstationary dynamics. Another direction is to investigate hybrid architectures that blend ESN-style reservoir dynamics with recent structured state space models. Finally, evaluating ESN-based MPC on large-scale, high-dimensional, or experimental control systems will be critical to assessing its practical impact and guiding further algorithmic development.

Chapter 5

ORC: GPU-ACCELERATED RESERVOIR COMPUTING IN JAX

5.1 Introduction

The preceding chapter argued that echo state networks (ESNs) provide a particularly attractive class of surrogate model for data-driven approximations of control systems. To realize these benefits, however, performant and differentiable implementations of ESNs are necessary. In this chapter, we develop OpenReservoirComputing (ORC), a Python package that provides several key benefits over existing reservoir computing (RC) libraries for both these control applications and reservoir computing more broadly. ORC is written in JAX [19] and Equinox [87]. JAX is a Python library for high-performance numerical computing that enables automatic differentiation, just-in-time (JIT) compilation, and GPU/TPU acceleration, while Equinox is a neural network framework for JAX. Recall that RC is a form of machine learning that functions by lifting a low-dimensional sequence or signal into a high-dimensional dynamical system and training a simple, linear readout layer from the high-dimensional dynamics back to a lower-dimensional quantity of interest. The most common application of RC is time-series forecasting, where the goal is to predict a signal's future evolution. RC has achieved state-of-the-art performance on this task, particularly when applied to chaotic dynamical systems. In addition, RC approaches can be adapted to perform classification and control tasks. ORC provides both modular components for building custom RC models and built-in models for forecasting, classification, and control. By building on JAX and Equinox, ORC offers GPU acceleration, JIT compilation, and automatic vectorization. These capabilities make prototyping new models faster and enable larger and more powerful reservoir architectures. End-to-end differentiability also enables seamless integration with other deep learning models built with Equinox.

5.2 Statement of Need

Time-series prediction, classification, and control are fundamental tasks across science and engineering, arising in applications from climate modeling and fluid dynamics to robotics and neuroscience. Deep learning approaches to these tasks typically require large datasets, long training times, and expensive tuning of optimization hyperparameters. RC offers a compelling alternative. Since only the readout layer is trained via a single ridge regression, RC models can be trained in a fraction of the time required by comparable recurrent neural networks, often with less data and fewer hyperparameters to tune [110]. This makes RC particularly attractive for rapid prototyping, real-time applications, and settings where training data is limited. However, realizing these benefits in practice requires software that is both efficient and adaptable.

ORC's built-in models provide an easy entry point for users new to the field. In particular, a new user can supply their own time-series data to instantiate, train, and forecast in three simple lines of code. Built-in visualization tools make it easy to evaluate model performance. Varying the hyperparameters of built-in models lets users explore how RC performance depends on configuration choices. While other RC libraries allow for easy use with low-dimensional systems, ORC's JAX foundation makes extending to higher-dimensional systems via parallel reservoirs equally simple (also achievable in three lines of code). For spatiotemporal forecasting problems, JAX's `vmap` transformation enables efficient vectorization across parallel reservoirs that decompose a spatial domain into overlapping subdomains, while JIT compilation eliminates Python overhead in the reservoir state evolution loop. Since RC approaches are so fast to train, this provides an easy way for users to train lightweight surrogate models.

Much RC research is aimed at designing performant reservoir architectures. ORC makes this easy through its use of abstract base classes for Embedding, Driver, and Readout layers. Users need only define forward pass logic to integrate a new reservoir topology or readout strategy, while reusing the rest of the framework. This modular approach also allows for easy ablation studies comparing how different components affect RC performance. It also makes it straightforward to reuse existing architectures, without the additional complexity

of writing teacher forcing or autoregressive prediction functions from scratch.

Because of ORC's functional approach in JAX, built-in and user-created models provide end-to-end differentiability by default. This enables gradient-based optimization of input sequences for control problems. This also makes ORC well suited to integrate with deep learning models such as those presented in [128, 129]. ORC models are simply Equinox modules, allowing them to be composed with other Equinox models such as standard neural networks, neural operators [87], etc.

5.3 State of the Field

	Lang.	GPU	Auto. Diff.	Parallel./Vect.	Forecast.	Classif.	Control	Cont. Time
ORC	Python	✓	✓	✓/✓	✓	✓	✓	✓
ReservoirPy	Python	✓*	×	×/×	✓	✓	×	×
ReservoirComputing.jl	Julia	✓	✓	✓/×	✓	✓	×	×

Table 5.1: Comparison of reservoir computing libraries across key features. ✓ indicates full support; × indicates no support. *ReservoirPy's GPU support is available via its JAX backend (v0.4.0+) but does not fully exploit JAX's functional programming model. *Parallelizable* denotes native support for parallel RC architectures as in [136] and *vectorizable* denotes native support for vectorization (e.g. `vmap`).

The most commonly used open-source library for reservoir computing is ReservoirPy [155]. Much like ORC, ReservoirPy provides a variety of built-in architectures, as well as an easy-to-use API for designing one's own layers. ReservoirPy was initially built on NumPy and SciPy with the maintainers adding a JAX backend in v0.4.0. However, ORC differs from ReservoirPy in several important ways.

First, ORC was *designed* on top of JAX [19] and Equinox [87], which provide a functional programming model for numerical computing. JAX's composable transformations include `vmap` (automatic vectorization, which maps a function over batched inputs without explicit loop code), `grad` (automatic differentiation for computing gradients), and `jit` (just-in-time

compilation to hardware-accelerated code). These cannot be retrofitted into a NumPy-based architecture; NumPy's imperative, stateful design relies on mutable objects and side effects that are incompatible with JAX's requirement for pure functions. While the JAX backend of ReservoirPy does improve performance, the API cannot fully exploit JAX's capabilities for this reason. As an example, ORC's autoregressive forecast loop uses `jax.lax.scan` (a JIT-compatible loop primitive with explicit carry state that avoids the Python-level overhead of a `for` loop), whereas ReservoirPy's object-oriented design makes this impossible. Moreover, the compatibility of ORC models with `vmap` makes the implementation of ensembles efficient and easy.

Second, ORC has a different built-in feature set. ORC supports continuous-time reservoir dynamics via Diffrax [86], a JAX-based library for solving differential equations with adaptive-step integrators, allowing users to define reservoir equations as ordinary differential equations. ORC also supports novel architectures such as Taylor-expanded and GRU-based drivers alongside standard echo state networks. ReservoirPy does not provide these functionalities.

Third, ORC models work seamlessly with other deep learning models implemented in Equinox. ReservoirPy is an outstanding library for standalone RC tasks that do not need to integrate with other deep learning frameworks. Working with NumPy rather than JAX may also be more accessible for users unfamiliar with functional programming. However, ORC's design priorities and advantages are nonetheless distinct from ReservoirPy's, as outlined above. ORC's syntax also is more consistent with modern deep learning libraries.

Other open source libraries for RC include Pytorch-ESN [123] and ReservoirComputing.jl [117]. Pytorch-ESN allows for integration with other PyTorch models, but is not as widely adopted and offers a more limited feature set than ORC or ReservoirPy. ReservoirComputing.jl is a widely used Julia library for RC that influenced many of ORC's design choices. In particular, ORC's modular design draws heavily from ReservoirComputing.jl. Since Python dominates much of machine learning research, bringing these capabilities to Python is valuable for the broader ML community. A summary of the functionality of ORC, ReservoirPy, and ReservoirComputing.jl is presented in Table 1.

5.4 Software Design

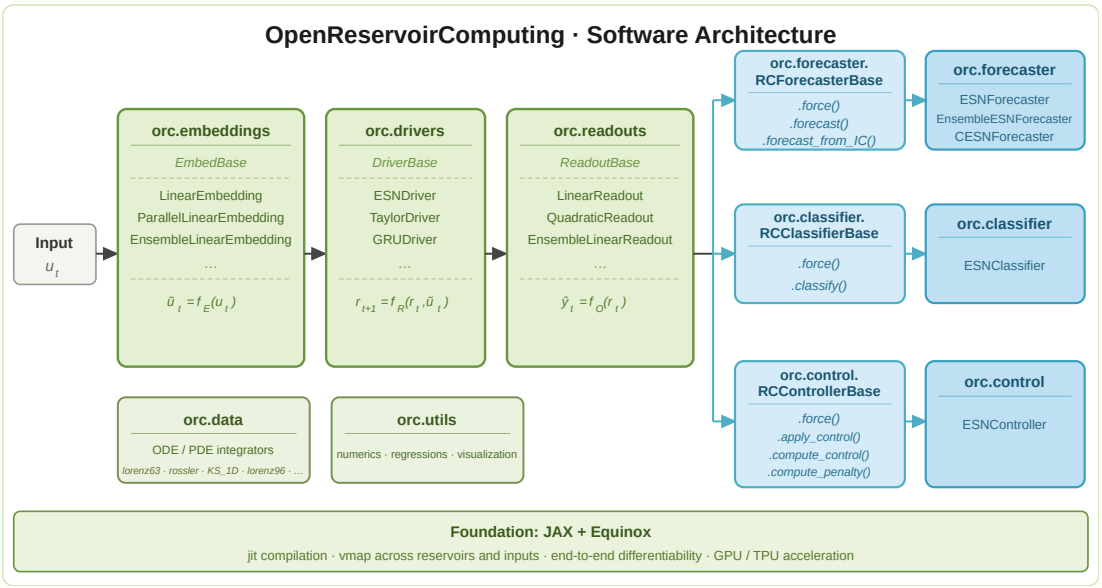


Figure 5.1: ORC three-layer pipeline architecture. Each reservoir computer (RC) is decomposed into (i) an "embedding" function that lifts a low-dimensional signal u_t to the reservoir dimension, (ii) a "driver" function that propagates the reservoir state, and (iii) a "readout" that maps back to a target y. For control and forecasting RCs, the target y is typically u_{t+1} (either in the presence of a forcing term or not) and for classification the target y is a label.

ORC models are decomposed into three components, illustrated in 5.1: (i) an embedding f_E that lifts a low-dimensional input signal u_t to a high-dimensional space, (ii) a driver f_R that propagates the high-dimensional state r_t, and (iii) a readout f_O that maps the latent state back to an approximation of some low-dimensional signal y_t. Depending on the task at hand, y_t may be a future time-step of u_t, a label associated with input data, or some other target signal. ORC differs from many existing approaches that unify (i) and (ii). Separating the embedding from the reservoir state propagation allows for cleaner application of RC to non-standard tasks, such as acting as a surrogate model for model predictive control. It also allows for the easier incorporation of non-standard embeddings, including

stochastic embeddings that can arise in the study of physical RC systems. Moreover, the modular design lets components developed for one task (e.g., forecasting) be reused directly in another (e.g., classification).

All components are implemented as Equinox modules [87], which are immutable pytree-registered objects. Model parameters (reservoir weights, readout matrices) are stored as JAX arrays within the module, and parameter updates produce new module instances via `eqx.tree_at` rather than mutating the state in place. This functional design enables JAX's composable transformations to operate directly on model objects and allows ORC models to be composed with other Equinox modules.

ORC supports parallel reservoirs [136] by default via a `chunks` parameter in each module. This slightly complicates built-in training support and introduces an extra tensor dimension that can make the API less intuitive at first. However, we believe this tradeoff is worthwhile because it enables spatiotemporal RC methods unavailable in other libraries. The inclusion of this tensor dimension also allows for simpele batching of parallel reservoirs during training, avoiding excessive GPU VRAM requirements.

ORC provides unified training functions that work with any model inheriting from the corresponding base class, including user-defined models with custom components. These functions delegate shape handling to the readout layer and accept keyword arguments that are forwarded to the model's `force` method, allowing the same training function to handle both discrete and continuous-time models transparently.

The library provides three built-in model classes: `ESNForecaster` for time series prediction, `ESNClassifier` for sequence classification, and `ESNController` for learning control policies with exogenous control inputs. Each composes embedding, driver, and readout components and provides task-specific methods (`forecast`, `classify`, `apply_control`). Users who need custom architectures can subclass the abstract base classes, define only the components that differ, and immediately use the unified training functions without reimplementing teacher forcing, autoregressive prediction, or ridge regression.

ORC also includes a data generation module with ODE and PDE integrators for standard benchmark systems, including the Lorenz-63 attractor, Rössler system, double pendulum, Lorenz-96 model, and the Kuramoto-Sivashinsky equation, all implemented using Diffrax.

5.5 *Research Impact Statement*

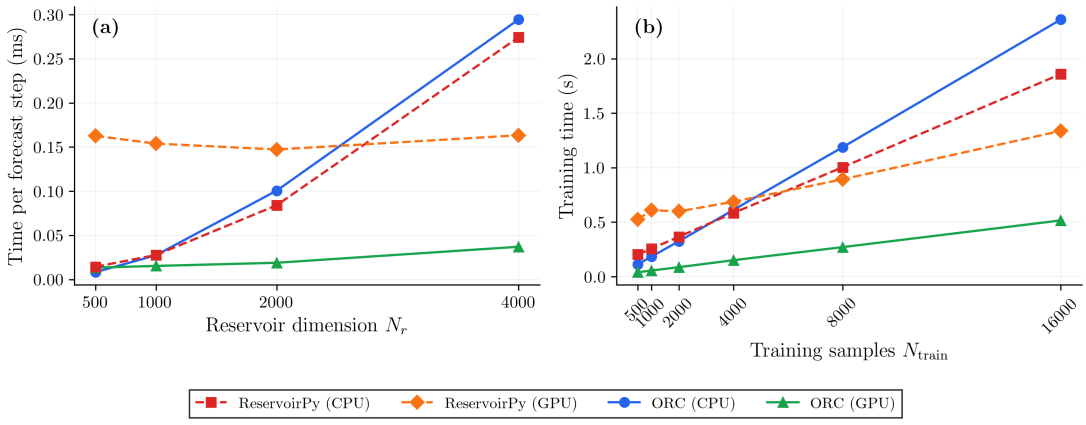


Figure 5.2: ORC with GPU acceleration enables significantly faster performance than ReservoirPy, even when using ReservoirPy's JAX backend. Panel (a) shows the time per forecast step of RC models with varying reservoir dimension trained to forecast the Lorenz system, while panel (b) shows training tine for RC models with fixed reservoir dimension of 2000 but varying number of training samples. Performance of the two libraries with and without GPU acceleration are shown. GPU results were obtained running on an NVIDIA A40 GPU and CPU results were obtained with an Apple M2 chip.

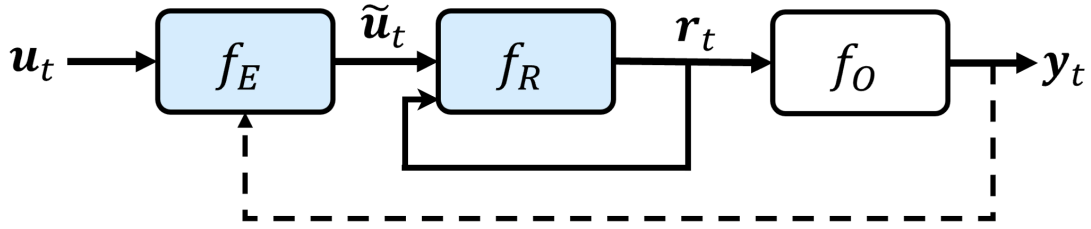
ORC addresses an immediate need for RC researchers; it allows for easy reimplementation of architectures such as (Özalp et al. 2023, 2025) that integrate reservoir computing with larger neural network architectures, and makes it easier to iterate on these ideas than any existing library. We also benchmark ORC against ReservoirPy in Fig. 5.2, across reservoir sizes and number of training samples. We find that with GPU acceleration, ORC scales far more favorably than ReservoirPy. Moreover, ORC is the only current package that supports training RC for control tasks, thanks to its efficiency and the end-to-end differentiability that JAX and Equinox provide. RC-based control has recently been shown to be advantageous over other popular forms of RNN-based MPC [168]. ORC's performance also makes it well suited for the parallel architectures needed to extend RC to higher-dimensional

data.

5.6 RC Background

The purpose of this notebook is to provide an introduction to reservoir computing (RC) by walking through the creation of an RC model within the ORC framework. RC was initially proposed independently by Maas, Natschläger, and Markram [1], and Jaeger [2] in the early 2000s, largely as a means of avoiding issues with training recurrent neural networks (RNNs) via backpropagation and gradient descent. Since then, the field of RC has grown rapidly, introducing many RC inspired architectures and new applications of RC approaches.

One of the central premises of ORC is that the vast majority of RC architectures can be decomposed into three components: an embedding that lifts a low-dimensional input signal u_t to a high-dimensional space, a driver that propagates a high-dimensional latent state r_t, and a readout that maps the latent state back to a lower dimensional signal $\hat{y}_t$. More concretely, we can typically represent the embedding f_E, driver f_R, and readout f_O as functions1. Schematically, a reservoir can be illustrated with these functions as a part of the below control loop. The shaded blue boxes are instantiated independent of training data, simplifying the training of f_O.



More formally, each function can be defined:

$$f_E : \mathbb{R}^{N_u} \to \mathbb{R}^{N_r} \quad \text{with} \quad \tilde{u}_t = f_E(u_t), \quad \text{(embedding)}$$

$$f_R : \mathbb{R}^{N_r} \times \mathbb{R}^{N_r} \to \mathbb{R}^{N_r} \quad \text{with} \quad r_{t+1} = f_R(r_t, \tilde{u}_t), \quad \text{(driver)}$$

and

$$f_O : \mathbb{R}^{N_y} \to \mathbb{R}^{N_r} \quad \text{with} \quad \hat{y}_t = f_O(r_t). \quad \text{(readout)}$$

We denote N_u as the input dimension, N_r the reservoir dimension, and N_y the output dimension. RC makes the assumption that if the reservoir dimension is sufficiently large ($N_r \gg N_u$) and the dynamics of the driver are sufficiently "expressive," then the parameters/weights of both the embedding and driver can be set independent of any training data; only the readout layer is trained. This greatly simplifies the training of an RC in comparison to RNNs as it completely avoids backpropagation through time. It also allows for the implementation of a wider variety of embeddings and drivers, including physical implementations of RC.

Throughout the rest of this notebook, we illustrate how ORC can be used to create and train a RC for forecasting with user-defined embedding, driver, and readout functions. For this example, we use a discrete time reservoir, however, ORC is also capable of using continuous time drivers.

```
# imports
import equinox as eqx
import jax
import jax.numpy as jnp

import orc
from orc.utils.regressions import ridge_regression
```

5.6.1 Defining an embedding

The most common choice for embedding function takes the form $f_E(u) = W_E u$, where $W_E \in \mathbb{R}^{N_r \times N_u}$ with entries drawn independently according to some distribution. To illustrate the flexibility of ORC, we instead define f_E as a two-layer MLP with ELU activation. This is accomplished by inheriting from `orc.embeddings.EmbedBase`, which requires specifying an input dimension `in_dim`, a reservoir dimension `res_dim`, and defining a method `embed` that maps from `in_dim` to `res_dim`.

```
# define embedding function
class ELUEmbedding(orc.embeddings.EmbedBase):
```

```
    W1: jnp.ndarray  # weight matrix for first layer of ELU MLP
    W2: jnp.ndarray  # weight matrix for second layer of ELU MLP
    b1: jnp.ndarray  # bias for first layer of ELU MLP
    b2: jnp.ndarray  # bias for second layer of ELU MLP

    def __init__(self, in_dim, res_dim, seed=0):
        super().__init__(in_dim=in_dim, res_dim=res_dim)
        rkey = jax.random.key(seed)
        W1key, W2key, b1key, b2key = jax.random.split(rkey, 4)
        # random initialization of parameters of ELU MLP
        self.W1 = jax.random.normal(
            W1key, shape=(res_dim // 2, in_dim)
        )
        self.W1 = self.W1 / jnp.sqrt((res_dim // 2) * in_dim)
        self.W2 = jax.random.normal(
            W2key,
            shape=(res_dim, res_dim // 2)
        )
        self.W2 = self.W2 / jnp.sqrt(res_dim * (res_dim // 2))
        self.b1 = jax.random.normal(
            b1key,
            shape=(res_dim // 2)
        )
        self.b1 = self.b1 / jnp.sqrt(res_dim // 2)
        self.b2 = jax.random.normal(b1key, shape=(res_dim))
        self.b2 = self.b2 / jnp.sqrt(res_dim)

    def embed(self, in_state):
        in_state = self.W1 @ in_state + self.b1
        in_state = jax.nn.elu(in_state)
        in_state = self.W2 @ in_state + self.b2
        in_state = jax.nn.elu(in_state)
        return in_state
```

5.6.2 Defining a driver

For the driver, we use the update equations of a gated recurrent unit (GRU). The GRU is already implemented in `equinox` so we need only wrap the existing `eqx.Module` in a `DriverBase` class. Although we redefine it here for illustrative purposes, this driver is actually built into `ORC`.

```
# define driver function
class GRUDriver(orc.drivers.DriverBase):

    gru: eqx.Module

    def __init__(self, res_dim, seed=0):
        super().__init__(res_dim=res_dim)
        key = jax.random.key(seed)
        self.gru = eqx.nn.GRUCell(res_dim, res_dim, key=key)

    def advance(self, res_state, in_state):
        return self.gru(in_state, res_state)
```

5.6.3 Defining a readout

While there is great flexibility in the assumed form of f_E and f_R, f_O is more restricted if we want to retain the ability to train the RC with a linear regression. We assume the output operator is linear, that is, $f_O(r_t) = W_O r_t$ where $W_O \in \mathbb{R}^{N_y \times N_r}$. We initialize the readout layer of our RC by subclassing `orc.readouts.ReadoutBase` and defining a `readout` method. We initialize W_O with zeros, although the initialization won't affect training.

```
# define readout function
class Readout(orc.readouts.ReadoutBase):
    res_dim: int
    out_dim: int
    wout: jnp.ndarray

    def __init__(self, out_dim, res_dim):
        super().__init__(out_dim, res_dim)
```

```
        self.wout = jnp.zeros((out_dim, res_dim))

    def readout(self, res_state):
        return self.wout @ res_state
```

5.6.4 Defining a forecaster

We are now able to combine these three components into a `Forecaster` that inherits from `orc.rc.RCForecasterBase`.

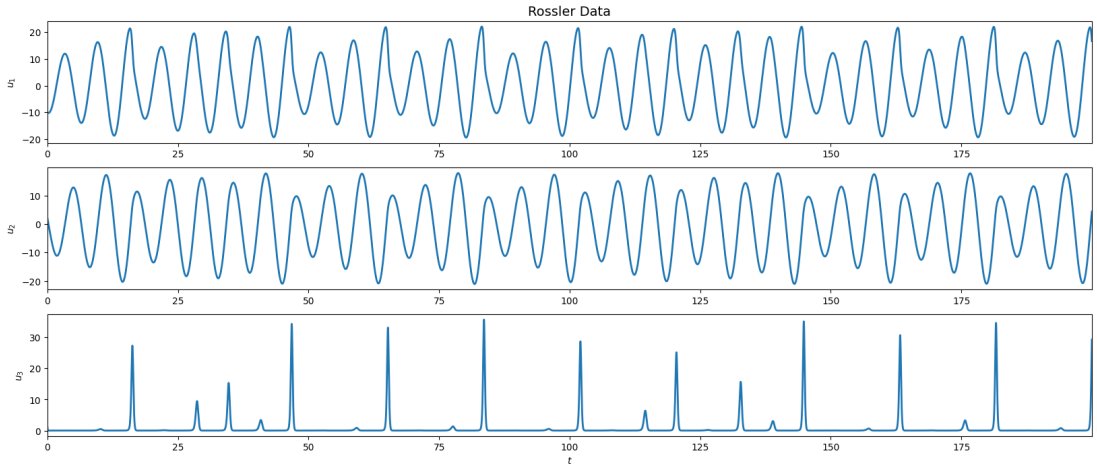
```
# define RC as embedding + driver + readout
class Forecaster(orc.forecaster.RCForecasterBase):
    driver: orc.drivers.DriverBase
    readout: orc.readouts.ReadoutBase
    embedding: orc.embeddings.EmbedBase
```

5.6.5 Training

Now that the input, driver, and readout functions are defined and packaged into a `Forecaster` object, we can turn to the training procedure for the model. For example data, we use a numerical integration of the Rossler system.

```
# integrate Rossler system
tN = 200
dt = 0.01
u0 = jnp.array([-10, 2, 1], dtype=jnp.float64)
U,t = orc.data.rossler(tN=tN, dt=dt, u0=u0)
split_idx = int(U.shape[0] * 0.8)
U_train = U[:split_idx]
U_test = U[split_idx:]
t_test = t[split_idx:]
orc.utils.visualization.plot_time_series(
    U,
    t,
    state_var_names=["$u_1$", "$u_2$", "$u_3$"],
    title="Rossler Data",
```

```
    x_label= "$t$",
)
```



The desired input (and output) dimension is that of the Rossler system, $N_y = N_u = 3$. We'll take $N_R = 500$ for this example.

```
# create RC
Nr = 500
Nu = 3
driver = GRUDriver(Nr)
embedding = ELUEmbedding(Nu, Nr)
readout = Readout(Nu, Nr)
model = Forecaster(driver, readout, embedding)
```

Our training data can be represented in a data matrix as

$$U_{\text{data}} = \begin{bmatrix} | & | & & | \\ u_1 & u_1 & \cdots & u_T \\ | & | & & | \end{bmatrix}.$$

Now, setting $r_0 = \mathbf{0}$, we can compute $r_1 = f_R(r_0, f_E(u_0))$, $r_2 = f_R(r_1, f_E(u_1))$, ..., $r_T = f_R(r_{T-1}, f_E(u_{T-1}))$ to generate

$$R_{\text{data}} \begin{bmatrix} | & | & & | \\ r_1 & r_1 & \cdots & r_T \\ | & | & & | \end{bmatrix}.$$

Learning W_O now just amounts to performing a Ridge regression to satisfy

$$R_{data}^T W_O^T = U_{data}^T.$$

In practice, we discard the first `spinup` columns of R_{data} and U_{data} to remove the effect of our (arbitrary) choice $r_0 = \mathbf{0}$. The ability to discard this initial transient relies on the principle of generalized synchronization and is frequently referred to as the echo state property in RC literature.

Since the `model` instance inherits from `RCForecasterBase`, generating R_{data} is easily accomplished in a single line.

```
# teacher force the reservoir
forced_seq = model.force(U_train[:-1], res_state=jnp.zeros((Nr)))

# shift the indices of the target sequence of training data
target_seq = U_train[1:]

# set transient to discard
spinup = 200

# learn wout
readout_mat = ridge_regression(
    forced_seq[spinup:],
    target_seq[spinup:],
    beta=1e-7
)
```

The `readout` attribute of `model` now needs to be updated to the learned `readout_mat`. However, many ORC objects are instances of `equinox.Module`, which are immutable. Thus, we need to create a new `Forecaster` object with `readout.W_O` set to `readout_mat`. This can be done with `equinox`'s convenient `tree_at` function.

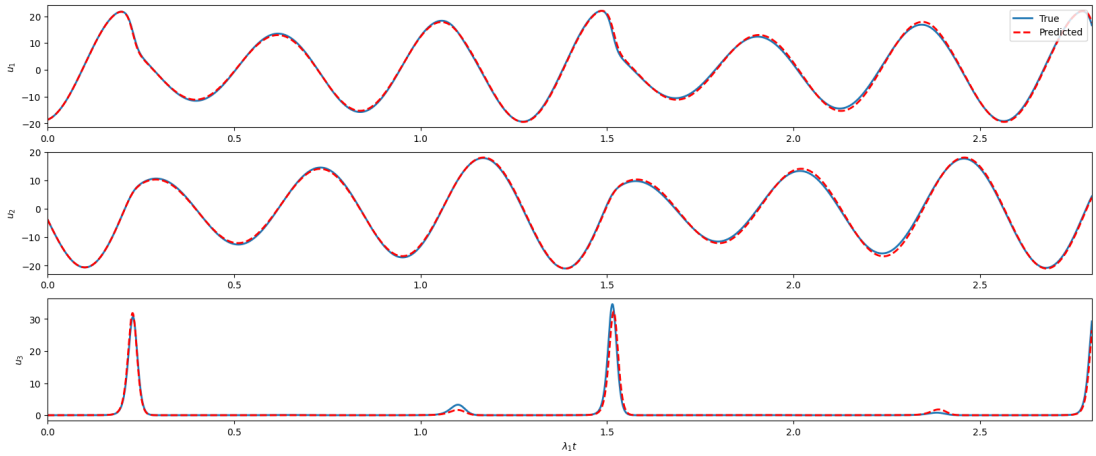
```
# define where in the forecaster model we need to update
def where(model: Forecaster):
    return model.readout.wout
model = eqx.tree_at(where, model, readout_mat)
```

```
# Alternatively
model, forced_seq = orc.forecaster.train_RCForecaster(
    model,
    train_seq=U_train,
    spinup=spinup,
    beta=1e-7
)
```

We can now perform a forecast and evaluate our trained RC:

```
# perform forecast
U_pred = model.forecast_from_IC(
    fcast_len=U_test.shape[0],
    spinup_data=U_train[-spinup:]
)

# plot forecast
orc.utils.visualization.plot_time_series(
    [U_test, U_pred],
    (t_test - t_test[0]) * 0.07,
    state_var_names=["$u_1$", "$u_2$", "$u_3$"],
    time_series_labels=["True", "Predicted"],
    line_formats=["-", "r--"],
    x_label= r"$\lambda _1 t$",
)
```



In summary, this example illustrates the modularity of ORC: given fixed input, output, and reservoir dimensions, any component of the `Forecaster` (embedding, driver, or readout) can be independently substituted with alternative implementations from ORC or user-defined mappings. This modularity is a central design principle of ORC, enabling rapid exploration of reservoir computing architectures without modifying the broader training and evaluation pipeline.

5.7 ORC Control

This notebook demonstrates using an ESN as a learned model for model-predictive control (MPC). The idea is to train an ESN to predict how a system responds to control inputs, then use that learned model to optimize control sequences that drive the system toward a desired reference trajectory.

The controller minimizes a cost function over a finite forecast horizon:

$$L(\mathbf{u}) = \alpha_1 \|\mathbf{y}(\mathbf{u}) - \mathbf{y}_{\text{ref}}\|^2 + \alpha_2 \|\mathbf{u}\|^2 + \alpha_3 \|\Delta\mathbf{u}\|^2,$$

where $\mathbf{y}(\mathbf{u})$ is the ESN's predicted output given control sequence $\mathbf{u}$, $\mathbf{y}_{\text{ref}}$ is the reference trajectory, and $\Delta\mathbf{u}$ penalizes non-smooth control. Optimization is performed via BFGS.

```
import orc
import jax
import jax.numpy as jnp
import matplotlib.pyplot as plt
```

5.7.1 Plant: Two-Mass Spring-Damper System

Our test system is a wall-coupled two-mass spring-damper:

```
Wall ---[k1, c1]--- m1 ---[k2, c2]--- m2
```

The state vector is $[x_1, x_2, v_1, v_2]$ (positions and velocities of both masses). A force applied to m_1 is the control input, and we observe the positions $[x_1, x_2]$.

```
class TwoMassSpringSystem:
    """
```

```
    Two-mass spring-damper system connected to a wall.

    Wall ---[k1, c1]--- m1 ---[k2, c2]--- m2

    State vector: [x1, x2, v1, v2] (positions and velocities)
    Input: Force applied to mass 1
    Output: Positions of both masses [x1, x2]
    """

    def __init__(
        self,
        m1=1.0,
        m2=1.0,
        k1=1.0,
        k2=1.0,
        c1=0.0,
        c2=0.0,
        dt=0.01
    ):
        self.m1 = m1
        self.m2 = m2
        self.k1 = k1
        self.k2 = k2
        self.c1 = c1
        self.c2 = c2
        self.dt = dt

        # Continuous-time state matrix with damping
        # State: [x1, x2, v1, v2]
        A_cont = jnp.array([
            [0, 0, 1, 0],
            [0, 0, 0, 1],
            [-(k1 + k2) / m1, k2 / m1, -(c1 + c2) / m1, c2 / m1],
            [k2 / m2, -k2 / m2, c2 / m2, -c2 / m2]
        ])
```

```
        # Continuous-time input matrix (force on m1)
        B_cont = jnp.array([[0], [0], [1 / m1], [0]])

        # Discretize using Euler method
        self.A = jnp.eye(4) + dt * A_cont
        self.B = dt * B_cont

        # Output matrix: observe positions [x1, x2]
        self.C = jnp.array([
            [1, 0, 0, 0],
            [0, 1, 0, 0]
        ])
        self.D = jnp.zeros((2, 1))

        self.state_dim = 4
        self.output_dim = 2

    def step(self, x, u):
        """Single timestep."""
        u = jnp.atleast_1d(u).reshape(-1, 1)
        x_next = self.A @ x + self.B @ u
        y = self.C @ x_next + self.D @ u
        return x_next, y.flatten()

    def simulate(self, u_sequence, x0=None):
        """Simulate over input sequence."""
        T = len(u_sequence)
        x = x0 if x0 is not None else jnp.zeros((self.state_dim, 1))

        states = []
        outputs = []

        for t in range(T):
            x, y = self.step(x, u_sequence[t])
            states.append(x.flatten())
            outputs.append(y)
```

```
        return jnp.stack(states), jnp.stack(outputs)
```

5.7.2 Generate Training Data

To train the ESN controller, we need input-output pairs that explore the system's behavior. We apply a random piecewise-constant force to m_1 and record the resulting positions. This gives the ESN a diverse set of dynamics to learn from.

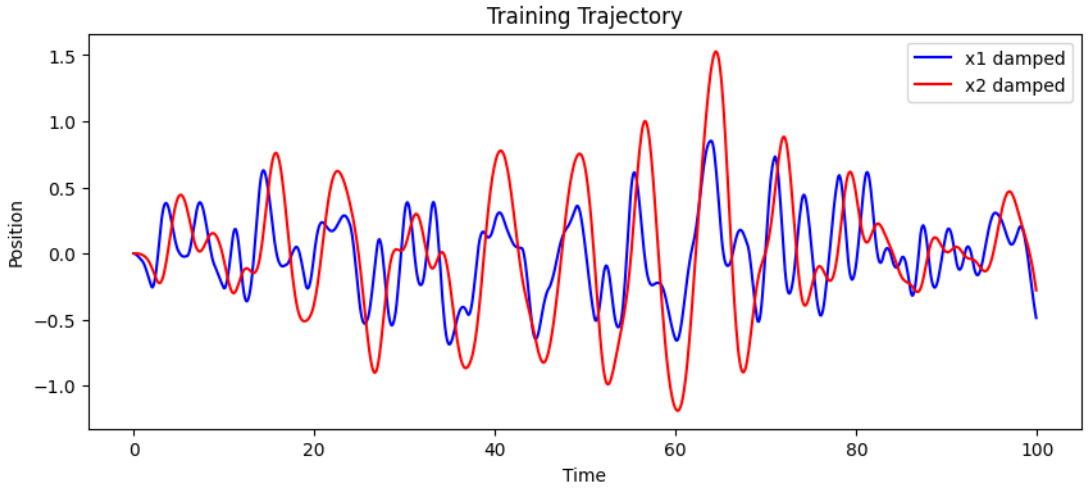
```
system_damped = TwoMassSpringSystem(
    m1=1.0,
    m2=1.0,
    k1=2.0,
    k2=1.0,
    c1=0.4,
    c2=0.4,
    dt=0.1
)

# Impulse input
T = 1000
t = jnp.arange(T) * 0.1
u = jax.random.uniform(
    jax.random.key(0),
    shape=(100,),
    minval=-1,
    maxval=1
)
u = jnp.repeat(u, 10).reshape(-1, 1)
print(u.shape)

states_damped, outputs_damped = system_damped.simulate(u)

plt.figure(figsize=(10, 4))
plt.plot(t, outputs_damped[:, 0], 'b-', label='x1 damped')
plt.plot(t, outputs_damped[:, 1], 'r-', label='x2 damped')
```

```
plt.xlabel('Time')
plt.ylabel('Position')
plt.legend()
plt.title('Training Trajectory')
plt.show()
```



5.7.3 Train the ESN Controller

`ESNController` takes both the observed outputs and control inputs as input to the reservoir embedding (concatenated). Training uses ridge regression, just like the forecasting models.

```
control_model = orc.control.ESNController(
    data_dim=2,
    control_dim=1,
    res_dim=1000
)
control_model, R = orc.control.train_RCController(
    control_model,
    outputs_damped[:-1],
    u[1:]
)
```

5.7.4 Closed-Loop Control

We run the controller in a receding-horizon loop. At each step:

1. `compute_control()` optimizes a 20-step control sequence to track the reference trajectory (all zeros — we want to bring the system to rest).
2. Only the first control input is applied to the real system.
3. The reservoir state is updated with the observed output and applied control via `force()`.
4. The process repeats from the new state.

This is the standard model-predictive control (MPC) pattern, with the ESN serving as the predictive model.

```
ref_traj = jnp.zeros((500, 2))
r0 = R[-1]
state, output = states_damped[-1:], outputs_damped[-1:]
state_list = []
control_list = []

for i in range(250):
    comp_control = control_model.compute_control(
        jnp.zeros((20, 1)), res_state=r0, ref_traj=ref_traj[i:i+20]
    )

    r0 = control_model.force(output, comp_control[0:1,0:1], r0)[0]

    state, output = system_damped.simulate(
        comp_control[0:1,0:1],
        state.reshape(4,-1)
    )
    state_list.append(state)
    control_list.append(comp_control[0,0])


state_list = jnp.array(state_list)
```

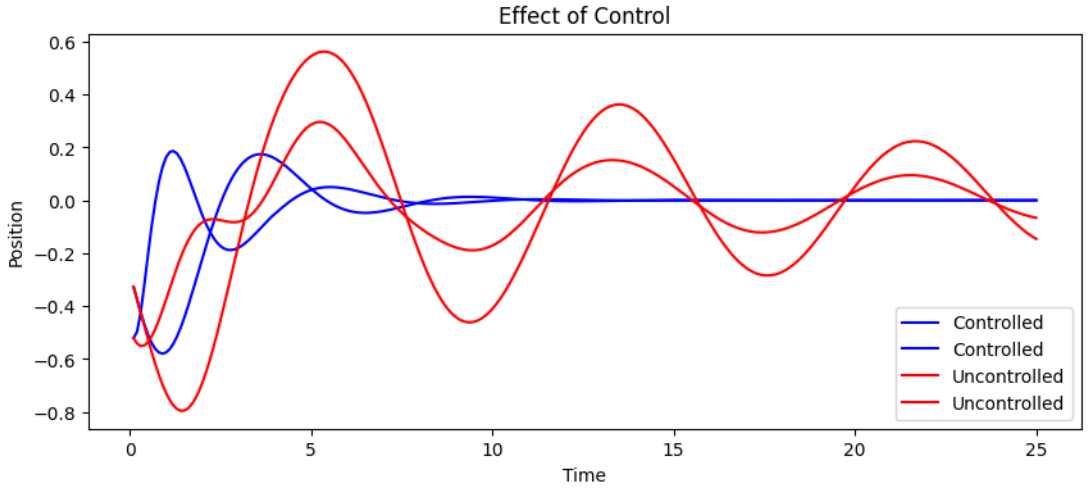
```
control_list = jnp.array(control_list)
```

5.7.5 Compare Controlled vs. Uncontrolled

For comparison, we simulate the system from the same initial state with zero input (no control). The controlled system should converge to the reference (zero displacement), while the uncontrolled system oscillates freely.

```
uncontrolled_states, uncontrolled_outputs = system_damped.simulate(
    jnp.zeros((250,1)),
    states_damped[-1:].reshape(4,-1)
)
```

```
t = jnp.arange(1, 251) * 0.1
plt.figure(figsize=(10, 4))
plt.plot(t, state_list[:, 0, [0,1]], 'b-', label='Controlled')
plt.plot(t, uncontrolled_outputs, 'r-', label='Uncontrolled')
plt.xlabel('Time')
plt.ylabel('Position')
plt.legend()
plt.title('Effect of Control')
plt.show()
```



5.8 ORC Classification

This notebook demonstrates how to use ORC's classifier module to classify time-series data using an Echo State Network (ESN). We generate synthetic time-series belonging to distinct classes, train an `ESNClassifier`, and evaluate its performance. This is a common task in areas like activity recognition, medical signal analysis, and sensor data classification.

As with other ORC modules, only `jax`, `orc`, and `equinox` are needed. We also import `matplotlib` for visualization.

```
import jax
import jax.numpy as jnp
import equinox as eqx
import matplotlib.pyplot as plt
import orc
```

5.8.1 Generate Synthetic Classification Data

We create a simple 3-class classification problem where each class is a sinusoidal time-series with a distinct frequency. A small amount of noise is added to each sample. This mimics a common structure in real-world time-series classification tasks, where different classes exhibit different temporal patterns.

```
# Dataset parameters
data_dim = 3
n_classes = 3
seq_len = 200
n_train_per_class = 20
n_test_per_class = 10

key = jax.random.PRNGKey(0)

def generate_samples(key, n_per_class):
    seqs, labels = [], []
    t = jnp.linspace(0, 4 * jnp.pi, seq_len).reshape(-1, 1)
    for class_idx in range(n_classes):
```

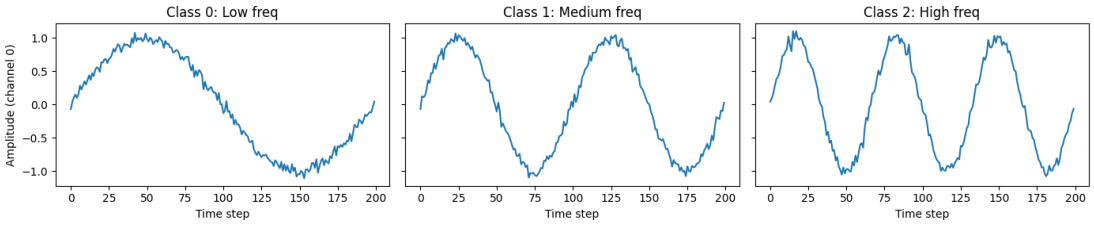
```
        freq = (class_idx + 1) * 0.5
        for _ in range(n_per_class):
            key, subkey = jax.random.split(key)
            noise = 0.05 * jax.random.normal(subkey, (seq_len, data_dim))
            seq = jnp.sin(freq * t * jnp.arange(1, data_dim + 1)) + noise
            seqs.append(seq)
            labels.append(class_idx)
    return jnp.stack(seqs), jnp.array(labels, dtype=jnp.int32), key

train_seqs, train_labels, key = generate_samples(key, n_train_per_class)
test_seqs, test_labels, key = generate_samples(key, n_test_per_class)

print(f"Training set: {train_seqs.shape}  (n_samples, seq_len, data_dim)")
print(f"Test set:     {test_seqs.shape}")
```

One example from each class is visualized to demonstrate how the temporal patterns differ.

```
fig, axes = plt.subplots(1, n_classes, figsize=(14, 3), sharey=True)
class_names = ["Low freq", "Medium freq", "High freq"]
for i in range(n_classes):
    idx = i * n_train_per_class
    axes[i].plot(train_seqs[idx, :, 0])
    axes[i].set_title(f"Class {i}: {class_names[i]}")
    axes[i].set_xlabel("Time step")
axes[0].set_ylabel("Amplitude (channel 0)")
plt.tight_layout()
plt.show()
```



5.8.2 Initialize the ESN Classifier

The `ESNClassifier` follows the same Embedding → Driver → Readout architecture as other ORC models. To instantiate a classifier, we specify:

- `data_dim`: the number of input channels per time-step
- `n_classes`: the number of classification classes
- `res_dim`: the dimension of the reservoir (latent state)

The `state_repr` parameter controls how reservoir states are summarized into a feature vector for classification:

- `"final"` (default): uses only the last reservoir state
- `"mean"`: averages all reservoir states (after an optional spinup period)

```
res_dim = 500

classifier = orc.classifier.ESNClassifier(
    data_dim=data_dim,
    n_classes=n_classes,
    res_dim=res_dim,
    seed=42,
)

print(f"Reservoir dimension: {classifier.res_dim}")
print(f"Input dimension:     {classifier.in_dim}")
print(f"Output dimension:    {classifier.out_dim}")
print(f"State representation: {classifier.state_repr}")
```

5.8.3 Train the Classifier

Training is performed by `orc.classifier.train_ESNClassifier`. Under the hood, this:

1. Forces each training sequence through the reservoir (in parallel via `jax.vmap`)
2. Extracts a feature vector per sequence (final state or mean state)

3. Solves ridge regression from features to one-hot encoded class labels
4. Updates the readout weights

The `beta` parameter controls Tikhonov regularization.

```
trained_classifier = orc.classifier.train_RCClassifier(
    classifier,
    train_seqs=train_seqs,
    labels=train_labels,
    beta=1e-6,
)

print("Training complete.")
```

5.8.4 Evaluate on Training Data

We can classify individual sequences using `classify`, which by default zero-initializes the reservoir state and returns a vector of class probabilities (via softmax). The predicted class is the one with the highest probability.

```
probs = jax.vmap(trained_classifier.classify)(train_seqs)

train_preds = jnp.argmax(probs, axis=1)
train_acc = jnp.mean(train_preds == train_labels)
print(f"Training accuracy: {train_acc:.1%}")
```

5.8.5 Evaluate on Test Data

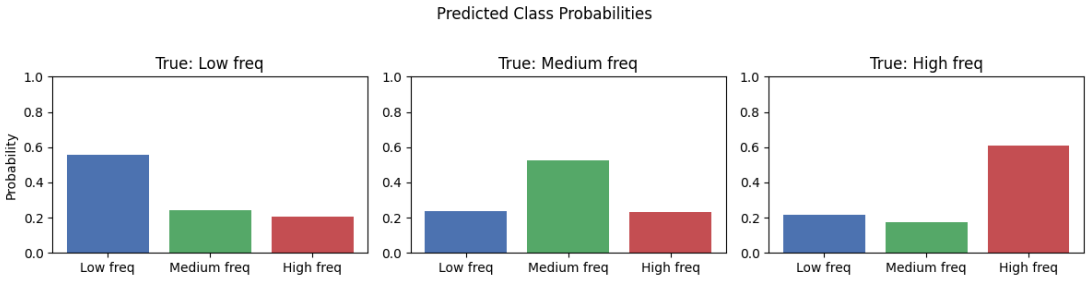
The real test of the classifier is on unseen data. We classify each test sequence and compute the accuracy.

```
probs = jax.vmap(trained_classifier.classify)(test_seqs)

test_preds = jnp.argmax(probs, axis=1)
test_acc = jnp.mean(test_preds == test_labels)
print(f"Test accuracy: {test_acc:.1%}")
```

The predicted probabilities for a few test samples assess the classifier confidence.

```
fig, axes = plt.subplots(1, 3, figsize=(12, 3))
for i, class_idx in enumerate(range(n_classes)):
    sample_idx = class_idx * n_test_per_class
    probs = trained_classifier.classify(test_seqs[sample_idx])
    axes[i].bar(
        range(n_classes),
        probs,
        color=["#4C72B0", "#55A868", "#C44E52"]
    )
    axes[i].set_xticks(range(n_classes))
    axes[i].set_xticklabels(class_names)
    axes[i].set_ylim(0, 1)
    axes[i].set_title(f"True: {class_names[class_idx]}")
    axes[i].set_ylabel("Probability" if i == 0 else "")
plt.suptitle("Predicted Class Probabilities", y=1.02)
plt.tight_layout()
plt.show()
```



5.8.6 Using Mean State Representation

Instead of using only the final reservoir state for classification, we can average the reservoir states over the sequence. This is often more robust when the discriminative information is spread throughout the time-series rather than concentrated at the end — for example, when class identity is determined by frequency content or statistical properties of the entire signal. The `spinup` parameter discards early transient states before averaging, since the reservoir needs time to synchronize with the input.

```
classifier_mean = orc.classifier.ESNClassifier(
    data_dim=data_dim,
    n_classes=n_classes,
    res_dim=res_dim,
    seed=42,
    state_repr="mean",
)

trained_mean = orc.classifier.train_RCClassifier(
    classifier_mean,
    train_seqs=train_seqs,
    labels=train_labels,
    spinup=20,
    beta=1e-6,
)

in_seq = test_seqs
res_state = None
spinup = 20
probs = eqx.filter_vmap(trained_mean.classify)(
    in_seq,
    res_state,
    spinup
    )

mean_preds = jnp.argmax(probs, axis=1)
mean_acc = jnp.mean(mean_preds == test_labels)
print(f"Test accuracy (mean state): {mean_acc:.1%}")
```

5.8.7 Summary

This notebook demonstrates the core classification workflow in ORC:

1. **Initialize** an `ESNClassifier` with `orc.classifier.ESNClassifier`
2. **Train** with `orc.classifier.train_ESNClassifier`, which uses ridge regression for a closed-form solution

3. **Classify** new sequences with `classify` (reservoir state defaults to zero if not provided)

Key parameters to tune:

- `res_dim`: larger reservoirs can capture more complex dynamics
- `beta`: regularization strength (increase if overfitting)
- `state_repr`: `"final"` vs `"mean"` depending on where class-discriminative information lies in the sequence
- `spinup`: number of transient states to discard when using `state_repr="mean"`

Chapter 6

SHRED-ESN

6.1 Introduction

The preceding chapters developed methods for two tasks that arise throughout the study and application of data-driven modeling of dynamical systems. Namely, Chapter 3 introduced SHRED as a recurrent architecture that reconstructs high-dimensional states from sparse sensors, and Chapters 4 and 5 developed echo state networks as efficient surrogate models of dynamical systems. These contributions considered the state estimation and system identification problems separately, but there is great value to addressing the problems jointly [29, 56, 112]. The task at hand then becomes to forecast the future evolution of the state using only a small number of noisy sensors and without a first principles dynamical model. Existing approaches to reduced order modeling based on autoencoders function by learning a nonlinear dimensionality reduction and training a time stepper in the resulting latent space. Consequently, this means that the full state must first be reconstructed from partial observations before the forecasting pipeline can be applied.

This chapter unifies the sensing and modeling directions of the preceding chapters into a single approach for forecasting from sparse sensors. A trained SHRED model encodes time-lagged sensor trajectories into a latent representation. The latent states extracted from the training data form a time series in a low-dimensional space upon which an echo state network is trained to propagate autonomously. At deployment, SHRED encodes an initial window of sensor measurements into a latent state, the ESN iterates its own predictions forward to produce a latent state forecast, and the decoder of the SHRED network estimates the high-dimensional state at any time point along the way. We demonstrate this approach on the Kuramoto-Sivashinsky equation at varying domain lengths L and on the two-dimensional Kolmogorov flow at Reynolds number 30. Beyond short-term forecasting accuracy, we show that the latent space time stepper replicates the Lyapunov spectrum of the true dynamics

for several of the systems studied. The replication of the Lyapunov spectrum provides strong empirical evidence that the latent representation faithfully captures the geometric and statistical structure of the underlying system.

6.2 Methods

The forecasting pipeline developed in this chapter combines the SHRED architecture introduced in Chapter 3 with the echo state networks developed in Chapters 4 and 5. The pipeline proceeds in three stages. First, a SHRED model is trained on the system of interest, following the procedure described in Chapter 3. Given time-lagged trajectories of sparse sensor measurements as input, the GRU encoder maps each trajectory to a latent state $z_k \in \mathbb{R}^r$, and the shallow decoder is trained to reconstruct the full high-dimensional state $x_k \in \mathbb{R}^n$ from z_k. Once the SHRED model is trained, the encoder is run over the full training dataset to extract the sequence of latent states $z_1, z_2, \ldots, z_m$. These latent states form a time series in $\mathbb{R}^r$, with $r \ll n$, that captures the temporal evolution of the system in a compressed representation. In the second stage, an echo state network is trained on this latent time series to learn the autonomous map $z_{k+1} \approx F(z_k)$, where F is the input-driven ESN update followed by a linear readout, as described in Chapter 4 and implemented with the software developed in Chapter 5. At deployment, the pipeline operates as follows: an initial window of sparse sensor measurements is fed through the trained SHRED encoder to produce a latent state z_0. The ESN then iterates its own predictions forward autonomously, generating a latent-space trajectory $z_1, z_2, \ldots, z_T$ without any further sensor input. The full high-dimensional state can be recovered at any point along this trajectory by applying the SHRED decoder, $\hat{x}_k = \mathcal{D}(z_k)$.

The SHRED models in this chapter use transposed convolutional layers instead of feed-forward layers for the shallow decoder. Additionally, the encoder is a GRU network that processes time-lagged sensor trajectories of length L_{lag} and produces a latent state of dimension r. The model is trained by minimizing the mean squared reconstruction error on the training data, as in chapter 3.

Once the SHRED model is trained, the latent time series is extracted by running a forward pass of the encoder over the training data. The resulting sequence is normalized to

zero mean and unit variance along each coordinate before being used to train the ESN. The ESN is configured with a reservoir of size N, spectral radius ρ, input scaling σ_{in}, leaking rate α, and Tikhonov regularization parameter β. An initial spinup period of N_{spin} time steps is discarded to allow the reservoir state to become independent of its initialization. Hyperparameters are selected via grid search using the OpenReservoirComputing library, with the search conducted over a validation set held out from the training data.

The primary evaluation metric in this chapter is the Lyapunov spectrum of the learned latent dynamics, which we now define. Consider a dynamical system evolving on a state space $\mathbb{R}^d$, and let z_0 be a point on the attractor. A Lyapunov exponent measures the average rate of exponential divergence or convergence of trajectories that are infinitesimally close to the reference trajectory starting at z_0. More formally, the i-th Lyapunov exponent λ_i characterizes the asymptotic growth rate along the i-th direction:

$$\lambda_i = \lim_{T\to\infty} \frac{1}{T} \ln \frac{\|J_T\, e_i\|}{\|e_i\|}, \tag{6.1}$$

where $J_T = \prod_{k=0}^{T-1} \frac{\partial F}{\partial z}\big|_{z_k}$ is the product of Jacobians along the trajectory and e_i is the i-th direction identified by the limiting process. The full ordered set $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ is the Lyapunov spectrum. The spectrum encodes fundamental properties of the attractor. Positive exponents correspond to directions along which nearby trajectories diverge, and their number characterizes the dimension of the unstable manifold. The sum of all exponents is related to the fractal dimension of the attractor through the Kaplan-Yorke conjecture.

We compute the Lyapunov spectrum using a standard QR-based algorithm [1,138]. The algorithm proceeds as follows. An orthonormal set of perturbation vectors is initialized and evolved forward along the trajectory by multiplication by the local Jacobian $\frac{\partial F}{\partial z}\big|_{z_k}$ at each time step. Periodically, a QR decomposition is applied to re-orthonormalize the perturbation vectors, preventing them from collapsing onto the direction of the maximal Lyapunov exponent. After a sufficiently long trajectory, the time-averaged logarithms converge to the Lyapunov exponents. For the true system, the Jacobian is computed from the governing equations evaluated along a reference trajectory on the attractor. For the learned latent dynamics, the Jacobian of the ESN update map is used in place of the true Jacobian, and the algorithm is applied along a trajectory generated by the autonomous ESN.

Replication of the Lyapunov spectrum is a substantially stronger test of a learned model than short-term forecasting accuracy [138]. A model can produce accurate predictions over short horizons while failing to capture the long-term geometric and statistical properties of the attractor. Two dynamical systems can agree on individual trajectories for finite times and yet have fundamentally different Lyapunov spectra. If the learned latent dynamics replicate the spectrum of the true system, it provides evidence that the latent representation has captured the intrinsic dynamical structure rather than just fitting particular trajectories.

6.3 Experimental Systems

The Kuramoto-Sivashinsky (KS) equation is a canonical model of spatiotemporal chaos [94, 149]. The equation governs the evolution of a scalar field $u(t, x)$ on a periodic domain of length L:

$$u_t + uu_x + u_{xx} + u_{xxxx} = 0, \quad x \in [0, L], \quad u(x, 0) = u_0(x), \tag{6.2}$$

with periodic boundary conditions $u(t, x + L) = u(t, x)$. The second-order term u_{xx} is destabilizing and drives energy production at long wavelengths, while the fourth-order term u_{xxxx} is dissipative and damps short wavelengths. The nonlinear advection uu_x transfers energy between scales. The balance among these three effects produces chaotic dynamics whose complexity is controlled by the domain length L. As L increases, more unstable modes fit within the domain, the attractor dimension grows, and the dynamics become increasingly complex [72].

We study the KS equation at two domain lengths, $L = 22$ and $L = 44$. The $L = 22$ case is a standard benchmark in the dynamical systems and machine learning literature, producing low-dimensional chaotic dynamics with a small number of positive Lyapunov exponents. The $L = 44$ case roughly doubles the number of positive Lyapunov exponents, yielding a higher-dimensional attractor and more complex spatiotemporal behavior. This pair of domain lengths provides a controlled comparison since the governing equation is the same, but the intrinsic complexity of the dynamics increases substantially between the two cases. In turn, this allows us to test how the forecasting pipeline and Lyapunov spectrum replication scale with attractor dimension.

For each domain length, we collect m snapshots of the spatially discretized state $u_k \in \mathbb{R}^n$ at uniform time intervals from a trajectory on the attractor, after discarding an initial transient. Sparse sensor measurements are obtained by selecting p components of the state vector at random spatial locations. The same random sensor locations are used across all snapshots. In all experiments with the KS system, we take $p = 25$ and $n = 128$, following [130]. The SHRED model is trained on trajectories of these p sensor measurements, with the latent dimension r and lag length L_{lag} specified below.

The other benchmark system we consider is the Kolmogorov flow. The Kolmogorov flow is a two-dimensional incompressible flow driven by a sinusoidal body force, governed by the Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{u} + \mathbf{f}, \quad \nabla \cdot \mathbf{u} = 0, \tag{6.3}$$

on a periodic domain $[0, 2\pi]^2$, where $\mathbf{u} = (u, v)$ is the velocity field, p is the pressure, Re is the Reynolds number, and $\mathbf{f}$ is a steady forcing, typically of the form $\mathbf{f} = (\sin(k_f y), 0)$ for some forcing wavenumber k_f. We take $k_f = 4$ and study the flow at $Re = 30$, a regime in which the flow is quasi periodic. Data is obtained using the pseudo-spectral method. Compared to the KS equation, this Kolmogorov flow is a more challenging test case in the sense that there is a larger discrepancy between the number of available sensors and the state dimension, but easier in the sense that dynamics are not chaotic.

Training data is collected by sampling 100,000 snapshots at uniform time intervals from a trajectory on the attractor, after discarding an initial transient to eliminate dependence on the initial condition. As in the KS experiments, $p = 144$ sensors are placed at random spatial locations and the same locations are used across all snapshots.

6.4 Results

We first compare the reconstruction performance of SHRED with that of a convolutional autoencoder (CAE) [130]. The CAE serves as an upper bound on reconstruction performance since it has access to the full state as input, while SHRED operates only on trajectories of sparse sensor measurements. At a fixed latent dimension, the gap between the two quantifies the cost of operating without full-state information.

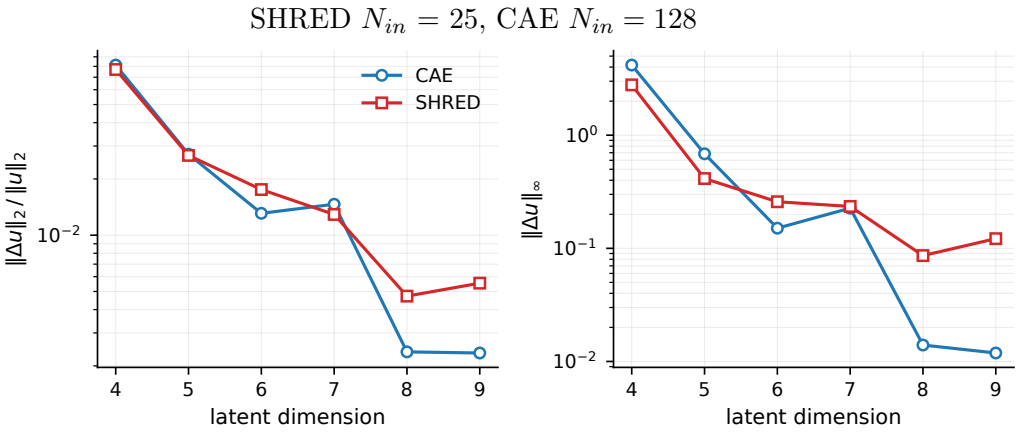


Figure 6.1: Reconstruction error of the convolutional autoencoder (CAE) and shallow recurrent decoder (SHRED) on the KS system with $L = 22$ as a function of latent dimension d. N_{in} denotes the input dimension for each model. Left: aggregate relative ℓ_2 error. Right: maximum pointwise error.

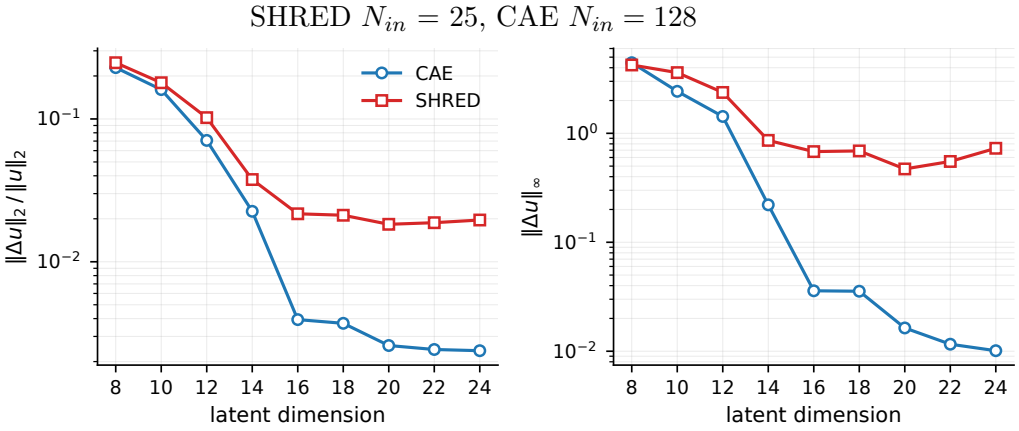


Figure 6.2: Reconstruction error of CAE and SHRED on the KS system with $L = 44$ as a function of latent dimension d. N_{in} denotes the input dimension for each model. Left: aggregate relative ℓ_2 error. Right: maximum pointwise error.

Figures 6.1 and 6.2 show the aggregate relative ℓ_2 error and maximum pointwise error for both architectures on the KS system at $L = 22$ and $L = 44$, respectively. At $L = 22$, both architectures achieve comparable reconstruction error at low latent dimensions, but

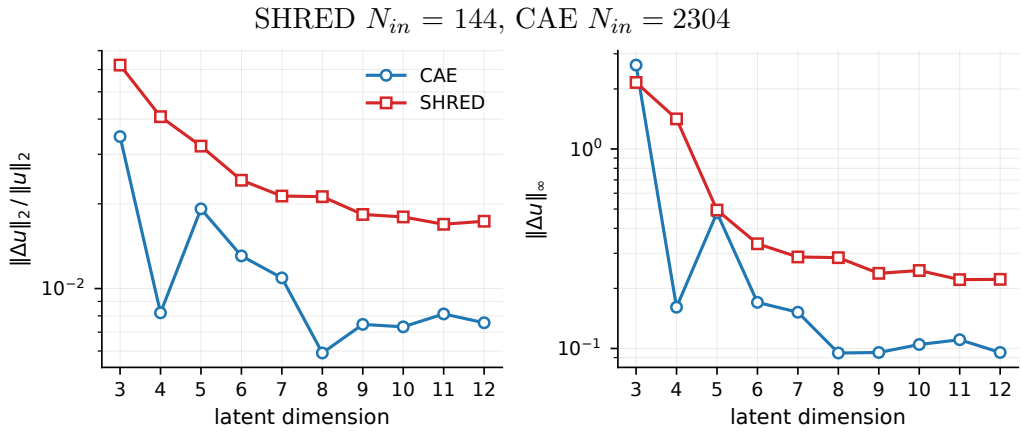


Figure 6.3: Reconstruction error of the CAE and SHRED on the Kolmogorov flow at $Re = 30$ as a function of latent dimension d. N_{in} denotes the input dimension for each model. Left: aggregate relative ℓ_2 error. Right: maximum pointwise error.

plateau beyond $d = 8$, with the CAE plateauing at roughly one-third the aggregate relative ℓ_2 error of SHRED. At $L = 44$, the CAE consistently outperforms SHRED for $d > 14$ by approximately an order of magnitude. Despite this gap in reconstruction error, both approaches produce spatiotemporal fields that are visually indistinguishable from the ground truth over the time intervals considered, as shown in Figures 6.4 and 6.5. The pointwise error plots reveal that the SHRED error, while small in magnitude relative to the field amplitude, is larger than that of the CAE. For all subsequent experiments, we fix the latent dimension at $d = 8$ for $L = 22$ and $d = 18$ for $L = 44$.

Figures 6.3 and 6.6 report the same comparisons for the Kolmogorov flow at $Re = 30$. The CAE again achieves lower reconstruction error across latent dimensions, with both approaches plateauing around $d = 7$. All subsequent Kolmogorov flow experiments use $d = 7$. The reconstructions show the same qualitative behavior: both are visually indistinguishable from the ground truth, with the SHRED discrepancies small in magnitude. Across all three systems, the SHRED reconstructions are sufficiently accurate to capture the relevant spatiotemporal dynamics from sparse sensors alone, despite exhibiting higher reconstruction error than the CAE.

We next evaluate the forecasting performance of the CAE + ESN and SHRED + ESN

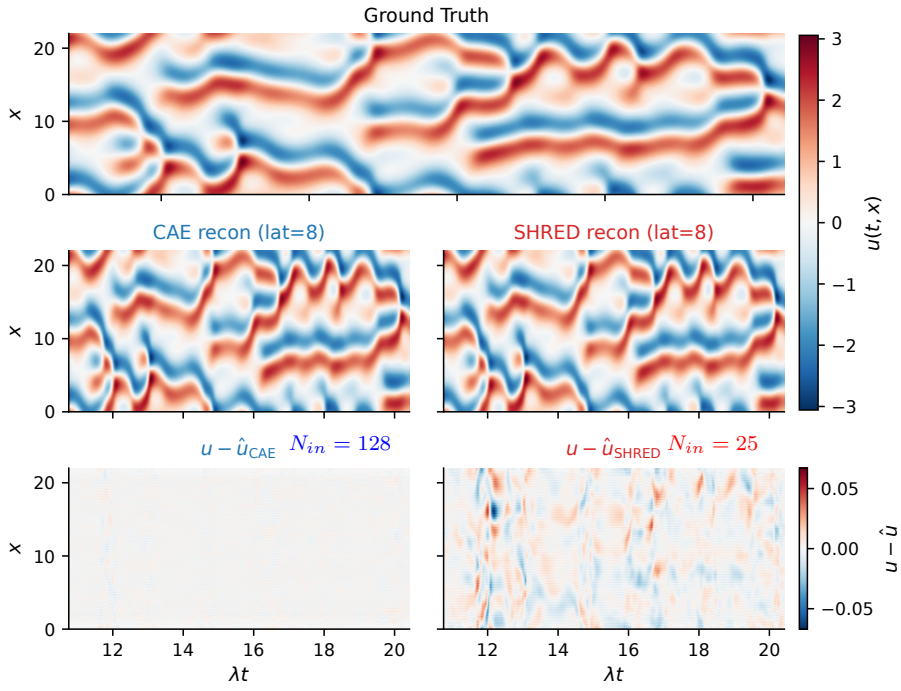


Figure 6.4: Example reconstructions from CAE and SHRED on the KS system with $L = 22$ over 10 Lyapunov times. N_{in} denotes the input dimension for each model. Top: ground truth. Middle: reconstructions. Bottom: pointwise errors. Both models use latent dimension $d = 8$.

pipelines. In both cases, an echo state network is trained to autonomously propagate the latent state forward in time, and the decoder recovers the full state. Figure 6.7 shows the forecasting error for the KS system at $L = 22$ evaluated over $N = 50$ randomly selected initial conditions from a withheld test trajectory. The two approaches provide comparable forecasting accuracy, despite the fact that the CAE outperformed the SHRED model in terms of reconstruction alone. Figure 6.8 presents the same comparison for $L = 44$. Here, the SHRED + ESN approach outperforms the CAE + ESN. The valid prediction time for both approaches is shorter than at $L = 22$, as is to be expected from the increased complexity. Still, the SHRED + ESN performance advantage persists throughout the 5 Lyapunov time forecast horizon. This result is notable given that the CAE achieves substantially

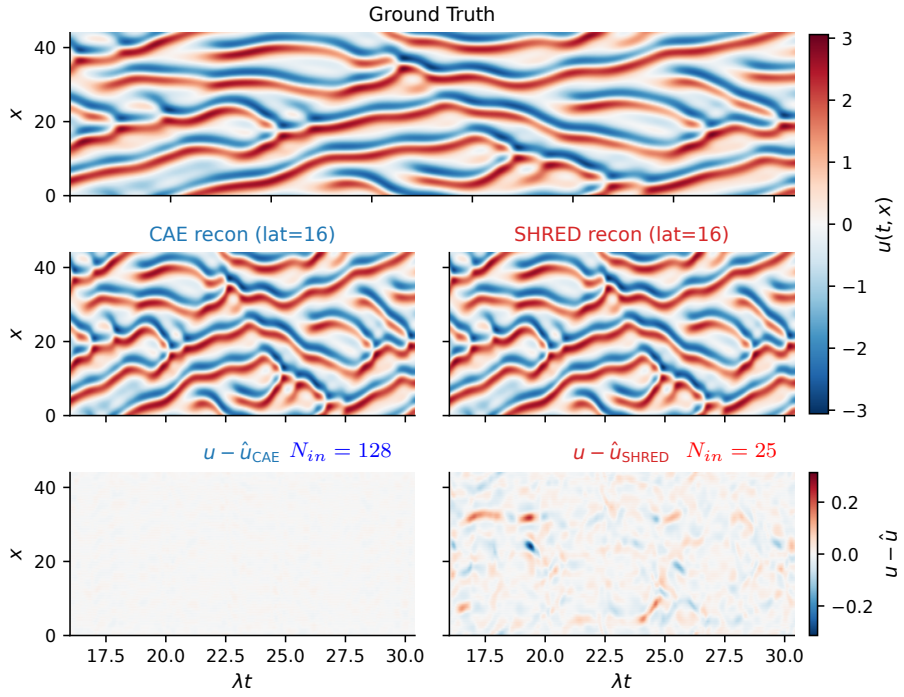


Figure 6.5: Example reconstructions from CAE and SHRED on the KS system with $L = 44$ over 10 Lyapunov times. N_{in} denotes the input dimension for each model. Top: ground truth. Middle: reconstructions. Bottom: pointwise errors. Both models use latent dimension $d = 16$.

lower reconstruction error when forecasting is not considered. The fact that SHRED + ESN produces better forecasts suggests that reconstruction accuracy alone does not determine forecasting performance. The recurrent structure of the SHRED encoder produces a latent representation whose dynamics are more structured and amenable to learning by the ESN. In other words, the temporal conditioning provided by the recurrent encoder structures the latent space such that it is more amenable to autonomous propagation, even if the resulting reconstruction is less accurate.

Fig. 6.9 shows the forecasting comparison for the Kolmogorov flow at $Re = 30$ evaluated over $N = 30$ initial conditions. The CAE + ESN outperforms SHRED + ESN with lower median error throughout the duration of the forecast horizon. However, both approaches

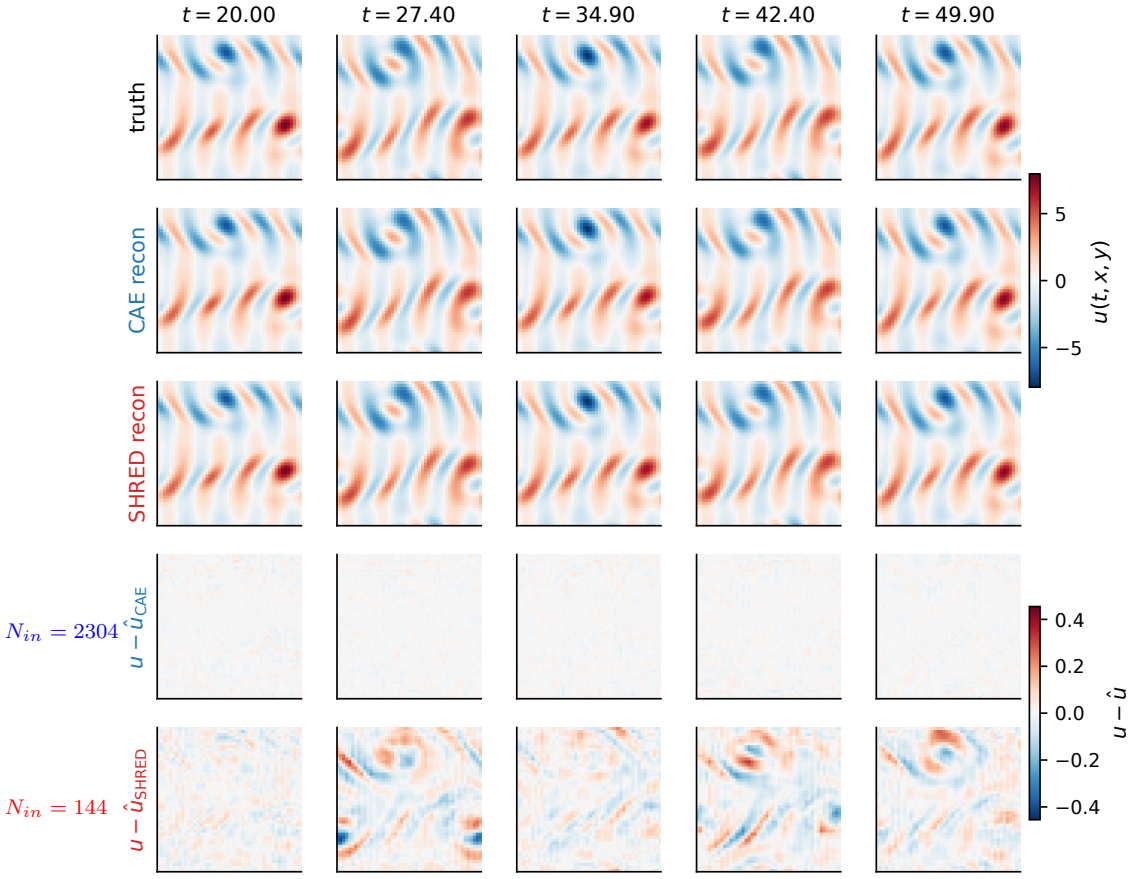


Figure 6.6: Example reconstructions from CAE and SHRED on the Kolmogorov flow at $Re = 30$ over 50 dimensionless time units. N_{in} denotes the input dimension for each model. Top: ground truth. Middle rows: CAE and SHRED reconstructions. Bottom rows: pointwise errors. Both models use latent dimension $d = 8$.

yield accurate forecasts over long forecast horizons (150 nondimensional time units), with errors that stay roughly flat rather than growing exponentially as in the KS system. This observation is consistent with the fact that the considered flow is quasi-periodic.

Across the three example systems, we observe that as dynamical complexity increases, the performance of the SHRED + ESN approach improves in comparison to the CAE + ESN pipeline. This suggests that accurate reconstructions are not the sole determining factor for reduced order model performance. The structure of the learned latent state is

also critical.

To that end, we now consider the Lyapunov spectrum of the learned latent dynamics of the SHRED + ESN approach. Fig. 6.10 presents the Lyapunov spectra for the KS equation at both domain lengths, computed from the true dynamics and from the learned ESN. Following the convention of [130, 138], the shown spectra are augmented with two additional zero Lyapunov exponents.

For both $L = 22$ and $L = 44$, the SHRED + ESN spectrum matches the ground truth closely across the entire range of considered exponents. Plots of the Lyapunov spectra of the Kolmogorov flow are omitted as there are no positive exponents. The replication of the true dynamics in the SHRED latent space provides strong empirical support for a Takens-like interpretation that the latent representation, derived from sparse measurement trajectories, preserves the dynamical invariants of the original high-dimensional system.

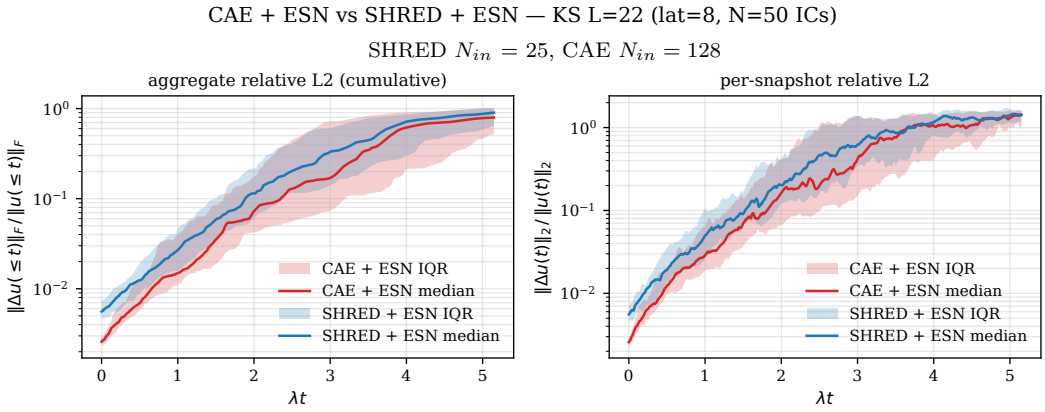


Figure 6.7: Forecasting error for the KS system with $L = 22$ with a CAE + ESN and SHRED + ESN architecture, evaluated over $N = 50$ initial conditions. N_{in} denotes the input dimension for each model. Left: cumulative relative ℓ_2 error. Right: per-snapshot relative ℓ_2 error. Shaded regions denote interquartile ranges.

6.5 Discussion

The results of this chapter demonstrate that the SHRED + ESN pipeline enables autonomous forecasting of chaotic dynamical systems directly from sparse sensor measure-

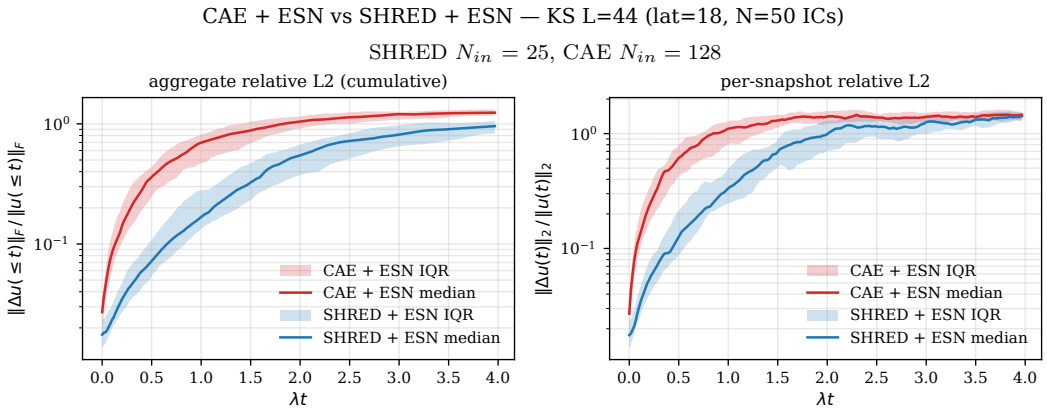


Figure 6.8: Forecasting error for the KS system with $L = 44$ with a CAE + ESN and SHRED + ESN architecture, evaluated over $N = 50$ initial conditions. N_{in} denotes the input dimension for each model. Left: cumulative relative ℓ_2 error. Right: per-snapshot relative ℓ_2 error. Shaded regions denote interquartile ranges.

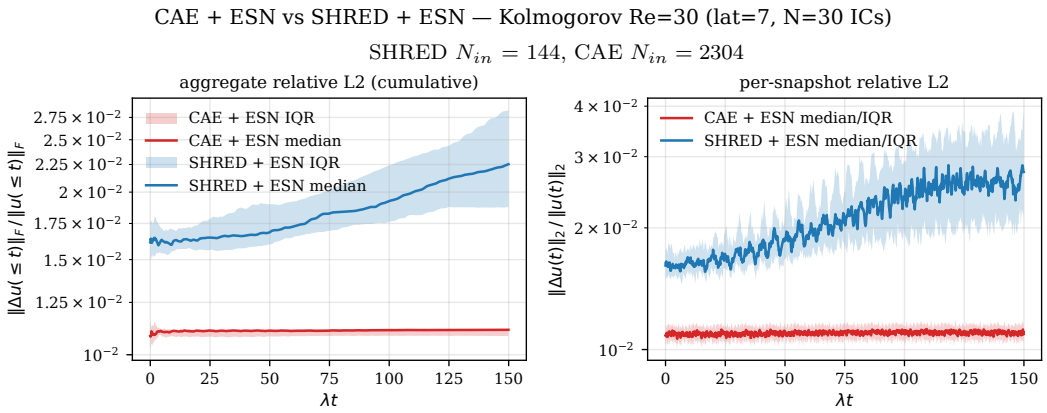


Figure 6.9: Forecasting error for the Kolmogorov flow with a CAE + ESN and SHRED + ESN architecture, evaluated over $N = 30$ initial conditions. N_{in} denotes the input dimension for each model. Left: cumulative relative ℓ_2 error. Right: per-snapshot relative ℓ_2 error. Shaded regions denote interquartile ranges.

ments, without requiring access to the full state at deployment. On the Kuramoto-Sivashinsky equation, the pipeline produces accurate short-term forecasts at both $L = 22$ and $L = 44$,

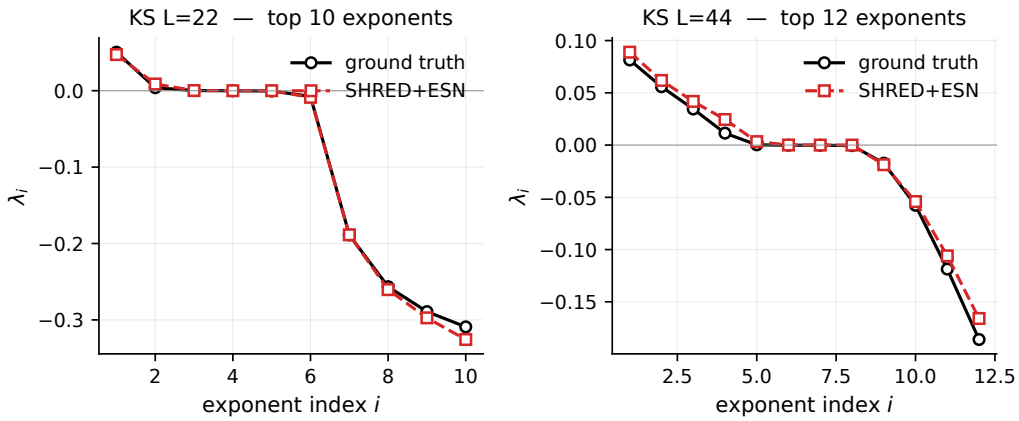


Figure 6.10: Lyapunov spectra of the true KS dynamics (black) and the learned SHRED + ESN latent dynamics (red, dashed) for $L = 22$ (left, top 10 exponents) and $L = 44$ (right, top 12 exponents). Spectra are augmented with two additional zero exponents following [138].

with the valid prediction time decreasing as the attractor dimension grows. On the Kolmogorov flow at $Re = 30$, the pipeline extends to a system with two-dimensional spatial structure and a substantially higher state dimension, producing forecasts that capture the spatial structure of the flow field over the prediction horizon. Most significantly, the autonomous latent-space time stepper replicates the Lyapunov spectrum of the true high-dimensional dynamics for the systems studied.

The SHRED-based pipeline offers a practical advantage over forecasting approaches built on autoencoders. In an autoencoder-based pipeline, the encoder takes the full state as input, which means that in a sensor-constrained setting, the full state must first be reconstructed from partial observations before the forecasting pipeline can be applied. The state estimation problem is itself nontrivial, and errors introduced at this stage propagate through the encoder into the latent representation and corrupt downstream forecasts. The SHRED-based pipeline avoids this issue entirely. The encoder operates directly on sparse sensor trajectories, and the latent representation it produces is conditioned on the available measurements from the start. This advantage becomes more significant as the gap between the number of available sensors and the state dimension grows. This is exactly the regime

encountered in most real-world applications.

Several limitations of the present work suggest directions for future investigation. The KS equation was studied at two domain lengths, and the degradation in forecasting accuracy from $L = 22$ to $L = 44$ raises the question of how the pipeline scales to systems with significantly higher-dimensional attractors. The Kolmogorov flow was studied at a single Reynolds number for which the flow was quasi-periodic. At higher Reynolds numbers, the attractor dimension grows and the dynamics become increasingly turbulent, presenting a natural axis along which to test the limits of the approach. More broadly, the systems studied here are statistically stationary. How the pipeline performs on systems with intermittent dynamics, transient chaos, or slowly drifting parameters has not been explored. The hyperparameter sensitivity of the ESN, while manageable for the systems considered in this chapter, may become a more significant challenge as the complexity of the latent dynamics increases, motivating the hybrid reservoir approaches discussed in Chapter 7.

Chapter 7

CONCLUSION

This dissertation has argued that RNN approaches provide a computationally efficient and expressive framework for data-driven modeling of dynamical systems from sparse, partial observations. The problems addressed included state estimation from sparse sensors, surrogate modeling for control, and autonomous forecasting. While distinct tasks, in all cases we seek to learn a compact, temporal representation of high-dimensional dynamical system from limited data. The methods developed across Chapters 3 through 6 are unified by the insight that the recurrent structure of RNNs provides an inductive bias that is fundamentally well-matched to this class of problems. The sequential, causal processing of inputs mirrors the time-ordered evolution of physical systems. This structural alignment, rather than brute-force size or training data, is what enables the methods in this dissertation to operate effectively in the low-data, sparse-observation regimes that are typical of science and engineering.

The contributions of this dissertation build to the work presented in Chapter 6. SHRED [170] demonstrated that recurrent processing of sensor trajectories enables accurate, nonlinear state estimation from sparse measurements. By operating on trajectories rather than snapshots, SHRED incorporates temporal filtering that is absent from classical snapshot-based methods like gappy POD, while its learned nonlinear decoder avoids the limitations of a fixed linear basis. Echo state networks, developed in the context of system identification and control, demonstrated that fixing the recurrent weights and training only a linear output layer produces surrogate models that are accurate enough to capture nonlinear dynamics and cheap enough to evaluate thousands of times within an MPC optimization loop [168]. OpenReservoirComputing made these reservoir computing methods computationally practical, providing a modular, GPU-accelerated platform for constructing, training, and deploying reservoir models at scale [169]. Finally, the integration of SHRED with

echo state networks in Chapter 6 unified the sensing and reduced order modeling aspects of the dissertation into a single pipeline for autonomous forecasting directly from sparse sensor streams. The replication of the Lyapunov spectrum of the true high-dimensional dynamics by the learned latent-space time stepper provided strong empirical evidence that this approach captures the fundamental structure of the underlying attractor.

Several observations are consistent across the individual contributions. First, the structural alignment between RNNs and dynamical systems has concrete benefits. It is what allows a trained SHRED model to perform temporal filtering without an explicit dynamical model, what makes an echo state network interpretable as a high-dimensional dynamical system whose attractors and stability properties can be analyzed with classical tools, and why the latent dynamics of a model that receives only sparse measurements at input still replicates the underlying Lyapunov spectrum. Second, treating the sensing and modeling problems jointly has important benefits. The SHRED + ESN pipeline produces better forecasts than either component alone precisely because the latent representation learned by SHRED is already structured in a way that is amenable to temporal propagation. Third, the reservoir computing paradigm, despite its simplicity, proves surprisingly competitive with fully trained networks for the class of problems considered here. This suggests that for dynamical systems modeling, the inductive bias provided by a rich, fixed nonlinear dynamical system can compensate for the lack of task-specific weight optimization, with the caveat that appropriate hyperparameter selection remains critical.

The methods developed in this dissertation have natural limitations that suggest directions for future work. The systems studied here, while spanning a range of complexities, are predominantly low- to moderate-dimensional attractors embedded in high-dimensional state spaces. Whether the SHRED + ESN forecasting pipeline and its Lyapunov spectrum replication extend to systems with significantly higher-dimensional attractors or intermittent, transient dynamics remains an open question. The reservoir computing framework, while computationally efficient, is sensitive to hyperparameter choices, and the cost of searching over reservoir configurations can become significant for complex systems. Stability is also not always guaranteed. Hybrid approaches that allow partial adaptation of reservoir weights are a promising direction for reducing this sensitivity. The MPC framework developed in

Chapter 4 considered unconstrained or simply constrained control problems, but extending it to systems with complex state constraints, longer planning horizons, or partially observed states would broaden its applicability. Additionally, the methods in this dissertation are purely data-driven, making no use of governing equations even when partial physics knowledge is available. Integrating known physics through constraints on the architecture, regularization informed by governing equations, or hybrid models that combine learned and analytical components is a natural next step that could further reduce data requirements.

As data-driven methods for dynamical systems continue to develop, a central question is not simply how to build more powerful models but how to build models whose structure reflects the structure of the phenomena they represent. The recurrent architectures and reservoir computing methods developed in this dissertation demonstrate that when the architecture of the model is aligned with the mathematics of the problem, such lightweight models can produce results that compete with much larger models.

BIBLIOGRAPHY

[1] Henry Abarbanel. *Analysis of Observed Chaotic Data*. Springer Science & Business Media, December 2012.

[2] Osama Ahmed, Felix Tennie, and Luca Magri. Prediction of chaotic dynamics and extreme events: A recurrence-free quantum reservoir computing approach. *Physical Review Research*, 6(4):043082, November 2024. Publisher: American Physical Society.

[3] Osama Ahmed, Felix Tennie, and Luca Magri. Optimal training of finitely sampled quantum reservoir computers for forecasting of chaotic dynamics. *Quantum Machine Intelligence*, 7(1):31, February 2025.

[4] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control Barrier Functions: Theory and Applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431, June 2019.

[5] Abdul Fatir Ansari, Lorenzo Stella, Ali Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Bernie Wang. Chronos: Learning the Language of Time Series. *Transactions on Machine Learning Research*, May 2024.

[6] Hassan Arbabi and Igor Mezic. Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the koopman operator. *SIAM Journal on Applied Dynamical Systems*, 16(4):2096–2126, 2017.

[7] Jorge F. Arinez, Qing Chang, Robert X. Gao, Chengying Xu, and Jianjing Zhang. Artificial Intelligence in Advanced Manufacturing: Current Status and Future Outlook. *Journal of Manufacturing Science and Engineering*, 142(110804), August 2020.

[8] Luca Bugliari Armenio, Enrico Terzi, Marcello Farina, and Riccardo Scattolini. Model Predictive Control Design for Dynamical Systems Learned by Echo State Networks. *IEEE Control Systems Letters*, 3(4):1044–1049, October 2019. Conference Name: IEEE Control Systems Letters.

[9] A. Bahi, I. Gasmi, and S. Bentrad. Deep learning for smart grid stability in energy transition. In *Proceedings of the Fourth International Conference on Technological Advances in Electrical Engineering (ICTAEE'23)*, May 2023.

[10] Joseph Bakarji, Kathleen Champion, J Nathan Kutz, and Steven L Brunton. Discovering governing equations from partial measurements with deep delay autoencoders. *arXiv preprint arXiv:2201.05136*, 2022.

[11] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, November 2004.

[12] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[13] Isabelle Bey, Daniel J. Jacob, Robert M. Yantosca, Jennifer A. Logan, Brendan D. Field, Arlene M. Fiore, Qinbin Li, Honguy Y. Liu, Loretta J. Mickley, and Martin G. Schultz. Global modeling of tropospheric chemistry with assimilated meteorology: Model description and evaluation. *Journal of Geophysical Research: Atmospheres*, 106(D19):23073–23095, 2001.

[14] Katharina Bieker, Sebastian Peitz, Steven L. Brunton, J. Nathan Kutz, and Michael Dellnitz. Deep model predictive flow control with limited sensor data and online learning. *Theoretical and Computational Fluid Dynamics*, 34(4):577–591, August 2020.

[15] Thomas Bolton and Laure Zanna. Applications of Deep Learning to Ocean Data Inference and Subgrid Parameterization. *Journal of Advances in Modeling Earth Systems*, 11(1):376–399, 2019.

[16] Fabio Bonassi, Marcello Farina, Jing Xie, and Riccardo Scattolini. On Recurrent Neural Networks for learning-based control: Recent results and ideas for future developments. *Journal of Process Control*, 114:92–104, June 2022.

[17] Fabio Bonassi, Alessio La Bella, Marcello Farina, and Riccardo Scattolini. Nonlinear MPC design for incrementally ISS systems with application to GRU networks. *Automatica*, 159:111381, January 2024.

[18] Stephen Boyd and Lieven Vandenberghe. Convex Optimization, March 2004. ISBN: 9780511804441 Publisher: Cambridge University Press.

[19] James Bradbury, Roy Frostig, Peter Hawkins, Matthew J. Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. Jax: Composable transformations of python+numpy programs. Released software, 2018. urlhttp://github.com/jax-ml/jax.

[20] Daniel Brunner, Bhavin J. Shastri, Mohammed A. Al Qadasi, H. Ballani, Sylvain Barbay, Stefano Biasi, Peter Bienstman, Simon Bilodeau, Wim Bogaerts, Fabian Böhm, G. Brennan, Sonia Buckley, Xinlun Cai, Marcello Calvanese Strinati, B. Canakci, Benoit Charbonnier, Mario Chemnitz, Yitong Chen, Stanley Cheung, Jeff Chiles, Suyeon Choi, Demetrios N. Christodoulides, Lukas Chrostowski, J. Chu, J. H. Clegg, D. Cletheroe, Claudio Conti, Qionghai Dai, Luigi Di Lauro, Nikolaos Panteleimon Diamantopoulos, Niyazi Ulas Dinc, Jacob Ewaniuk, Shanhui Fan, Lu Fang, Riccardo Franchi, Pedro Freire, Silvia Gentilini, Sylvain Gigan, Gian Luca Giorgi, C. Gkantsidis, J. Gladrow, Elena Goi, M. Goldmann, A. Grabulosa, Min Gu, Xianxin Guo, Matěj Hejda, F. Horst, Jih Liang Hsieh, Jianqi Hu, Juejun Hu, Chaoran Huang, Antonio Hurtado, Lina Jaurigue, K. P. Kalinin, Morteza Kamalian Kopae, D. J. Kelly, Mercedeh Khajavikhan, H. Kremer, Jeremie Laydevant, Joshua C. Lederman, Jongheon Lee, Daan Lenstra, Gordon H. Y. Li, Mo Li, Yuhang Li, Xing Lin, Zhongjin Lin, Mieszko Lis, Kathy Lüdge, Alessio Lugnan, Alessandro Lupo, A. I. Lvovsky, Egor Manuylovich, Alireza Marandi, Federico Marchesin, Serge Massar, Adam N. McCaughan, Peter L. McMahon, Miltiadis Moralis Pegios, Roberto Morandotti, Christophe Moser, David J. Moss, Avilash Mukherjee, Mahdi Nikdast, B. J. Offrein, Ilker Oguz, Bakhrom Oripov, G. O'Shea, Aydogan Ozcan, F. Parmigiani, Sudeep Pasricha, Fabio Pavanello, Lorenzo Pavesi, Nicola Peserico, L. Pickup, Davide Pierangeli, Nikos Pleros, Xavier Porte, Bryce A. Primavera, Paul Prucnal, Demetri Psaltis, Lukas Puts, Fei Qiao, B. Rahmani, Fabrice Raineri, Carlos A. Ríos Ocampo, Joshua Robertson, Bruno Romeira, Charles Roques Carmes, Nir Rotenberg, A. Rowstron, Steffen Schoenhardt, Russell L. T. Schwartz, Jeffrey M. Shainline, Sudip Shekhar, Anas Skalli, Mandar M. Sohoni, Volker J. Sorger, Miguel C. Soriano, James Spall, Ripalta Stabile, Birgit Stiller, Satoshi Sunada, Anastasios Tefas, Bassem Tossoun, Apostolos Tsakyridis, Sergei K. Turitsyn, Guy Van der Sande, Thomas Van Vaerenbergh, Daniele Veraldi, Guy Verschaffelt, E. A. Vlieg, Hao Wang, Tianyu Wang, Gordon Wetzstein, Logan G. Wright, Changming Wu, Chu Wu, Jiamin Wu, Fei Xia, Xingyuan Xu, Hangbo Yang, Weiming Yao, Mustafa Yildirim, S. J. Ben Yoo, Nathan Youngblood, Roberta Zambrini, Haiou Zhang, and Weipeng Zhang. Roadmap on Neuromorphic Photonics, January 2025. arXiv:2501.07917 [cs].

[21] Steven L Brunton, Bingni W Brunton, Joshua L Proctor, Eurika Kaiser, and J Nathan Kutz. Chaos as an intermittently forced linear system. *Nature communications*, 8(1):1–9, 2017.

[22] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, Cambridge, 1st edition edition, April 2019.

[23] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[24] Steven L. Brunton, Nicholas Zolman, J. Nathan Kutz, and Urban Fasel. Machine learning for sparse nonlinear modeling and control. *Annual Review of Control, Robotics, and Autonomous Systems*, 8(1):1–27, 2025.

[25] Jared L. Callaham, Kazuki Maeda, and Steven L. Brunton. Robust flow reconstruction from limited measurements via sparse representation. *Physical Review Fluids*, 4(10):103907, October 2019. Publisher: American Physical Society.

[26] Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[27] Douglas W. Carter, Francis De Voogt, Renan Soares, and Bharathram Ganapathisubramani. Data-driven sparse reconstruction of flow over a stalled aerofoil using experimental data. *Data-Centric Engineering*, 2:e5, 2021. Publisher: Cambridge University Press.

[28] W. F. Caselton and J. V. Zidek. Optimal monitoring network designs. *Statistics & Probability Letters*, 2(4):223–227, August 1984.

[29] Kathleen Champion, Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

[30] Ashesh Chattopadhyay, Pedram Hassanzadeh, and Devika Subramanian. Data-driven predictions of a multiscale Lorenz 96 chaotic system using machine-learning methods: reservoir computing, artificial neural network, and long short-term memory network. *Nonlinear Processes in Geophysics*, 27(3):373–389, July 2020. Publisher: Copernicus GmbH.

[31] Saifon Chaturantabut and Danny C. Sorensen. Nonlinear Model Reduction via Discrete Empirical Interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, January 2010. Publisher: Society for Industrial and Applied Mathematics.

[32] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31, pages 6572–6583. Curran Associates, Inc., 2018.

[33] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[34] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, December 2014. arXiv:1412.3555 [cs].

[35] Emily Clark, Travis Askham, Steven L. Brunton, and J. Nathan Kutz. Greedy Sensor Placement With Cost Constraints. *IEEE Sensors Journal*, 19(7):2642–2656, April 2019. Conference Name: IEEE Sensors Journal.

[36] Emily Clark, Steven L. Brunton, and J. Nathan Kutz. Multi-Fidelity Sensor Selection: Greedy Algorithms to Place Cheap and Expensive Sensors With Cost Constraints. *IEEE Sensors Journal*, 21(1):600–611, January 2021. Conference Name: IEEE Sensors Journal.

[37] Emily Clark, J. Nathan Kutz, and Steven L. Brunton. Sensor Selection With Cost Constraints for Dynamically Relevant Bases. *IEEE Sensors Journal*, 20(19):11674–11687, October 2020. Conference Name: IEEE Sensors Journal.

[38] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[39] Brian M. de Silva, Krithika Manohar, Emily Clark, Bingni W. Brunton, Steven L. Brunton, and J. Nathan Kutz. PySensors: A Python package for sparse sensor placement. *Journal of Open Source Software*, 6(58):2828, 2021.

[40] N. a. K. Doan, W. Polifke, and L. Magri. Short- and long-term predictions of chaotic flows and extreme events: a physics-constrained reservoir computing approach. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2253):20210135, September 2021. Publisher: Royal Society.

[41] David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[42] A. Draeger, S. Engell, and H. Ranke. Model predictive control using neural networks. *IEEE Control Systems Magazine*, 15(5):61–66, October 1995. Conference Name: IEEE Control Systems Magazine.

[43] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A. Theodorou, and James M. Rehg. Aggressive Deep Driving: Combining Convolutional Neural Networks and Model Predictive Control. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 133–142. PMLR, October 2017. ISSN: 2640-3498.

[44] Jan Drgona, Aaron Tuor, James Koch, Madelyn Shapiro, Bruno Jacob, and Draguna Vrabie. NeuroMANCER: Neural Modules with Adaptive Nonlinear Constraints and Efficient Regularizations. *GitHub*, 2023.

[45] Zlatko Drmač and Serkan Gugercin. A New Selection Operator for the Discrete Empirical Interpolation Method—Improved A Priori Error Bound and Extensions. *SIAM Journal on Scientific Computing*, 38(2):A631–A648, January 2016. Publisher: Society for Industrial and Applied Mathematics.

[46] Megan R. Ebers, Jan P. Williams, Katherine M. Steele, and J. Nathan Kutz. Leveraging arbitrary mobile sensor trajectories with shallow recurrent decoder networks for full-state reconstruction, 2023.

[47] Peter J. Ehlers, Hendra I. Nurdin, and Daniel Soh. Stochastic reservoir computers. *Nature Communications*, 16(1):3070, March 2025. Publisher: Nature Publishing Group.

[48] N. Benjamin Erichson, Lionel Mathelin, Zhewei Yao, Steven L. Brunton, Michael W. Mahoney, and J. Nathan Kutz. Shallow neural networks for fluid flow reconstruction with limited sensors. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2238):20200097, June 2020. Publisher: Royal Society.

[49] Geir Evensen. The ensemble Kalman filter: theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367, 2003.

[50] R. Everson and L. Sirovich. Karhunen–Loève procedure for gappy data. *JOSA A*, 12(8):1657–1664, August 1995. Publisher: Optica Publishing Group.

[51] Mohammad Farazmand. Sparse discrete empirical interpolation method: State estimation from few sensors. *SIAM Journal on Scientific Computing*, 46(6):A3658–A3680, 2024.

[52] Mohammad Farazmand. State estimation using sparse DEIM and recurrent neural networks. *Proceedings of the Royal Society A*, 482:20250591, 2026.

[53] Mohammad Farazmand and Arvind K. Saibaba. Tensor-based flow reconstruction from optimally located sensor measurements. *Journal of Fluid Mechanics*, 962:A27, 2023.

[54] Gerald B Folland. *Introduction to partial differential equations*, volume 102. Princeton university press, 1995.

[55] Keisuke Fujii and Kohei Nakajima. Harnessing Disordered-Ensemble Quantum Dynamics for Machine Learning. *Physical Review Applied*, 8(2):024030, August 2017. Publisher: American Physical Society.

[56] Mars Liyao Gao, Jan P. Williams, and J. Nathan Kutz. Sparse identification of nonlinear dynamics and koopman operators with shallow recurrent decoder networks. *Proceedings of the National Academy of Sciences*, 123(16):e2508144123, 2026.

[57] Daniel J. Gauthier, Erik Bollt, Aaron Griffith, and Wendson A. S. Barbosa. Next generation reservoir computing. *Nature Communications*, 12(1):5564, September 2021. Publisher: Nature Publishing Group.

[58] Matan Gavish and David L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, 2014.

[59] William Gilpin. Deep reconstruction of strange attractors from time series. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 204–216. Curran Associates, Inc., 2020.

[60] Francisco J. Gonzalez and Maciej Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. *arXiv preprint arXiv:1808.01346*, 2018.

[61] Issa Hammoud, Sebastian Hentzelt, Thimo Oehlschlaegel, and Ralph Kennel. Long-Horizon Direct Model Predictive Control Based on Neural Networks for Electrical Drives. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pages 3057–3064, October 2020. ISSN: 2577-1647.

[62] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. ISSN: 1063-6919.

[63] John D. Hedengren, Reza Asgharzadeh Shishavan, Kody M. Powell, and Thomas F. Edgar. Nonlinear modeling, estimation and predictive control in APMonitor. *Computers & Chemical Engineering*, 70:133 – 148, 2014. Manfred Morari Special Issue.

[64] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, January 2021.

[65] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, January 2021.

[66] Lukas Hewing, Julen Kabzan, and Melanie N. Zeilinger. Cautious model predictive control using Gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2020.

[67] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[68] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[69] Philip Holmes, John L. Lumley, Gahl Berkooz, and Clarence W. Rowley. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, 2nd edition, 2012.

[70] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[71] Keke Huang, Ke Wei, Fanbiao Li, Chunhua Yang, and Weihua Gui. LSTM-MPC: A Deep Learning Based Predictive Control Method for Multimode Process Control. *IEEE Transactions on Industrial Electronics*, 70(11):11544–11554, November 2023. Conference Name: IEEE Transactions on Industrial Electronics.

[72] James M. Hyman, Basil Nicolaenko, and Stéphane Zaleski. Order and complexity in the Kuramoto–Sivashinsky model of weakly turbulent interfaces. *Physica D: Nonlinear Phenomena*, 23(1–3):265–292, 1986.

[73] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks – with an Erratum note. *GMD Report 148*, 2001.

[74] Herbet Jaeger. "the 'echo state' approach to analyzing and training recurrent neural networks". Technical report, German National Research Center for Information Technology, Technical Report GMD 148, 2001.

[75] Byung-Ki Jeon and Eui-Jong Kim. LSTM-Based Model Predictive Control for Optimal Temperature Set-Point Planning. *Sustainability*, 13(2):894, January 2021. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[76] Jean P. Jordanou, Eduardo Camponogara, Eric Aislan Antonelo, and Marco Aurélio Schmitz Aguiar. Nonlinear Model Predictive Control of an Oil Well with Echo State Networks. *IFAC-PapersOnLine*, 51(8):13–18, January 2018.

[77] Siddharth Joshi and Stephen Boyd. Sensor Selection via Convex Optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, February 2009. Conference Name: IEEE Transactions on Signal Processing.

[78] E. Kaiser, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2219):20180335, November 2018. Publisher: Royal Society.

[79] Lev Kakasenko, Alen Alexanderian, Mohammad Farazmand, and Arvind Krishna Saibaba. Bridging the gap between deterministic and probabilistic approaches to state estimation. *Physica D: Nonlinear Phenomena*, 490:135150, 2026.

[80] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[81] Mason Kamb, Eurika Kaiser, Steven L Brunton, and J Nathan Kutz. Time-delay observables for koopman: Theory and applications. *SIAM Journal on Applied Dynamical Systems*, 19(2):886–917, 2020.

[82] Niharika Karnik, Yash Bhangale, Mohammad G. Abdo, Andrei A. Klishin, Joshua J. Cogliati, Bingni W. Brunton, J. Nathan Kutz, Steven L. Brunton, and Krithika Manohar. PySensors 2.0: A Python package for sparse sensor placement. *arXiv preprint arXiv:2509.08017*, 2025.

[83] Niharika Karnik, Congjian Wang, Palash K. Bhowmik, Joshua J. Cogliati, Silvino A. Balderrama Prieto, Changhu Xing, Andrei A. Klishin, Richard Skifton, Musa Moussaoui, Charles P. Folsom, Joe J. Palmer, Piyush Sabharwall, Krithika Manohar, and Mohammad G. Abdo. Leveraging optimal sparse sensor placement to aggregate a network of digital twins for nuclear subsystems. *Energies*, 17(13), 2024.

[84] Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. Deep Encoder, Shallow Decoder: Reevaluating Non-autoregressive Machine Translation, June 2021. arXiv:2006.10369 [cs].

[85] Saeed Ahmed Khan, Fangjun Hu, Gerasimos Angelatos, and Hakan E. Türeci. Physical reservoir computing using finitely-sampled quantum systems, November 2021. arXiv:2110.13849 [quant-ph].

[86] Patrick Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

[87] Patrick Kidger and Cristian Garcia. Equinox: Neural networks in jax via callable pytrees and filtered transformations. Differentiable Programming Workshop at NeurIPS, 2021.

[88] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[89] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

[90] Andrei A. Klishin, J. Nathan Kutz, and Krithika Manohar. Data-induced interactions of sparse sensors using statistical physics. *arXiv preprint arXiv:2307.11838*, 2023.

[91] Huan Yee Koh, Yizhen Zheng, Madeleine Yang, Rohit Arora, Geoffrey I. Webb, Shirui Pan, Li Li, and George M. Church. AI-driven protein design. *Nature Reviews Bioengineering*, pages 1–23, September 2025. Publisher: Nature Publishing Group.

[92] Basil Kouvaritakis and Mark Cannon. *Model Predictive Control: Classical, Robust, and Stochastic*. Advanced Textbooks in Control and Signal Processing. Springer, Cham, 2016.

[93] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *The Journal of Machine Learning Research*, 9:235–284, June 2008.

[94] Yoshiki Kuramoto. Diffusion-induced chaos in reaction systems. *Progress of Theoretical Physics Supplement*, 64:346–367, 1978.

[95] J. Nathan Kutz. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*. Oxford University Press, Oxford, 1st edition edition, September 2013.

[96] J. Nathan Kutz, Steven L. Brunton, Bingni W. Brunton, and Joshua L. Proctor. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, 2016.

[97] Felix Köster, Dhruvit Patel, Alexander Wikner, Lina Jaurigue, and Kathy Lüdge. Data-informed reservoir computing for efficient time-series prediction. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(7):073109, July 2023.

[98] Alan S. Lapedes and Robert Farber. Nonlinear signal processing using neural networks: Prediction and system modelling. In *Proceedings of the IEEE International Conference on Neural Networks*, 1987. OSTI Report Number 5470451.

[99] Xuan-Hien Le, Hung Viet Ho, Giha Lee, and Sungho Jung. Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting. *Water*, 11(7):1387, July 2019. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[100] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[101] Ching-Hung Lee and Ching-Cheng Teng. Identification and control of dynamic systems using recurrent fuzzy neural networks. *IEEE Transactions on Fuzzy Systems*, 8(4):349–366, August 2000. Conference Name: IEEE Transactions on Fuzzy Systems.

[102] Kookjin Lee and Kevin T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

[103] Seungjoon Lee, Felix Dietrich, George E Karniadakis, and Ioannis G Kevrekidis. Linking gaussian process regression with data-driven manifold embeddings for nonlinear data fusion. *Interface focus*, 9(3):20180083, 2019.

[104] Yi Li, Eric Perlman, Minping Wan, Yunke Yang, Charles Meneveau, Randal Burns, Shiyi Chen, Alexander Szalay, and Gregory Eyink. A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, 9:N31, January 2008.

[105] Yuxuan Liang, Haomin Wen, Yuqi Nie, Yushan Jiang, Ming Jin, Dongjin Song, Shirui Pan, and Qingsong Wen. Foundation Models for Time Series Analysis: A Tutorial and Survey. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pages 6555–6565, New York, NY, USA, August 2024. Association for Computing Machinery.

[106] Liangyue Cao, Yiguang Hong, Haiping Fang, and Guowei He. Predicting chaotic time series with wavelet networks. *Physica D: Nonlinear Phenomena*, 85(1):225–238, July 1995.

[107] Benjamin Lindemann, Timo Müller, Hannes Vietz, Nasser Jazdi, and Michael Weyrich. A survey on long short-term memory networks for time series prediction. *Procedia CIRP*, 99:650–655, January 2021.

[108] D. V. Lindley. On a Measure of the Information Provided by an Experiment. *The Annals of Mathematical Statistics*, 27(4):986–1005, 1956. Publisher: Institute of Mathematical Statistics.

[109] Zhixin Lu, Jaideep Pathak, Brian Hunt, Michelle Girvan, Roger Brockett, and Edward Ott. Reservoir observers: Model-free inference of unmeasured variables in chaotic systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(4):041102, April 2017.

[110] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[111] Mantas Lukoševičius. A Practical Guide to Applying Echo State Networks. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade: Second Edition*, pages 659–686. Springer, Berlin, Heidelberg, 2012.

[112] Bethany Lusch, J. Nathan Wehmeyer, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018.

[113] Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences*, 69(4):593–616, December 2004.

[114] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14(11):2531–2560, November 2002.

[115] Krithika Manohar, Bingni W. Brunton, J. Nathan Kutz, and Steven L. Brunton. Data-Driven Sparse Sensor Placement for Reconstruction: Demonstrating the Benefits of Exploiting Known Patterns. *IEEE Control Systems Magazine*, 38(3):63–86, June 2018. Conference Name: IEEE Control Systems Magazine.

[116] Krithika Manohar, Eurika Kaiser, Steven L. Brunton, and J. Nathan Kutz. Optimized Sampling for Multiscale Dynamics. *Multiscale Modeling & Simulation*, 17(1):117–136, January 2019. Publisher: Society for Industrial and Applied Mathematics.

[117] Francesco Martinuzzi, Chris Rackauckas, Anas Abdelrehim, Miguel D. Mahecha, and Karin Mora. ReservoirComputing.jl: An efficient and modular library for reservoir computing models. *Journal of Machine Learning Research*, 23(288):1–8, 2022. http://jmlr.org/papers/v23/22-0611.html.

[118] Daniele Masti and Alberto Bemporad. Learning nonlinear state–space models using autoencoders. *Automatica*, 129:109666, July 2021.

[119] Ali Mesbah. Stochastic Model Predictive Control: An Overview and Perspectives for Future Research. *IEEE Control Systems Magazine*, 36(6):30–44, December 2016. Conference Name: IEEE Control Systems Magazine.

[120] Ali Momeni, Babak Rahmani, Benjamin Scellier, Logan G. Wright, Peter L. McMahon, Clara C. Wanjura, Yuhang Li, Anas Skalli, Natalia G. Berloff, Tatsuhiro Onodera, Ilker Oguz, Francesco Morichetti, Philipp del Hougne, Manuel Le Gallo, Abu Sebastian, Azalia Mirhoseini, Cheng Zhang, Danijela Marković, Daniel Brunner, Christophe Moser, Sylvain Gigan, Florian Marquardt, Aydogan Ozcan, Julie Grollier, Andrea J. Liu, Demetri Psaltis, Andrea Alù, and Romain Fleury. Training of Physical Neural Networks, June 2024. arXiv:2406.03372 [physics].

[121] Jeremy Morton, Freddie D. Witherden, Antony Jameson, and Mykel J. Kochenderfer. Deep dynamical modeling and control of unsteady fluid flows. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 9278–9288, Red Hook, NY, USA, December 2018. Curran Associates Inc.

[122] Pere Mujal, Rodrigo Martínez-Peña, Gian Luca Giorgi, Miguel C. Soriano, and Roberta Zambrini. Time-series quantum reservoir computing with weak and projective measurements. *npj Quantum Information*, 9(1):1–10, February 2023. Publisher: Nature Publishing Group.

[123] Stefano Nardo. PyTorch-ESN: An echo state network module for PyTorch. GitHub repository, 2018. https://github.com/stefanonardo/pytorch-esn.

[124] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, March 1990.

[125] Mario Ohlberger and Stephan Rave. Reduced basis methods: Success, limitations and future challenges. *Proceedings of the Conference Algoritmy*, pages 1–12, 2016.

[126] Amel Ourici and Bahi Abderaouf. Maximum Power Point Tracking in A Photovoltaic System Based on Artificial Neurons. *Indian Journal of Science and Technology*, 16(23):1760–1767, January 2023. Publisher: The Indian Society of Education and Environment.

[127] Amel Ourici and Abderaouf Bahi. Maximum power point tracking in a photovoltaic system based on artificial neurons. *Indian Journal of Science and Technology*, 16(23):1760–1767, 2023.

[128] Elise Özalp, Georgios Margazoglou, and Luca Magri. Reconstruction, forecasting, and stability of chaotic dynamics from partial data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(9):093107, 2023.

[129] Elise Özalp, Andrea Nóvoa, and Luca Magri. Real-time forecasting of chaotic dynamics from sparse data and autoencoders. *Computer Methods in Applied Mechanics and Engineering*, 450:118600, 2025.

[130] Elise Ozalp, Andrea Novoa, and Luca Magri. Real-time forecasting of chaotic dynamics from sparse data and autoencoders. *Computer Methods in Applied Mechanics and Engineering*, 450:118600, 2026.

[131] Shaowu Pan and Karthik Duraisamy. On the structure of time-delay embedding in linear models of non-linear dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(7):073135, 2020.

[132] Yunpeng Pan and Jun Wang. Model Predictive Control of Unknown Nonlinear Dynamical Systems Based on Recurrent Neural Networks. *IEEE Transactions on Industrial Electronics*, 59(8):3089–3101, August 2012. Conference Name: IEEE Transactions on Industrial Electronics.

[133] Liam Paninski. Asymptotic theory of information-theoretic experimental design. *Neural Computation*, 17(7):1480–1507, July 2005.

[134] Eric J. Parish and Kevin T. Carlberg. Time-series machine-learning error models for approximate solutions to parameterized dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 365:112990, June 2020.

[135] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1310–1318, 2013.

[136] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical Review Letters*, 120(2):024102, 2018.

[137] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach. *Physical Review Letters*, 120(2):024102, January 2018.

[138] Jaideep Pathak, Zhixin Lu, Brian R. Hunt, Michelle Girvan, and Edward Ott. Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(12):121102, December 2017.

[139] Jason A. Platt, Stephen G. Penny, Timothy A. Smith, Tse-Chun Chen, and Henry D. I. Abarbanel. A systematic exploration of reservoir computing for forecasting complex spatiotemporal dynamics. *Neural Networks*, 153:530–552, September 2022.

[140] Joshua L. Proctor, Steven L. Brunton, and J. Nathan Kutz. Dynamic Mode Decomposition with Control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, January 2016. Publisher: Society for Industrial and Applied Mathematics.

[141] Richard W. Reynolds, Nick A. Rayner, Thomas M. Smith, Diane C. Stokes, and Wanqiu Wang. An Improved In Situ and Satellite SST Analysis for Climate. *Journal of Climate*, 15:1609–1625, July 2002. ADS Bibcode: 2002JCli...15.1609R.

[142] Stefano Riva, Carolina Introini, Antonio Cammi, Dean Price, Alexey Yermakov, Yue Zhao, Philippe M. Wyder, Judah Goldfeder, Jan Williams, Amy Sara Rude, Matteo Tomasetto, Joe Germany, Joseph Bakarji, Georg Maierhofer, Miles Cranmer, and J. Nathan Kutz. Ctf4nuclear: Common task framework for nuclear fission and fusion models, 2026.

[143] David E. Rumelhart and James L. McClelland. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 318–362. MIT Press, 1987. Conference Name: Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations.

[144] Shervin Sahba, Christopher C. Wilcox, Austin McDaniel, Benjamin D. Shaffer, Steven L. Brunton, and J. Nathan Kutz. Wavefront sensor fusion via shallow decoder neural networks for aero-optical predictive control. In *Interferometry XXI*, volume 12223, pages 11–17. SPIE, October 2022.

[145] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[146] Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: an engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5):1327–1349, November 2021.

[147] Paola Sebastiani and Henry P. Wynn. Maximum Entropy Sampling and Optimal Bayesian Experimental Design. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 62(1):145–157, 2000. Publisher: [Royal Statistical Society, Wiley].

[148] Alen Senanian, Sridhar Prabhu, Vladimir Kremenetski, Saswata Roy, Yingkang Cao, Jeremy Kline, Tatsuhiro Onodera, Logan G. Wright, Xiaodi Wu, Valla Fatemi, and Peter L. McMahon. Microwave signal processing using an analog quantum reservoir computer. *Nature Communications*, 15(1):7490, August 2024. Publisher: Nature Publishing Group.

[149] Gregory I. Sivashinsky. Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations. *Acta Astronautica*, 4(11–12):1177–1206, 1977.

[150] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Westview Press, 2nd edition, 2015.

[151] Floris Takens. Detecting strange attractors in turbulence. In David Rand and Lai-Sang Young, editors, *Dynamical Systems and Turbulence, Warwick 1980*, Lecture Notes in Mathematics, pages 366–381, Berlin, Heidelberg, 1981. Springer.

[152] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123, July 2019.

[153] Enrico Terzi, Fabio Bonassi, Marcello Farina, and Riccardo Scattolini. Learning model predictive control with long short-term memory networks. *International Journal of Robust and Nonlinear Control*, 31(18):8877–8896, 2021. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rnc.5519.

[154] Enrico Terzi, Fabio Bonassi, Marcello Farina, and Riccardo Scattolini. Learning model predictive control with long short-term memory networks. *International Journal of Robust and Nonlinear Control*, 31(18):8877–8896, 2021. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rnc.5519.

[155] Nathan Trouvain, Luca Pedrelli, Thanh Trung Dinh, and Xavier Hinaut. ReservoirPy: An efficient and user-friendly library to design echo state networks. In *Artificial Neural Networks and Machine Learning – ICANN 2020*, pages 494–505, 2020.

[156] Gonzalo Uribarri and Gabriel B. Mindlin. Dynamical time series embeddings in recurrent neural networks. *Chaos, Solitons & Fractals*, 154:111612, January 2022.

[157] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. arXiv:1706.03762 [cs].

[158] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6000–6010, Red Hook, NY, USA, December 2017. Curran Associates Inc.

[159] Meghana Velegar, N. Benjamin Erichson, Christoph A. Keller, and J. Nathan Kutz. Scalable diagnostics for global atmospheric chemistry using Ristretto library (version 1.0). *Geoscientific Model Development*, 12(4):1525–1539, April 2019. Publisher: Copernicus GmbH.

[160] David Verstraeten, Benjamin Schrauwen, Michiel D'Haene, and Dirk Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 2007.

[161] P. R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. Backpropagation algorithms and Reservoir Computing in Recurrent Neural Networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 126:191–217, June 2020.

[162] Pantelis R. Vlachas, Wonmin Byeon, Zhong Y. Wan, Themistoklis P. Sapsis, and Petros Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. *Proceedings of the Royal Society A: Mathematical,*

Physical and Engineering Sciences, 474(2213):20170844, May 2018. Publisher: Royal Society.

[163] Jeen-Shing Wang and Yen-Ping Chen. A fully automated recurrent neural network for unknown dynamic system identification and control. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 53(6):1363–1372, June 2006. Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers.

[164] Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[165] Alexander Wikner, Joseph Harvey, Michelle Girvan, Brian R. Hunt, Andrew Pomerance, Thomas Antonsen, and Edward Ott. Stabilizing machine learning prediction of dynamics: Novel noise-inspired regularization tested with reservoir computing. *Neural Networks*, 170:94–110, February 2024.

[166] Jan Williams, Olivia Zahn, and J. Nathan Kutz. Data-driven sensor placement with shallow decoder networks, February 2022. arXiv:2202.05330 [math].

[167] Jan P. Williams, J. Nathan Kutz, and Krithika Manohar. Reservoir computing for system identification and predictive control with limited data. arXiv preprint, 2024.

[168] Jan P. Williams, J. Nathan Kutz, and Krithika Manohar. Reservoir computing for system identification and model predictive control. *Neural Networks*, page 109031, 2026.

[169] Jan P. Williams, Dima Tretiak, Steven L. Brunton, J. Nathan Kutz, and Krithika Manohar. Openreservoircomputing: Gpu-accelerated reservoir computing in jax. *arXiv*, 2026.

[170] Jan P. Williams, Olivia Zahn, and J. Nathan Kutz. Sensing with shallow recurrent decoder networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 480(2298):20240054, 09 2024.

[171] Matthew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

[172] Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pages 53140–53164, Vienna, Austria, July 2024. JMLR.org.

[173] Philippe Martin Wyder, Judah Goldfeder, Alexey Yermakov, Yue Zhao, Stefano Riva, Jan P. Williams, David Zoro, Amy Sara Rude, Matteo Tomasetto, Joe Germany, Joseph Bakarji, Georg Maierhofer, Miles Cranmer, and J. Nathan Kutz. Common task framework for a critical evaluation of scientific machine learning algorithms. ICML 2025 Workshop on Scientific Machine Learning (CODEML), June 09 2025. OpenReview.

[174] Alexey Yermakov, Yue Zhao, Marine Denolle, Yiyu Ni, Philippe M. Wyder, Judah Goldfeder, Stefano Riva, Jan Williams, David Zoro, Amy Sara Rude, Matteo Tomasetto, Joe Germany, Joseph Bakarji, Georg Maierhofer, Miles Cranmer, and J. Nathan Kutz. The seismic wavefield common task framework, 2026.

[175] Charles D. Young and Michael D. Graham. Deep learning delay coordinate dynamics for chaotic attractors from partial observable data. *Phys. Rev. E*, 107:034215, Mar 2023.

[176] Jian Yu and Jan S. Hesthaven. Flowfield Reconstruction Method Using Artificial Neural Network. *AIAA Journal*, 57(2):482–498, February 2019. Publisher: American Institute of Aeronautics and Astronautics.

[177] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A Survey of Large Language Models, March 2025. arXiv:2303.18223 [cs].